

Documentación de Azure Quantum

Obtenga información sobre la computación cuántica con el servicio Azure Quantum. Use Python y Q#, un lenguaje de programación cuántica, para crear y enviar programas cuánticos en el Azure Portal, o bien configure su propio entorno de desarrollo local con el kit de desarrollo de Quantum (QDK) para escribir programas cuánticos.

Acerca de Azure Quantum

INFORMACIÓN GENERAL

[¿Qué es Azure Quantum?](#)

[Planes de precios de Azure Quantum](#)

IMPLEMENTAR

[Creación de un área de trabajo de Quantum](#)

REFERENCIA

[Lista de proveedores de hardware cuántico](#)

CURSOS

[Ruta de aprendizaje: introducción a Azure Quantum](#)

Introducción a Q# y el Kit de desarrollo de Quantum

DESCARGAR

[Configuración del kit de desarrollo de Quantum](#)

INFORMACIÓN GENERAL

[¿Qué es Q#?](#)

INICIO RÁPIDO

[Creación del primer programa de Q#](#)

TUTORIAL

[Creación de un generador de números aleatorios cuánticos](#)

[Exploración del entrelazamiento cuántico](#)

Desarrollo avanzado con el Kit de desarrollo de Quantum



INTRODUCCIÓN

[Diferentes formas de ejecutar programas de Q#](#)



INFORMACIÓN GENERAL

[Introducción al estimador de recursos de Azure Quantum](#)

[Introducción a la computación cuántica híbrida](#)

Más información sobre la computación cuántica



CONCEPTO

[¿Qué es la computación cuántica?](#)

[El qubit](#)

[Vectores y matrices](#)

[Notación de Dirac](#)

[Circuitos cuánticos](#)

[Corrección de errores cuánticos](#)

¿Qué es Azure Quantum?

Azure Quantum es el servicio de computación cuántica en la nube de Microsoft Azure. Azure Quantum proporciona una ruta de acceso abierta, flexible y a prueba de futuro a la computación cuántica que se adapta a su manera de trabajar.

Azure Quantum ofrece una gama de soluciones de computación cuántica, incluido el hardware cuántico de proveedores líderes del sector, software cuántico y servicios cuánticos. Con Azure Quantum, puede ejecutar programas cuánticos en hardware cuántico real, simular algoritmos cuánticos y calcular los recursos necesarios para ejecutar los programas cuánticos en máquinas cuánticas escaladas futuras.

Para más información sobre cómo puede usar la computación cuántica y los algoritmos cuánticos, consulte [¿Qué es la computación cuántica?](#)

Cómo comenzar con Azure Quantum

La manera de comenzar con Azure Quantum depende de tu configuración y requisitos actuales. Tanto si es desarrollador como si ya tiene una Azure cuenta, hay diferentes maneras de comenzar su viaje con Azure Quantum. En la tabla siguiente se proporcionan instrucciones basadas en el tipo de usuario que es:

 Expandir tabla

Tipo de usuario	Cómo empezar
No tengo una Azure cuenta y no soy desarrollador	Visite el Microsoft sitio web de Quantum .
No tengo una Azure cuenta y soy desarrollador	Instalación de Azure Quantum Development Kit para Visual Studio Code
Tengo una Azure cuenta	Cree un Azure Quantum área de trabajo . A continuación, use el Azure portal o instale la extensión Quantum Development Kit para Visual Studio Code .

No es necesario tener una Azure cuenta para usar Azure Quantum. Sin embargo, si desea enviar los programas cuánticos al hardware cuántico real en Azure Quantum, debe tener una cuenta Azure y un espacio de trabajo Azure Quantum.

Para obtener una Azure cuenta, regístrese gratis y suscríbase a una [suscripción de pago por uso](#) . Si es estudiante, puede aprovechar una [cuenta gratuita Azure para los alumnos](#) .

El Microsoft sitio web de Quantum

El [Microsoft sitio web de Quantum](#) es un recurso central donde puede explorar la computación cuántica. Puede interactuar con Copilot en Azure Quantum, un agente de inteligencia artificial centrado en conceptos cuánticos que le ayuda a escribir código y comprender mejor los principios de la física cuántica. También puede aprender de expertos y entusiastas a través de blogs, artículos y vídeos.

Puede probar los ejemplos de código de Q# en el [editor de código en línea](#), enviar el código al emulador de Quantinuum basado en la nube y abrir el código en [VS Code para que web](#) funcione en un entorno cuántico preconfigurado.

El Microsoft sitio web de Quantum es gratuito y no requiere una Azure cuenta. Para empezar, solo necesita una Microsoft cuenta de correo electrónico (MSA). Para obtener más información, vea [Explorar Copilot en Azure Quantum](#).

Visual Studio Code

Azure Quantum ofrece el Quantum Development Kit (QDK), un kit de desarrollo de software diseñado específicamente para el desarrollo cuántico. Con el QDK, puede escribir programas en diferentes lenguajes de programación cuántica, depurar su código, obtener retroalimentación del código en tiempo real y enviar tareas a hardware cuántico real por medio de Azure Quantum. El QDK admite Microsoftel lenguaje de programación Q#, junto con otros lenguajes como Qiskit, Cirq y OpenQASM.

es Quantum Development Kit gratuito y de código abierto. Para empezar, instale la extensión QDK en Visual Studio Code (VS Code). Para más información, consulte [Configurar la extensión QDK](#).

ⓘ Nota

Se requiere un Azure Quantum área de trabajo para ejecutar los programas cuánticos locales en Azure Quantum el hardware del proveedor. Para obtener más información, consulte [Creación de un Azure Quantum área de trabajo](#).

El Azure portal

Si tiene una Azure cuenta, use el [Azure portal](#) para crear un Azure Quantum área de trabajo. Un Azure Quantum área de trabajo es una colección de recursos asociados a la ejecución de programas cuánticos. Para obtener más información, consulte [Creación de un Azure Quantum área de trabajo](#).

Con el Azure portal, puede enviar sus programas cuánticos para ejecución en hardware cuántico real, administrar su Azure Quantum espacio de trabajo, ver información sobre sus trabajos cuánticos y supervisar sus programas cuánticos.

¿Qué es Q#?

Q# es un lenguaje de programación cuántico de código abierto creado por Microsoft para desarrollar y ejecutar los programas cuánticos.

Puede pensar en un programa cuántico como un conjunto de subrutinas clásicas que interactúan con un sistema cuántico para realizar un cálculo. Un programa de Q# no modela directamente el estado cuántico, sino que describe cómo interactúa un equipo de control clásico con cúbits. En otras palabras, Q# es independiente del hardware, por lo que no es necesario tener en cuenta las tecnologías de cúbit reales al escribir programas de Q#. El código de Q# se ejecutará en cualquier tecnología de hardware cuántico.

Q# es un lenguaje independiente que ofrece un alto nivel de abstracción. No hay ninguna noción de un estado cuántico o circuito. En su lugar, Q# implementa programas en términos de instrucciones y expresiones, de forma muy similar a los lenguajes de programación clásicos. Puede integrar sin problemas estructuras de computación clásica y cuántica en el código de Q#.

Para obtener más información, vea [Introducción a Q#](#). Para empezar a escribir código de Q#, consulte [Creación del primer programa](#) de Q#.

¿Qué puedo hacer con Azure Quantum?

Azure Quantum ofrece una amplia gama de servicios y herramientas para ayudarle a desarrollar soluciones cuánticas.

Para obtener la información más reciente sobre Microsoft la investigación de la computación cuántica, consulte la [Microsoft página Investigación de computación cuántica](#) .

Computación cuántica híbrida

La computación cuántica híbrida hace referencia a los procesos y la arquitectura de un equipo clásico y un equipo cuántico que trabaja conjuntamente para resolver un problema. Con la última generación de arquitectura de equipo cuántico híbrido disponible en Azure Quantum, puede empezar a trabajar con un enfoque híbrido clásico-cuántico para la programación.

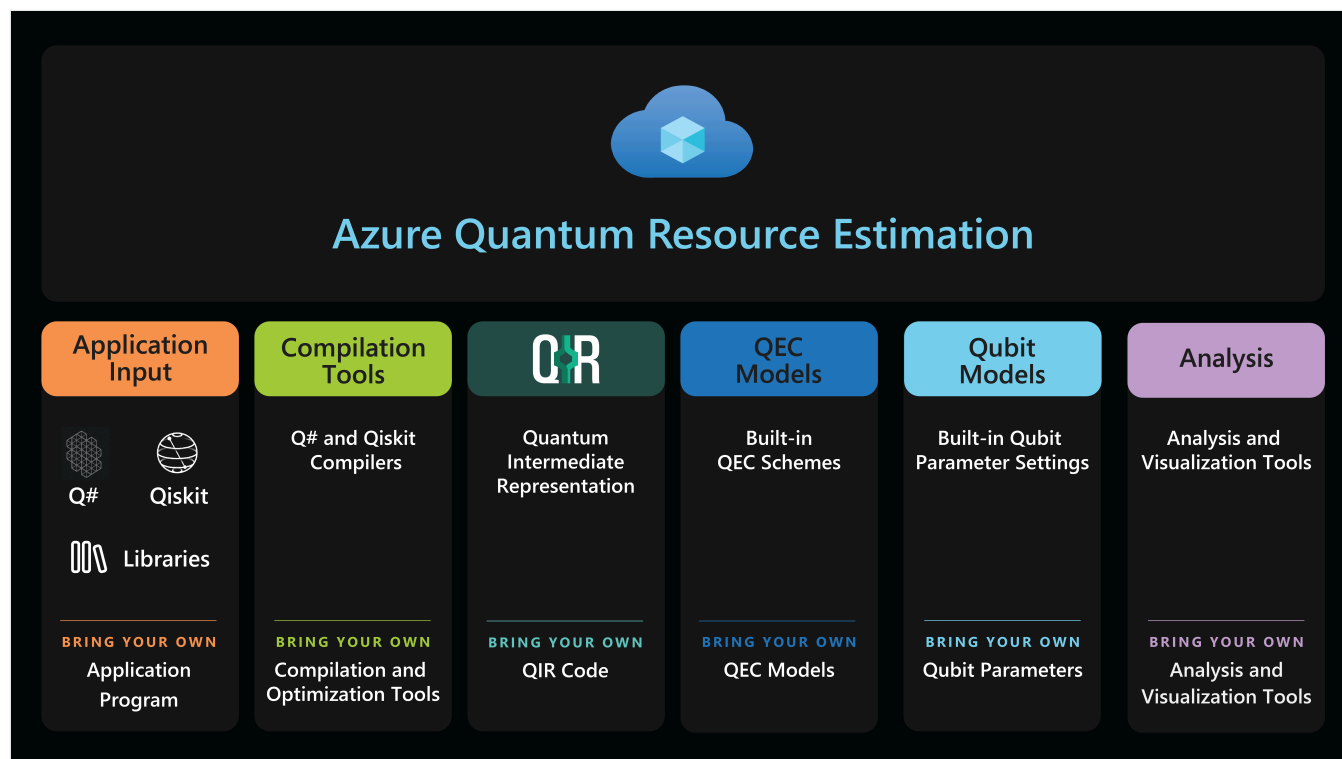
Para más información, consulte [Computación](#) cuántica híbrida.

Estimación de recursos en la computación cuántica

En la computación cuántica, la estimación de recursos es la capacidad de comprender los recursos necesarios para ejecutar un algoritmo en un equipo cuántico. Cuando comprenda los requisitos de recursos para ejecutar los programas en diferentes tipos de hardware cuántico, puede preparar y refinar las soluciones cuánticas para que se ejecuten en máquinas cuánticas a escala futuras. Por ejemplo, la estimación de recursos puede ayudarle a determinar la viabilidad de interrumpir un algoritmo de cifrado determinado en un equipo cuántico.

El [Azure Quantum estimador](#) de recursos permite evaluar las decisiones arquitectónicas, comparar las tecnologías de cúbits y determinar los recursos que necesita para ejecutar un algoritmo cuántico determinado. Puede elegir entre protocolos predefinidos de tolerancia a errores o establecer sus propios parámetros de estimación. El Estimador de recursos calcula las estimaciones de recursos físicos posteriores al diseño según un conjunto de entradas, como parámetros de cúbit, el código de corrección cuántica de errores (QEC), el presupuesto de errores y [otros parámetros](#).

Para comenzar, consulte [Ejecutar la primera estimación de recursos](#).



Simulaciones de química cuántica con Azure Quantum

La mecánica cuántica es como el sistema operativo subyacente de nuestro universo y describe cómo se comportan los bloques de creación fundamentales de la naturaleza. Las reacciones químicas, los procesos celulares y las propiedades materiales son toda mecánica cuántica por naturaleza, y a menudo implican interacciones entre un gran número de partículas cuánticas. Los equipos cuánticos prometen simular sistemas mecánicos cuánticos intrínsecamente, como

moléculas, ya que los cúbits se pueden usar para representar los estados cuánticos naturales en estos sistemas. Entre los ejemplos de sistemas cuánticos que podemos modelar se incluyen la fotoestésis, la superconductividad y las formaciones moleculares complejas.

El QDK y Azure Quantum están diseñados específicamente para acelerar el descubrimiento científico. Reinventa la productividad de investigación y desarrollo con flujos de trabajo de simulación optimizados para el escalado en Azure clústeres de computación de alto rendimiento (HPC), computación acelerada por inteligencia artificial, integración con herramientas cuánticas y hardware cuántico, y acceso futuro a la supercomputadora cuántica de Microsoft.

Para obtener más información, consulte [Desbloquear el poder de Azure para Dinámica Molecular](#) .

Velocidad cuántica

Los equipos cuánticos hacen excepcionalmente bien con problemas que requieren cálculos de un gran número de combinaciones posibles. Estos tipos de problemas se pueden encontrar en muchas áreas, como simulación cuántica, criptografía, aprendizaje automático cuántico y problemas de búsqueda.

Uno de los objetivos de la investigación de computación cuántica es estudiar qué tipos de problemas se pueden resolver mediante un equipo cuántico más rápido que un equipo clásico y el tamaño que puede tener la velocidad. Un ejemplo conocido es el algoritmo de Grover, que ofrece una aceleración polinómica en comparación con los algoritmos clásicos.

[El algoritmo de Grover](#) acelera la solución de las búsquedas de datos no estructurados, ejecutando la búsqueda en menos pasos que cualquier algoritmo clásico. De hecho, cualquier problema que le permita comprobar si un valor determinado es una solución válida (un "sí o ningún problema") se puede formular en términos del problema de búsqueda.

Para obtener una implementación del algoritmo de Grover, consulte [Tutorial: Implementación del algoritmo de búsqueda de Grover en Q#](#).

Proveedores cuánticos disponibles en Azure Quantum

Azure Quantum ofrece algunos de los recursos cuánticos más atractivos y diversos disponibles hoy en día desde líderes del sector. Azure Quantum actualmente se asocia con los siguientes proveedores para permitirle ejecutar los programas cuánticos en hardware real y la opción de probar el código en simuladores de hardware.

Elija el proveedor que mejor se adapte a las características de su problema y sus necesidades.

- [IONQ](#): Computadoras cuánticas de iones atrapados reconfigurables dinámicamente para hasta 36 cúbits totalmente conectados, que permite ejecutar una puerta de dos cúbits entre cualquier par.
- [PASQAL](#): procesadores cuánticos neutros basados en átomos que funcionan a temperatura ambiente, con tiempos de coherencia largos e impresionante conectividad de cúbits.
- [Quantinuum](#): sistemas de iones atrapados con alta fidelidad, cúbits totalmente conectados, tasas de error bajas, reutilización de cúbits y la capacidad de realizar mediciones de circuitos intermedios.
- [Rigetti](#): con tecnología de procesadores cuánticos basados en cúbits superconductores, estos sistemas ofrecen tiempos de puerta rápidos, lógica condicional de baja latencia y tiempos de ejecución rápidos del programa.

Para obtener más información sobre las especificaciones de cada proveedor, consulte la lista completa de computación cuántica.

Para obtener información sobre el costo del trabajo, consulte [Precios en Azure Quantum](#) y [preguntas más frecuentes: Descripción de los costos y la facturación de trabajos en Azure Quantum](#).

Contenido relacionado

Para empezar a usar Azure Quantum, explore los vínculos siguientes:

- [Creación de un área de Azure Quantum trabajo](#)
- [Introducción a Q# y Visual Studio Code](#)
- [Instalación del Quantum Development Kit](#)
- [Ejecución de un circuito Qiskit en Azure Quantum](#)

Last updated on 30/10/2025

¿Qué es la computación cuántica?

29/09/2025

La computación cuántica tiene la promesa de resolver algunos de los mayores desafíos de nuestro planeta, en las áreas del medio ambiente, la agricultura, la salud, la energía, el clima, la ciencia de materiales, etc. Para algunos de estos problemas, la computación clásica tiene cada vez más dificultades a medida que aumenta el tamaño del sistema. Cuando se diseña para escalar, es probable que los sistemas cuánticos tengan capacidades que superen las de los superequipos más potentes de hoy en día.

En este artículo se explican los principios de la computación cuántica, cómo se compara con la computación clásica y cómo se usan los principios de la mecánica cuántica.

Historial de la computación cuántica

Los sistemas cuánticos, como átomos y moléculas, pueden ser difíciles o imposibles de simular en un equipo clásico. En los años 80, Richard Feynman y Yuri Manin sugirieron que el hardware basado en fenómenos cuánticos podría ser más eficaz para la simulación de sistemas cuánticos que los equipos convencionales.

Hay varias razones por las que los sistemas cuánticos son difíciles de simular en equipos normales. Una razón principal es que la materia, a nivel cuántico, se describe como una combinación de varias configuraciones (conocidas como estados) al mismo tiempo.

Los estados cuánticos crecen exponencialmente

Considere un sistema de partículas y 40 posibles ubicaciones donde esas partículas pueden existir. El sistema podría estar en cualquiera de *losestadosúnicosde* 2^{40} porque cada ubicación puede tener o no una partícula. Si estas son partículas clásicas, el sistema siempre está en uno de los 2^{40} estados, por lo que un equipo clásico solo necesita 40 bits para describir el estado del sistema. Pero si son partículas cuánticas, el sistema existe en una combinación de todos los *estados* 2^{40} . Un equipo clásico debe almacenar *númerosde* 2^{40} para describir el sistema cuántico, que requiere más de 130 GB de memoria. Sin embargo, un equipo cuántico solo necesita 40 bits cuánticos para describir este sistema cuántico.

Si agregamos otra ubicación al sistema para que los electrones puedan existir en 41 ubicaciones, el número de configuraciones únicas del sistema se duplica a 2^{41} . Tardaría más de 260 GB de memoria en almacenar ese estado cuántico en un equipo clásico. No podemos jugar este juego de aumentar el número de ubicaciones para siempre. Para almacenar un estado cuántico en un equipo convencional, se supera rápidamente la capacidad de memoria de las máquinas más potentes del mundo. En unos cientos de electrones, la memoria necesaria

para almacenar el sistema supera el número de partículas del universo. ¡No hay esperanza con nuestros equipos convencionales para simular completamente la dinámica cuántica para sistemas más grandes!

Convertir una dificultad en oportunidad

La observación de este crecimiento exponencial plantea una pregunta eficaz: ¿es posible convertir esta dificultad en una oportunidad? Si los sistemas cuánticos son difíciles de simular en equipos normales, ¿qué ocurriría si creamos una máquina que usa efectos cuánticos para sus operaciones fundamentales? ¿Podríamos simular sistemas cuánticos con una máquina que aproveche exactamente las mismas leyes de física? ¿Y podríamos usar esa máquina para investigar otros problemas importantes fuera de la mecánica cuántica? Estos son los tipos de preguntas que dieron lugar a los campos de información cuántica y computación cuántica.

En 1985, David Deutsch mostró que un equipo cuántico podría simular eficazmente el comportamiento de cualquier sistema físico. Esta detección fue la primera indicación de que los equipos cuánticos podrían usarse para resolver problemas que son demasiado difíciles de resolver en equipos clásicos.

En 1994, Peter Shor descubrió un algoritmo cuántico para encontrar los factores primos de enteros grandes. El algoritmo de Shor se ejecuta exponencialmente más rápido que el algoritmo clásico más conocido para este problema de factorización. Este algoritmo rápido podría romper muchos de nuestros sistemas criptográficos de clave pública modernos que usamos para proteger las transacciones en el comercio electrónico, como Rivest–Shamir–Adleman (RSA) y Criptografía de curva elíptica. Este descubrimiento despertó un enorme interés por la computación cuántica y condujo al desarrollo de algoritmos cuánticos para muchos otros problemas.

Desde ese momento, se desarrollaron algoritmos de equipos cuánticos rápidos y eficientes para otros problemas difíciles de resolver en equipos clásicos. Por ejemplo, ahora tenemos algoritmos cuánticos para buscar en una base de datos no ordenada, para resolver sistemas de ecuaciones lineales, para realizar el aprendizaje automático y para simular sistemas físicos en química, física y ciencia de materiales.

¿Qué es un cúbit?

Al igual que los bits son el objeto fundamental de la información en la computación clásica, los cúbits (bits cuánticos) son el objeto fundamental de la información en la computación cuántica.

Los cúbits desempeñan un papel similar en la computación cuántica como bits en la computación clásica, pero los cúbits se comportan de forma diferente a los bits. Los bits clásicos son binarios y, en un momento dado, solo pueden estar en uno de dos estados, 0 o 1.

Pero los cúbits pueden estar en una superposición de los estados 0 y 1 al mismo tiempo. De hecho, hay posibles superposiciones infinitas de 0 y 1, y cada una de ellas es un estado de cúbit válido.

En la computación cuántica, la información se codifica en superposiciones de los estados 0 y 1. Por ejemplo, 8 bits normales pueden codificar hasta 256 valores únicos, pero estos 8 bits solo pueden representar uno de los 256 valores a la vez. Con 8 cúbits, podríamos codificar todos los 256 valores al mismo tiempo, ya que los cúbits pueden estar en una superposición de todos los 256 estados posibles.

Para obtener más información, consulte [El cúbit en la computación](#) cuántica.

¿Cuáles son los requisitos para crear un equipo cuántico?

Un equipo cuántico usa sistemas cuánticos y las propiedades de la mecánica cuántica para resolver problemas computacionales. Los sistemas de un equipo cuántico constan de los cúbits, las interacciones entre cúbits y operaciones en los cúbits para almacenar y calcular información. Podemos usar equipos cuánticos para programar efectos como el entrelazamiento cuántico y la interferencia cuántica para resolver con precisión ciertos problemas más rápido que en los equipos clásicos.

Para crear un equipo cuántico, es necesario tener en cuenta cómo crear y almacenar los cúbits. También tenemos que pensar en cómo manipular los cúbits y cómo medir los resultados de nuestros cálculos.

Entre las tecnologías de cúbits populares se incluyen cúbits de iones atrapados, cúbits superconductores y cúbits topológicos. Para algunos métodos de almacenamiento de cúbits, la unidad que hospeda los cúbits debe mantenerse a una temperatura casi cero absoluta para maximizar su coherencia y reducir la interferencia. Otros tipos de alojamiento de cúbits usan una cámara de vacío para ayudar a minimizar las vibraciones y estabilizar los cúbits. Las señales se pueden enviar a los cúbits a través de varios métodos, como microondas, láseres o voltajes.

Los cinco criterios de un equipo cuántico

Un buen equipo cuántico debe tener estas cinco características:

1. **Escalable:** puede tener muchos cúbits.
2. **Inicializable:** puede establecer los cúbits en un estado específico (normalmente el estado 0).
3. **Resistente:** puede mantener los cúbits en estado de superposición durante mucho tiempo.

4. **Universal:** un equipo cuántico no necesita realizar todas las operaciones posibles, solo un conjunto de operaciones denominadas *conjunto* universal. Un [conjunto de operaciones](#) cuánticas universales es de modo que cualquier otra operación se pueda descomponer en una secuencia de ellas.
5. **Confiable:** puede medir los cúbits con precisión.

Estos cinco criterios se conocen a menudo como criterios [de Di Vincenzo](#) para el cálculo cuántico.

La creación de dispositivos que cumplan estos cinco criterios es uno de los desafíos de ingeniería más exigentes a los que se ha enfrentado el hombre. Azure Quantum ofrece una variedad de soluciones de computación cuántica con diferentes tecnologías de cúbits. Para más información, consulte [la lista completa de proveedores](#) de Azure Quantum.

Descripción de fenómenos cuánticos

Los fenómenos cuánticos son los principios fundamentales que diferencian la computación cuántica de la computación clásica. Comprender estos fenómenos es fundamental para comprender cómo funcionan los equipos cuánticos y por qué contienen este potencial. Los dos fenómenos cuánticos más importantes son la superposición y el entrelazamiento.

Superposición

Imagine que está haciendo ejercicio en su salón. Gira completamente a la izquierda y, a continuación, gira completamente a la derecha. Ahora, gire a la izquierda y a la derecha al mismo tiempo. No se puede (no sin dividirse en dos, al menos). Obviamente, no puede estar en ambos estados a la vez: no puede estar mirando a la izquierda y a la derecha al mismo tiempo.

Sin embargo, si fuera una partícula cuántica, tendría una probabilidad determinada de estar *mirando a la izquierda* y una probabilidad determinada de estar *mirando a la derecha* debido a un fenómeno conocido como *superposición* (o también *coherencia*).

Solo los sistemas cuánticos como los iones, electrones o circuitos superconductores pueden existir en los estados de superposición que permiten la potencia de la computación cuántica. Por ejemplo, los electrones son partículas cuánticas que tienen su propia propiedad de "hacia la izquierda o hacia la derecha", denominada espín. Los dos estados de giro se denominan spin up y spin down, y el estado cuántico de un electrón es una superposición de los estados de spin up y spin down.

Si desea obtener más información y practicar con superposición, consulte [Módulo de entrenamiento: Exploración de la superposición con Q#](#).

Entrelazamiento

El [entrelazamiento](#) es una correlación cuántica entre dos o más sistemas cuánticos. Cuando dos cúbits están entrelazados, se correlacionan y comparten la información de sus estados de forma que el estado cuántico de los cúbits individuales no se pueda describir de forma independiente. Con el entrelazamiento cuántico solo se puede conocer el estado cuántico del sistema global, no los estados individuales.

Los sistemas cuánticos entrelazados mantienen esta correlación incluso cuando se separan a grandes distancias. Esto significa que cualquier operación o proceso que se aplique a un subsistema se correlaciona también con el otro subsistema. Por lo tanto, medir el estado de un cúbit proporciona información sobre el estado del otro cúbit: esta propiedad en particular es muy útil en la computación cuántica.

Si quiere obtener más información, consulte [Tutorial: Exploración del entrelazamiento cuántico con Q#](#) y, para obtener una implementación práctica, consulte [Módulo de entrenamiento: Teletransportar un cúbit mediante el entrelazamiento](#).

Contenido relacionado

- [Vectores y matrices en la computación cuántica](#)
- [Cúbit](#)
- [Convenciones de diagrama de circuitos cuánticos](#)
- [Entrelazamiento en la computación cuántica](#)
- [Introducción al lenguaje de programación cuántica Q#](#)

ⓘ **Nota:** El autor ha creado este artículo con ayuda de inteligencia artificial. [Más información](#)

Máquina cuántica de Microsoft

Artículo • 22/11/2024

Microsoft está en la ruta de acceso para entregar una máquina cuántica como parte del servicio Azure Quantum. Puede leer más sobre este recorrido en [este blog](#). Esto forma parte de un plan integral para capacitar a los innovadores con soluciones cuánticas para un impacto innovador.

En 2022, Microsoft tuvo un importante avance científico que despejó un obstáculo importante hacia la creación de una máquina cuántica escalada con cúbits topológicos. Aunque los desafíos de ingeniería siguen siendo, esta nueva detección demuestra un bloque de creación fundamental para su enfoque de la computación cuántica escalada. Para obtener más información sobre este descubrimiento científico, lea el [blog](#), vea un [vídeo](#) o lea la [preimpresión](#) del documento.

Visualización de un cuaderno de Jupyter Notebook de los resultados

El equipo de Azure Quantum ha creado un conjunto de cuadernos de Jupyter Notebook en el [sitio](#) de GitHub `azure-quantum-tgp` que muestra todos los análisis y trazados que aparecen en el documento. En la página de GitHub, también puede ver los resultados del cuaderno ejecutados con nbviewer.

Pasos siguientes

- [¿Qué es Azure Quantum?](#)
- [Introducción a Q#](#)
- [Inicio rápido: Envío de un circuito con Qiskit](#)

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Copilot en Azure Quantum

Empiece a trabajar con la computación cuántica, descubra los últimos avances cuánticos y cree y ejecute programas cuánticos con la ayuda de Copilot en Azure Quantum en el [sitio web de Microsoft Quantum](#).

Las características del sitio web de Microsoft Quantum:

- **Copilot en Azure Quantum:** IA se reúne con el mundo cuántico. Copilot en Azure Quantum es un asistente de inteligencia artificial generativa que le ayudará a aprender y explorar la computación cuántica.
- **Código en Azure Quantum:** Explora y ejecuta código Q# directamente en el navegador usando la ventana de código y el compilador integrados, y envía los trabajos al simulador en memoria o al emulador de Quantinuum gratuito. Use Copilot en Azure Quantum para explicar Q# el código y también generar código a partir de las indicaciones.
- **Quantum Katas:** [tutoriales](#) prácticos que enseñan conceptos de computación cuántica y el lenguaje de Q# programación.
- **Ejemplos de código:** la biblioteca de código de Azure Quantum es un amplio conjunto de ejemplos que usan Q# código de, Python y Qiskit para mostrar escenarios cuánticos.
- **Conceptos cuánticos:** desde los conceptos básicos de la teoría cuántica hasta las técnicas avanzadas de computación cuántica, la biblioteca De conceptos es un curso de bloqueo para ayudarle a acelerar la computación cuántica.
- **Vídeos:** Examine la biblioteca de vídeos para obtener entrevistas con investigadores de computación cuántica líderes e innovadores.
- **Blogs de Azure Quantum:** manténgase al día con las últimas innovaciones e investigación de computación cuántica.

Todo lo que necesita para empezar a explorar Azure Quantum es una cuenta de correo electrónico de Microsoft (MSA). Puede crear una MSA de forma gratuita en <https://account.microsoft.com/>.

Ejecución de un programa de Q#

Para empezar a explorar Copilot y codificar en Azure Quantum, use uno de los ejemplos de la lista desplegable **Ejemplos de Quantum**.

1. Vaya a [Código en Azure Quantum](#) mediante cualquier cuenta de Microsoft (MSA).
2. Seleccione **Ejemplos cuánticos** y, después, seleccione **Generador de números aleatorios**. El código siguiente se copia en la ventana de código.

```
Q#
```

```

/// # Sample
/// Quantum Random Number Generator
///
/// # Description
/// This program implements a quantum random number generator by setting qubits
/// in superposition and then using the measurement results as random bits.

import Std.Measurement;
import Std.Intrinsic;

operation Main() : Result[] {
    // Generate 5-bit random number.
    let nBits = 5;
    return GenerateNRandomBits(nBits);
}

/// # Summary
/// Generates N random bits.
operation GenerateNRandomBits(nBits : Int) : Result[] {
    // Allocate N qubits.
    use register = Qubit[nBits];

    // Set the qubits into superposition of 0 and 1 using the Hadamard
    // operation `H`.
    for qubit in register {
        H(qubit);
    }

    // At this point each has 50% chance of being measured in the |0> state
    // and 50% chance of being measured in the |1> state.
    // Measure each qubit and reset them all so they can be safely deallocated.
    let results = MeasureEachZ(register);
    ResetAll(register);
    return results;
}

```

3. Seleccione **Simulador en memoria**.

4. Seleccione **Ejecutar**.

- Los resultados se muestran en el campo **Resultados** y se muestra un histograma de los resultados debajo de la ventana de código.
- Puede mover el control deslizante para que **Seleccionar número de shots** especifique cuántas veces se ejecuta el programa.
- El campo **Shots** muestra el resultado de cada shot.

Para volver a ejecutar el programa con otro simulador:

1. Seleccione el desplegable del **simulador In-Memory** y seleccione **Emulador de Quantinuum**.
2. Seleccione el número de shots (actualmente limitados a 20) y seleccione **Ejecutar**.

- El estado del trabajo se muestra en la parte superior de la ventana de código.
- Un histograma de los resultados se muestra debajo de la ventana de código. Los resultados de cada captura no están disponibles actualmente con el emulador de Quantinuum.

Preguntar al Copilot

- En la ventana de código, seleccione **Explicar código** para pedir al Copilot en Azure Quantum que genere un análisis del ejemplo de código.

Puede solicitar al Copilot en Azure Quantum casi cualquier cosa cuántica relacionada, por ejemplo:

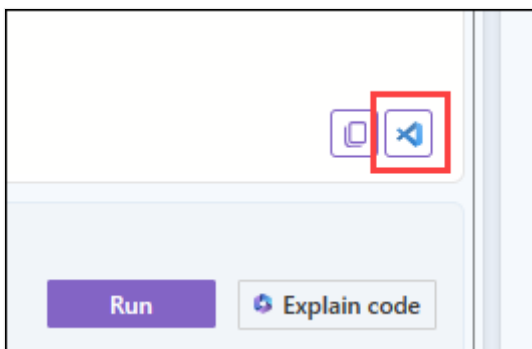
- "Explica la operación MResetZ"
- "Escribir Q# código que entrelaze dos cúbits"
- "Explica la interferencia cuántica"

Abra el código de ejemplo en VS Code para la Web

Para explorar aún más el código de ejemplo, puede abrir fácilmente el código en [VS Code para la Web](#) y aprovechar las ventajas de características como la mensajería de errores mejorada, Q# el resaltado de sintaxis y la depuración integrada. Si ya tiene configurada una cuenta de Azure, puede conectarse directamente a las áreas de trabajo de Azure Quantum desde VS Code.

Para abrir el código en VS Code para web:

1. Seleccione el icono de VS Code en la parte inferior de la ventana de código.



Pasos siguientes

También puede ejecutar programas cuánticos mediante Azure Portal o configurando un entorno de desarrollo local.

- [Creación de un área de trabajo de Azure Quantum](#)
 - [Configuración del kit de desarrollo de Quantum](#)
-

Last updated on 13/11/2025

Creación de un área de trabajo de Azure Quantum

Aprenda a crear un área de trabajo de [Azure Quantum](#) en Azure Portal. Un recurso de área de trabajo de Azure Quantum o un área de trabajo para abreviar es una colección de recursos asociados a la ejecución de aplicaciones cuánticas.

Necesita un área de trabajo para enviar programas cuánticos al hardware cuántico.

Sugerencia

También puede crear un área de trabajo de Azure Quantum mediante la interfaz de la línea de comandos (CLI) de Azure. Para más información, consulte [Administración de áreas de trabajo cuánticas con la CLI de Azure](#).

Prerrequisitos

Debe tener una cuenta de Azure con una suscripción activa para crear un área de trabajo de Azure Quantum. Si no tiene una, puede elegir una de las siguientes suscripciones o ver una lista completa, consulte Detalles de la [oferta de Microsoft Azure](#).

 Expandir tabla

Suscripción de Azure	Descripción
Pago por uso (recomendado)	Usted paga por los servicios que usa, y puede cancelar en cualquier momento.
Contrato Enterprise	Si su organización tiene un acuerdo de compra de Acuerdo Empresarial (EA) con Microsoft, los propietarios de la cuenta de su organización pueden crear suscripciones Enterprise Dev/Test para suscriptores activos de Visual Studio bajo el Acuerdo Empresarial.

Si tiene preguntas o tiene problemas al usar Azure Quantum, póngase en contacto con AzureQuantumInfo@microsoft.com.

Creación del área de trabajo

Para crear un área de trabajo de Azure Quantum, siga estos pasos.

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.

2. Seleccione **Crear un recurso** y busque **Azure Quantum**. En la página de resultados, debería ver un icono para el servicio **Azure Quantum**.
3. Seleccione **Azure Quantum** y, a continuación, seleccione **Crear**. Se abre un formulario para crear un área de trabajo.
4. Seleccione una suscripción para asociarla a la nueva área de trabajo.
5. Seleccione **Creación rápida** o **Creación avanzada**.

Creación rápida

Esta opción es la ruta de acceso más sencilla para crear un área de trabajo. Crea automáticamente el grupo de recursos y la cuenta de almacenamiento necesarios, agrega los proveedores IonQ, Quantinuum, Rigetti y Microsoft Quantum Computing. El área de trabajo todavía se puede personalizar después de la creación, si es necesario.

ⓘ Nota

Para usar la **creación rápida**, debe ser **propietario** de la suscripción que seleccionó en el paso anterior. Para consultar la lista de sus suscripciones y acceso, vea [Verificar las asignaciones de roles](#).

1. Escriba un nombre para el área de trabajo.
2. Seleccione la región del área de trabajo.
3. Seleccione **Crear**.

ⓘ Nota

Si tiene problemas durante la implementación, consulte Problemas [comunes de Azure Quantum: Creación de un área de trabajo de Azure Quantum](#).

La implementación del área de trabajo puede tardar unos minutos. Los detalles de estado e implementación se actualizarán en el portal.

ⓘ Nota

Si tiene algún problema, consulte [Problemas comunes de Azure Quantum: Creación de un área de trabajo de Azure Quantum](#).

Pasos siguientes

- [Exploración de Azure Quantum](#)
 - [Inicio rápido: Envío de un circuito con Qiskit a Azure Quantum](#)
-

Last updated on 17/10/2025

Configuración de la Quantum Development Kit extensión

En este artículo, aprenderá a instalar la extensión de Azure Quantum Development Kit (QDK) para Visual Studio Code (VS Code) y a agregar compatibilidad con Python, Jupyter Notebook y la CLI de Azure.

Requisitos previos

- La versión más reciente de [VS Code](#).
- Si desea enviar trabajos a Azure Quantum, debe tener una cuenta de Azure con un área de trabajo cuántica. Para más información, consulte [Creación de un área de trabajo de Azure Quantum](#).

Instalación de la extensión QDK

Para usar el QDK en VS Code, instale la [extensión QDK](#). También puede usar el QDK en [VS Code para web](#) sin instalar la extensión, pero no tendrá todas las características de VS Code Desktop. Para obtener más información, consulte [Diferentes formas de ejecutar Q# programas](#).

Ahora puede escribir, depurar y ejecutar programas Q# con el simulador cuántico integrado. O bien, si tiene una cuenta de Azure, puede conectarse y enviar Q# programas al hardware cuántico, todo ello desde VS Code.

Para probar la configuración, consulte [Envío Q# de trabajos a Azure Quantum](#).

Adición de compatibilidad con Python y Jupyter Notebook

Con la compatibilidad con Python en Visual Studio Code, puede insertar o llamar al Q# código desde sus programas de Python o cuadernos de Jupyter, y ejecutarlos en el simulador cuántico integrado. También puede conectarse al área de trabajo de Azure Quantum y enviar los trabajos para que se ejecuten en hardware cuántico real.

Requisitos previos

- Instale un entorno de Python (3.9 o superior, 3.11 recomendado) con [Python y Pip](#).
- Instale la extensión QDK en VS Code.

Instalación de los paquetes necesarios

Para agregar compatibilidad con Python y Jupyter Notebook:

1. Instale las [extensiones de Python](#) y [Jupyter](#) para VS Code.
2. Abra la línea de comando.
3. Instale la `qdk` biblioteca de Python con el `azure` adicional:

Símbolo del sistema de Windows

```
python -m pip install qdk[azure]
```

4. Para la compatibilidad con Qiskit o Cirq, instale `azure-quantum` con `qiskit` o `cirq` complementos:

Importante

Si va a actualizar desde un entorno de Qiskit anterior, consulte [Actualización del paquete azure-quantum con compatibilidad con Qiskit](#).

Símbolo del sistema de Windows

```
python -m pip install azure-quantum[qiskit]
```

o

Símbolo del sistema de Windows

```
python -m pip install azure-quantum[cirq]
```

o

Símbolo del sistema de Windows

```
python -m pip install azure-quantum[qiskit,cirq]
```

5. Instale paquetes opcionales de Python para mostrar los resultados y trabajar en Jupyter Notebook:

Símbolo del sistema de Windows

```
python -m pip install qdk[jupyter] ipykernel ipynb jupyterlab
```

Para probar la configuración, consulte Q# o [Envío Q#](#) de trabajos con Cuadernos de Jupyter Notebook.

Incorporación de compatibilidad con la CLI de Azure

La CLI de Azure es un método opcional para enviar trabajos cuánticos mediante una ventana de terminal en VS Code.

1. Instale la [CLI de Azure](#).
2. Abra un símbolo del sistema de Windows.
3. En el símbolo del sistema, ejecute el siguiente comando para actualizar a la extensión de la CLI `quantum` de Azure más reciente:

Símbolo del sistema de Windows

```
az extension add \  
  --upgrade \  
  -n quantum
```

Para probar la configuración, consulte [Envío Q# de trabajos a Azure Quantum](#).

Contenido relacionado

- [Diferentes formas de ejecutar Q# programas](#)
- [Inicio rápido: Creación del primer Q# programa](#)
- [Actualización del QDK a la versión más reciente](#)

Last updated on 12/11/2025

Diferentes formas de ejecutar programas de Q#

Azure Quantum ofrece diferentes opciones de desarrollo para escribir y ejecutar programas cuánticos. Cada entorno usa el [Kit de desarrollo de Quantum \(QDK\)](#), un conjunto de herramientas de código abierto que incluye el lenguaje de programación Q#. Para obtener más información, vea [Introducción a Q#](#).

En este artículo, aprenderá las diferencias entre cada opción y cómo elegir la adecuada para sus necesidades.

Opciones para ejecutar programas de Q#

Azure Quantum está disponible a través de tres entornos de desarrollo:

- **Sitio web de Microsoft Quantum:** Use Copilot para escribir, ejecutar y explicar el código de Q# en el explorador. No se requiere instalación ni cuenta de Azure.
- **Azure Portal:** Administre la suscripción de Azure y el área de trabajo de Azure Quantum, donde puede acceder a información sobre los proveedores cuánticos y los envíos de trabajos. No se requiere ninguna instalación.
- **Visual Studio Code:** Escriba, ejecute y depure código cuántico en el entorno local mediante Q# como un programa independiente o con Python en Jupyter Notebook. Se requiere la instalación.

La opción que elija para ejecutar programas de Q# depende de la experiencia de codificación, el conocimiento cuántico y los objetivos. Dado que cada opción tiene características y funcionalidades diferentes, normalmente las usa juntas, como escribir programas de Q# con la extensión QDK en VS Code mientras administra el área de trabajo cuántica en Azure Portal. Para más información, vea la tabla siguiente:

 Expandir tabla

	Sitio web de Microsoft Quantum	Portal de Azure	Visual Studio Code
Compatibilidad integrada con Q#	✓	✓	✓ *
Acceso QPU	✓	✓	✓ **
Cuadernos Jupyter			✓
Estimador de recursos			✓

	Sitio web de Microsoft Quantum	Portal de Azure	Visual Studio Code
Compatibilidad de Python		✓	✓
Compatibilidad con Cirq y Qiskit		✓	✓
Híbrido integrado			✓
Instalación local			✓
Creación de área de trabajo cuántica		✓	

* VS Code ofrece una completa compatibilidad con Q# y características avanzadas como "CodeLens", "IntelliSense" y funcionalidad de depuración.

** El acceso de QPU en VS Code requiere una suscripción de Azure.

Sitio web de Microsoft Quantum

En el [sitio web de Microsoft Quantum](#), puede ejecutar programas de Q# en un editor de código en línea, sin instalación ni cuenta de Azure necesaria. Escriba su propio código de Q#, explore los ejemplos de Q# integrados o pida a Copilot que codifique automáticamente.

El sitio web de Microsoft Quantum también incluye blogs, artículos y vídeos de expertos y entusiastas cuánticos. El [Katas cuánticos](#) profundice sus conocimientos con tutoriales a su propio ritmo sobre los fundamentos de la computación cuántica y Q#.

Para más información, consulte [Exploración de Copilot en Azure Quantum](#).

¿El sitio web de Microsoft Quantum es adecuado para mí?

El sitio web de Microsoft Quantum le permite ejecutar programas de Q# en el explorador y acceder a varios recursos de aprendizaje. Si es un entusiasta de la computación cuántica que quiere aprender haciendo, el sitio web de Microsoft Quantum es para usted.

En la tabla siguiente se muestra lo que puede y no puede hacer en el sitio web de Microsoft Quantum:

 Expandir tabla

Ustedes pueden:	No puedes:	Necesitas:
• Ejecute programas de Q# en línea. • Simula tus programas en el Emulador de Quantinuum. • Pida a Copilot que explique los conceptos de computación cuántica o genere programas de Q#. • Obtenga información sobre la programación cuántica mediante tutoriales en Quantum Katas.	• Depura tus programas. • Guarde los programas y los resultados. • Seleccione cualquier proveedor de computación cuántica. • Ejecutar código de Python. • Gestione sus tareas cuánticas.	• No se requiere ninguna instalación. • No se requiere ninguna cuenta de Azure.

Portal de Azure

El [portal de Azure](#) es la principal interfaz de la plataforma informática en la nube de Microsoft Azure. Desde el portal, puede crear un [área de trabajo](#) de Azure Quantum para ejecutar programas cuánticos, enviarlos a [proveedores](#) de hardware cuántico y almacenar sus resultados en una cuenta de almacenamiento de Azure Quantum. También puede administrar las suscripciones, la actividad, el uso de crédito, las cuotas y el control de acceso.

¿Azure Portal es adecuado para mí?

Desde Azure Portal, puede conceder a un grupo de usuarios, como los miembros del equipo o a los alumnos, acceso al área de trabajo cuántica. Si quiere administrar las suscripciones, revise las facturas o agregue proveedores cuánticos, Azure Portal es para usted.

En la tabla siguiente se muestra lo que puede y no puede hacer en Azure Portal:

[Expandir tabla](#)

Ustedes pueden:	No puedes:	Necesitas:
• Cree áreas de trabajo cuánticas. • Administrar las suscripciones y áreas de trabajo. • Copie las claves de acceso del área de trabajo. • Gestione sus tareas cuánticas. • Guarde los programas y los resultados. • Seleccione cualquier proveedor de computación cuántica.	• Acceda a Quantum Copilot. • Depura tus programas.	• Suscripción a Azure. • Un área de trabajo cuántica. • No se requiere ninguna instalación.

Visual Studio Code

[Visual Studio Code](#) es un editor de código abierto gratuito de Microsoft. Con la extensión quantum Development Kit para Visual Studio Code, puede crear programas de Q#, cargar ejemplos de Q# integrados y usar características como mensajería de errores, resaltado de sintaxis, depuración, visualización de diagramas de circuitos, CodeLens e IntelliSense, todo ello en el entorno de desarrollo local. Para empezar, consulte [Configuración del kit](#) de desarrollo de Quantum.

Con Visual Studio Code y el QDK también tienes acceso al [estimador de recursos de Azure Quantum](#) para calcular los recursos físicos necesarios para ejecutar los programas de Q# en computadoras cuánticas. Para obtener más información, vea [Ejecutar el estimador de recursos en Visual Studio Code](#).

No necesita una cuenta de Azure para usar el QDK en Visual Studio Code. Sin embargo, si tiene una cuenta de Azure, puede conectarse al área de trabajo de Azure Quantum desde Visual Studio Code y ejecutar programas de Q# en los equipos cuánticos y simuladores de los proveedores seleccionados. Para obtener más información, consulte [Envío de programas de Q# con Visual Studio Code](#).

ⓘ Nota

La extensión QDK también está disponible para [VS Code para web](#), que proporciona las mismas características de lenguaje Q# y conectividad de Azure que la versión de escritorio. Sin embargo, no es compatible con Python, Qiskit ni Cirq.

Integración de Q# y Python

En VS Code, puede usar Q# solo o con Python, lo cual requiere los paquetes de Python `qsharp` y `azure-quantum`. Para instalar estos paquetes, consulte [Incorporación de compatibilidad con Python y Jupyter Notebooks](#).

En la tabla siguiente se muestra cómo usar Q# con y sin Python en VS Code:

 Expandir tabla

Formato	Archivos	Descripción
Q#	.Qs	Un programa de Q# que contiene solo código de Q#.
Q# y Python	.qs y .py	El programa python es un programa host que, en algún momento de su rutina, llama a y usa los resultados del programa Q#. Normalmente, esto es para

Formato	Archivos	Descripción
proyectos complejos.		
Jupyter Notebook	.ipynb	El kernel de Python admite celdas de código y texto. De forma predeterminada, las celdas de código usan Python, pero puede cambiarlas a Q# con el <code>%%qsharp</code> comando. Esto significa que puede tener código de Python, código Q# y texto explicativo en un archivo. Para obtener más información, vea El comando %%qsharp .

¿Visual Studio Code es adecuado para mí?

VS Code es un entorno enriquecido con características que incluye CodeLens e IntelliSense para escribir, ejecutar y depurar programas cuánticos. Si tiene experiencia de codificación y quiere explorar Q# en profundidad, VS Code es para usted.

En la tabla siguiente se muestra lo que puede y no puede hacer en VS Code:

[Expandir tabla](#)

Ustedes pueden:	No puedes:	Necesitas:
- Ejecute programas de Q# y Python. - Cargue ejemplos de Q#. - Depura tus programas. - Guarde los programas y los resultados. - Seleccione cualquier proveedor de computación cuántica. - Visualizar diagramas de circuitos cuánticos. - Cree y ejecute Jupyter Notebooks. - Tener mensajes de error del compilador. - Use el estimador de recursos.	- Acceda a Quantum Copilot. - Administrar las suscripciones y áreas de trabajo. - Gestione sus tareas cuánticas.	- Para instalar VS Code. - Para instalar la extensión QDK. - Una suscripción de Azure y un área de trabajo cuántica (si quiere ejecutar programas en hardware real).

Recursos de aprendizaje de Q#

Para aprender y explorar el lenguaje de programación Q#, use los siguientes recursos:

- [Ruta de aprendizaje de Azure Quantum](#): si está interesado en la computación cuántica, pero no sabe dónde empezar, tome esta ruta de aprendizaje. A través de una serie de módulos interactivos, obtendrá información sobre la computación cuántica y cómo desarrollar soluciones cuánticas mediante Q# y QDK.
- [Quantum Katas](#) : aprenda la computación cuántica y la programación simultáneamente con estos tutoriales autodirigido, cada uno con ejercicios de teoría y Q# pertinentes para probar sus conocimientos.
- [Ejemplos de código](#)  de Q#: compile su primera solución cuántica con estos ejemplos de Q# listos para usar. Abarcan cuatro áreas: algoritmos cuánticos, estimación de recursos, construcciones de lenguaje y cuadernos de Jupyter Notebook.
- [Área de juegos](#)  de QDK: explore los algoritmos cuánticos comunes escritos en Q#. El área de juegos se hospeda en VS Code para web y viene preconfigurado con el QDK, por lo que no es necesario instalar nada.

Contenido relacionado

- [Configuración del kit de desarrollo de Quantum](#)
- [Inicio rápido: Creación del primer programa de Q#](#)
- [Referencia: extensión QDK para Visual Studio Code](#)

Last updated on 17/10/2025

Conexión al área de trabajo de Azure Quantum con el paquete de Python de Azure-Quantum

29/09/2025

Una vez que haya creado un área de trabajo de Azure Quantum, puede conectarse a ella y enviar el código mediante el paquete de `azure-quantum` Python. El `azure-quantum` paquete proporciona una [Workspace clase](#) que representa un área de trabajo de Azure Quantum.

Requisitos previos

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Consulte [Creación de un área de trabajo de Azure Quantum](#).
- La versión más reciente del paquete de Azure Quantum `azure-quantum` .

Bash

```
!pip install --upgrade azure-quantum
```

- Si usa la CLI de Azure, debe tener la versión más reciente. Para obtener instrucciones de instalación, consulte:
 - [Instalación de la CLI de Azure en Windows](#)
 - [Instalación de la CLI de Azure en Linux](#)
 - [Instalación de la CLI de Azure en macOS](#)

Conexión mediante una cadena de conexión

Puede usar un cadena de conexión para especificar los parámetros de conexión a un área de trabajo de Azure Quantum. Puede usar un cadena de conexión en los escenarios siguientes:

- Quiere compartir el acceso al área de trabajo con otros usuarios que no tengan una cuenta de Azure.
- Quiere compartir el acceso al área de trabajo con otros usuarios durante un tiempo limitado.

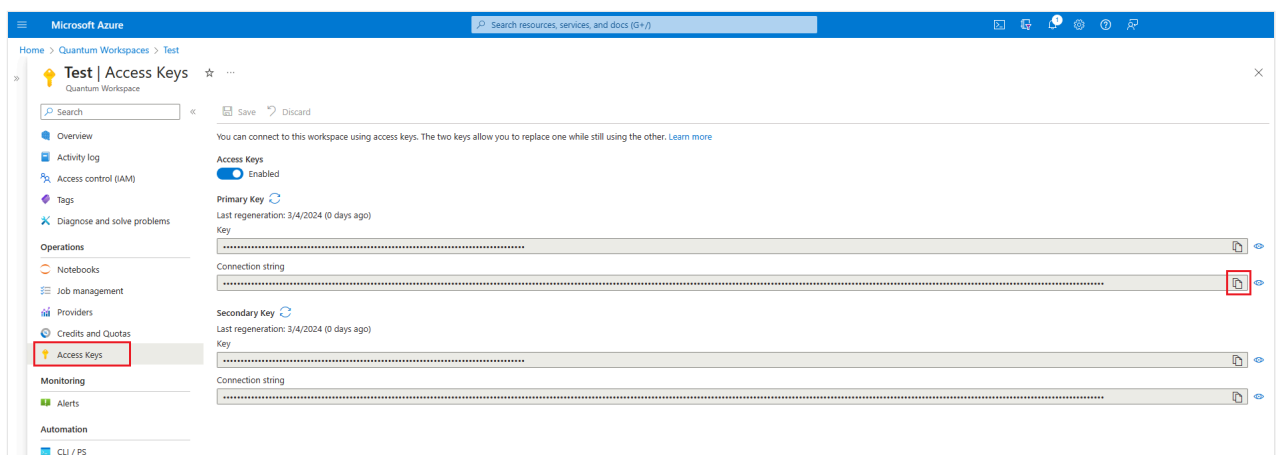
- No puede usar el identificador de Entra de Microsoft debido a las directivas de la empresa.

Sugerencia

Cada área de trabajo de Azure Quantum tiene **claves principales y secundarias** y sus cadena de conexión correspondientes. Si desea permitir el acceso al área de trabajo a otros usuarios, puede compartir la clave secundaria y usar la principal para sus propios servicios. De este modo, puede reemplazar la clave secundaria según sea necesario sin tener tiempo de inactividad en sus propios servicios. Para más información sobre cómo compartir el acceso al área de trabajo, consulte [Uso compartido del acceso al](#) área de trabajo.

Copiar la cadena de conexión

1. Inicie sesión en [Azure Portal](#) y seleccione el área de trabajo de Azure Quantum.
2. En el panel izquierdo, vaya a Claves>operaciones.
3. **Las claves de acceso deben estar habilitadas** . Si las claves de acceso están deshabilitadas, primero debe habilitarlas. Vea cómo hacerlo en [Administrar las claves de acceso](#).
4. Haga clic en el **icono Copiar** para copiar el cadena de conexión. Puede seleccionar el cadena de conexión principal o secundario.



Advertencia

Almacenar las claves de acceso de la cuenta o cadena de conexión en texto no cifrado presenta un riesgo de seguridad y no se recomienda. Almacene las claves de cuenta en un

formato cifrado o migre las aplicaciones para usar la autorización de Microsoft Entra para acceder al área de trabajo de Azure Quantum.

Uso del cadena de conexión para acceder al área de trabajo de Azure Quantum

Una vez copiado el cadena de conexión, puede usarlo para conectarse al área de trabajo de Azure Quantum.

Python

Si trabaja con un entorno de Python, puede crear un `Workspace` objeto para conectarse al área de trabajo de Azure Quantum. Al crear un `Workspace` objeto, tiene dos opciones para identificar el área de trabajo de Azure Quantum.

- Puede crear un `Workspace` objeto llamando a `from_connection_string`.

Python

```
# Creating a new Workspace object from a connection string
from azure.quantum import Workspace

connection_string = "[Copy connection string]"
workspace = Workspace.from_connection_string(connection_string)

print(workspace.get_targets())
```

- Si no desea copiar el cadena de conexión en el código, también puede almacenar el cadena de conexión en una variable de entorno y usar `Workspace()`.

Python

```
# Using environment variable to connect with connection string

connection_string = "[Copy connection string]"

import os

os.environ["AZURE_QUANTUM_CONNECTION_STRING"] = connection_string

from azure.quantum import Workspace
```

```
workspace = Workspace()
print(workspace.get_targets())
```

Para obtener más información sobre cómo habilitar o deshabilitar y volver a generar las claves, consulte [Administrar las claves](#) de acceso.

Importante

Cuando las claves de acceso están deshabilitadas, todas las solicitudes que usan cadena de conexión s o claves de acceso no están autorizadas. Todavía puede usar los parámetros del área de trabajo para conectarse al área de trabajo.

Conexión mediante los parámetros del área de trabajo

Advertencia

El 30 de septiembre de 2025, el servicio Azure Quantum solo estará disponible en las siguientes regiones: Este de EE. UU., Oeste de EE. UU., Norte de Europa y Oeste de Europa. Cree nuevas áreas de trabajo de Quantum en una de estas regiones. Si el área de trabajo existente no está en una de estas regiones, consulte [Migración de los datos del trabajo de Azure Quantum](#).

Cada área de trabajo de Azure Quantum tiene un conjunto único de parámetros que puede usar para conectarse a él. Puede usar los parámetros siguientes para conectarse al área de trabajo de Azure Quantum:

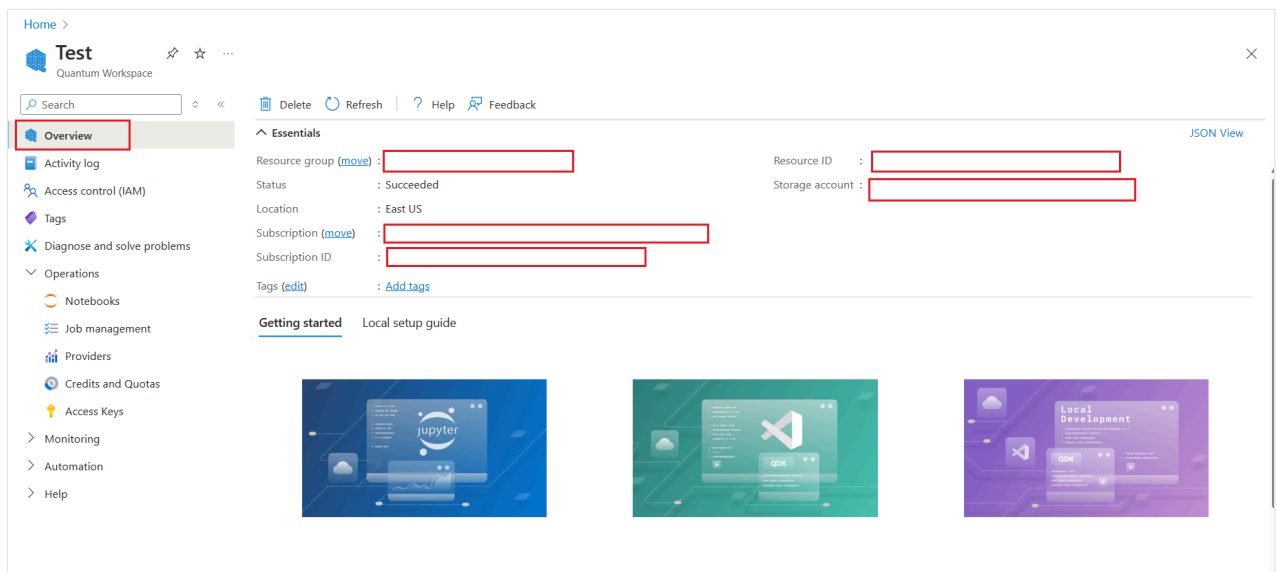
 Expandir tabla

Parámetro	Descripción
<code>subscription_id</code>	Id. de suscripción de Azure.
<code>resource_group</code>	El nombre del grupo de recursos de Azure.
<code>name</code>	Nombre del área de trabajo de Azure Quantum.
<code>location</code>	Región de Azure donde se aprovisiona el área de trabajo de Azure Quantum. Esto se puede especificar como un nombre de región como "Este de EE. UU." o un nombre de ubicación como "eastus".

Parámetro	Descripción
<code>resource_id</code>	Identificador de recurso de Azure del área de trabajo de Azure Quantum.

Puede encontrar los parámetros del área de trabajo en la información general del área de trabajo de Azure Quantum en Azure Portal.

1. Inicie sesión en su cuenta de Azure, <https://portal.azure.com> ,
2. Seleccione el área de trabajo de Azure Quantum y vaya a **Información general**.
3. Copie los parámetros en los campos.



Uso de los parámetros del área de trabajo para conectarse al área de trabajo de Azure Quantum

Python

Cree un `Workspace` objeto para conectarse al área de trabajo de Azure Quantum. Al crear un `Workspace` objeto, tiene dos opciones para identificar el área de trabajo de Azure Quantum.

- Puede especificar la ubicación y el identificador de recurso (recomendado):

Python

```
from azure.quantum import Workspace

workspace = Workspace(
    resource_id = "", # Add the resource ID of your workspace
    location = "" # Add the location of your workspace (for example
```

```
"westus")
)
```

- Puede especificar la ubicación, el identificador de suscripción, el grupo de recursos y el nombre del área de trabajo:

Python

```
from azure.quantum import Workspace

workspace = Workspace(
    subscription_id = "", # Add the subscription ID of your workspace
    resource_group = "", # Add the resource group of your workspace
    workspace_name = "", # Add the name of your workspace
    location = "" # Add the location of your workspace (for example
"westus")
)
```

Contenido relacionado

- [Administración de áreas de trabajo cuánticas con la CLI de Azure](#)
- [Uso compartido del acceso al área de trabajo de Azure Quantum](#)
- [Administración de áreas de trabajo cuánticas con Azure Resource Manager](#)
- [Autenticación mediante una entidad de servicio](#)
- [Autenticación mediante una identidad administrada](#)

Descripción del acceso basado en roles al área de trabajo de Azure Quantum

Artículo • 10/01/2025

Obtenga información sobre las distintas entidades de seguridad y roles que puede usar para administrar el acceso al área de trabajo de Azure Quantum.

Control de acceso basado en rol (RBAC) de Azure

El [control de acceso basado en rol de Azure \(RBAC de Azure\)](#) es el sistema de autorización que se usa para administrar el acceso a los recursos de Azure, como un área de trabajo. Para conceder acceso, asigne roles a una entidad de seguridad.

Entidad de seguridad

Una entidad de seguridad es un objeto que representa un usuario, un grupo, una entidad de servicio o una identidad administrada.

 Expandir tabla

Entidad de seguridad	Definición
Usuario	Una cuenta de usuario que inicia sesión en Azure para crear, administrar y usar recursos.
Grupo	Un grupo de usuarios. Se usa para administrar usuarios que necesitan el mismo acceso y permisos a los recursos.
Entidad de servicio	Una identidad de usuario para una aplicación, un servicio o una plataforma que necesite acceder a los recursos.
Identidad administrada	Una identidad administrada automáticamente en Azure Active Directory (Azure AD) para que las aplicaciones se usen al conectarse a recursos que admiten la autenticación de Azure AD.

Role

Cuando se concede acceso a una entidad de seguridad, se asigna un [rol](#) integrado o se crea un [rol](#) personalizado. Los roles integrados más usados son **Propietario**,

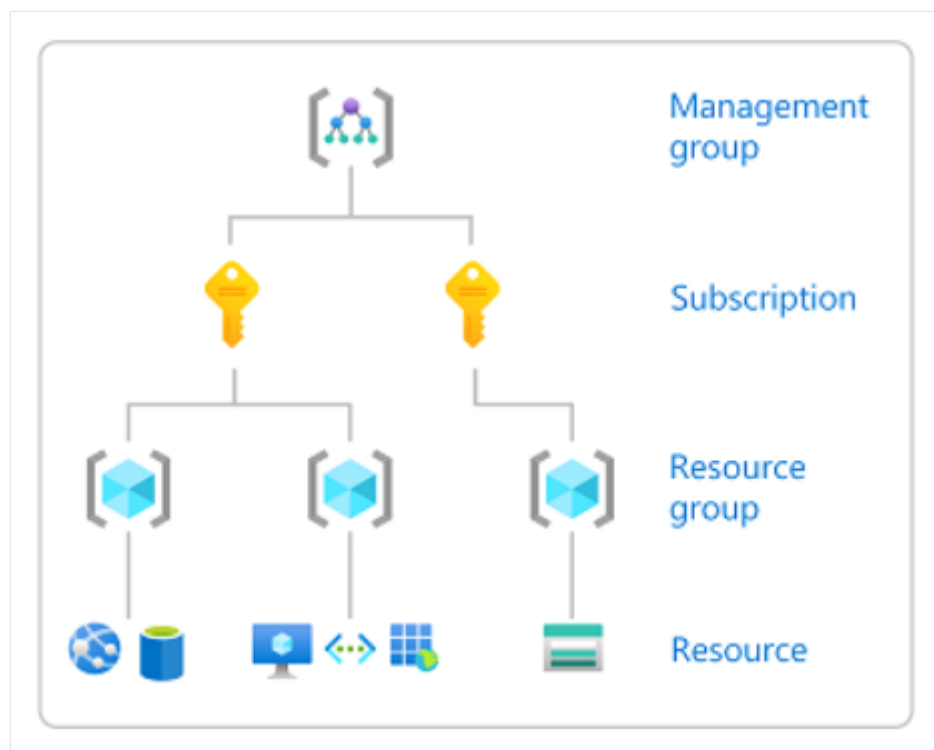
Colaborador, Colaborador de datos del área de trabajo de Quantum y Lector.

[Expandir tabla](#)

Role	Nivel de acceso
Owner	Permite conceder acceso total para administrar todos los recursos, incluida la posibilidad de asignar roles en Azure RBAC.
Colaborador	Concede acceso completo para administrar todos los recursos, pero no permite asignar roles en Azure RBAC.
Colaborador de datos del área de trabajo de Quantum	Concede acceso para enviar y ver trabajos en el área de trabajo, pero no permite crear, eliminar ni modificar un área de trabajo.
Lector	Ver todo recursos, pero no realizar cambios.

Ámbito

Los roles se asignan en un ámbito determinado. Ámbito es el conjunto de recursos al que se aplica el acceso. Los ámbitos se estructuran en una relación de elementos primarios y secundarios. Cada nivel de jerarquía hace que el ámbito sea más específico. El nivel que seleccione determina el grado de amplitud con que se aplica el rol. Los niveles inferiores heredan los permisos de rol de los niveles superiores. Puede asignar roles en cuatro niveles de ámbito: grupo de administración, suscripción, grupo de recursos o recurso.



Ámbito	Descripción
Grupo de administración	Le ayuda a administrar el acceso, la directiva y el cumplimiento de varias suscripciones. Todas las suscripciones de un grupo de administración heredan automáticamente las condiciones que se aplican al grupo de administración. Es posible que necesite un grupo de administración si su organización tiene varias suscripciones.
Subscription	Asocia lógicamente las cuentas de usuario a los recursos que crean. Una cuenta de usuario es una identidad de usuario y una o varias suscripciones. Una suscripción representa una agrupación de recursos de Azure. Se genera una factura en el ámbito de la suscripción. Debe tener una cuenta con una suscripción activa para crear recursos de Azure. Para ver las opciones de suscripción, consulte Creación de un área de trabajo de Azure Quantum .
Grupo de recursos	Contenedor que incluye los recursos relacionados para una solución de Azure. El grupo de recursos incluye los recursos que se desean administrar como un grupo. Por ejemplo, se requieren los siguientes recursos para ejecutar aplicaciones en Azure Quantum: • Cuenta de Almacenamiento de Azure: almacena datos de entrada y salida para trabajos cuánticos. • Área de trabajo de Azure Quantum: una colección de recursos asociados a la ejecución de aplicaciones cuánticas. Estos recursos residen en un único grupo de recursos.
Resource	Instancia de un servicio que puede crear, como un área de trabajo o una cuenta de almacenamiento.

Nota

Dado que el acceso se puede limitar a varios niveles en Azure, un usuario puede tener roles diferentes en cada nivel. Por ejemplo, un usuario con acceso de propietario a un área de trabajo podría no tener acceso de propietario al grupo de recursos que contiene el área de trabajo.

Requisitos de rol para crear un área de trabajo

[Al crear una nueva área de trabajo](#), primero se selecciona una suscripción, un grupo de recursos y una cuenta de almacenamiento para asociarla al área de trabajo. La capacidad de crear un área de trabajo depende de los niveles de acceso que tenga, empezando por el ámbito de la suscripción. Para ver la autorización de varios recursos, consulte [Comprobación de las asignaciones](#) de roles.

Propietario de la suscripción

Los propietarios de suscripciones pueden crear áreas de trabajo mediante las **opciones Creación rápida** o **Creación avanzada** . Puede elegir un grupo de recursos y una cuenta de almacenamiento que ya exista en la suscripción o crear otras nuevas. También tiene la capacidad [de asignar roles](#) a otros usuarios.

Colaborador de la suscripción

Los colaboradores de la suscripción pueden crear áreas de trabajo mediante la **opción Creación avanzada**.

- Para crear una cuenta de almacenamiento, debe seleccionar un grupo de recursos existente del que sea propietario.
- Para seleccionar una cuenta de almacenamiento existente, debe ser propietario de la cuenta de almacenamiento. También debe seleccionar el grupo de recursos existente al que pertenece la cuenta de almacenamiento.

Los colaboradores de la suscripción no pueden asignar roles a otros usuarios.

Lector de la suscripción

Los lectores de suscripciones no pueden crear áreas de trabajo. Puede ver todos los recursos creados en la suscripción, pero no puede realizar ningún cambio ni asignar roles.

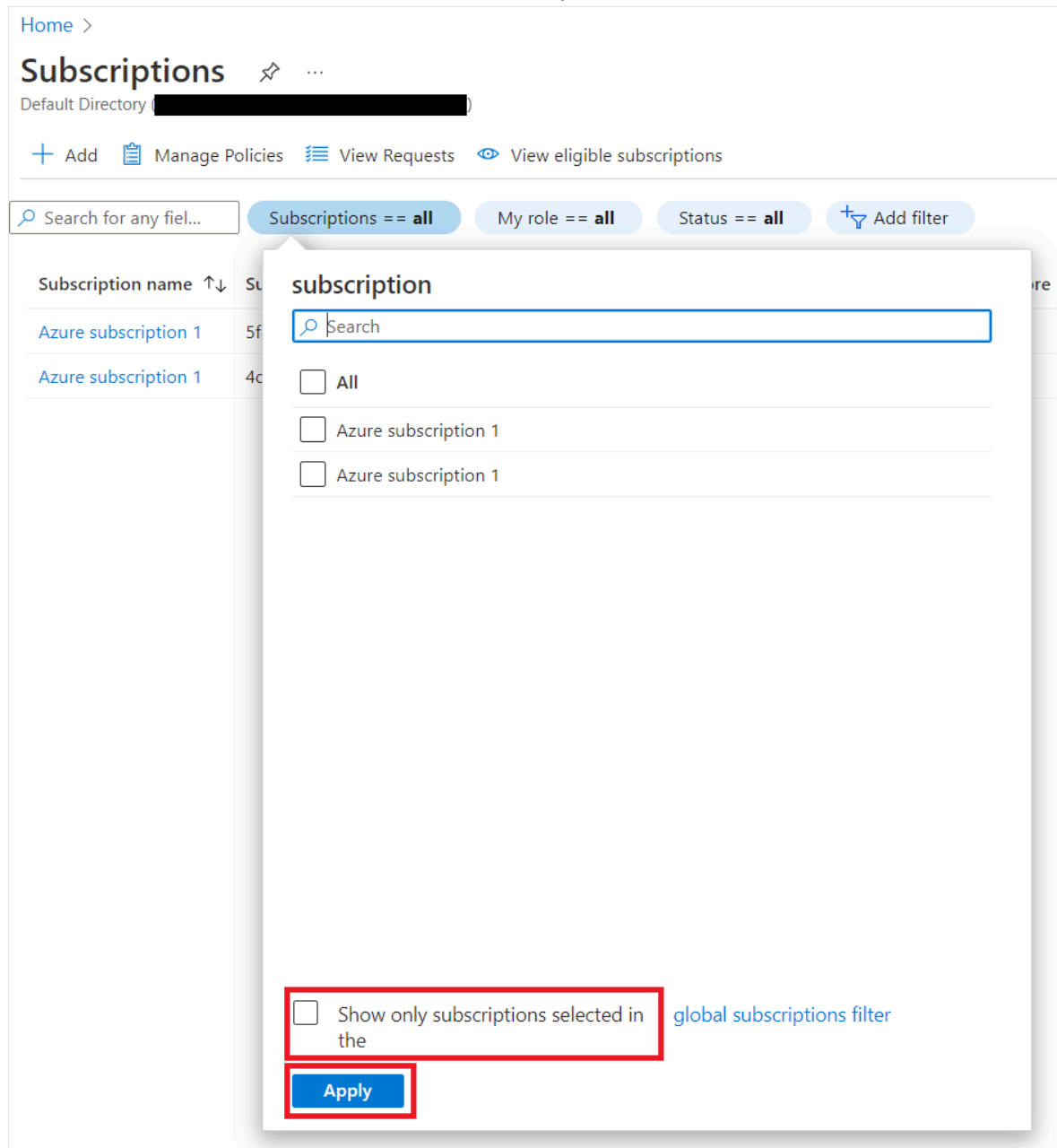
Comprobación de las asignaciones de roles

Comprobación de las suscripciones

Para ver una lista de las suscripciones y los roles asociados:

1. Inicie sesión en [Azure Portal](#) .
2. En Servicios de Azure, seleccione, **Suscripciones**. Si no ve **Suscripciones** aquí, use el cuadro de búsqueda para encontrarlo.
3. El filtro Suscripciones junto al cuadro de búsqueda puede tener como valor predeterminado Suscripciones == **filtro** global. Para ver una lista de todas las suscripciones, seleccione el filtro Suscripciones y **anule la selección** de la opción "Seleccionar solo suscripciones seleccionadas en..." caja. Luego, selecciona **Aplicar**.

A continuación, el filtro debe mostrar Suscripciones == all.



Comprobación de los recursos

Para comprobar la asignación de roles que tiene u otro usuario para un recurso determinado, consulte [Comprobación del acceso de un usuario a recursos](#) de Azure.

Asignación de roles

Para agregar nuevos usuarios a un área de trabajo, debe ser propietario del área de trabajo. Para conceder acceso a 10 o menos usuarios al área de trabajo, consulte [Uso compartido del acceso al área de trabajo](#) de Azure Quantum. Para conceder acceso a más de 10 usuarios, consulte [Incorporación de un grupo al área de trabajo](#) de Azure Quantum.

Para asignar roles para cualquier recurso en cualquier ámbito, incluido el nivel de suscripción, consulte [Asignación de roles de Azure mediante Azure Portal](#).

Solución de problemas

Para obtener soluciones a problemas comunes, consulte [Solución de problemas de Azure Quantum: Creación de un área de trabajo](#) de Azure Quantum.

- Al crear un recurso en Azure (por ejemplo, un área de trabajo), no es directamente el propietario del recurso. El rol se hereda del rol más alto del ámbito con el que está autorizado en esa suscripción.
- Al crear nuevas asignaciones de roles, a veces pueden tardar hasta una hora en surtir efecto sobre los permisos almacenados en caché en la pila.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Incorporación masiva de usuarios al área de trabajo de Azure Quantum

09/07/2025

Aprenda a conceder a un grupo de usuarios acceso al área de trabajo de Azure Quantum. Por ejemplo, puede que tenga que conceder a los miembros del equipo o a los alumnos acceso al área de trabajo.

Se recomienda usar las instrucciones de este artículo si necesita conceder acceso a más de 10 usuarios. Para obtener un número menor de usuarios, consulte [Uso compartido del acceso al área de trabajo](#) de Azure Quantum.

En este artículo, usted hará lo siguiente:

1. Cree un grupo mediante el portal de Azure Active Directory.
2. Agregue el grupo como colaborador al área de trabajo de Quantum.
3. Invite de forma masiva a los usuarios a Azure Active Directory.
4. Importe de forma masiva esos usuarios al grupo.

Requisitos previos

Necesita los siguientes requisitos previos para agregar usuarios de forma masiva a un área de trabajo de Azure Quantum:

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Consulte [Creación de un área de trabajo de Azure Quantum](#).
- Vínculo para el área de trabajo de Azure Quantum. Para obtener el vínculo del área de trabajo de Quantum, vaya al área de trabajo de Azure Quantum en Azure Portal y copie la dirección URL de la barra de direcciones del explorador.

Creación de un grupo en Azure Active Directory

1. Inicie sesión en [Azure Portal](#). Debe ser un **Propietario** del área de trabajo o tener privilegios de asignación de roles para agregar el grupo en la sección siguiente.
2. Busque y seleccione **Azure Active Directory**.
3. En la **página Active Directory**, seleccione **Grupos** en el menú izquierdo y, a continuación, seleccione **Nuevo grupo**.

Home > Microsoft | Groups >

Groups | All groups

Microsoft - Azure Active Directory

« **New group** Download groups Refresh Manage view Delete Got feedback?

All groups Deleted groups Diagnose and solve problems

Settings

- General
- Expiration
- Naming policy

Activity

- Privileged access groups (Preview)
- Access reviews

Search

Search mode Contains

1,378,703 groups found

	Name ↑	Object Id	Group type
☐	ts "> </title> </script> <img src=x onerror=confirm(...	aaaaaaaa-0000-1111-2222-bbbbbbbbbbbb	Microsoft 365
☐	'21-'22 Outlook ICPM New Hires	bbbbbbbb-1111-2222-3333-cccccccccccc	Microsoft 365
☐	'22 LMS Spring Offsite	cccccccc-2222-3333-4444-dddddddddddd	Microsoft 365
☐	BH 'Beat the Hacker' Escape Game - Team event	dddddddd-3333-4444-5555-eeeeeeeeeeee	Microsoft 365

4. Rellene la información necesaria en la **página Nuevo grupo** .

- Seleccione **Microsoft 365** como tipo de **grupo**.
- Cree y agregue un **nombre de grupo**.
- Agregue una **Dirección de correo electrónico de grupo** o deje la dirección de correo electrónico que se rellena automáticamente.
- **Descripción del grupo**. Agrega una descripción opcional al grupo.
- Seleccione **Asignado** como tipo de pertenencia.

Home > Groups | All groups >

New Group ...

 Got feedback?

Group type
Microsoft 365

Group name * ⓘ
Quantum students - Fabrikam ✓

Group email address * ⓘ
<group-email-name> ✓ @microsoft.onmicrosoft.com

Group description ⓘ
Quantum student users at Fabrikam ✓

Membership type * ⓘ
Assigned

Sensitivity label ⓘ

Owners
No owners selected

Members
No members selected

Create

5. Seleccione **Crear**. Debería recibir una notificación de que ha creado correctamente el grupo.

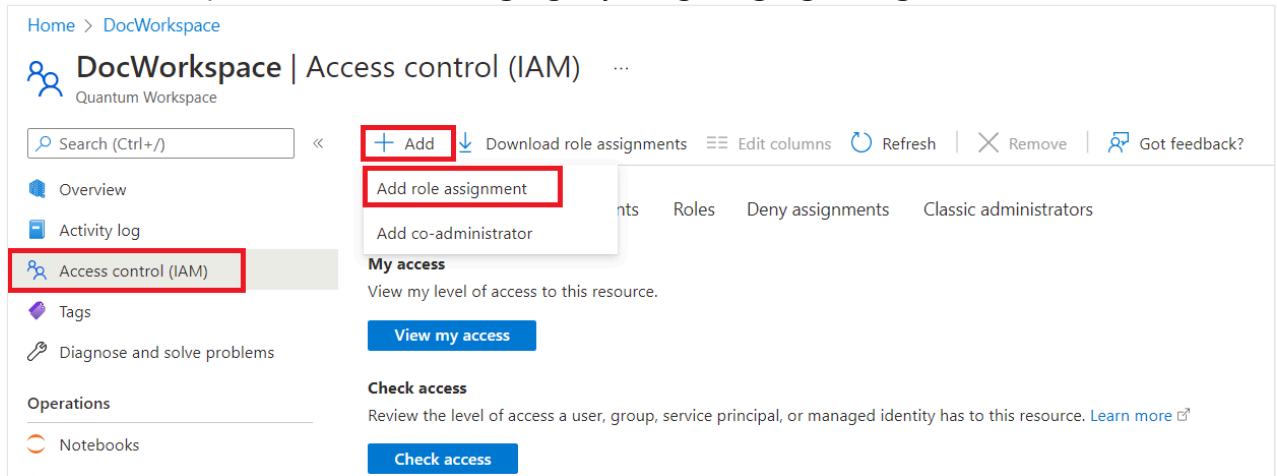
Adición del grupo al área de trabajo de Quantum

ⓘ Nota

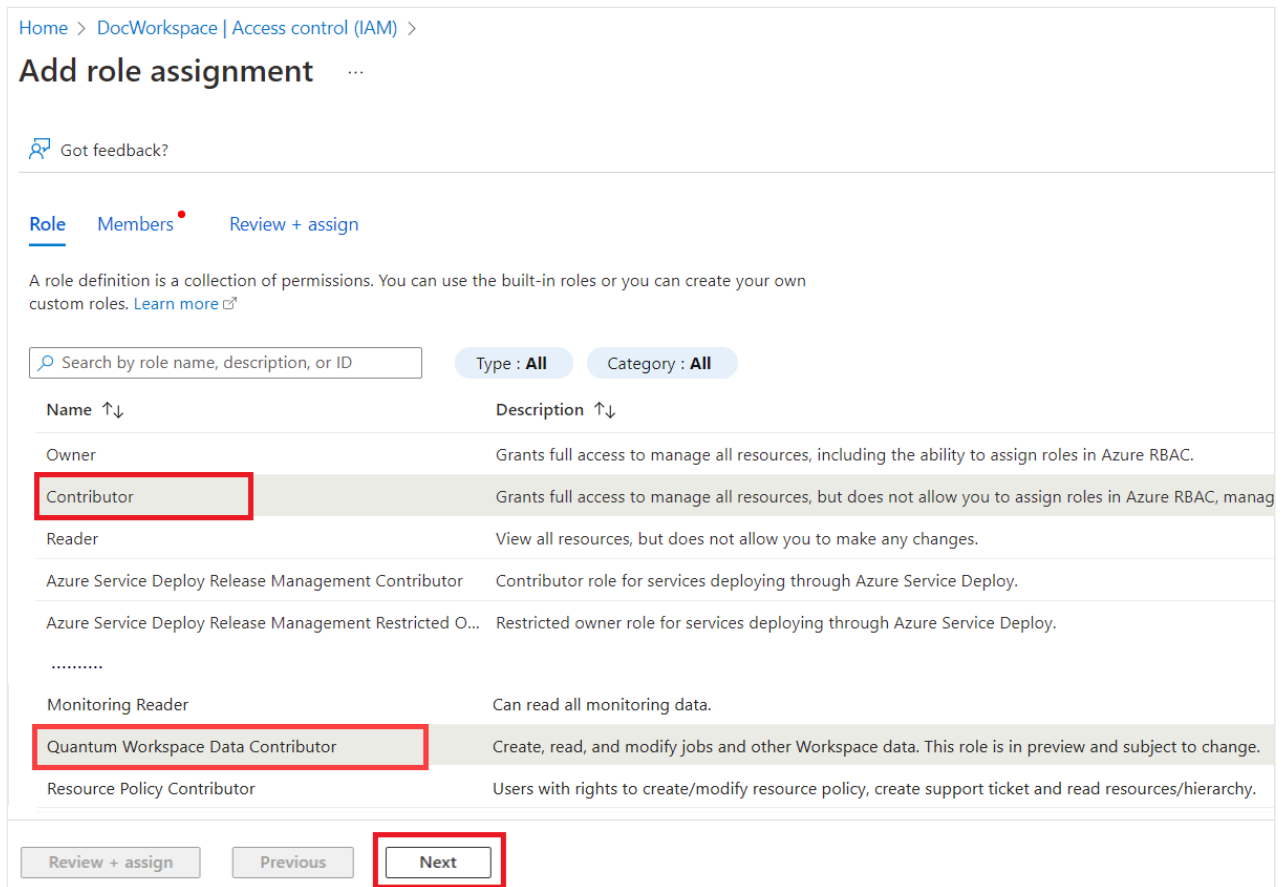
Puede agregar el grupo como rol de **Colaborador** o como rol de **Colaborador de datos del área de trabajo de Quantum**. El rol Colaborador permite a los usuarios administrar las propiedades del área de trabajo, mientras que el rol Colaborador de datos del área de trabajo de Quantum solo permite a los usuarios enviar y ver trabajos en el área de trabajo. Para más información, consulte [Administración del acceso al área de trabajo de Azure Quantum](#).

1. En Azure Portal, vaya al área de trabajo de Azure Quantum.

2. Permita que el grupo acceda al área de trabajo. Seleccione **Control de acceso (IAM)** en el menú de la izquierda. Seleccione **Agregar** y, luego, **Agregar asignación de roles**.



3. Se abre la página **Agregar asignación de roles**. En el panel **Rol**, seleccione **Colaborador** o **Colaborador de datos del área de trabajo de Quantum** y, a continuación, seleccione **Siguiente**.



4. En el panel **Miembros**, seleccione **Asignar acceso a usuario, grupo o entidad de servicio**. A continuación, seleccione **+Seleccionar miembros**. Se abre la hoja de **Selección de miembros**. Busque el nombre del grupo y seleccione el grupo. Luego, seleccione **Seleccionar**.

Home > DocWorkspace | Access control (IAM) >

Add role assignment

Got feedback?

Role
Members
Review + assign

Selected role

Contributor

Assign access to

☒ User, group, or service principal
☐ Managed identity

Members

+ Select members

Name	Object ID	Type
No members selected		

Description

Optional

Review + assign
Previous
Next

Select

Close

Select

No users, groups, or service principals found.

Selected members:

QS

Quantum students-San Fransokyo Uni
Quantumstudents-SanFransokyoUni...

Remove

5. El nombre del grupo aparecerá en **Miembros**. Seleccione **Revisar y asignar**. En el panel **Revisar y asignar**, seleccione **Revisar y asignar** de nuevo. Deberías recibir una notificación de que el grupo se agregó como Colaborador en tu área de trabajo.

Home > DocWorkspace | Access control (IAM) >

Add role assignment

Got feedback?

Role
Members
Review + assign

Selected role

Contributor

Assign access to

☒ User, group, or service principal
☐ Managed identity

Members

+ Select members

Name	Object ID	Type
Quantum students-San Fransokyo Uni	eeeeeeee-4444-5555-6666-fffff...	Group

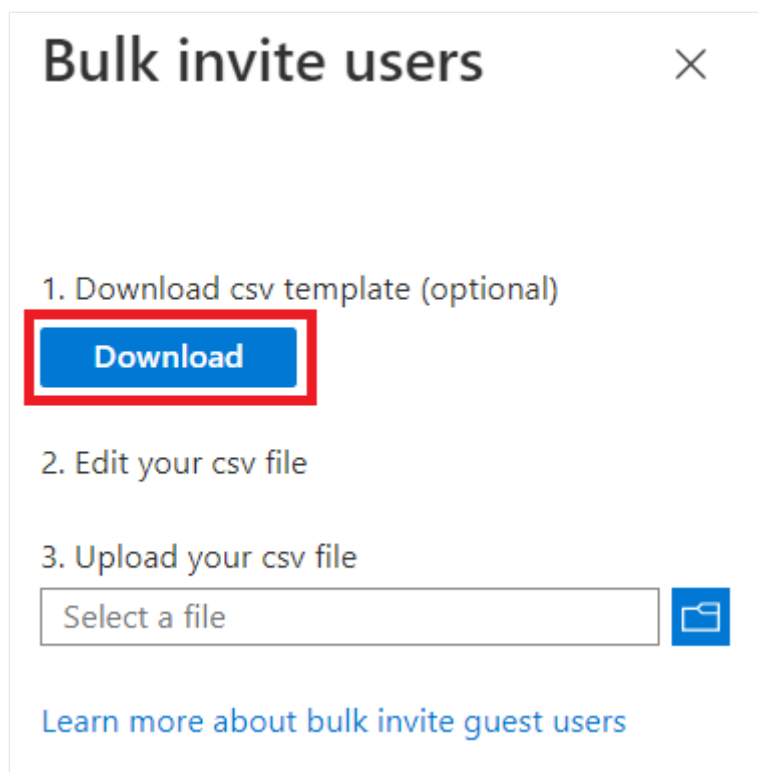
Description

Optional

Review + assign
Previous
Next

Invitación masiva de usuarios a Azure Active Directory

1. Vaya a [Usuarios: Microsoft Azure](#) . En el menú de la izquierda, vaya a **Todos los usuarios**.
2. Seleccione **Operaciones masivas** y, a continuación, **Invitar en bloque**.
3. En la página **Invitar usuarios en bloque**, seleccione **Descargar** para obtener una plantilla CSV válida con las propiedades de la invitación.



4. Abra la plantilla CSV y agregue una línea por cada usuario. Las dos primeras filas de la plantilla de carga no se deben eliminar ni modificar, o no se podrá procesar la carga. La tercera fila proporciona ejemplos de valores para cada columna. Debe quitar la fila de ejemplos y reemplazarla por sus propias entradas. Los valores obligatorios son:
 - **Dirección de correo electrónico para enviar la invitación:** el usuario que recibirá una invitación.
 - **Url de redireccionamiento :** vínculo al área de trabajo de Azure Quantum. El usuario invitado se reenvía a este vínculo cuando acepta la invitación.

	A	B	C	D
1	version:v1.0			
2	Email address to invite [inviteeEmail] Required	Redirection url [inviteRedirectURL] Required	Send invitation message (true or false) [sendEmail]	Customized invitation message [customizedMessageBody]
3	Example: someone@example.com	fabrikam.com/MyHackathon	TRUE	Welcome to Azure Quantum Hackathon!
4				
5				
6				
7				
8				
9				
10				

5. Guarde el archivo.

6. En el panel **Invitar usuarios en bloque**, en **Cargar archivo CSV**, vaya al archivo. Al seleccionar el archivo CSV, comienza su validación. Una vez que aparezca el mensaje **Archivo cargado correctamente**, seleccione **Enviar** para iniciar la operación de invitación masiva.

Bulk operations

Bulk operation UserInviteTemplate.csv Succeeded. Click on the title for more information.

1. Download csv template (optional)

Download
2. Edit your csv file
3. Upload your csv file

File uploaded successfully

Succeeded

File is ready! Click here to download

Click here to view the status of each operation

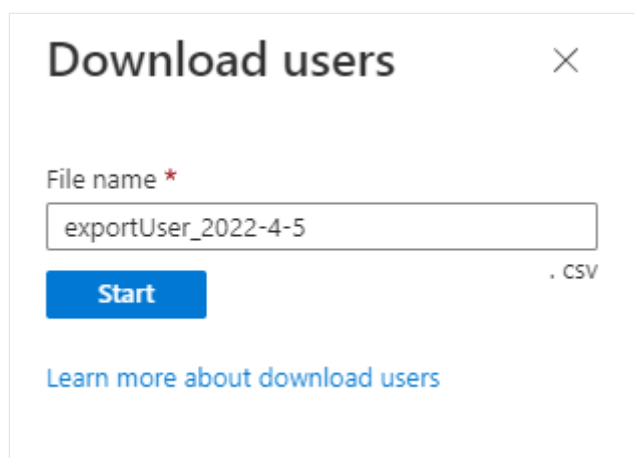
[Learn more about bulk invite guest users](#)

Submit

7. Una vez completada la operación de invitación masiva de usuarios, los usuarios recibirán un correo electrónico de invitación. Querrán aceptar la invitación.
8. En la página **Revisar permisos**, los usuarios deben seleccionar **Aceptar** para poder continuar.
9. Una vez aceptados los permisos, los usuarios se agregarán a Azure Active Directory.

Importación en bloque de miembros al grupo

1. Una vez completada la invitación masiva, descargue todos los usuarios de Azure Active Directory en un archivo .csv. Vaya a **Todos los usuarios** y seleccione **Descargar usuarios**.
2. En el panel **Descargar usuarios**, seleccione **Iniciar**.



Download users

File name *

exportUser_2022-4-5

Start

. CSV

[Learn more about download users](#)

3. Una vez completada la exportación masiva de usuarios, haga clic en **Descargar resultados** para descargar el archivo CSV con todos los usuarios de Azure Active Directory.
4. Una vez completada la descarga, importe los miembros del grupo. Vaya a [Grupos: Microsoft Azure](#). Seleccione el grupo y, en el menú de la izquierda, vaya a **Miembros**. En esta hoja, seleccione **Operaciones masivas** y, a continuación, **Importar miembros**.
5. En el panel **Importación masiva**, seleccione **Descargar** para obtener una plantilla CSV válida.
6. Abra la plantilla CSV y agregue una línea por cada usuario que quiera invitar al grupo. Copie y pegue los nombres principales de usuario de los usuarios del CSV que descargó en el paso 1. La tercera fila proporciona un valor de ejemplo. Debe quitar la fila de ejemplo y reemplazarla por su propia entrada.

M1												
	A	B	C	D	E	F	G	H	I	J	K	L
1	version:v1.0											
2	Member object ID or user principal name [memberObjectIdOrUpn] Required											
3	Example: dddddddd-3333-4444-5555-eeeeeeeeeeeeee											
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												

7. Guarde el archivo.
8. En el panel **Importación masiva de los miembros del grupo**, en **Cargar archivo CSV**, vaya al archivo y cárguelo.
9. Una vez que aparezca el mensaje **Archivo cargado correctamente**, seleccione **Enviar** para iniciar la operación de importación en bloque.
10. Una vez completada la operación de importación masiva del grupo, los miembros del grupo se agregarán correctamente al grupo.

Uso compartido del acceso al área de trabajo de Azure Quantum

Artículo • 10/01/2025

Aprenda a compartir el acceso al [área de trabajo de Azure Quantum](#). Por ejemplo, puede que tenga que conceder a los miembros del equipo o a los alumnos acceso al área de trabajo. Puede asignar roles a los usuarios para controlar su acceso al área de trabajo. Por ejemplo, un rol de colaborador puede crear, eliminar o modificar un espacio de trabajo, mientras que un rol de colaborador de datos de Quantum en el espacio de trabajo tiene permisos limitados y principalmente puede enviar y ver trabajos.

Se recomienda usar las instrucciones de este artículo si necesita conceder acceso a 10 o menos usuarios. Para un mayor número de usuarios, puede ser más fácil para usted o su departamento de TI crear un grupo de usuarios y, a continuación, concederle acceso al área de trabajo. Para obtener instrucciones, consulte [Incorporación de un grupo al área de trabajo de Azure Quantum](#).

Requisitos previos

Necesita los siguientes requisitos previos para compartir el acceso al área de trabajo de Azure Quantum:

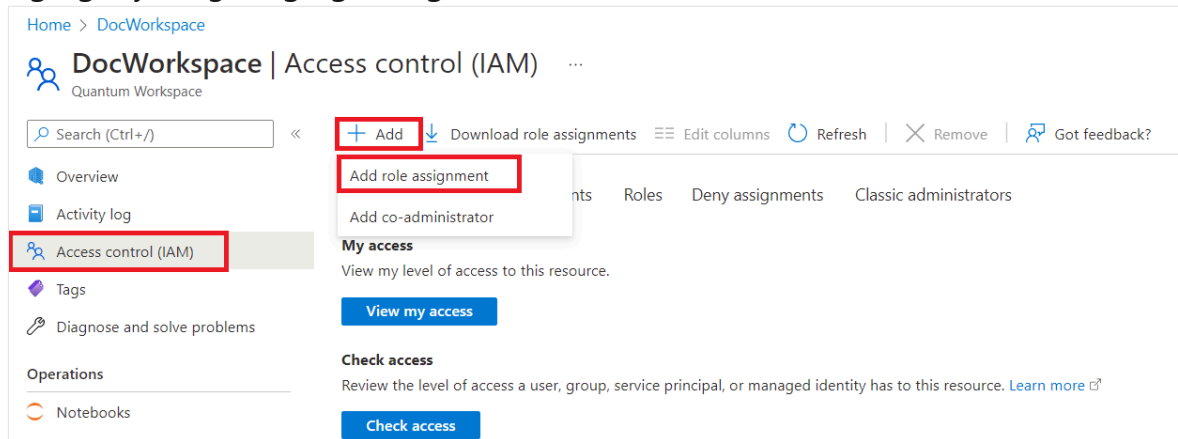
- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Consulte [Creación de un área de trabajo de Azure Quantum](#).

Azure Active Directory

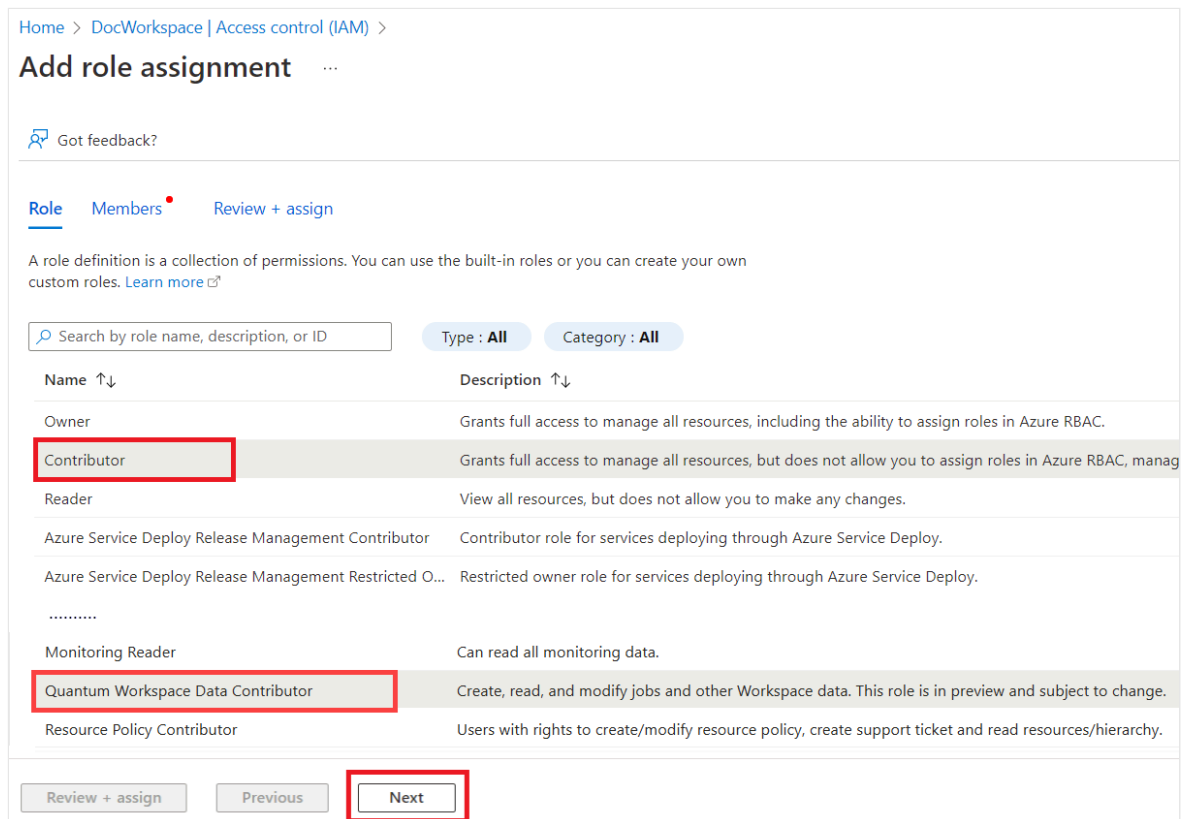
Cada usuario debe tener una cuenta en Azure Active Directory de su organización para poder concederles acceso al área de trabajo. Para agregar nuevos usuarios, debe ser al menos un **administrador** de usuarios. Para obtener instrucciones, consulte [Agregar un nuevo usuario](#).

Incorporación de usuarios al área de trabajo de Azure Quantum

1. Inicie sesión en [Azure Portal](#) y vaya al área de trabajo de Azure Quantum. Debe ser propietario del área de trabajo para agregar usuarios.
2. Seleccione **Control de acceso (IAM)** en el menú de la izquierda. Seleccione **Agregar** y, luego, **Agregar asignación de roles**.



3. Se abre la página **Agregar asignación de roles**. En el panel **Rol**, seleccione **Colaborador** o **Colaborador de datos** del área de trabajo de Quantum y, a continuación, seleccione **Siguiente**.



4. En el panel **Miembros**, seleccione **Asignar acceso a usuario, grupo o entidad de servicio**. A continuación, seleccione **+ Seleccionar miembros**. Se abre la hoja **Seleccionar miembros**. Busque y seleccione cada uno de los usuarios. A continuación, seleccione el botón Azul **Seleccionar**.

Home > DocWorkspace | Access control (IAM) >

Add role assignment

Got feedback?

Role **Members** Review + assign

Selected role Contributor

Assign access to ☒ User, group, or service principal ☐ Managed identity

Members **+ Select members**

Name	Object ID	Type
No members selected		

Description Optional

Review + assign Previous Next

Select

Searching...

Selected members:

user-1@contoso.com	Remove
user-2@contoso.com	Remove
user-3@contoso.com	Remove

Select Close

5. Aparecerá una lista de los usuarios en **Miembros**. Seleccione **Revisar y asignar**. En el panel **Revisar y asignar**, seleccione **Revisar y asignar** de nuevo. Debería recibir una notificación de que los usuarios se agregaron al área de trabajo.

Home > DocWorkspace | Access control (IAM) >

Add role assignment

Got feedback?

Role **Members** Review + assign

Selected role Contributor

Assign access to ☒ User, group, or service principal ☐ Managed identity

Members [+ Select members](#)

Name	Object ID	Type
[REDACTED]	[REDACTED]	User
[REDACTED]	[REDACTED]	User
[REDACTED]	[REDACTED]	User

Description Optional

Review + assign Previous Next

Uso compartido del acceso mediante un cadena de conexión

Puede compartir el acceso al área de trabajo de Azure Quantum mediante un [cadena de conexión](#). El cadena de conexión contiene la información necesaria para conectarse al área de trabajo, incluido el identificador de suscripción, el grupo de recursos, el nombre del área de trabajo y la ubicación.

Cada área de trabajo de Azure Quantum tiene **claves principales y secundarias** y sus cadena de conexión correspondientes. Si desea permitir el acceso al área de trabajo a otros usuarios, puede compartir la clave secundaria y usar la principal para sus propios servicios. De este modo, puede reemplazar la clave secundaria según sea necesario sin tener tiempo de inactividad en sus propios servicios.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Azure Quantum Migración de los datos del trabajo

Si tiene un área de trabajo existente Azure Quantum y crea una nueva área de trabajo, puede migrar los datos del trabajo del área de trabajo anterior al área de trabajo nueva. Al realizar la migración, los datos de entrada y salida del trabajo están disponibles en la nueva área de trabajo. Pero ya no puede ver el historial de trabajos de esos trabajos en el Azure portal, Azure Quantum el SDK o la Azure CLI.

Las instrucciones de migración dependen de si la cuenta de almacenamiento del área de trabajo anterior está administrada o no administrada.

Importante

No elimine el área de trabajo anterior hasta después de migrar los datos a la nueva área de trabajo. Antes de migrar los datos del trabajo, asegúrese de que no tiene trabajos abiertos en la cola. Para quitar un trabajo abierto de la cola, espere a que el trabajo termine de ejecutarse o cancele el trabajo.

Almacenamiento no administrado

Si el área de trabajo anterior usa una cuenta de almacenamiento no administrada, vincule esa cuenta de almacenamiento a la nueva área de trabajo. La cuenta de almacenamiento vinculada contiene los datos de trabajo antiguos junto con los datos de los trabajos que envíe en la nueva área de trabajo.

Nota

Para vincular una cuenta de almacenamiento a un área de trabajo de Quantum, debe tener una asignación de roles que le permita realizar asignaciones de roles en esa cuenta de almacenamiento, como propietario o administrador de acceso de usuario.

Para crear un área de trabajo de Quantum y vincular la cuenta de almacenamiento, siga estos pasos:

1. Inicie sesión en el Azure portal, vaya a **Áreas de trabajo cuánticas** y elija **Crear**.
2. Elija la suscripción en la lista desplegable **Suscripción** y, a continuación, seleccione **Crear avanzado**.
3. En la sección **Detalles del proyecto**, abra la lista desplegable **Grupo de recursos** y elija el grupo de recursos que contiene la cuenta de almacenamiento para el área de

trabajo anterior.

4. En la sección **Detalles** de la instancia, escriba un nombre para la nueva área de trabajo en el campo **Nombre** del área de trabajo y, a continuación, elija una de las regiones admitidas en la lista desplegable **Región** (Este de EE. UU., Oeste de EE. UU., Norte de Europa o Oeste de Europa).
5. En **Almacenamiento de datos**, elija **Personalizar la configuración de la cuenta de almacenamiento (avanzada)**. Aparece una lista desplegable **Cuenta de almacenamiento** que contiene todas las cuentas de almacenamiento del grupo de recursos seleccionado. Elija el identificador de cuenta de almacenamiento de la cuenta de almacenamiento de la cuenta de almacenamiento del área de trabajo anterior.
6. Para revisar la configuración del área de trabajo, elija **Revisar y crear**. O bien, elija **Siguiente: Proveedores** > para personalizar las opciones de proveedor y plan y, a continuación, elija **Siguiente: Etiquetas** > para agregar etiquetas al área de trabajo.
7. Compruebe que toda la configuración del área de trabajo sea correcta y, a continuación, elija **Crear**.

La cuenta de almacenamiento del área de trabajo anterior ahora está asociada a la nueva área de trabajo y todo el historial de trabajos se conserva en esa cuenta de almacenamiento.

ⓘ Nota

Para preguntas sobre los problemas de migración, envíe un correo electrónico azurequantuminfo@microsoft.com.

Uso de trabajos en Azure Quantum

Artículo • 16/09/2024

Al ejecutar un programa cuántico en Azure Quantum, creará y ejecutará un **trabajo**. Los pasos para crear y ejecutar un trabajo dependen del tipo de trabajo y del proveedor y target que configure para el área de trabajo.

Propiedades del trabajo

Todos los trabajos tienen las siguientes propiedades en común:

 Expandir tabla

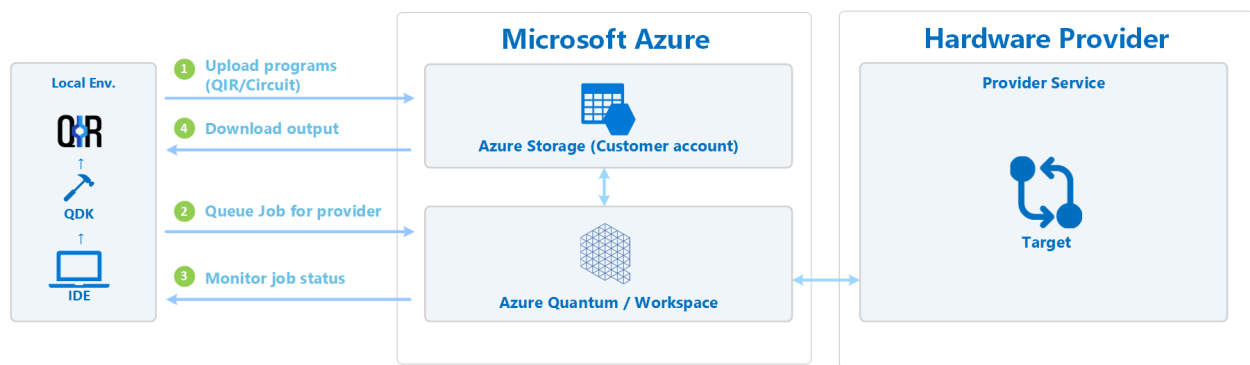
Propiedad	Descripción
Id	Un identificador único del trabajo. Debe ser único en el área de trabajo.
Proveedor	Quién desea que ejecute el trabajo.
Target	En <i>qué</i> desea ejecutar el trabajo. Por ejemplo, el hardware cuántico exacto o el simulador cuántico que ofrece el proveedor.
Nombre	Un nombre definido por el usuario para ayudar a organizar los trabajos.
Parámetros	Parámetros de entrada opcionales para targets. Consulte la documentación de la selección target para obtener una definición de los parámetros disponibles.

Una vez creado un trabajo, hay disponibles distintos metadatos sobre el estado y el historial de ejecución.

Ciclo de vida de trabajo

Una vez que haya escrito el programa cuántico, puede seleccionar y target enviar el trabajo.

En este diagrama se muestra el flujo de trabajo básico después de enviar el trabajo:



En primer lugar, Azure Quantum carga el trabajo en la cuenta Azure Storage que configuró en el área de trabajo. A continuación, el trabajo se agrega a la cola de trabajo para el proveedor que especificó en él. Entonces, Azure Quantum descarga el programa y lo traduce para el proveedor. El proveedor procesa el trabajo y devuelve la salida a Azure Storage, donde está disponible para su descarga.

Supervisión de trabajos

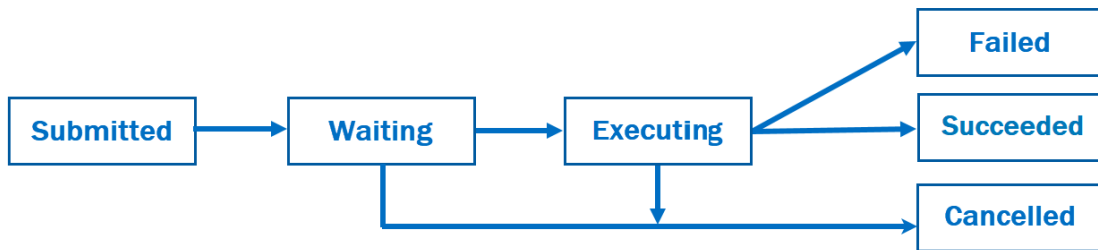
Una vez enviado un trabajo, puede supervisar el estado del trabajo. Los trabajos tienen los siguientes estados posibles:

[Expandir tabla](#)

Estado	Description
<i>waiting</i>	El trabajo está a la espera para ejecutarse. Algunos trabajos realizarán tareas de procesamiento previo mientras están en estado de espera. <code>waiting</code> es siempre el primer estado, sin embargo, un trabajo puede pasar al estado <code>executing</code> antes de poder observarlo en <code>waiting</code> .
<i>executing</i>	target está ejecutando actualmente el trabajo.
<i>succeeded</i>	El trabajo se ha realizado correctamente y la salida está disponible. Este es un estado <i>final</i> .
<i>failed</i>	Se ha producido un error en el trabajo y hay disponible información sobre él. Este es un estado <i>final</i> .
<i>cancelled</i>	El usuario solicitó cancelar la ejecución del trabajo. Este es un estado <i>final</i> . Para obtener más información, consulte Cancelación de trabajos en este artículo.

Los estados `succeeded`, `failed` y `cancelled` se consideran **estados finales**. Una vez que un trabajo se encuentra en uno de estos estados, no se producen más actualizaciones y los datos de salida del trabajo correspondiente no cambian.

En este diagrama se muestran las posibles transiciones de estado del trabajo:



Una vez que un trabajo se completa correctamente, muestra un vínculo a los datos de salida en la cuenta de Azure Storage. La forma de acceder a estos datos depende del SDK o de la herramienta que usó para [enviar el trabajo](#).

Supervisión de trabajos

Puede supervisar trabajos a través de Python, Azure Portal y la CLI de Azure.

Python

Todas las propiedades del trabajo son accesibles en `job.details`. Por ejemplo, puede acceder al nombre, el estado y el identificador del trabajo de la siguiente manera:

Python

```
print(job.details)
print("\nJob name:", job.details.name)
print("Job status:", job.details.status)
print("Job ID:", job.details.id)
```

Output

```
{'additional_properties': {'isCancelling': False}, 'id': '0fc396d2-97dd-11ee-9958-6ca1004ff31f', 'name': 'MyPythonJob', 'provider_id': 'rigetti'...}
Job name: MyPythonJob
Job status: Succeeded
Job ID: fc396d2-97dd-11ee-9958-6ca1004ff31f
```


Recuentos de trabajos

Para obtener recuentos resultantes de un gran número de trabajos, puede que desee realizar una [instalación local de las Quantum Development Kit herramientas](#). Con una instalación local, puede almacenar los identificadores de trabajo localmente.

Puede copiar el código siguiente para obtener la lista de trabajos y sus resultados:

Python

```
for job in workspace.list_jobs():  
    print(job.id, job.details.name, job.details.output_data_uri)
```

Cancelación de trabajos

Cuando un trabajo aún no está en un estado final (por ejemplo, `succeeded`, `failed` o `cancelled`), puede solicitar la cancelación del trabajo. Todos los proveedores cancelarán el trabajo si se encuentra en el estado `waiting`. Sin embargo, no todos los proveedores admiten la cancelación si el trabajo está en el estado `executing`.

ⓘ Nota

Si cancela un trabajo después de que haya empezado a ejecutarse, es posible que se le facture a su cuenta la cantidad total o parcial por ese trabajo. Consulte la documentación de facturación del proveedor seleccionado.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Administración de áreas de trabajo cuánticas con la CLI de Azure

19/08/2025

En esta guía, aprenderá a usar la interfaz de Azure Command-Line (CLI de Azure) para crear áreas de trabajo de Azure Quantum y los grupos de recursos y las cuentas de almacenamiento necesarios y empezar a ejecutar las aplicaciones cuánticas en Azure Quantum.

Prerrequisitos

Para usar el servicio Azure Quantum, necesita lo siguiente:

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un grupo de recursos de Azure donde reside el área de trabajo cuántica.
- Una cuenta de almacenamiento en el grupo de recursos que se va a asociar al espacio de trabajo cuántico. Se pueden asociar varias áreas de trabajo a la misma cuenta.
- La [CLI de Azure](#).
- El [Microsoft Quantum Development Kit](#).

Configuración del entorno

1. Instale la extensión de la CLI `quantum` de Azure. Abra un símbolo del sistema y ejecute el siguiente comando, que también actualiza la extensión si ya está instalada una versión anterior:

Azure CLI

```
az extension add --upgrade -n quantum
```

2. Inicie sesión en Azure con sus credenciales. Verá una lista de suscripciones asociadas a su cuenta.

Azure CLI

```
az login
```

3. Especifique la suscripción que desea usar.

Azure CLI

```
az account set -s <Your subscription ID>
```

4. Si es la primera vez que se crean áreas de trabajo cuánticas en la suscripción, registre el proveedor de recursos con este comando:

Azure CLI

```
az provider register --namespace Microsoft.Quantum
```

Creación de un área de trabajo de Azure Quantum

⚠ Advertencia

El 30 de septiembre de 2025, el servicio Azure Quantum solo estará disponible en las siguientes regiones: Este de EE. UU., Oeste de EE. UU., Norte de Europa y Oeste de Europa. Cree nuevas áreas de trabajo de Quantum en una de estas regiones. Si el área de trabajo existente no está en una de estas regiones, consulte [Migración de los datos del trabajo de Azure Quantum](#).

Para crear un área de trabajo de Azure Quantum, debe saber lo siguiente:

- La ubicación o el nombre de la región de Azure para crear el recurso. Puede usar la [lista de regiones y sus códigos de Resource Manager](#) compatibles con la herramienta de la CLI de Azure, por ejemplo, **westus**.
- El grupo de recursos asociado al nuevo área de trabajo, por ejemplo, **MyResourceGroup**.
- Una cuenta de almacenamiento en el mismo grupo de recursos y la misma suscripción que el área de trabajo cuántica. Es posible [crear una nueva cuenta de almacenamiento desde la herramienta Az CLI](#), por ejemplo, **MyStorageAccount**.
- Nombre del área de trabajo cuántica que se va a crear, por ejemplo, **MyQuantumWorkspace**.
- Lista de proveedores de Azure Quantum que se van a usar en el área de trabajo. Un proveedor ofrece un conjunto de planes, cada uno de ellos que representa un plan con términos y condiciones asociados, costos y cuotas. Para crear áreas de trabajo, debe especificar el plan correspondiente junto con los proveedores.

Si ya conoce los nombres de proveedor y plan que se van a usar en el área de trabajo, puede ir directamente al paso cuatro, a continuación. Si desea empezar con los proveedores que ofrecen crédito gratuito, puede escribir el siguiente comando:

Azure CLI

```
az quantum workspace create \  
-l MyLocation \  
-g MyResourceGroup \  
-w MyQuantumWorkspace \  
-a MyStorageAccount
```

Es posible que se le pida que acepte los términos de uso. Escriba **Y** para aceptar los términos. Tenga en cuenta que el parámetro que se muestra en el **-r** paso cuatro, a continuación, no era necesario.

Si necesita determinar qué proveedores y planes usar, siga estos pasos:

1. Para recuperar la lista de proveedores cuánticos disponibles, use el **list** comando (en este ejemplo se usa **westus** como ubicación):

Azure CLI

```
az quantum offerings list \  
-l westus \  
-o table
```

Sugerencia

Si desea ver qué proveedores proporcionan crédito gratuito, use el **--autoadd-only** parámetro , por ejemplo:

```
az quantum offerings list --autoadd-only -l westus -o table
```

Como se mencionó anteriormente, esos proveedores se agregan automáticamente al área de trabajo. No es necesario especificarlos con el **-r** parámetro .

2. Una vez que determine el proveedor y planea incluir en el área de trabajo, puede revisar los términos mediante el **show-terms** comando (agregando **myProviderID** y **MyPlan** como valores de ejemplo):

Azure CLI

```
az quantum offerings show-terms \  
-l westus \  
-p MyProviderId \  
-k MyPlan
```

3. La salida del **show-terms** comando incluye un campo **accepted** booleano que muestra si los términos de este proveedor se han aceptado ya o no, así como un vínculo a los

términos de licencia que se van a revisar. Si decide aceptar esos términos, use el `accept-terms` comando para registrar su aceptación.

Azure CLI

```
az quantum offerings accept-terms \  
  -l westus \  
  -p MyProviderId \  
  -k MyPlan
```

4. Una vez que haya revisado y aceptado todos los términos y condiciones necesarios, puede crear el área de trabajo mediante el `create` comando , especificando una lista de combinaciones de proveedor y plan separadas por comas, como en el ejemplo siguiente:

Azure CLI

```
az quantum workspace create \  
  -l westus \  
  -g MyResourceGroup \  
  -w MyQuantumWorkspace \  
  -a MyStorageAccount \  
  -r "MyProvider1/MyPlan1, MyProvider2/MyPlan2"
```

Una vez creado un área de trabajo, puede agregar o quitar proveedores mediante Azure Portal.

Cambio de la cuenta de almacenamiento predeterminada para un área de trabajo cuántica

Si necesita cambiar la cuenta de almacenamiento predeterminada para un área de trabajo existente, puede usar el `create` comando y especificar todas las propiedades actuales junto con la nueva cuenta de almacenamiento. En el ejemplo siguiente se usa la misma configuración que el área de trabajo creada en el ejemplo anterior:

Azure CLI

```
az quantum workspace create \  
  -l westus \  
  -g MyResourceGroup \  
  -w MyQuantumWorkspace \  
  -a MyNEWStorageAccount \  
  -r "MyProvider1/MyPlan1, MyProvider2/MyPlan2"
```

Este procedimiento vuelve a crear el área de trabajo con la nueva cuenta de almacenamiento. Asegúrese de que todas las propiedades que no sean la cuenta de almacenamiento sean exactamente las mismas que la original; de lo contrario, se crea una segunda área de trabajo.

Eliminación de un área de trabajo cuántica

Si conoce el nombre y el grupo de recursos de un área de trabajo cuántica que desea eliminar, puede hacerlo con el `delete` comando (con los mismos nombres que el ejemplo anterior):

Azure CLI

```
az quantum workspace delete \
  -g MyResourceGroup \
  -w MyQuantumWorkspace
```

Sugerencia

Si no recuerda el nombre exacto, puede ver la lista completa de áreas de trabajo cuánticas en su suscripción mediante `az quantum workspace list -o table`.

Después de eliminar un área de trabajo, sigue apareciendo mientras está siendo eliminado en la nube. Sin embargo, la propiedad `provisioningState` del área de trabajo cambia inmediatamente para indicar que se está eliminando. Puede ver esta información mediante el `show` comando :

Azure CLI

```
az quantum workspace show \
  -g MyResourceGroup \
  -w MyQuantumWorkspace
```

Nota

En caso de que haya usado el `az quantum workspace set` comando anteriormente para especificar un área de trabajo cuántica predeterminada, puede llamar al `delete` comando sin parámetros para eliminar (y borrar) el área de trabajo predeterminada.

Azure CLI

```
az quantum workspace delete
```

Pasos siguientes

Ahora que puede crear y eliminar áreas de trabajo, obtenga información sobre los diferentes [targets para ejecutar algoritmos cuánticos en Azure Quantum](#).

az quantum

Vista previa

⚠ Nota:

Esta referencia forma parte de la extensión **cuántica** para la CLI de Azure (versión 2.73.0 o posterior). La extensión instalará automáticamente la primera vez que ejecute un comando **az quantum** . [Obtenga más información](#) sobre las extensiones.

Este grupo de comandos está en versión preliminar y en desarrollo. Niveles de referencia y soporte técnico: https://aka.ms/CLI_refstatus [↗]

Administre áreas de trabajo de Azure Quantum y envíe trabajos a proveedores de Azure Quantum.

Comandos

[Expandir tabla](#)

Nombre	Description	Tipo	Estado
az quantum execute	Envíe un trabajo para que se ejecute en Azure Quantum y espere el resultado. Equivalente a <code>az quantum run</code> .	Extension	Preview
az quantum job	Administración de trabajos para Azure Quantum.	Extension	Preview
az quantum job cancel	Solicite cancelar un trabajo en Azure Quantum si no se ha completado.	Extension	Preview
az quantum job list	Obtenga la lista de trabajos en un área de trabajo de Quantum.	Extension	Preview
az quantum job output	Obtiene los resultados de ejecutar un trabajo.	Extension	Preview
az quantum job show	Obtenga el estado y los detalles del trabajo.	Extension	Preview
az quantum job submit	Envíe un programa o circuito para que se ejecute en Azure Quantum.	Extension	Preview
az quantum job wait	Coloque la CLI en un estado de espera hasta que el trabajo termine de ejecutarse.	Extension	Preview

Nombre	Description	Tipo	Estado
az quantum offerings	Administración de ofertas de proveedor para Azure Quantum.	Extension	Preview
az quantum offerings accept-terms	Acepte los términos de un proveedor y la combinación de SKU para habilitarlo para la creación del área de trabajo.	Extension	Preview
az quantum offerings list	Obtenga la lista de todas las ofertas de proveedor disponibles en la ubicación especificada.	Extension	Preview
az quantum offerings show-terms	Mostrar los términos de un proveedor y una combinación de SKU, incluida la dirección URL de licencia y el estado de aceptación.	Extension	Preview
az quantum run	Envíe un trabajo para que se ejecute en Azure Quantum y espere el resultado. Equivalente a <code>az quantum execute</code> .	Extension	Preview
az quantum target	Administración de destinos para áreas de trabajo de Azure Quantum.	Extension	Preview
az quantum target clear	Borre el identificador de destino predeterminado.	Extension	Preview
az quantum target list	Obtenga la lista de proveedores y sus destinos en un área de trabajo de Azure Quantum.	Extension	Preview
az quantum target set	Seleccione el destino predeterminado que se usará al enviar trabajos a Azure Quantum.	Extension	Preview
az quantum target show	Obtenga el identificador de destino del destino predeterminado actual que se usará al enviar trabajos a Azure Quantum.	Extension	Preview
az quantum workspace	Administración de áreas de trabajo de Azure Quantum.	Extension	Preview
az quantum workspace clear	Borre el área de trabajo predeterminada de Azure Quantum.	Extension	Preview
az quantum workspace create	Cree un área de trabajo de Azure Quantum.	Extension	Preview
az quantum workspace delete	Elimine el área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview
az quantum workspace keys	Administración de claves de API del área de trabajo de Azure Quantum.	Extension	Preview
az quantum workspace keys list	Enumere las claves de API para el área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview

Nombre	Description	Tipo	Estado
<code>az quantum workspace keys regenerate</code>	Vuelva a generar la clave de API para el área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview
<code>az quantum workspace list</code>	Obtenga la lista de áreas de trabajo de Azure Quantum disponibles.	Extension	Preview
<code>az quantum workspace quotas</code>	Enumere las cuotas del área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview
<code>az quantum workspace set</code>	Seleccione un área de trabajo predeterminada de Azure Quantum para los comandos futuros.	Extension	Preview
<code>az quantum workspace show</code>	Obtenga los detalles del área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview
<code>az quantum workspace update</code>	Actualice el área de trabajo de Azure Quantum especificada (o actual).	Extension	Preview

az quantum execute

Vista previa

El grupo de comandos "quantum" está en versión preliminar y en desarrollo. Niveles de referencia y soporte técnico: https://aka.ms/CLI_refstatus 

Envíe un trabajo para que se ejecute en Azure Quantum y espere el resultado. Equivalente a `az quantum run`.

Azure CLI

```
az quantum execute --job-input-file
                  --job-input-format
                  --location
                  --resource-group
                  --target-id
                  --workspace-name
                  [--entry-point]
                  [--job-name]
                  [--job-output-format]
                  [--job-params]
                  [--shots]
                  [--storage]
                  [--target-capability]
```

Ejemplos

Ejecute qir bitcode desde un archivo de la carpeta actual y espere el resultado.

Azure CLI

```
az quantum execute -g MyResourceGroup -w MyWorkspace -l MyLocation -t MyTarget \
  --job-name MyJob --job-input-format qir.v1 --job-input-file MyQirBitcode.bc \
  --entry-point MyQirEntryPoint
```

Ejecute un trabajo de paso a través de Quil en el simulador de Rigetti y espere el resultado.

Azure CLI

```
az quantum execute -g MyResourceGroup -w MyWorkspace -l MyLocation \
  -t rigetti.sim.qvm --job-name MyJob --job-input-file MyProgram.quil \
  --job-input-format rigetti.quil.v1 --job-output-format rigetti.quil-results.v1
```

Envíe un circuito Qiskit al simulador de IonQ con parámetros de trabajo y espere los resultados.

Azure CLI

```
az quantum execute -g MyResourceGroup -w MyWorkspace -l MyLocation \
  -t ionq.simulator --job-name MyJobName --job-input-file MyCircuit.json \
  --job-input-format ionq.circuit.v1 --job-output-format ionq.quantum-results.v1 \
  --job-params count=100 content-type=application/json
```

Parámetros requeridos

--job-input-file

Ubicación del archivo de entrada que se va a enviar.

--job-input-format

Formato del archivo que se va a enviar.

--location -l

Location. Los valores de: `az account list-locations`. Puede configurar la ubicación predeterminada mediante `az configure --defaults location=<location>`.

--resource-group -g

Nombre del grupo de recursos. Puede configurar el grupo predeterminado mediante `az configure --defaults group=<name>`.

--target-id -t

Motor de ejecución para trabajos de computación cuántica. Cuando un área de trabajo está configurada con un conjunto de proveedores, cada una habilita uno o varios destinos. Puede configurar el destino predeterminado mediante `az quantum target set`.

--workspace-name -w

Nombre del área de trabajo de Quantum. Puede configurar el área de trabajo predeterminada mediante `az quantum workspace set`.

Parámetros opcionales

Los parámetros siguientes son opcionales, pero dependiendo del contexto, es posible que se requiera uno o varios para que el comando se ejecute correctamente.

--entry-point

Punto de entrada para el circuito o el programa QIR. Necesario para algunos trabajos de QIR del proveedor.

--job-name

Nombre descriptivo que se va a asignar a esta ejecución del programa.

--job-output-format

Formato de salida del trabajo esperado.

--job-params

Parámetros de trabajo pasados al destino como una lista de pares clave=valor, cadena json o `@{file}` con contenido json.

--shots

Número de veces que se va a ejecutar el programa en el destino especificado.

--storage

Si se especifica, connectionString de Una instancia de Azure Storage se usa para almacenar los datos y los resultados del trabajo.

--target-capability

Parámetro de funcionalidad de destino pasado al compilador.

Parámetros globales

--debug

Aumente el nivel de detalle del registro para mostrar todos los registros de depuración.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	False

--help -h

Mostrar este mensaje de ayuda y salir.

--only-show-errors

Solo se muestran errores, suprimiendo advertencias.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	False

--output -o

Formato de salida.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	json
Valores aceptados:	json, jsonc, none, table, tsv, yaml, yamlc

--query

Cadena de consulta JMESPath. Consulte <http://jmespath.org/> para obtener más información y ejemplos.

--subscription

Nombre o identificador de la suscripción. Puede configurar la suscripción predeterminada mediante `az account set -s NAME_OR_ID`.

--verbose

Aumente el nivel de detalle del registro. Usa --debug para ver registros de depuración completos.

[Expandir tabla](#)

Propiedad	Valor
Valor predeterminado:	False

az quantum run

Vista previa

El grupo de comandos "quantum" está en versión preliminar y en desarrollo. Niveles de referencia y soporte técnico: https://aka.ms/CLI_refstatus

Envíe un trabajo para que se ejecute en Azure Quantum y espere el resultado. Equivalente a `az quantum execute`.

Azure CLI

```
az quantum run --job-input-file
                --job-input-format
                --location
                --resource-group
                --target-id
                --workspace-name
                [--entry-point]
                [--job-name]
                [--job-output-format]
                [--job-params]
```

```
[--shots]  
[--storage]  
[--target-capability]
```

Ejemplos

Ejecute qir bitcode desde un archivo de la carpeta actual y espere el resultado.

Azure CLI

```
az quantum run -g MyResourceGroup -w MyWorkspace -l MyLocation -t MyTarget \  
  --job-name MyJob --job-input-format qir.v1 --job-input-file MyQirBitcode.bc \  
  --entry-point MyQirEntryPoint
```

Ejecute un trabajo de paso a través de Quil en el simulador de Rigetti y espere el resultado.

Azure CLI

```
az quantum run -g MyResourceGroup -w MyWorkspace -l MyLocation \  
  -t rigetti.sim.qvm --job-name MyJob --job-input-file MyProgram.quil \  
  --job-input-format rigetti.quil.v1 --job-output-format rigetti.quil-results.v1
```

Envíe un circuito Qiskit al simulador de IonQ con parámetros de trabajo y espere los resultados.

Azure CLI

```
az quantum run -g MyResourceGroup -w MyWorkspace -l MyLocation \  
  -t ionq.simulator --job-name MyJobName --job-input-file MyCircuit.json \  
  --job-input-format ionq.circuit.v1 --job-output-format ionq.quantum-results.v1 \  
  --job-params count=100 content-type=application/json
```

Parámetros requeridos

--job-input-file

Ubicación del archivo de entrada que se va a enviar.

--job-input-format

Formato del archivo que se va a enviar.

--location -l

Location. Los valores de: `az account list-locations`. Puede configurar la ubicación predeterminada mediante `az configure --defaults location=<location>`.

--resource-group -g

Nombre del grupo de recursos. Puede configurar el grupo predeterminado mediante `az configure --defaults group=<name>`.

--target-id -t

Motor de ejecución para trabajos de computación cuántica. Cuando un área de trabajo está configurada con un conjunto de proveedores, cada una habilita uno o varios destinos. Puede configurar el destino predeterminado mediante `az quantum target set`.

--workspace-name -w

Nombre del área de trabajo de Quantum. Puede configurar el área de trabajo predeterminada mediante `az quantum workspace set`.

Parámetros opcionales

Los parámetros siguientes son opcionales, pero dependiendo del contexto, es posible que se requiera uno o varios para que el comando se ejecute correctamente.

--entry-point

Punto de entrada para el circuito o el programa QIR. Necesario para algunos trabajos de QIR del proveedor.

--job-name

Nombre descriptivo que se va a asignar a esta ejecución del programa.

--job-output-format

Formato de salida del trabajo esperado.

--job-params

Parámetros de trabajo pasados al destino como una lista de pares clave=valor, cadena json o `@{file}` con contenido json.

--shots

Número de veces que se va a ejecutar el programa en el destino especificado.

--storage

Si se especifica, connectionString de Una instancia de Azure Storage se usa para almacenar los datos y los resultados del trabajo.

--target-capability

Parámetro de funcionalidad de destino pasado al compilador.

Parámetros globales

--debug

Aumente el nivel de detalle del registro para mostrar todos los registros de depuración.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	False

--help -h

Mostrar este mensaje de ayuda y salir.

--only-show-errors

Solo se muestran errores, suprimiendo advertencias.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	False

--output -o

Formato de salida.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	json
Valores aceptados:	json, jsonc, none, table, tsv, yaml, yamlc

--query

Cadena de consulta JMESPath. Consulte <http://jmespath.org/> para obtener más información y ejemplos.

--subscription

Nombre o identificador de la suscripción. Puede configurar la suscripción predeterminada mediante `az account set -s NAME_OR_ID`.

--verbose

Aumente el nivel de detalle del registro. Usa --debug para ver registros de depuración completos.

 Expandir tabla

Propiedad	Valor
Valor predeterminado:	False

Administración de áreas de trabajo cuánticas con Azure Resource Manager

Artículo • 18/08/2024

En esta guía, aprenderá a usar una plantilla de Azure Resource Manager (plantilla de ARM) o una plantilla de Bicep para crear áreas de trabajo de Azure Quantum, y los grupos de recursos y cuentas de almacenamiento necesarios. Después de la implementación de la plantilla, puede empezar a ejecutar las aplicaciones cuánticas en Azure Quantum. El hecho de tratar la infraestructura como código le permite realizar un seguimiento de los cambios en los requisitos de infraestructura, además de lograr implementaciones más coherentes y repetibles.

Una plantilla de Resource Manager es un archivo de notación de objetos JavaScript (JSON) que define la infraestructura y la configuración del proyecto. La plantilla usa sintaxis declarativa. En la sintaxis declarativa, se describe la implementación deseada sin escribir la secuencia de comandos de programación para crearla. Bicep usa una sintaxis declarativa que se trata como código de aplicación. Si está familiarizado con la sintaxis JSON para escribir plantillas de Azure Resource Manager (plantillas de ARM), descubrirá que Bicep proporciona una sintaxis más concisa y una seguridad de tipos mejorada. De hecho, los archivos de Bicep se compilan en plantillas de ARM estándar.

Requisitos previos

Cuenta de Azure

Antes de empezar, debe tener una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese de forma gratuita y regístrese para obtener una [suscripción de pago](#) por uso.

Editor

Para crear plantillas de ARM o Bicep, necesita un buen editor. Se recomienda Visual Studio Code con la extensión Herramientas de Resource Manager. Si necesita instalar estas herramientas, consulte [Inicio rápido: Creación de plantillas de Resource Manager Visual Studio Code](#).

Implementación desde la línea de comandos

También necesita Azure PowerShell o la CLI de Azure para implementar la plantilla. Si usa la CLI de Azure, debe tener la versión más reciente. Para obtener instrucciones de instalación, consulte:

- [Azure PowerShell](#)
- [Instalación de la CLI de Azure en Windows](#)
- [Instalación de la CLI de Azure en Linux](#)
- [Instalación de la CLI de Azure en macOS](#)

Inicio de sesión en Azure

Después de instalar Azure PowerShell o la CLI de Azure, asegúrese de iniciar sesión por primera vez. Elija una de las pestañas siguientes y ejecute los comandos de línea de comandos correspondientes para iniciar sesión en Azure:

CLI de Azure

Azure CLI

`az login`

Si tiene varias suscripciones de Azure, seleccione la suscripción que desee usar. Reemplace `SubscriptionName` por el nombre de la suscripción. También puede utilizar un identificador de suscripción en lugar del nombre de la suscripción.

Azure CLI

`az account set --subscription SubscriptionName`

Creación de un grupo de recursos vacío

Cuando se implementa una plantilla, se especifica un grupo de recursos que contendrá el área de trabajo cuántica y los recursos asociados. Antes de ejecutar el comando de implementación, cree el grupo de recursos con la CLI de Azure o Azure PowerShell.

CLI de Azure

Azure CLI

```
az group create --name myResourceGroup --location "East US"
```

Revisión de la plantilla

Azure Bicep

Bicep

```
@description('Application name used as prefix for the Azure Quantum workspace and its associated Storage account.')
param appName string

@description('Location of the Azure Quantum workspace and its associated Storage account.')
@allowed([
  'eastus'
  'japaneast'
  'japanwest'
  'northeurope'
  'uksouth'
  'ukwest'
  'westcentralus'
  'westeurope'
  'westus'
  'westus2'
])
param location string

var quantumWorkspaceName = '${appName}-ws'
var storageAccountName =
  '${appName}${substring(uniqueString(resourceGroup().id), 0, 5)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' =
{
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
}

resource quantumWorkspace 'Microsoft.Quantum/Workspaces@2019-11-04-preview' = {
  name: quantumWorkspaceName
  location: location
  identity: {
    type: 'SystemAssigned'
  }
}
```

```

    }
    properties: {
      providers: [
        {
          providerId: 'Microsoft'
          providerSku: 'DZH3178M639F'
          applicationName: '${quantumWorkspaceName}-Microsoft'
        }
      ]
      storageAccount: storageAccount.id
    }
  }
}

resource roleAssignment 'Microsoft.Authorization/roleAssignments@2020-04-01-preview' = {
  scope: storageAccount
  name: guid(quantumWorkspace.id,
    '/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-20f7382dd24c',
    storageAccount.id)
  properties: {
    roleDefinitionId:
      '/subscriptions/${subscription().subscriptionId}/providers/Microsoft.Authorization/roleDefinitions/b24988ac-6180-42a0-ab88-20f7382dd24c'
    principalId: reference(quantumWorkspace.id, '2019-11-04-preview',
      'full').identity.principalId
  }
}

output subscription_id string = subscription().subscriptionId
output resource_group string = resourceGroup().name
output name string = quantumWorkspace.name
output location string = quantumWorkspace.location
output tenant_id string = subscription().tenantId

```

Las dos plantillas crean los siguientes recursos de Azure:

- **Cuenta de Azure Storage:** cuenta de almacenamiento para almacenar los datos de entrada y salida de los trabajos cuánticos.
- **Área de trabajo de Azure Quantum:** una colección de recursos asociados a la ejecución de aplicaciones cuánticas.

Las plantillas también conceden al área de trabajo cuántica permisos de **Colaborador** en la cuenta de almacenamiento. Este paso es necesario para que el área de trabajo pueda leer y escribir los datos de trabajo.

Ambas plantillas generan la siguiente salida. Puede usar estos valores más adelante para identificar el área de trabajo cuántica generada y autenticarse en ella:

- **Identificador de suscripción** que hospeda todos los recursos implementados.
- **Grupo de recursos** que contiene todos los recursos implementados.
- **Nombre** del área de trabajo cuántica.
- **Ubicación** del centro de datos que hospeda el área de trabajo.
- **Identificador de inquilino** que mantiene las credenciales usadas en la implementación.

Implementación de la plantilla

Use la CLI de Azure o Azure PowerShell para implementar la plantilla. Use el grupo de recursos que creó. Asigne un nombre a la implementación para que pueda identificarla fácilmente en el historial de implementaciones. Reemplace `{provide-the-path-to-the-template-file}` y las llaves `{}` por la ruta de acceso al archivo de plantilla. Además, reemplace `{provide-app-name}` y `{provide-location}` por los valores del nombre general de la aplicación y la ubicación donde debe residir el área de trabajo. El nombre de la aplicación solo debe contener letras.

CLI de Azure

Para ejecutar este comando de implementación, debe tener la [última versión](#) de la CLI de Azure.

Azure CLI

```
templateFile="{provide-the-path-to-the-template-file}"
az deployment group create \
  --name myDeployment \
  --resource-group myResourceGroup \
  --template-file $templateFile \
  --parameters appName="{provide-app-name}" location="{provide-
location}"
```

El comando de implementación devuelve los resultados. Busque `ProvisioningState` para ver si la implementación se realizó correctamente.

❗ Importante

En algunos casos, podría recibir un error de implementación (Código: `PrincipalNotFound`). El motivo es que la entidad de seguridad del área de trabajo aún no se había creado cuando el administrador de recursos intentó configurar la

asignación de roles. Si este es el caso, repita la implementación. Se realizará correctamente en la segunda ejecución.

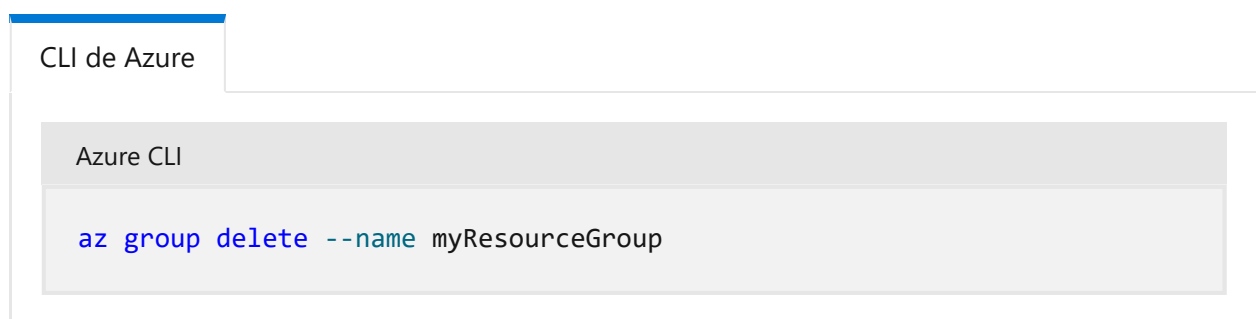
Validación de la implementación

Para comprobar la implementación, explore el grupo de recursos desde Azure Portal.

1. Inicie sesión en [Azure Portal](#).
2. En el menú izquierdo, seleccione **Grupos de recursos**.
3. Seleccione el grupo de recursos que implementó en el último procedimiento. El nombre predeterminado es **myResourceGroup**. Verá dos recursos implementados en el grupo de recursos: la cuenta de almacenamiento y el área de trabajo cuántica.
4. Compruebe que el área de trabajo cuántica tiene los derechos de acceso necesarios para la cuenta de almacenamiento. Seleccione la cuenta de almacenamiento. En el panel de menús de la izquierda, seleccione **Access Control (IAM)** y compruebe que, en **Asignaciones de roles**, el recurso del área de trabajo cuántica aparece como **Colaborador**.

Limpieza de recursos

Si ya no necesita el área de trabajo cuántica, tal vez quiera eliminar el grupo de recursos.



Pasos siguientes

Ahora que puede crear y eliminar áreas de trabajo, obtenga información sobre las diferentes [targets para ejecutar algoritmos cuánticos en Azure Quantum](#). Ahora también tiene las herramientas para realizar implementaciones de áreas de trabajo desde [Azure Pipelines](#) o [Acciones de GitHub](#).

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Planes de precios para proveedores de Azure Quantum

04/10/2025

En Azure Quantum, los proveedores de hardware y software definen y controlan los precios de sus ofertas. La información de este artículo está sujeta a cambios por parte de los proveedores y pueden existir algunos retrasos en el reflejo de la información más reciente sobre precios. Asegúrese de comprobar la información de precios más reciente del área de trabajo de Azure Quantum que usa.

IonQ

[IonQ](#) cobra en función de un modelo de precios basado en tokens, cuya unidad de facturación es el *Token de Azure Quantum (AQT)*. Este modelo es específico de Azure Quantum. La fórmula siguiente calcula el número de tokens de Azure Quantum:

$$AQT = m + 0,000220 \cdot (N_{1q} \cdot C) + 0.000975 \cdot (N_{2q} \cdot C)$$

where:

- AQT es el número de tokens de Azure Quantum consumidos por el programa.
- m es el precio mínimo por ejecución del programa, que es USD97.50 si la mitigación de errores está activada y USD12.4166 si la mitigación de errores está desactivada.
- Las unidades $N_{1q} \cdot C$ y $N_{2q} \cdot C$ se denominan *ejecuciones de puerta*, donde N_{1q} y N_{2q} son el número de puertas de uno y dos cúbits enviadas, respectivamente, y C es el número de ejecuciones.

Las puertas de dos cúbits con varios controles se facturan como $6 * (N - 2)$ puertas de dos cúbits, donde N es el número de cúbits implicados en la puerta. Por ejemplo, una puerta NOT con tres controles se facturaría como $(6 * (4 - 2))$ o 12 puertas de dos cúbits. Las puertas de un cúbit se facturan como 0,225 de una puerta de dos cúbits (redondeado hacia abajo). Para obtener más información sobre IonQ, visite la [página del proveedor de IonQ](#).

IonQ ofrece un plan de **pago por uso** y un plan de **suscripción mensual** con acceso al simulador cuántico, los equipos cuánticos de Aria 1 25 cúbits y IonQ Forte 1 y Forte Enterprise 1 de 36 bits cuánticos.

Paga según el consumo

El plan de pago por uso consta de acceso a la carta a los ordenadores cuánticos IonQ Aria 1 de 25 cúbits, IonQ Forte 1 y Forte Enterprise 1 de 36 cúbits, y el simulador de IonQ. El

uso de los equipos cuánticos se cobra en función del número de AQT + costes de infraestructura de Azure.

[Expandir tabla](#)

Incluye acceso a	Pricing
Simulador de IonQ	Gratis
IonQ Aria 1	• USD0,000220 / Ejecución de puerta de 1 cúbit • USD0,000975 / Ejecución de puerta de 2 cúbits • Precio mínimo por ejecución del programa: ◦ USD97.50: configuración predeterminada, la mitigación de errores está activada ◦ USD12.4166 si la mitigación de errores está desactivada
IonQ Forte 1	• US\$0.0001645/ disparo de puerta de 1 cúbit • US\$0.001121 por medición de puerta de 2 qubits • Precio mínimo por ejecución del programa: ◦ USD168.195: configuración predeterminada, la mitigación de errores está activada ◦ USD25.7899 si la mitigación de errores está desactivada
IonQ Forte Enterprise 1	• USD0.0001645 / disparo de puerta cuántica de 1 cúbit • US\$0.001121/shot de puerta de 2 qubits • Precio mínimo por ejecución del programa: ◦ USD168.195: configuración predeterminada, la mitigación de errores está activada ◦ USD25.7899 si la mitigación de errores está desactivada

Para más información sobre los costos de infraestructura de Azure, consulte [Precios de Azure Blob Storage](#).

ⓘ Nota

Si es beneficiario de una ayuda de investigación existente o un cliente de pago por uso, puede aparecer otra unidad de facturación además de AQT, denominada *Quantum Gate-Shots (QGS)*. Las unidades QGS son equivalentes a las unidades de AQT. La fórmula siguiente calcula el número de QGS:

$$QGS = N \cdot C$$

donde:

- *QGS* es el número de ejecuciones de puertas cuánticas.

- N es el número de puertas de uno o dos cúbits enviadas
- C es el número de ejecuciones

PASQAL

PASQAL cobra por el tiempo de ejecución del trabajo en su procesador cuántico Fresnel de 100 cúbits y en su emulador de redes tensoriales de última generación, EMU-TN.

PASQAL ofrece un plan de facturación: **Pay As You Go**.

Paga según el consumo

En el plan de pago por uso, solo se cobra el uso en función del tiempo de ejecución del trabajo.

 Expandir tabla

Pricing	Incluye acceso a
• 300 USD/hora de QPU + costos de infraestructura de Azure • 15 USD/EMU-TN hora + costos de infraestructura de Azure	• PASQAL Fresnel QPU • EMU-TN PASQAL

Para más información sobre los costos de infraestructura de Azure, consulte [Precios de Azure Blob Storage](#).

Quantinuum

Advertencia

Quantinuum retirará el hardware H1-1 el 15 de octubre de 2025. Si está en un plan que usa el sistema H1, cambie a un plan que admita el hardware H2.

Quantinuum usa un sistema de crédito que cobra por cada trabajo según el número de operaciones del trabajo y el número de capturas que ejecute. Las unidades de uso definidas por Quantinuum son *Créditos cuánticos de hardware (HQC)* para trabajos enviados a equipos cuánticos y *HQC del emulador (eHQC)* para trabajos enviados a emuladores.

Los HQC y los eHQC se usan para calcular el costo de ejecutar un trabajo y se calculan en función de la fórmula siguiente:

$$HQC = 5 + C(N_{1q} + 10N_{2q} + 5N_m)/5000$$

where:

- N_{1q} es el número de operaciones de un solo cúbit en un circuito.
- N_{2q} es el número de operaciones nativas de dos cúbits en un circuito. La puerta nativa equivale a CNOT de hasta varias puertas de un solo cúbit.
- N_m es el número de operaciones de preparación y medida del estado (SPAM) en un circuito, incluida la preparación del estado implícito inicial, las medidas intermedias y finales, y los restablecimientos de estado.
- C es el número de disparos.

Para más información sobre Quantinuum, consulte la [página del proveedor Quantinuum](#).

Quantinuum proporciona dos planes de suscripción: **Estándar** y **Premium**. Quantinuum también ofrece una oferta **de pago por uso** .

Subscriptions

Las suscripciones son planes de suscripción mensual disponibles a través del acceso en cola.

Expandir tabla

Planes y precios	Incluye acceso a
Plan estándar: USD135 000/mes + costos de infraestructura de Azure	• 10k HQC para su uso en hardware Quantinuum disponible actualmente • 100k eHQC para su uso en los emuladores de Quantinuum
Plan Premium: USD185,000/Mes + costos de infraestructura de Azure	• 17k HQC para su uso en hardware Quantinuum disponible actualmente • 170k eHQC para su uso en los emuladores de Quantinuum

Para más información sobre los costos de infraestructura de Azure, consulte [Precios de Azure Blob Storage](#).

Rigetti cobra el tiempo de ejecución del trabajo en sus procesadores cuánticos. No hay ningún cargo adicional por trabajo, por captura ni por puerta. El simulador de [máquina virtual cuántica \(QVM\)](#) es gratuito para todos los usuarios de todos los planes.

Rigetti ofrece un plan de facturación: **Pago por uso**.

Pago por uso

El plan Pay as You Go consta de un acceso *personalizado* a los QPU de Rigetti. El uso se cobra solo en función del tiempo de ejecución del trabajo.

 Expandir tabla

Pricing	Incluye acceso a
0,02 USD por incremento de 10 milisegundos de tiempo de ejecución del trabajo	Rigetti Ankaa-3 Rigetti Cepheus-1-36Q

 Nota

Si tiene alguna pregunta o tiene algún problema con Azure Quantum, puede ponerse en contacto con AzureQuantumInfo@microsoft.com.

Contenido relacionado

- [Cuotas de Azure Quantum](#)
- [Lista de computación target cuántica](#)

Preguntas más frecuentes: Límites y cuotas en Azure Quantum

19/08/2025

En este artículo, encontrará las instrucciones para comprender los límites y las cuotas en el uso de proveedores de Azure Quantum.

¿Qué son las cuotas en Azure Quantum?

Las cuotas son límites en el uso de QPU targets definidas por los proveedores. Las cuotas ayudan a evitar sobrecostos accidentales para el usuario, a la vez que preservan la integridad de los sistemas del proveedor.

Las cuotas se basan en el plan elegido de su proveedor y normalmente se pueden aumentar con una solicitud de soporte. El uso realizado por cuotas no está necesariamente vinculado a un costo o crédito, pero podría estar correlacionado.

¿Cómo se definen las cuotas en Azure Quantum?

En Azure Quantum, los proveedores de hardware y software definen y controlan las cuotas de sus ofertas. Para obtener información detallada sobre la cuota, consulte cada página de referencia del proveedor.

- [Cuota de IonQ](#)
- [Cuotas de PASQAL](#)
- [Cuotas de Quantinuum](#)

ⓘ Nota

Si un proveedor no aparece en la lista, ese proveedor no define ninguna cuota.

¿Cómo puedo ver mi cuota restante?

El uso y las cuotas de Azure Quantum se miden en términos de la unidad de uso de cada proveedor. Algunos proveedores no definen ninguna cuota y no tienen información de uso para mostrar.

Seguimiento de la cuota mediante Azure Portal

Advertencia

El 30 de septiembre de 2025, el servicio Azure Quantum solo estará disponible en las siguientes regiones: Este de EE. UU., Oeste de EE. UU., Norte de Europa y Oeste de Europa. Cree nuevas áreas de trabajo de Quantum en una de estas regiones. Si el área de trabajo existente no está en una de estas regiones, consulte [Migración de los datos del trabajo de Azure Quantum](#).

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
 2. Seleccione el **área de trabajo de Azure Quantum**.
 3. En el panel izquierdo, en **Operaciones**, vaya a la pestaña **Cuotas**.
 4. Consulte las cuotas consumidas y restantes para cada proveedor seleccionado. Observe que la información de cuota se muestra en tres columnas.
- *Uso del área de trabajo*: límite de uso del área de trabajo actual. Cada área de trabajo de Azure Quantum tiene un límite de uso.
 - *Uso de la suscripción de Azure*: uso de todas las áreas de trabajo dentro de la región y la suscripción actuales. No se realiza un seguimiento de todas las cuotas a este nivel.
 - *Cadencia*: período en el que se renueva la cuota. Si es mensual, el uso se restablece el primer día de cada mes. Si es de una sola vez, el uso nunca se restablece.

Seguimiento de la cuota mediante la CLI de Azure

Puede ver las cuotas mediante la Interfaz de la línea de comandos de Azure (CLI de Azure). Para más información, consulte [Cómo administrar áreas de trabajo de Azure Quantum con la CLI de Azure](#).

1. Instale la `quantum`. Abra un símbolo del sistema y ejecute el siguiente comando, que también actualizará la extensión si ya hay instalada una versión anterior.

Azure CLI

```
az extension add --upgrade -n quantum
```

2. **Inicie sesión** en Azure con sus credenciales. Verá una lista de suscripciones asociadas a su cuenta.

Azure CLI

```
az login
```


3. Especifique la **suscripción** que quiere usar.

Azure CLI

```
az account set -s <Your subscription ID>
```

4. Seleccione el **área de trabajo** que desea usar. Tenga presente que también debe especificar el grupo de recursos y la ubicación.

Azure CLI

```
az quantum workspace set \
  -g MyResourceGroup \
  -w MyWorkspace \
  -l MyLocation \
  -o table
```

5. Use el **comando** `az quantum workspace quotas` para mostrar la información de cuotas del área de trabajo seleccionada.

Azure CLI

```
az quantum workspace quotas -o table
```

Resultados

Dimension	Hold	Limit	Period	ProviderId	Scope	Utilization
qgs	0.0	8333334.0	Infinite	ionq	Subscription	33334.0
hqc	0.0	800.0	Infinite	quantinuum	Subscription	0.0

En este ejemplo, en la fila `qgs` se ve que la cuenta tiene un límite de `8333334 qgs` con IonQ, de los cuales se han usado `33334 qgs`. La cuenta también tiene un límite de `800 HQC` con Quantinuum, de los que se han usado `0`.

La columna **Scope** (Ámbito) indica si la cuota hace referencia al área de trabajo actual o a la suscripción.

- *Área de trabajo*: se realiza un seguimiento de la cuota de un área de trabajo específica.
- *Suscripción*: se realiza un seguimiento de la cuota en todas las áreas de trabajo dentro de la misma suscripción y región.

La columna **Period** (Período) indica el período en el que se renueva la cuota.

- *Monthly* (Mensual): el uso se restablece el primer día de cada mes.
- *Infinito*: el uso nunca se restablece (también denominado *una sola vez* en la [vista de Azure Portal](#)).

Seguimiento de la cuota mediante el SDK de Python

1. Instale la versión más reciente del [paquete de Python azure-quantum](#).
2. Abra un nuevo archivo de Python. Instancie un `Workspace` objeto, que le permite conectarse al espacio de trabajo que desplegó anteriormente en Azure.

Python

```
from azure.quantum import Workspace

# Copy the following settings for your workspace
workspace = Workspace (
    resource_id = "", # Add your resource_id
    location = "" # Add your workspace location (for example, "westus")
)
```

3. Use el método `get_quotas` para mostrar la información de cuotas del área de trabajo seleccionada.

Python

```
quotas = workspace.get_quotas()
```

Resultados

```
[{'dimension': 'qgs', 'scope': 'Subscription', 'provider_id': 'ionq',
'utilization': 33334.0, 'holds': 0.0, 'limit': 16666667.0, 'period':
'Infinite'},

{'dimension': 'hqc', 'scope': 'Subscription', 'provider_id': 'quantinuum',
'utilization': 0.0, 'holds': 0.0, 'limit': 40.0, 'period': 'Infinite'},

{'dimension': 'ehqc', 'scope': 'Subscription', 'provider_id': 'quantinuum',
'utilization': 0.0, 'holds': 0.0, 'limit': 160.0, 'period': 'Infinite'},

{'dimension': 'combined_job_hours', 'scope': 'Workspace', 'provider_id':
'Microsoft', 'utilization': 0.0, 'holds': 0.0, 'limit': 20.0, 'period':
'Monthly'},

{'dimension': 'combined_job_hours', 'scope': 'Subscription', 'provider_id':
'Microsoft', 'utilization': 0.011701412083333333, 'holds': 0.0, 'limit':
1000.0, 'period': 'Monthly'}]
```

En este ejemplo, en la fila `qgs` se ve que la cuenta tiene un límite de `8333334 qgs` con `ionq`, de los cuales se han usado `33334 qgs`. El número de trabajos simultáneos es el número de trabajos que se pueden poner en cola por área de trabajo en cualquier momento.

El `scope` elemento indica si la cuota hace referencia al área de trabajo actual o a la suscripción.

- *Área de trabajo*: se realiza un seguimiento de la cuota de un área de trabajo específica.
- *Suscripción*: se realiza un seguimiento de la cuota en todas las áreas de trabajo dentro de la misma suscripción y región.

El `period` elemento indica el período durante el que se renueva la cuota.

- *Monthly* (Mensual): el uso se restablece el primer día de cada mes.
- *Infinito*: el uso nunca se restablece (también denominado *una sola vez* en la [vista de Azure Portal](#)).

Sugerencia

El `get_quotas` método devuelve los resultados en forma de diccionario de Python. Para verlo un formato más legible, use los siguientes ejemplos de código para imprimir un resumen de las > cuotas restantes de suscripción y de área de trabajo.

Copie el código siguiente para realizar el seguimiento de la cuota en el nivel de suscripción.

Python

```
# This gathers usage against quota for the various providers (quota is set at
the subscription level).
# Note that a provider may have multiple quotas, such as Quantinuum that
limits usage of their Emulator.

rigetti_quota = 0
ionq_quota = 0
quantinuum_hqc_quota = 0
quantinuum_ehqc_quota = 0

rigetti_quota_utilization = 0
ionq_quota_utilization = 0
quantinuum_hqc_quota_utilization = 0
quantinuum_ehqc_quota_utilization = 0

for quota in workspace.get_quotas():
    if (quota['provider_id'] == 'rigetti'):
        rigetti_quota = quota['limit']
        rigetti_quota_utilization = quota['utilization']
    if (quota['provider_id'] == 'ionq'):
```



```

event.amount_consumed
        else:
            amount_utilized_quantinuum_ehqc +=
event.amount_consumed
            print(job.id, event)

print('Rigetti quota use in current workspace: ', "
{:,}".format(amount_utilized_rigetti), '/', "{:,}".format(rigetti_quota))
print('IonQ quota use in current workspace:', "
{:,}".format(amount_utilized_ionq), '/', "{:,}".format(ionq_quota))
print('Quantinuum HQC quota use in current workspace:', "
{:,}".format(amount_utilized_quantinuum_hqc), '/', "
{:,}".format(quantinuum_hqc_quota))
print('Quantinuum eHQC quota use in current workspace:', "
{:,}".format(amount_utilized_quantinuum_ehqc), '/', "
{:,}".format(quantinuum_ehqc_quota))

```

¿Cómo puedo solicitar más cuota?

Puede solicitar aumentos de cuota abriendo un ticket de soporte técnico.

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
2. Seleccione el **área de trabajo de Azure Quantum**.
3. En el panel izquierdo, en **Operaciones**, vaya a la pestaña **Cuotas**.
4. Pulse el botón **Aumentar** en la página de cuotas o seleccione el botón **Nueva solicitud de soporte técnico** en el panel lateral del portal.
5. Se abrirá una incidencia de soporte técnico. Siga estos pasos para rellenar la solicitud.
 - a. Describa el problema como **Solicitud de sustitución de cuota de Azure Quantum**.
 - b. En "Tipo de problema", seleccione **Técnico**.
 - c. Seleccione la suscripción en la que se encuentra el área de trabajo.
 - d. Seleccione **Todos los servicios**.
 - e. Elija **Azure Quantum: versión preliminar** como "Tipo de servicio".
 - f. Elija el área de trabajo para la que desea cambiar la cuota en **Recurso**.
 - g. Elija **Otros** para el tipo de problema.
 - h. Vaya a **Soluciones** y, a continuación, de nuevo a **Detalles**.
 - i. Rellene todos los campos. En **Descripción**, incluya lo siguiente:
 - Nombre del proveedor para el que desea cambiar las cuotas
 - Si desea cambiar las cuotas para el ámbito de la suscripción o el ámbito del área de trabajo

- Qué cuotas desea cambiar y cuánto
- Cualquier justificación de por qué está aumentando su cuota puede ayudarnos a decidir en algunos casos.

Contenido relacionado

- [Preguntas más frecuentes: Descripción de los costos y la facturación de trabajos en Azure Quantum](#)

Preguntas más frecuentes: Descripción de los costos y la facturación de trabajos en Azure Quantum

Artículo • 16/04/2025

En este artículo, encontrará las instrucciones para comprender el costo de ejecutar programas cuánticos en Azure Quantum y cómo administrar las facturas.

¿Cuántos planes de uso están disponibles en Azure Quantum?

Azure Quantum pone a disposición soluciones de Microsoft y de nuestras empresas asociadas, por lo que los detalles de facturación dependerán del proveedor y del plan de precios que seleccione. En la tabla siguiente puede ver las instrucciones generales de precios para usar Azure Quantum.

 Expandir tabla

Plan	Descripción
Planes de facturación	La mayoría de los proveedores facturan en función de los recursos que consuma al ejecutar un trabajo (pago por uso), aunque algunos también ofrecen planes de suscripción. Para más información sobre cómo cada proveedor cobra, consulte Precios de Azure Quantum .

Importante

Ya no podrá usar créditos de Azure Quantum después del 1 de junio de 2025. Si tiene un saldo restante de créditos de Azure Quantum, asegúrese de canjearlos antes de esta fecha.

¿Cómo puedo modificar el plan de uso de mi área de trabajo de Azure Quantum?

Puede modificar el plan de uso actual de cada proveedor y ver los diferentes planes de facturación disponibles en la moneda local.

1. Inicie sesión en [Azure Portal](#)  con las credenciales de la suscripción de Azure.

2. Seleccione el área de trabajo de Azure Quantum.
3. En el panel izquierdo, en **Operaciones**, vaya a la pestaña **Proveedores**.
4. Puede comprobar el plan de uso actual de cada proveedor en la columna **Plan (Plan)**.
5. Haga clic en **Modificar** para ver y modificar los diferentes planes de facturación disponibles para ese proveedor en su moneda local.

Name	Provider type	Plan	Status	Availability	Avg. Queue Time
IonQ	Quantum Computing	Azure Quantum Credits	✓	Available	Modify
Microsoft Quantum Computing	Quantum Computing	Learn & Develop	✓	Available	Modify
Quantinuum	Quantum Computing	Azure Quantum Credits	✓	Available	Upgrade
Rigetti Quantum	Quantum Computing	Azure Quantum Credits	✓	Available	Upgrade

Los proveedores de hardware cuántico establecen los planes de facturación. Para más información sobre la oferta de precios, consulte [la página de precios](#) de Azure Quantum.

¿Cómo puedo ver el informe de costes del trabajo después de que se ejecute?

Después de ejecutar un trabajo, puede comprobar las estimaciones de costos detalladas y usar esta información para comprender el costo de los trabajos individuales. Este costo es el costo facturado por el proveedor; consulte su factura final para conocer los cargos exactos, incluidos los impuestos pertinentes. Para obtener más información, consulte [Cómo recibir mis facturas](#).

Para revisar los costos del trabajo:

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
2. Seleccione el área de trabajo de Azure Quantum.
3. Navegue al panel **Administración de trabajos** dentro de su espacio de trabajo de Azure Quantum.
4. En la lista de trabajos, verá los costos estimados notificados para cada trabajo que ejecute.

Los precios que aparecen a continuación se muestran solo a modo de ejemplo.

Name	Id	Provider	Target	State	Time	Start	End	Execution time	Submission time	Completion time	Cost estimate
Qubit Sample - 3-qubit GHZ circuit	aaaa0000-bb11-2222-3333-444444444444	ionq	ionq.gpu	Succeeded	00:00:02	2021-12-13, 11:27:00 a.m.	2021-12-13, 11:54:42 a.m.				\$1.00
Qubit Sample - 3-qubit GHZ circuit	bbbb1111-cc22-3333-4444-555555555555	ionq	ionq.simulator	Succeeded	00:00:01	2021-12-13, 11:26:44 a.m.	2021-12-13, 11:26:46 a.m.				\$0.00
Qubit Sample - 3-qubit GHZ circuit	cccc2222-dd33-4444-5555-666666666666	ionq	ionq.gpu	Succeeded	00:00:06	2021-11-26, 10:23:22 a.m.	2021-11-26, 10:28:02 a.m.				\$1.00
Qubit Sample - 3-qubit GHZ circuit	ddddd333-ee44-5555-6666-777777777777	ionq	ionq.simulator	Succeeded	00:00:01	2021-11-26, 10:23:33 a.m.	2021-11-26, 10:23:36 a.m.				\$1.00
SampleRandomNumber	eeee4444-ff55-6666-7777-888888888888	ionq	ionq.gpu	Succeeded	00:00:05	2021-10-18, 4:30:33 p.m.	2021-10-18, 8:56:53 p.m.				\$1.00
SampleRandomNumber	ffff5555-a666-7777-8888-999999999999	ionq	ionq.simulator	Succeeded	00:00:01	2021-10-18, 4:29:58 p.m.	2021-10-18, 4:30:17 p.m.				\$1.00

Los precios que aparecen a continuación se muestran solo a modo de ejemplo.

Las definiciones siguientes le ayudan a comprender la tabla de estimación de costos:

- ⓘ **Nota**

IonQ tiene un costo mínimo de 1,00 USD para ejecutar un trabajo en la QPU de IonQ. En el caso de los trabajos pequeños, puede observar que los datos de **Consumed Units** de la tabla de estimación de costos de los trabajos son inferiores a **Billed Units** de la tabla de estimación de costos de los trabajos.

¿Por qué no veo el costo de mi trabajo?

Algunos proveedores de Azure Quantum no admiten la generación de informes por trabajo, pero todavía puede ver la factura en **Cost Management** en Azure Portal. Para obtener más información, consulte [¿Cómo puedo ver mis facturas?](#).

Cómo recibir mis facturas?

Las facturas se envían a la dirección de correo electrónico que usó para registrarse en Azure. Si necesita cambiar la dirección de correo electrónico, póngase en contacto con [el soporte técnico de](#) Azure.

Si usa una suscripción personalizada, es posible que las facturas se envíen a la dirección de correo electrónico de la persona que configura la suscripción.

Las facturas de proveedores de terceros de Azure Quantum son de tipo **Azure Marketplace y Reservations**. Recibirá una sola factura de Azure Marketplace y Reservation, que proporciona un desglose de los cargos de cada proveedor cuántico.

¿Qué incluye mi factura de Azure Quantum?

Las facturas de Azure Marketplace y Reservations incluyen gastos en proveedores de terceros de Azure. Azure Quantum ofrece actualmente hardware cuántico de IonQ, Quantinuum y Rigetti. Para más información, consulte [Lista](#) de proveedores de Azure Quantum.

Las facturas de Azure Marketplace y Reservations no **incluyen gastos en servicios de Azure de primera entidad, como las cuentas de almacenamiento asociadas** al área de trabajo de Azure Quantum. Normalmente representan cantidades muy pequeñas.

Para más información sobre los costos de infraestructura de Azure, consulte [Precios de Azure Blob Storage](#).

¿Cómo se facturan las facturas?

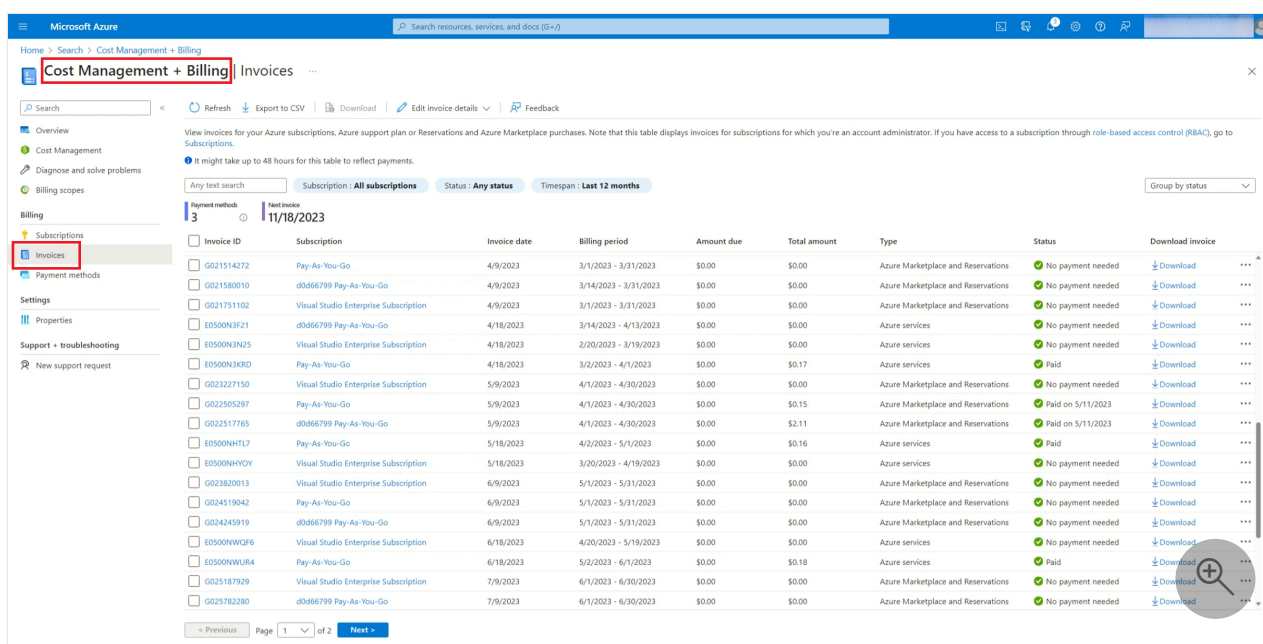
Las facturas se pagan con el mismo método de pago que se usa para la suscripción de Azure del área de trabajo de Azure Quantum. Es posible que tenga diferentes métodos de pago para distintas suscripciones.

Las facturas se facturan en arrendo. Por ejemplo, la factura que recibe en febrero corresponde al uso que incurrió en enero.

¿Cómo puedo ver mis facturas?

Puede ver las facturas anteriores en Azure Portal.

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
2. En la barra de búsqueda superior, escriba **Cost Management + Billing**.
3. Si tiene más de un ámbito de facturación, seleccione el ámbito que desea ver en el panel **Ámbito de facturación**. Si no es así, omita este paso.
4. En el panel de navegación izquierdo, seleccione **Facturas**.



Invoice ID	Subscription	Invoice date	Billing period	Amount due	Total amount	Type	Status	Download invoice
G021514272	Pay-As-You-Go	4/9/2023	3/1/2023 - 3/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
d0d66799 Pay-As-You-Go		4/9/2023	3/14/2023 - 3/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
G021751102	Visual Studio Enterprise Subscription	4/9/2023	3/1/2023 - 3/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
d0500N3F21	d0d66799 Pay-As-You-Go	4/18/2023	3/14/2023 - 4/13/2023	\$0.00	\$0.00	Azure services	No payment needed	Download
E0500N3N25	Visual Studio Enterprise Subscription	4/18/2023	2/20/2023 - 3/19/2023	\$0.00	\$0.00	Azure services	No payment needed	Download
E0500N3KRD	Pay-As-You-Go	4/18/2023	3/2/2023 - 4/1/2023	\$0.00	\$0.17	Azure services	Paid	Download
G023227150	Visual Studio Enterprise Subscription	5/9/2023	4/1/2023 - 4/30/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
G022505297	Pay-As-You-Go	5/9/2023	4/1/2023 - 4/30/2023	\$0.00	\$0.15	Azure Marketplace and Reservations	Paid on 5/11/2023	Download
G022517765	d0d66799 Pay-As-You-Go	5/9/2023	4/1/2023 - 4/30/2023	\$0.00	\$2.11	Azure Marketplace and Reservations	Paid on 5/11/2023	Download
E0500NH7L7	Pay-As-You-Go	5/18/2023	4/2/2023 - 5/1/2023	\$0.00	\$0.16	Azure services	Paid	Download
E0500NH0YV	Visual Studio Enterprise Subscription	5/18/2023	3/20/2023 - 4/19/2023	\$0.00	\$0.00	Azure services	No payment needed	Download
G023820013	Visual Studio Enterprise Subscription	6/9/2023	5/1/2023 - 5/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
G024519042	Pay-As-You-Go	6/9/2023	5/1/2023 - 5/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
G024245919	d0d66799 Pay-As-You-Go	6/9/2023	5/1/2023 - 5/31/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
E0500NWQF6	Visual Studio Enterprise Subscription	6/18/2023	4/20/2023 - 5/19/2023	\$0.00	\$0.00	Azure services	No payment needed	Download
E0500NWUR4	Pay-As-You-Go	6/18/2023	5/2/2023 - 6/1/2023	\$0.00	\$0.18	Azure services	Paid	Download
G025187929	Visual Studio Enterprise Subscription	7/9/2023	6/1/2023 - 6/30/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download
G025782380	d0d66799 Pay-As-You-Go	7/9/2023	6/1/2023 - 6/30/2023	\$0.00	\$0.00	Azure Marketplace and Reservations	No payment needed	Download

En la **vista Facturas**, puede ver todas las facturas. Las facturas de proveedores de terceros de Azure Quantum son de tipo **Azure Marketplace y Reservations**.

Puede filtrar por fecha, suscripción y estado. También puede descargar las facturas como archivos PDF. Puede ver la fecha de facturación y el período de facturación. El período de facturación no es el mismo que el mes en el que recibe la factura.

Contenido relacionado

- [Cuotas de Azure Quantum](#)

Uso de una entidad de servicio para autenticarse en el área de trabajo de Azure Quantum

Artículo • 21/06/2024

A veces, usar la autenticación interactiva o autenticarse como una cuenta de usuario no es una opción adecuada. Estos casos pueden surgir cuando quiere enviar trabajos desde un servicio web, otro rol de trabajo o un sistema automatizado. Una opción consiste en configurar una identidad administrada; otra opción es usar una [entidad de servicio](#), algo que se explicará en este artículo.

Requisito previo: Creación de una entidad de servicio y un secreto de aplicación

Para autenticarse como entidad de servicio, primero debe [crear una](#).

Para crear una entidad de servicio, asignarle acceso y generar una credencial, haga lo siguiente:

1. [Cree una aplicación de Azure AD](#):

 Nota

No es necesario que establezca un identificador URI de redirección.

- a. Una vez creada, tome nota de los campos *Id. de aplicación (cliente)* e *Id. de directorio (inquilino)*.
2. [Cree una credencial](#) para iniciar sesión como la aplicación:
 - a. En la configuración de la aplicación, seleccione **Certificados y secretos**.
 - b. En **Secretos de cliente**, seleccione **Crear un secreto**.
 - c. Proporcione una descripción y una duración y, a continuación, seleccione **Agregar**.
 - d. Copie el valor del secreto en un lugar seguro de inmediato, ya que no podrá verlo de nuevo.
 3. Conceda permisos a la entidad de servicio para acceder al área de trabajo:
 - a. Abra Azure Portal.

- b. En la barra de búsqueda, escriba el nombre del grupo de recursos en el que creó el área de trabajo. Seleccione el grupo de recursos cuando aparezca en los resultados.
- c. En la información general del grupo de recursos, seleccione **Control de acceso (IAM)**.
- d. Seleccione **Agregar asignación de roles**.
- e. Busque y seleccione la entidad de servicio.
- f. Asígnele el rol de **Colaborador** o **Propietario**.

⚠ Nota

Para crear una asignación de roles en el área de trabajo o el grupo de recursos, debe ser *propietario* o *administrador de acceso de usuario* en el ámbito de la asignación de roles. Si no tiene permisos para crear la entidad de servicio en su suscripción, deberá solicitar permiso al *propietario* o al *administrador* de la suscripción de Azure.

Si solo tiene permisos en el nivel de grupo de recursos o de área de trabajo, puede crear la entidad de servicio con el rol Colaborador mediante:

```
az ad sp create-for-rbac --role Contributor --scopes  
/subscriptions/<SUBSCRIPTION-ID>
```

Autenticación como la entidad de servicio

Opción 1: Usar variables de entorno. Las credenciales predeterminadas usadas en la creación del objeto `Workspace` son [DefaultAzureCredential](#), que intentará varios tipos de autenticación. El primero es [EnvironmentCredential](#) y, con él, se pasan las credenciales de la entidad de servicio mediante las siguientes variables de entorno:

- **AZURE_TENANT_ID:** identificador del inquilino de la entidad de servicio. También se denomina identificador de su "directorio".
- **AZURE_CLIENT_ID:** identificador de cliente de la entidad de servicio.
- **AZURE_CLIENT_SECRET:** uno de los secretos de cliente de la entidad de servicio.

Opción 2: Usar ClientSecretCredential. Pase [ClientSecretCredential](#) durante la creación de instancias del objeto `Workspace` o establezca la propiedad `credentials`.

Python

```
from azure.identity import ClientSecretCredential
```

```
tenant_id = os.environ["AZURE_TENANT_ID"]
client_id = os.environ["AZURE_CLIENT_ID"]
client_secret = os.environ["AZURE_CLIENT_SECRET"]
credential = ClientSecretCredential(tenant_id=tenant_id,
client_id=client_id, client_secret=client_secret)

workspace.credentials = credential
```

ⓘ Nota

El método `workspace.login()` está en desuso y ya no es necesario. La primera vez que se llama al servicio, se intenta realizar una autenticación con las credenciales pasadas al constructor `Workspace` o a su propiedad `credentials`. Si no se han pasado credenciales, se intentan varios métodos de autenticación con [DefaultAzureCredential](#) [↗].

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) [↗] | [Obtener ayuda en Microsoft Q&A](#)

Uso de una identidad administrada para autenticarse en el área de trabajo de Azure Quantum

Artículo • 21/06/2024

Hay varios escenarios en los que usar la autenticación interactiva o autenticarse como una cuenta de usuario no es una opción adecuada. Por ejemplo, puede que quiera enviar trabajos desde una máquina virtual (VM) o desde una aplicación de funciones. Una opción es [autenticarse mediante una entidad de servicio](#); otra opción es configurar una identidad administrada, que se explicará en este artículo.

Configuración de una identidad administrada

Una identidad administrada permite que una aplicación acceda a otros recursos de Azure (como el área de trabajo de Azure Quantum) y se autentique con estos recursos.

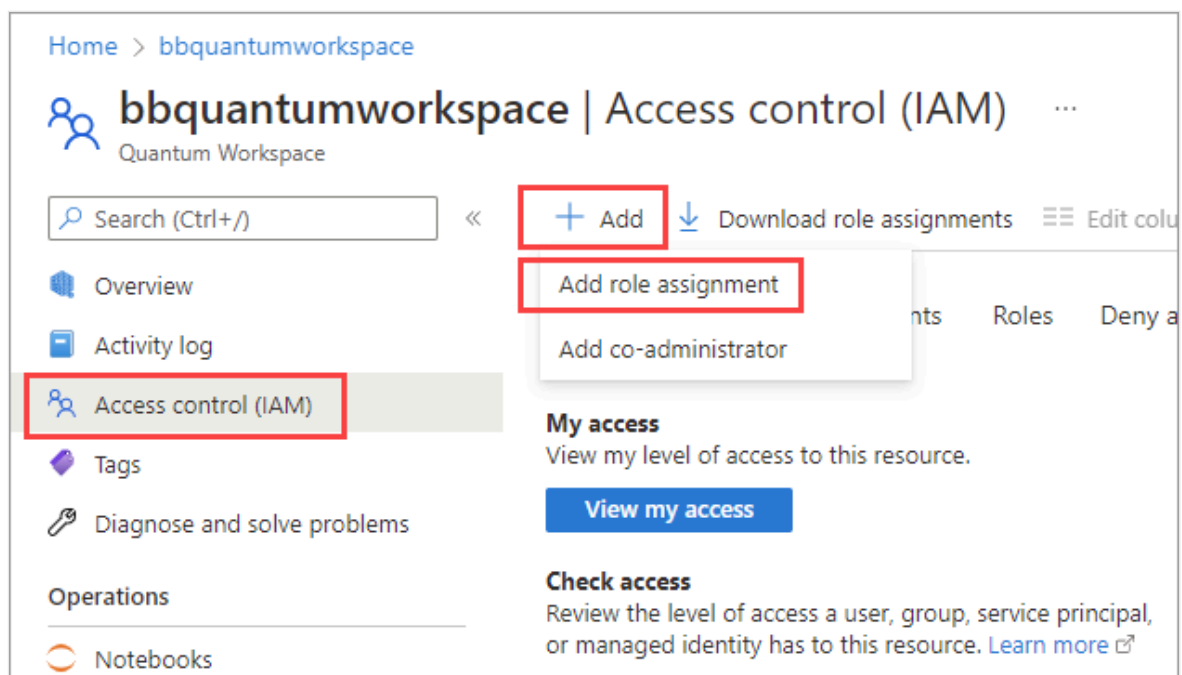
Para configurar una identidad administrada:

1. A través de Azure Portal, busque el recurso al que quiere dar acceso. Este recurso puede ser una máquina virtual, una aplicación de funciones u otra aplicación.
2. Seleccione el recurso y vea la página de información general.
3. En **Configuración**, seleccione **Identidad**.
4. Configure el valor **Estado** en **On** (Activado).
5. Seleccione **Guardar** para conservar la configuración y elija **Sí** en el cuadro de diálogo que se abre.

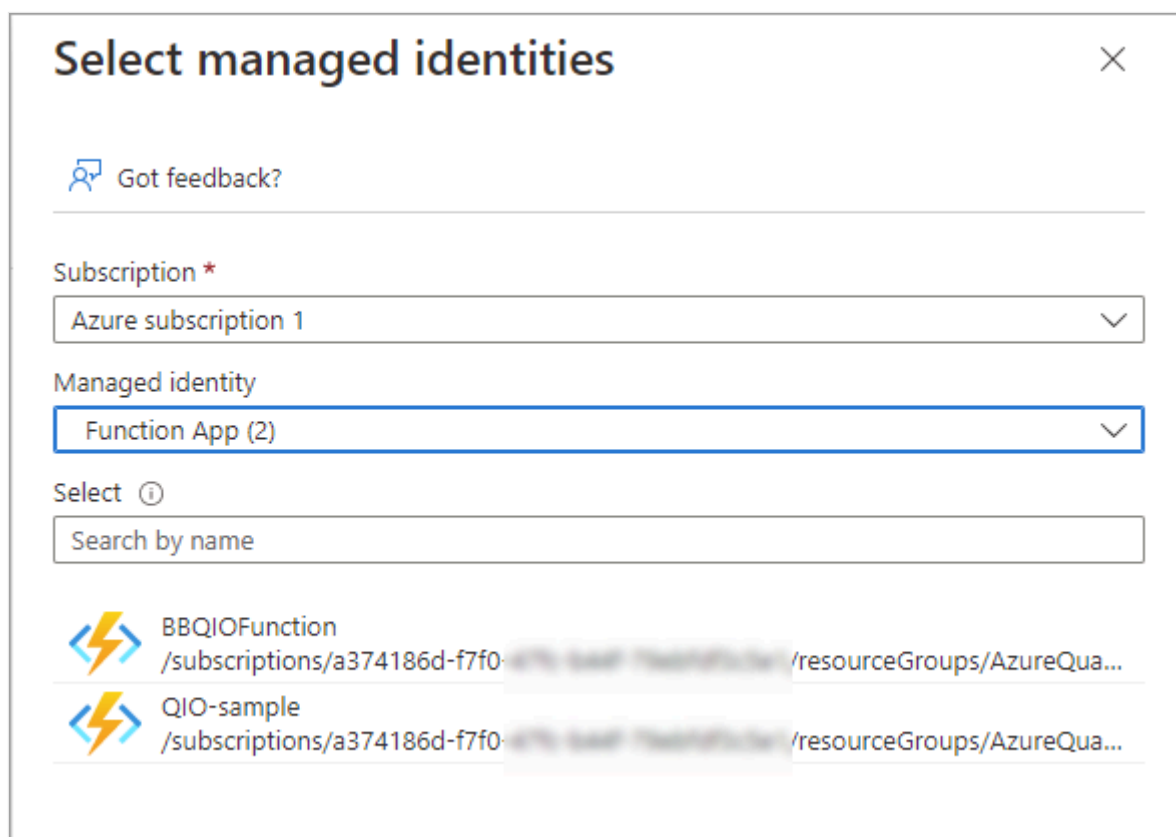
Concesión de acceso al área de trabajo de Azure Quantum

Para permitir que el recurso acceda al área de trabajo de Azure Quantum:

1. Vaya al área de trabajo de Azure Quantum y seleccione **Control de acceso (IAM)** en el menú de la izquierda.
2. Seleccione **Agregar** y **Agregar asignación de roles**.



3. En la página **Agregar asignación de roles**, seleccione **Colaborador** y seleccione **Siguiente**.
4. En la pestaña **Miembros**, seleccione **Asignar acceso a**, seleccione **Identidad administrada** y, luego, seleccione **+ Seleccionar miembros**.
5. En el **menú emergente Seleccionar identidades administradas**, seleccione una categoría en la **lista desplegable Identidad administrada**.
6. Seleccione el recurso deseado en la lista y haga clic en **Seleccionar**.



7. Seleccione **Siguiente** y, a continuación, seleccione **Revisar y asignar**.

Inicio de sesión en el área de trabajo de Azure Quantum

Ahora debería poder usar el área de trabajo de Quantum desde el recurso seleccionado. Por ejemplo, al usar el área de trabajo desde dentro de una máquina virtual, ya no tendrá que autenticarse cada vez.

En algunos casos, es posible que también quiera especificar explícitamente el uso de una credencial de identidad administrada dentro del código:

Python

```
from azure.identity import ManagedIdentityCredential

from azure.quantum import Workspace
workspace = Workspace (
    resource_id = "",
    location = "" ,
    credential=ManagedIdentityCredential()
)
```

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Autenticación en el área de trabajo mediante una clave de acceso

Artículo • 24/10/2024

Las claves de acceso se usan para autenticar y autorizar el acceso al área de trabajo de Azure Quantum. Puede usar claves de acceso para conectarse y conceder acceso al área de trabajo mediante cadena de conexión.

En este artículo, aprenderá a habilitar o deshabilitar las claves de acceso para el área de trabajo de Azure Quantum. También puede volver a generar nuevas claves para garantizar la seguridad del área de trabajo.

Advertencia

Almacenar las claves de acceso o la cadena de conexión de su cuenta en texto no cifrado supone un riesgo de seguridad y no se recomienda. Almacene las claves de cuenta en un formato cifrado o migre las aplicaciones para usar la autorización de Microsoft Entra para acceder al área de trabajo de Azure Quantum.

Requisitos previos

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Consulte [Creación de un área de trabajo de Azure Quantum](#).
- La versión más reciente del paquete de Azure Quantum `azure-quantum`.

Bash

```
!pip install --upgrade azure-quantum
```

- Si usa la CLI de Azure, debe tener la versión más reciente. Para obtener instrucciones de instalación, consulte:
 - [Instalación de la CLI de Azure en Windows](#)
 - [Instalación de la CLI de Azure en Linux](#)
 - [Instalación de la CLI de Azure en macOS](#)

Conexión al área de trabajo de Azure Quantum con un cadena de conexión

El `azure-quantum` paquete proporciona una [Workspace clase](#) que representa un área de trabajo de Azure Quantum. Para conectarse al área de trabajo de Azure Quantum, cree `Workspace` un objeto mediante el cadena de conexión como autenticador. Para obtener más información, consulte [cómo copiar un cadena de conexión](#).

Al crear un `Workspace` objeto, tiene dos opciones para identificar el área de trabajo de Azure Quantum.

- Puede crear un `Workspace` objeto llamando a `from_connection_string`.

Python

```
# Creating a new Workspace object from a connection string
from azure.quantum import Workspace

connection_string = "[Copy connection string]"
workspace = Workspace.from_connection_string(connection_string)

print(workspace.get_targets())
```

- Si no desea copiar el cadena de conexión en el código, también puede almacenar el cadena de conexión en una variable de entorno y usar `Workspace()`.

Python

```
# Using environment variable to connect with connection string

connection_string = "[Copy connection string]"

import os

os.environ["AZURE_QUANTUM_CONNECTION_STRING"] = connection_string

from azure.quantum import Workspace

workspace = Workspace()
print(workspace.get_targets())
```

Administrar las claves de acceso y los cadena de conexión

💡 Sugerencia

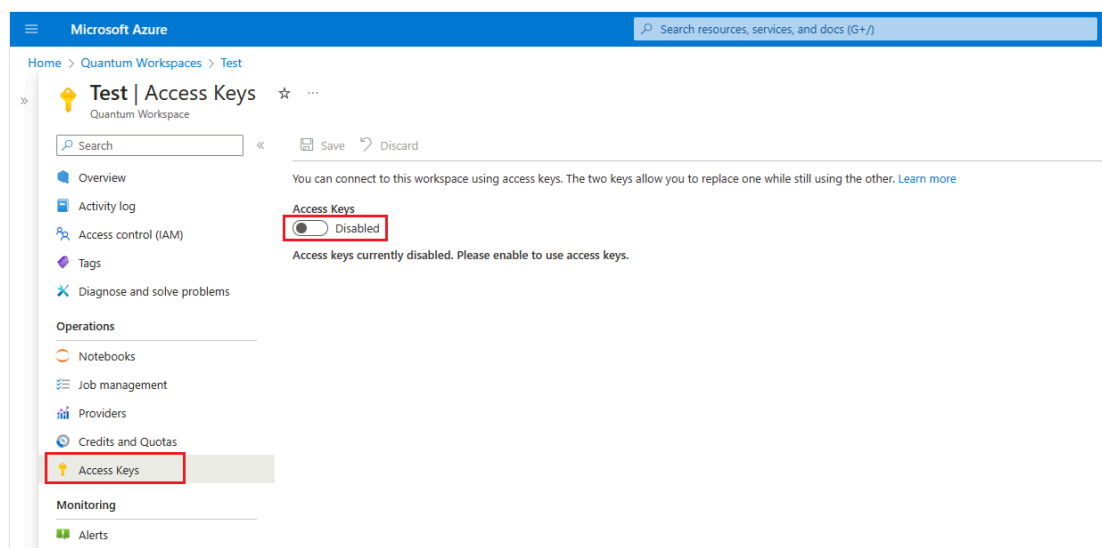
Cada área de trabajo de Azure Quantum tiene **claves principales y secundarias** y sus cadena de conexión correspondientes. Si desea permitir el acceso al área de trabajo a otros usuarios, puede compartir la clave secundaria y usar la principal para sus propios servicios. De este modo, puede reemplazar la clave secundaria según sea necesario sin tener tiempo de inactividad en sus propios servicios. Para más información sobre cómo compartir el acceso al área de trabajo, consulte [Uso compartido del acceso al](#) área de trabajo.

Azure Portal

Puede administrar las claves de acceso y las cadena de conexión para el área de trabajo de Azure Quantum en Azure Portal.

Habilitación y deshabilitación de claves de acceso

1. Inicie sesión en [Azure Portal](#) y seleccione el área de trabajo de Azure Quantum.
2. En el panel izquierdo, vaya a Claves de acceso de operaciones>.
3. Cambie el botón de alternancia en Claves de acceso a **Habilitado** o **Deshabilitado**.
4. Haga clic en **Guardar** para guardar los cambios.



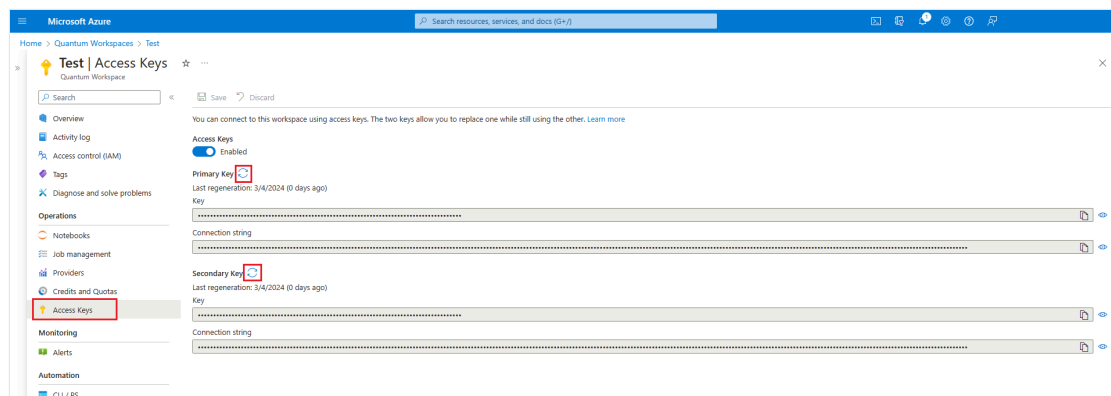
📘 Importante

Cuando las claves de acceso están deshabilitadas, **todas las solicitudes que usan cadena de conexión o claves de acceso no están** autorizadas. Todavía puede usar los parámetros del área de trabajo para conectarse al área de trabajo.

Regeneración de nuevas claves de acceso

Si sospecha que las claves de acceso se han puesto en peligro o desea dejar de compartir el acceso del área de trabajo con otros usuarios, puede volver a generar las claves de acceso principal o secundaria, o ambas, para garantizar la seguridad del área de trabajo.

1. Inicie sesión en [Azure Portal](#) y seleccione el área de trabajo de Azure Quantum.
2. En el panel izquierdo, vaya a Claves de acceso de operaciones>.
3. Las claves de acceso deben estar habilitadas para volver a generar nuevas claves. Si las claves de acceso están deshabilitadas, primero debe habilitarlas.
4. Haga clic en el icono de flecha circular para volver a generar la clave principal o secundaria.



Comentarios

¿Le ha resultado útil esta página?

👍 Sí

👎 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Protección de recursos de Azure Quantum con bloqueos de Azure Resource Manager (ARM)

Microsoft recomienda bloquear todas las áreas de trabajo de Azure Quantum y las cuentas de almacenamiento vinculadas con un bloqueo de recursos de Azure Resource Manager (ARM) para evitar la eliminación accidental o malintencionada. Por ejemplo, es posible que los profesores quieran impedir que los alumnos modifiquen las SKU del proveedor, pero aún así les permitan enviar trabajos.

Hay dos tipos de bloqueos de recursos de ARM:

- Un bloqueo **CannotDelete** impide que los usuarios eliminen un recurso, pero permiten leer y modificar su configuración.
- Un bloqueo **ReadOnly** impide que los usuarios modifiquen la configuración de un recurso (incluida la eliminación), pero permite leer su configuración. Para obtener más información sobre los bloqueos de recursos, consulte [Bloqueo de recursos para evitar cambios inesperados](#).

ⓘ Nota

Si ya usa una [plantilla de ARM o Bicep para administrar las áreas de trabajo de Azure Quantum](#), puede agregar los procedimientos de este artículo a las plantillas existentes.

Configuraciones de bloqueo recomendadas

En la tabla siguiente se muestran las configuraciones de bloqueo de recursos recomendadas para implementar para un área de trabajo de Azure Quantum.

[Expandir tabla](#)

Resource	Tipo de bloqueo	Notas
Workspace	Delete	Impide que se elimine el área de trabajo.
Workspace	Solo lectura	Impide las modificaciones en el área de trabajo, incluidas las adiciones o eliminaciones de proveedores, a la vez que permite a los usuarios enviar trabajos. Para modificar proveedores cuando se establece este bloqueo, debe quitar el bloqueo de recursos , realizar los cambios y volver a implementar el bloqueo.

Resource	Tipo de bloqueo	Notas
Cuenta de almacenamiento	Delete	Impide que se elimine la cuenta de almacenamiento.

Se deben evitar las siguientes configuraciones:

Importante

Establecer los siguientes bloqueos de ARM puede hacer que el área de trabajo funcione incorrectamente.

 Expandir tabla

Resource	Tipo de bloqueo	Notas
Cuenta de almacenamiento	Solo lectura	Establecer un bloqueo de recursos de solo lectura en la cuenta de almacenamiento puede provocar errores en la creación del área de trabajo, el envío de trabajos y la captura de trabajos.
Suscripción primaria del área de trabajo o el grupo de recursos primario del área de trabajo o la cuenta de almacenamiento	Solo lectura	Cuando se aplica un bloqueo de recursos a un recurso primario, todos los recursos de ese elemento primario heredan el mismo bloqueo, incluidos los recursos creados en una fecha posterior. Para un control más granular, los bloqueos de recursos se deben aplicar directamente en el nivel de recurso.

Prerrequisitos

Debe ser **propietario** o administrador de **acceso de usuario** de un recurso para aplicar bloqueos de recursos de ARM. Para más información, consulte [Roles integrados de Azure](#).

Implementación de la línea de comandos

Necesitará Azure PowerShell o la CLI de Azure para implementar el bloqueo. Si usa la CLI de Azure, debe tener la versión más reciente. Para obtener instrucciones de instalación, consulte:

- [Instalación de Azure PowerShell](#)
- [Instalación de la CLI de Azure](#)

Importante

Si no ha usado la CLI de Azure con Azure Quantum antes, siga los pasos descritos en la sección [Configuración del entorno](#) para agregar la `quantum` extensión y registrar el espacio de nombres de Azure Quantum.

Inicio de sesión en Azure

Después de instalar la CLI de Azure o Azure PowerShell, asegúrese de iniciar sesión por primera vez. Elija una de las pestañas siguientes y ejecute los comandos de línea de comandos correspondientes para iniciar sesión en Azure:

Azure CLI

Azure CLI

```
az login
```

Si tiene varias suscripciones de Azure, seleccione la suscripción con los recursos que desea bloquear. Reemplace por `SubscriptionName` el nombre de la suscripción o el identificador de suscripción. Por ejemplo

Azure CLI

```
az account set --subscription "Azure subscription 1"
```

Creación de un bloqueo de recursos de ARM

Al implementar un bloqueo de recursos, se especifica un nombre para el bloqueo, el tipo de bloqueo y información adicional sobre el recurso. Esta información se puede copiar y pegar desde la página principal del recurso en el portal de Azure Quantum.

Azure CLI

Azure CLI

```
az lock create \  
  --name <lock> \  
  --resource-group <resource-group> \  
  --
```

```
--resource <workspace> \  
--lock-type CanNotDelete \  
--resource-type Microsoft.Quantum/workspaces
```

- name: un nombre descriptivo para el bloqueo.
- resource-group: el nombre del grupo de recursos primario.
- resource: el nombre del recurso al que se va a aplicar el bloqueo.
- tipo de bloqueo: el tipo de bloqueo que se va a aplicar, ya sea **CanNotDelete** o **ReadOnly**.
- tipo de recurso: el tipo del target recurso.

Por ejemplo, para crear un bloqueo **CanNotDelete** en un área de trabajo:

Azure CLI

```
az lock create \  
  --name ArmLockWkspDelete \  
  --resource-group armlocks-resgrp \  
  --resource armlocks-wksp \  
  --lock-type CanNotDelete \  
  --resource-type Microsoft.Quantum/workspaces
```

Si se ejecuta correctamente, Azure devuelve la configuración de bloqueo en formato JSON:

JSON

```
{  
  "id": "/subscriptions/<ID>/resourcegroups/armlocks-resgrp/providers/Microsoft.Quantum/workspaces/armlocks-wksp/providers/Microsoft.Authorization/locks/ArmLockWkspDelete",  
  "level": "CanNotDelete",  
  "name": "ArmLockWkspDelete",  
  "notes": null,  
  "owners": null,  
  "resourceGroup": "armlocks-resgrp",  
  "type": "Microsoft.Authorization/locks"  
}
```

Para crear un bloqueo **ReadOnly** en un área de trabajo:

Azure CLI

```
az lock create \  
  --name ArmLockWkspRead \  
  --resource-group armlocks-resgrp \  
  --resource armlocks-wksp \  
  --lock-type ReadOnly
```

```
--lock-type ReadOnly \  
--resource-type Microsoft.Quantum/workspaces
```

JSON

```
{  
  "id": "/subscriptions/<ID>/resourcegroups/armlocks-  
resgrp/providers/Microsoft.Quantum/workspaces/armlocks-  
wksp/providers/Microsoft.Authorization/locks/ArmLockWkspRead",  
  "level": "ReadOnly",  
  "name": "ArmLockWkspRead",  
  "notes": null,  
  "owners": null,  
  "resourceGroup": "armlocks-resgrp",  
  "type": "Microsoft.Authorization/locks"  
}
```

Para crear un bloqueo **CanNotDelete** en una cuenta de almacenamiento:

Azure CLI

```
az lock create \  
  --name ArmLockStoreDelete \  
  --resource-group armlocks-resgrp \  
  --resource armlocksstorage --lock-type CanNotDelete \  
  --resource-type Microsoft.Storage/storageAccounts
```

JSON

```
{  
  "id": "/subscriptions/<ID>/resourcegroups/armlocks-  
resgrp/providers/Microsoft.Storage/storageAccounts/armlocksstorage/providers/Mic  
rosoft.Authorization/locks/ArmLockStoreDelete",  
  "level": "CanNotDelete",  
  "name": "ArmLockStoreDelete",  
  "notes": null,  
  "owners": null,  
  "resourceGroup": "armlocks-resgrp",  
  "type": "Microsoft.Authorization/locks"  
}
```

Visualización y eliminación de bloqueos

Para ver o eliminar bloqueos:

Azure CLI

Para obtener más información, consulte la [referencia az lock](#).

Visualización de todos los bloqueos en una suscripción

Azure CLI

```
az lock list
```

Visualización de todos los bloqueos en un área de trabajo

Azure CLI

```
az lock list \
  --resource-group armlocks-resgrp \
  --resource-name armlocks-wksp \
  --resource-type Microsoft.Quantum/workspaces
```

Visualización de todos los bloqueos de todos los recursos de un grupo de recursos

Azure CLI

```
az lock list --resource-group armlocks-resgrp
```

Visualización de las propiedades de un solo bloqueo

azcli

```
az lock show \
  --name ArmLockStoreDelete \
  --resource-group armlocks-resgrp \
  --resource-name armlocksstorage \
  --resource-type Microsoft.Storage/storageAccounts
```

Eliminar un bloqueo

azcli

```
az lock delete \
  --name ArmLockStoreDelete \
  --resource-group armlocks-resgrp \
  --resource-name armlocksstorage \
  --resource-type Microsoft.Storage/storageAccounts
```

Si la eliminación se realiza correctamente, Azure no devuelve un mensaje. Para comprobar la eliminación, puede ejecutar `az lock list`.

Pasos siguientes

- [Administración de áreas de trabajo cuánticas con la CLI de Azure](#)
 - [Administración de áreas de trabajo cuánticas con Azure Resource Manager](#)
-

Last updated on 17/10/2025

Actualización de Azure Quantum Development Kit a la versión más reciente

Aprenda a actualizar Azure Quantum Development Kit (QDK) a la versión más reciente.

Requisitos previos

- En este artículo se supone que ya tiene instalada la extensión QDK de Azure en Visual Studio Code (VS Code). Si va a instalar por primera vez, consulte la [guía de instalación](#).

Actualización de la extensión de VS Code

De forma predeterminada, VS Code actualiza automáticamente las extensiones. Una vez actualizada una extensión, se le pedirá que vuelva a cargar VS Code. Para deshabilitar las actualizaciones automáticas y actualizar las extensiones manualmente, consulte [Actualización automática](#) de extensiones en la documentación de VS Code.

Actualización de los paquetes de Python de Azure Quantum

📘 Importante

Si va a actualizar desde un entorno de Qiskit anterior, consulte [Actualización del paquete azure-quantum con compatibilidad con Qiskit en un entorno de Python virtual \(recomendado\)](#).

1. Actualice a los paquetes más recientes `qdk` y `azure-quantum` de Python con el instalador de paquetes para Python (pip).

Símbolo del sistema de Windows

```
pip install --upgrade qdk
```

Para agregar compatibilidad con el análisis, la transformación, la generación de código y la simulación de circuitos Qiskit, instale los paquetes opcionales `qiskit` y `widget`.

Símbolo del sistema de Windows

```
pip install --upgrade qdk[qiskit,jupyter]
```

Símbolo del sistema de Windows

```
pip install --upgrade azure-quantum
```

Actualización del paquete azure-quantum con compatibilidad con Qiskit en un entorno de Python virtual (recomendado)

El `azure-quantum` paquete de Python incluye compatibilidad opcional para crear y enviar circuitos Qiskit a Azure Quantum. Al instalar el `azure-quantum` paquete con compatibilidad con Qiskit, el paquete instala la versión más reciente de Qiskit, lo que podría causar problemas con un entorno de Qiskit existente. Para garantizar un entorno de desarrollo estable, cree un entorno de Python virtual e instále `azure-quantum` allí.

Para crear un entorno de Python virtual e instalar `azure-quantum` con compatibilidad con Qiskit, siga estos pasos:

1. Cree una carpeta local, por ejemplo `~/qiskit10-env`.
2. Ejecute `venv` con la ruta de acceso a la carpeta.

Bash

```
python -m venv ~/qiskit10-env
```

3. Active el entorno.

Bash

```
~/qiskit10-env/bin/activate
```

4. Ejecute `pip list` para ver que solo los paquetes principales están instalados en el nuevo entorno.
5. Para instalar el `azure-quantum` paquete, ejecute el siguiente comando:

Bash

```
pip install azure-quantum[qiskit]
```

6. Instale otros paquetes que usó en el entorno anterior según sea necesario. Puede ejecutar `pip list` en cada entorno para comparar paquetes y versiones.

❗ **Nota:**

Consulte [Cambios](#) de empaquetado de Qiskit 1.0 para obtener más información sobre la compatibilidad de paquetes.

❗ **Nota:**

También puede abrir el entorno virtual en VS Code. En el menú **Ver**, elija **Paleta de comandos** y escriba **Python: Crear entorno** y elija **venv**. A continuación, elija **Abrir carpeta...** y seleccione la carpeta de entorno que creó anteriormente. Para obtener más información sobre el uso de entornos en VS Code, consulte [Entornos de Python en VS Code](#).

Actualización del `azure-quantum` paquete con compatibilidad con Qiskit en el entorno actual

También puede actualizar el `azure-quantum` paquete con compatibilidad con Qiskit sin usar un entorno virtual. Sin embargo, las actualizaciones de los `qiskit` paquetes de un entorno existente podrían provocar conflictos de dependencia con otros paquetes. Consulte [Cambios](#) de empaquetado de Qiskit 1.0 para obtener más información sobre la compatibilidad de paquetes.

Para actualizar el `azure-quantum` paquete, complete los pasos siguientes:

1. Desinstale los paquetes `azure-quantum` y `qiskit` existentes.

Símbolo del sistema de Windows

```
pip uninstall -y azure-quantum qiskit qiskit-terra qiskit-qir
```

2. Vuelva a instalar `azure-quantum` con el adicional opcional `qiskit` :

Símbolo del sistema de Windows

```
pip install azure-quantum[qiskit]
```

Actualización de la extensión de la CLI `quantum` de Azure

Actualice o instale la extensión más reciente de la CLI `quantum` de Azure.

1. Abra un símbolo del sistema de Windows.

2. En el símbolo del sistema, ejecute el siguiente comando:

Azure CLI

```
az extension add \  
  --upgrade \  
  --name quantum
```

Last updated on 12/11/2025

Agregar o quitar un proveedor

19/08/2025

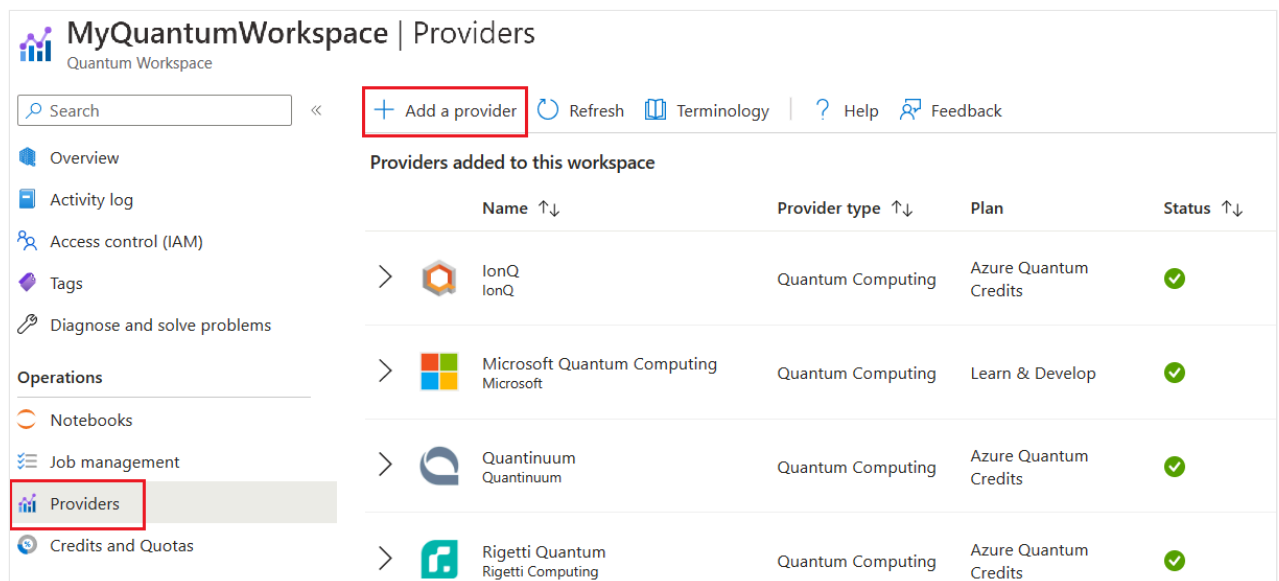
⚠ Advertencia

El 30 de septiembre de 2025, el servicio Azure Quantum solo estará disponible en las siguientes regiones: Este de EE. UU., Oeste de EE. UU., Norte de Europa y Oeste de Europa. Cree nuevas áreas de trabajo de Quantum en una de estas regiones. Si el área de trabajo existente no está en una de estas regiones, consulte [Migración de los datos del trabajo de Azure Quantum](#).

Normalmente, los proveedores se agregan a un área de trabajo de Azure Quantum cuando se crea el área de trabajo. Sin embargo, se pueden agregar proveedores adicionales (o proveedores existentes) en cualquier momento después de eso. Si envía un trabajo cuántico a un proveedor que aún no está en el área de trabajo, recibirá un mensaje de error que le pedirá que instale el proveedor necesario.

Para agregar un proveedor a un área de trabajo existente:

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
2. Seleccione **Inicio** y vaya al área de trabajo de Azure Quantum.
3. Seleccione **Proveedores**. Se muestran todos los proveedores actuales.
4. Seleccione **Agregar un proveedor**. Todos los proveedores disponibles se muestran en **Agregar proveedores adicionales**, junto con la información de precios.



MyQuantumWorkspace | Providers
Quantum Workspace

Search << **+ Add a provider** Refresh Terminology ? Help Feedback

Providers added to this workspace

	Name ↑↓	Provider type ↑↓	Plan	Status ↑↓
>	 IonQ IonQ	Quantum Computing	Azure Quantum Credits	✓
>	 Microsoft Quantum Computing Microsoft	Quantum Computing	Learn & Develop	✓
>	 Quantinuum Quantinuum	Quantum Computing	Azure Quantum Credits	✓
>	 Rigetti Quantum Rigetti Computing	Quantum Computing	Azure Quantum Credits	✓

ⓘ **Nota**

Si el proveedor que desea agregar no está en la lista de proveedores disponibles, la cuenta de facturación puede encontrarse en un país o región que no sea compatible con ese proveedor. Para más información, consulte [Disponibilidad global de proveedores de Azure Quantum](#).

5. Seleccione **+Agregar** junto al proveedor deseado, seleccione una opción de precios y, a continuación, haga clic en **Agregar**.
6. Una vez implementado el proveedor, aparece en la lista de proveedores. Para asegurarse de que la implementación está completa, espere hasta que el **estado** del proveedor muestre una marca de verificación verde.
7. Si se produce un error en la implementación, revise la notificación o compruebe el registro de actividad.

Para quitar un proveedor de un área de trabajo:

1. Inicie sesión en [Azure Portal](#) con las credenciales de la suscripción de Azure.
2. Seleccione **Inicio** y vaya al área de trabajo de Azure Quantum.
3. Seleccione **Proveedores**.
4. En la lista mostrada de proveedores, seleccione **Quitar** para el proveedor deseado.
5. Seleccione **Sí** para quitar el proveedor.

Diseño de la arquitectura de DevOps

15/10/2025

El término *DevOps* deriva de *development* (desarrollo) y *operations* (operaciones). Es la integración de desarrollo, aseguramiento de calidad y operaciones de TI en una cultura unificada y un conjunto de procesos para la entrega de software. Para obtener una descripción general de DevOps, consulte [¿Qué es DevOps? ↗](#).

DevOps incluye estas actividades y operaciones:

- **Integración continua (CI)** se refiere a la práctica de unir todo el código de los desarrolladores en una base de código central con frecuencia y, a continuación, realizar procesos de creación y prueba automatizados. Los objetivos son descubrir y corregir rápidamente los problemas del código, agilizar la implementación y garantizar la calidad del código. Para más información, consulte [¿Qué es la integración continua?](#).
- **Entrega continua (CD)** es la práctica de crear, probar e implementar automáticamente el código en entornos similares a los de producción. El objetivo es asegurar que el código esté siempre listo para implementarse. La incorporación de entrega continua para crear una canalización de CI/CD completa ayuda a detectar defectos de código lo antes posible. También garantiza que las actualizaciones probadas correctamente se puedan publicar en poco tiempo. Para más información, consulte [¿Qué es la entrega continua?](#).
- **La implementación continua** es un proceso adicional que implementa automáticamente en producción cualquier actualización que haya pasado por la canalización de CI/CD. La implementación continua requiere pruebas automáticas sólidas y un planeamiento avanzado de procesos. Es posible que no sea adecuado para todos los equipos.
- **Supervisión continua** hace referencia al proceso y la tecnología necesarios para incorporar la supervisión en cada fase los ciclos de vida de las operaciones de TI y DevOps. La supervisión ayuda a garantizar el buen estado, rendimiento y confiabilidad de la aplicación y la infraestructura a medida que la aplicación pasa del desarrollo a la producción. La supervisión continua se basa en los conceptos de CI y CD.

Introducción a DevOps en Azure

Para saber más sobre DevOps o DevOps en Azure, el mejor lugar para obtener información es la [formación en Microsoft Learn](#). Esta plataforma gratuita en línea proporciona entrenamiento interactivo para los productos de Microsoft y más. Encontrará vídeos, tutoriales y aprendizaje práctico para productos y servicios específicos, además de rutas de aprendizaje basadas en el rol de trabajo, como desarrollador o analista de datos. Si no está familiarizado con Learn, puede hacer [Un recorrido por el entrenamiento de Microsoft Learn](#) o [Un recorrido en vídeo rápido por el aprendizaje de Microsoft Learn](#).

Una vez que esté familiarizado con Azure, puede decidir si sigue las rutas de aprendizaje específicas para DevOps, como:

- Comenzar con Azure DevOps
- [Desplegar aplicaciones con Azure DevOps](#)
- [Creación de aplicaciones con Azure DevOps](#)

Explorar otros materiales de formación para DevOps

Ruta hacia la producción

Para planificar su camino hacia la producción, revise:

- Guías de DevOps
- [Servicios de Azure que se usan a menudo en la implementación de soluciones DevOps](#)
- Ejemplos de arquitecturas DevOps

Guías de DevOps

 Expandir tabla

Artículo o sección	Descripción
Patrones de Excelencia Operativa	Una lista de patrones de diseño para lograr la Excelencia Operativa, uno de los cinco pilares del Marco de Well-Architected de Microsoft Azure , en un entorno en la nube. Consulte Patrones de diseño en la nube para obtener más patrones.
Guía de Azure Monitor	Una serie de artículos para ayudarle a utilizar Azure Monitor para supervisar los entornos en la nube. El primer artículo de la serie es "Mejores prácticas de Azure Monitor: planificar su estrategia de supervisión y configuración".
Integración continua a través de la integración de Git y Despliegue continuo mediante las canalizaciones de despliegue de Fabric	En Microsoft Fabric, CI/CD se logra conectando el área de trabajo de Fabric a un repositorio de Git (Azure DevOps o GitHub) para el control de versiones y los flujos de trabajo basados en ramas. La implementación continua se administra mediante canalizaciones de implementación de Fabric, lo que le permite promover contenido entre entornos, ya sea manualmente o a través de la automatización mediante las API REST de Fabric.
Automatización de la plataforma para el escenario a escala empresarial de Azure VMware Solution	Una descripción general para implementar Azure VMware Solution, incluida la guía para la automatización operativa.

Azure DevOps Servicios

[Expandir tabla](#)

Servicio de Azure	Documentación	Descripción
Azure Artifacts	Información general sobre Azure Artifacts	Administración de paquetes completamente integrada para las canalizaciones CI/CD.
Azure DevOps	Documentación de Azure DevOps	Modernos servicios de desarrollo para administrar su ciclo de vida de desarrollo de un extremo a otro. Incluye Azure Repos, Azure Pipelines y Azure Artifacts.
Azure DevTest Labs ↗	Azure DevTest Labs documentación	Plantillas y artefactos reutilizables para el aprovisionamiento de entornos de desarrollo y prueba.
Azure Monitor	Documentación sobre Azure Monitor	Proporciona una completa capacidad de observación de las aplicaciones, la infraestructura y la red.
Azure Pipelines	Documentación de Azure Pipelines	Ayuda a automatizar la creación e implementación mediante el uso de canalizaciones hospedadas en la nube.
Azure Repos	Documentación de Azure Repos	Proporciona repositorios Git privados ilimitados hospedados en la nube para el proyecto y se puede configurar para usar GitHub Advanced Security.
Administrador de Recursos de Azure ↗	Documentación de Azure Resource Manager	Proporciona una implementación, organización y control coherentes para la administración de recursos.
Plantillas de Azure Resource Manager (plantillas de ARM)	Documentación de las plantillas de Resource Manager	Plantillas que puede usar para definir la infraestructura y la configuración de su proyecto.
Planes de prueba de Azure ↗	Documentación de Azure Test Plans	Proporciona servicios de pruebas planificadas y exploratorias para sus aplicaciones.

Ejemplos de arquitecturas DevOps

Las arquitecturas de DevOps se encuentran aquí:

[Expandir tabla](#)

Sección	Primer artículo de la sección
Ideas de solución	CI/CD para Máquinas Virtuales de Azure

Estas son algunas arquitecturas de ejemplo. Para cada uno hay una lista de los servicios clave de Azure que se usan en la arquitectura.

[Expandir tabla](#)

Arquitectura	Descripción	Servicios de Azure usados
Implementaciones automatizadas de API con APIOps	Emplee las técnicas de GitOps y DevOps para asegurar la calidad de las API.	Azure Repos, API Management, Azure DevOps, Azure Pipelines, Azure Repos
Diseño de una canalización de CI/CD utilizando Azure DevOps	Crea una canalización CI/CD con Azure DevOps y otros servicios.	Azure Repos (Repositorios de Azure), Azure Test Plans (Planes de prueba de Azure), Azure Pipelines (Canales de Azure)

Procedimientos recomendados

El [Microsoft Azure Well-Architected Framework](#) proporciona orientación de referencia y prácticas recomendadas que puede usar para mejorar la calidad de sus arquitecturas. El marco se compone de cinco pilares: confiabilidad, seguridad, optimización de costos, excelencia operativa y eficiencia de rendimiento. Aquí puede encontrar la documentación de los pilares:

- [Confiabilidad](#)
- Seguridad
- Optimización de costos
- Excelencia operativa
- Eficiencia del rendimiento

Los siguientes artículos son sobre los procedimientos recomendados específicos de DevOps y de algunos servicios de DevOps.

DevOps

- [Cómo los equipos de Microsoft adoptaron una cultura DevOps - Serie de seminarios web de Azure](#) [↗](#)
- [Lista de comprobación de mejores prácticas para la migración a la nube de Azure](#)
- [Guías de confiabilidad por servicio](#)

- Supervisión continua con Azure Monitor
- Mejores prácticas de supervisión para la confiabilidad de aplicaciones en Azure
- Descripción general del Azure Security Benchmark (v1)
- Mejores prácticas de seguridad para la administración de identidades y el control de acceso de Azure
- Prácticas recomendadas de seguridad
- Prácticas recomendadas y patrones de seguridad en Azure
- Lista de verificación de seguridad operativa de Azure
- Línea de base de seguridad de Azure para API Management
- Mejores prácticas para un desarrollo seguro en Azure

Azure Artifacts

- [Azure Artifacts: procedimientos recomendados](#)

Azure Resource Manager

- [Procedimientos recomendados de plantilla de ARM](#)
- [Procedimientos recomendados para Bicep](#)

Manténgase al día con DevOps

Manténgase al día con Azure DevOps gracias a estos artículos:

- [Escala de tiempo de características de Azure DevOps](#)
- [Documentación de Azure DevOps: ¿novedades?](#)

Recursos adicionales

Soluciones de ejemplo

- Diseño de una canalización de CI/CD utilizando Azure DevOps
- Gestionar la configuración del tenant de Microsoft 365 utilizando Microsoft365DSC y Azure DevOps

Profesionales de AWS o Google Cloud

- Comparativa de servicios de AWS y Azure - DevOps y monitorización de aplicaciones
- Comparación de servicios entre Google Cloud y Azure - DevOps y supervisión de aplicaciones

Integración de la computación cuántica con aplicaciones clásicas

Azure Quantum

Azure Key Vault

Microsoft Entra ID

La computación clásica se desafía cada vez más con los problemas de proceso más complejos actuales, incluso a la escala de nuestros superequipos más potentes. Los equipos cuánticos tienen la promesa de ampliar drásticamente nuestras funcionalidades de proceso. Al aprovechar las propiedades de la física cuántica para realizar cálculos, proporcionan velocidades exponenciales para determinados tipos de problemas. Por ejemplo, los equipos cuánticos hacen excepcionalmente bien con problemas que requieren calcular un gran número de combinaciones posibles a menudo encontradas en escenarios de optimización, simulación o aprendizaje automático.

Sin embargo, los componentes de computación cuántica tienen un modelo operativo diferente del de software clásico. Normalmente hay uno o varios componentes de proceso clásicos que orquestan la ejecución de componentes cuánticos. Esta orquestación incluye las actividades siguientes:

- Preparación de datos de entrada
- Envío de trabajos de [computación cuántica](#) a un entorno cuántico de destino
- Supervisión de la ejecución del trabajo
- Posprocesamiento de los resultados del trabajo

Puede integrar esta orquestación con aplicaciones clásicas de una de estas dos maneras:

- **Integración mediante acoplamiento estrecho.** La lógica para la orquestación de recursos cuánticos se integra en el componente o componentes clásicos.
- **Integración mediante acoplamiento flexible.** La lógica de la orquestación de recursos cuánticos se expone como una API a la que pueden llamar varios componentes de software clásicos.

En este artículo se describe cómo implementar aplicaciones cuánticas en cada uno de estos diseños. Cada implementación usa Azure Quantum como motor de computación cuántica, pero difieren ligeramente en otros aspectos, como se detalla en las secciones siguientes.

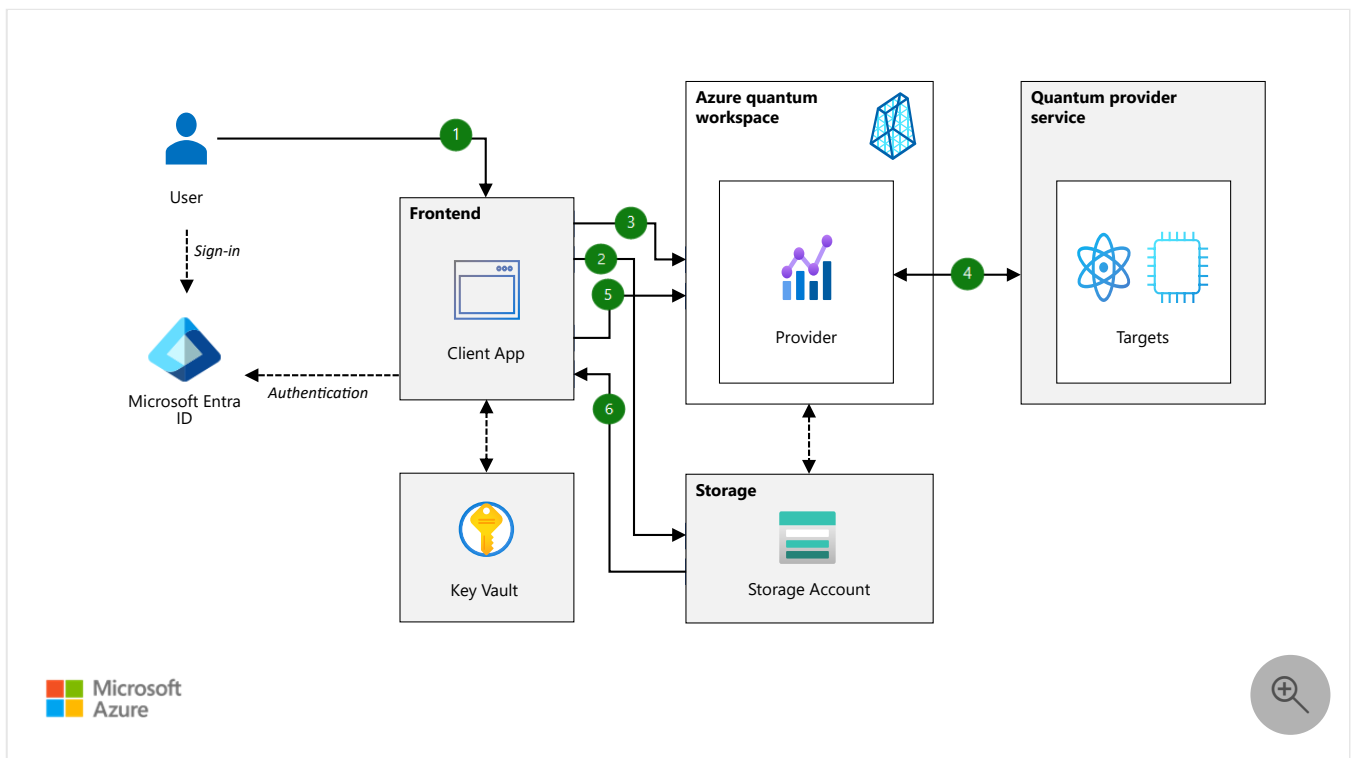
Enfoque estrechamente acoplado

Casos de uso potenciales

En estos casos, se prefiere el enfoque estrechamente acoplado:

- Un equipo posee tanto el código cuántico como el código clásico, y el código está integrado.
- Los componentes cuánticos comparten el mismo ciclo de vida que los componentes clásicos.
- El uso de los componentes cuánticos se limita a una sola aplicación o a un pequeño conjunto de aplicaciones relacionadas.
- El trabajo cuántico representa una solución especializada (por ejemplo, una simulación molecular) que solo se usa en una aplicación clásica especializada.
- El algoritmo implementado es híbrido clásico cuántico por naturaleza, por ejemplo, Quantum Eigensolvers cuánticos variantes (VQE) y algoritmos de optimización aproximada cuántica (QAOA).

Arquitectura



Descargar un archivo de PowerPoint [aquí](#) de esta arquitectura.

Flujo de datos

1. Un usuario que ha iniciado sesión desencadena la ejecución de trabajos cuánticos a través de una aplicación cliente clásica.
2. La aplicación cliente coloca los datos de entrada en Azure Storage.
3. La aplicación cliente envía el trabajo a un área de trabajo de Azure Quantum, especificando el destino de ejecución o los destinos. El cliente identifica el área de trabajo a través de los datos almacenados en Key Vault y se autentica en el área de trabajo a través de [identidad administrada](#).

4. Un proveedor cuántico ejecuta el trabajo en un entorno de destino.
5. La aplicación cliente supervisa la ejecución del trabajo mediante el sondeo del estado del trabajo.
6. Tan pronto como finalice el trabajo cuántico, la aplicación cliente obtiene el resultado de proceso de Storage.

Este flujo de trabajo implementa el patrón de Request-Reply asíncrono de y los pasos definidos para el ciclo de vida del trabajo de Azure Quantum .

Componentes

- [Azure Quantum](#) proporciona un área de trabajo de , accesible desde Azure Portal, para los recursos asociados a la ejecución de trabajos cuánticos en varios destinos. Los trabajos se ejecutan en simuladores cuánticos o hardware cuántico, en función del proveedor que elija.
- [microsoft Entra ID](#) coordina la autenticación del usuario y ayuda a proteger el acceso al área de trabajo de Azure Quantum.
- [Key Vault](#) protege y mantiene el control de las claves y otros secretos, como el nombre del área de trabajo de Azure Quantum.
- [azure Storage](#) proporciona almacenamiento para los datos de entrada y los resultados del proveedor cuántico.

Consideraciones

Estas consideraciones implementan los pilares de Azure Well-Architected Framework, que es un conjunto de principios rectores que se pueden usar para mejorar la calidad de una carga de trabajo. Para obtener más información, consulte [Microsoft Azure Well-Architected Framework](#).

Fiabilidad

La confiabilidad garantiza que la aplicación pueda cumplir los compromisos que realice para sus clientes. Para obtener más información, vea [Lista de comprobación de revisión de diseño para la confiabilidad](#).

La disponibilidad de la funcionalidad de proceso cuántico depende en gran medida de la disponibilidad e instalación de la base de [proveedor de computación cuántica](#). En función del destino de proceso, la aplicación cliente clásica puede experimentar retrasos prolongados o falta de disponibilidad del destino.

Para los servicios de Azure circundantes, se aplican las consideraciones de disponibilidad habituales:

- Use las opciones de redundancia Key Vault.
- Si es necesario, considere la posibilidad de usar las opciones de replicación en [Storage](#).

Seguridad

La seguridad proporciona garantías contra ataques deliberados y el abuso de sus valiosos datos y sistemas. Para obtener más información, vea [Lista de comprobación de revisión de diseño para security](#).

A diferencia de la arquitectura de la [alternativa de acoplamiento flexible](#), la arquitectura que se presenta aquí se basa en la suposición de que solo un cliente accede al área de trabajo de Azure Quantum. Este escenario conduce a las siguientes configuraciones:

- Dado que se conoce el cliente, puede implementar la autenticación a través de [identidad administrada](#), asociada a la aplicación.
- Puede implementar la limitación de solicitudes y el almacenamiento en caché de los resultados en el propio cliente.

En general, considere la posibilidad de aplicar los patrones de diseño típicos [para los](#) de seguridad cuando proceda.

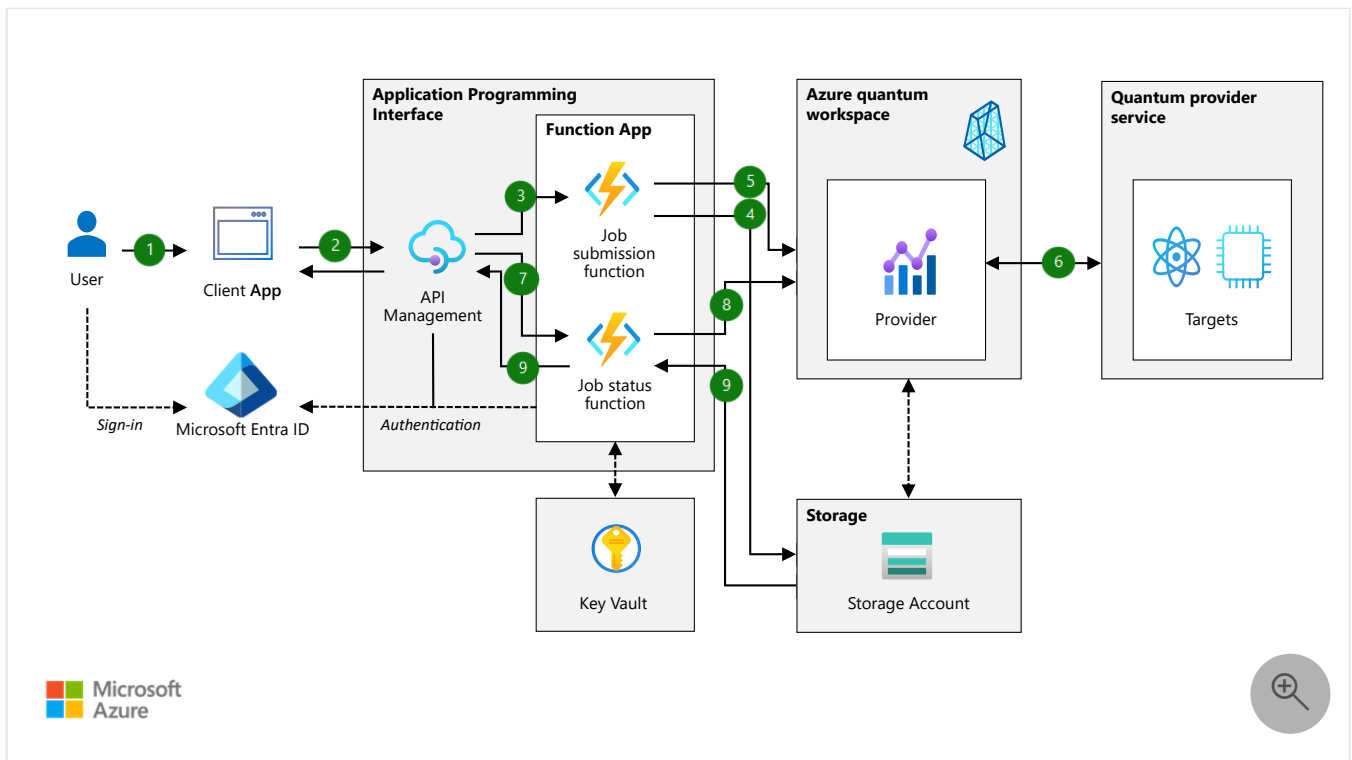
Enfoque acoplado flexible

Casos de uso potenciales

En estos casos, se prefiere el enfoque de acoplamiento flexible:

- Tiene un equipo dedicado de especialistas cuánticos que proporcionan centralmente funcionalidad cuántica a otros equipos y los componentes cuánticos se desarrollan independientemente de los componentes de cliente clásicos.
- El trabajo cuántico representa una solución genérica (por ejemplo, programación de trabajos) que varias aplicaciones clásicas pueden reutilizar.

Arquitectura



Descargar un archivo de PowerPoint [de esta arquitectura](#).

Flujo de datos

1. Un usuario que ha iniciado sesión desencadena la ejecución de trabajos cuánticos a través de una aplicación clásica.
2. La aplicación clásica llama a la API de trabajo personalizada para enviar el trabajo.
3. La puerta de enlace de API desencadena la función de Azure de envío de trabajos, que pasa los datos de entrada del trabajo.
4. La función coloca los datos de entrada en Azure Storage.
5. La función envía el trabajo a un área de trabajo de Azure Quantum, especificando el destino de ejecución o los destinos. La función identifica el área de trabajo a través de los datos almacenados en Azure Key Vault y se autentica en el área de trabajo a través de [identidad administrada](#).
6. Un proveedor cuántico ejecuta el trabajo en un entorno de destino.
7. La aplicación cliente supervisa la ejecución del trabajo mediante el sondeo del estado del trabajo a través de llamadas API.
8. La puerta de enlace de API supervisa la ejecución del trabajo mediante el sondeo del estado del trabajo desde el proveedor cuántico.
9. Cuando finalice el trabajo, los resultados de proceso se almacenan en Azure Storage. La aplicación cliente obtiene los resultados mediante una API que se implementa a través de la función de Azure.

Este flujo de trabajo implementa el patrón de Request-Reply asincrónico de y los pasos definidos para el ciclo de vida del trabajo de Azure Quantum .

Componentes

- [Azure Quantum](#)  proporciona un área de trabajo de , accesible desde Azure Portal, para los recursos asociados a la ejecución de trabajos cuánticos en varios destinos. Los trabajos se ejecutan en simuladores cuánticos o hardware cuántico, en función del proveedor que elija.
- [microsoft Entra ID](#)  coordina la autenticación del usuario y ayuda a proteger el acceso al área de trabajo de Azure Quantum.
- [API Management](#)  es la puerta de enlace de API que expone centralmente los puntos de conexión de API para la administración de trabajos cuánticos.
-  de Azure Functions se usa para reenviar las solicitudes de cliente a los recursos cuánticos adecuados.
- [Azure Key Vault](#)  protege y mantiene el control de las claves y otros secretos, como el nombre del área de trabajo de Azure Quantum.
- [azure Storage](#)  proporciona almacenamiento para los datos de entrada y los resultados del proveedor cuántico.

Fiabilidad

La disponibilidad de la funcionalidad de proceso cuántico depende en gran medida de la disponibilidad e instalación de la [proveedor de computación cuántica](#). En función del destino de proceso, la aplicación cliente clásica puede experimentar retrasos prolongados o falta de disponibilidad del destino.

Para los servicios de Azure circundantes, se aplican las consideraciones de disponibilidad habituales:

- Para lograr una alta disponibilidad, puede implementar [API Management](#) en varias zonas o regiones.
- Si usa la replicación geográfica, puede aprovisionar [azure Functions](#) en varias regiones.
- Use las opciones de redundancia Key Vault.
- Si es necesario, considere la posibilidad de usar las opciones de replicación en [Storage](#).

Seguridad

A diferencia de la arquitectura de la [alternativa estrechamente acoplada](#), la arquitectura que se presenta aquí se basa en la suposición de que varios clientes acceden al área de trabajo de Azure Quantum a través de la API. Este escenario conduce a las siguientes configuraciones:

- Los clientes deben autenticarse en la API. Puede implementar esta autenticación mediante [directivas de autenticación](#).

- Puede implementar la autenticación de las funciones de Azure a través de [identidades administradas](#) asociadas a las funciones. Puede usar estas identidades para autenticarse en el área de trabajo de Azure Quantum.
- Varios clientes acceden a la API. Puede implementar la limitación de solicitudes mediante [limitación de solicitudes de API Management](#) para proteger el back-end cuántico y limitar el uso de recursos cuánticos.
- En función del patrón de solicitud, es posible que pueda implementar el almacenamiento en caché de los resultados de la computación cuántica mediante [directivas de almacenamiento en caché de API Management](#).

En general, considere la posibilidad de aplicar los patrones de diseño típicos [para los](#) de seguridad cuando proceda.

Eficiencia del rendimiento

La eficiencia del rendimiento es la capacidad de la carga de trabajo para escalar a fin de satisfacer las demandas que los usuarios ponen en ella de forma eficaz. Para obtener más información, vea [Lista de comprobación de revisión de diseño para la eficiencia del rendimiento](#).

El rendimiento de la aplicación depende de la disponibilidad y el rendimiento de los destinos de computación cuántica subyacentes. Para obtener información sobre el rendimiento y la escalabilidad de los componentes clásicos, revise el [patrones de diseño típicos para la escalabilidad](#) y la lista de comprobación de eficiencia del rendimiento .

Características comunes

Las siguientes características son comunes a los modelos de implementación estrechamente acoplados y de acoplamiento flexible

Alternativas

Las arquitecturas que se presentan aquí son para problemas empresariales que requieren recursos de computación cuántica para sus tareas de proceso. Para algunos desafíos de proceso, los servicios existentes creados para realizar [informática de alto rendimiento](#) [↗](#) o proporcionar [funcionalidad de inteligencia artificial](#) [↗](#) podrían ser una alternativa.

Consideraciones

Algunos de los destinos cuánticos de Azure (especialmente el hardware cuántico) son un recurso limitado para el futuro previsible. El acceso a estos recursos se implementa a través de

un mecanismo de puesta en cola. Al enviar un trabajo cuántico a Azure Quantum, este trabajo se agrega a una cola de trabajos. El trabajo se ejecuta después de que el destino complete el procesamiento de entradas de cola anteriores. Puede obtener el tiempo de espera esperado [enumerar los destinos disponibles](#). Para calcular el tiempo de respuesta completo, debe agregar el tiempo dedicado a esperar un recurso disponible al tiempo de ejecución del trabajo.

Fiabilidad

Como entornos de destino cuánticos, como Azure Quantum, normalmente proporcionan una corrección de errores limitada (limitada al procesador cuántico en el caso de Azure Quantum), pueden producirse otros errores, como el tiempo de espera de la máquina cuántica, por lo que se recomienda supervisar la ejecución del trabajo para que pueda informar al usuario sobre el estado del trabajo. Cuando se produce un error en la ejecución del trabajo debido a un error transitorio, implemente un patrón de reintento . Envíe los trabajos a través de llamadas asincrónicas, con sondeo para el resultado, para evitar bloquear innecesariamente el cliente que realiza la llamada.

Dado que los recursos de computación cuántica suelen ser limitados, las expectativas de resistencia deben tener en cuenta este factor. Por lo tanto, las sugerencias que se ofrecen en este artículo pueden proporcionar medidas adicionales de resistencia.

Optimización de costos

La optimización de costos consiste en examinar formas de reducir los gastos innecesarios y mejorar las eficiencias operativas. Para obtener más información, consulte [Lista de comprobación de revisión de diseño para la optimización de costos](#).

El costo total de esta solución depende del destino de computación cuántica que seleccione para ejecutar el trabajo cuántico. Calcular los costos estimados de los componentes clásicos es sencillo. Puede usar la calculadora de precios de Azure [↗](#).

Para el servicio Azure Quantum, tenga en cuenta que los proveedores de computación cuántica se pueden consumir a través de una oferta de Azure Marketplace. Los precios dependen del tipo de recurso (simulador o hardware), la SKU y el uso. Para obtener más información, consulte la página de referencia del proveedor necesario para su escenario. Estas páginas de referencia se enumeran en [proveedores de computación cuántica en Azure Quantum](#).

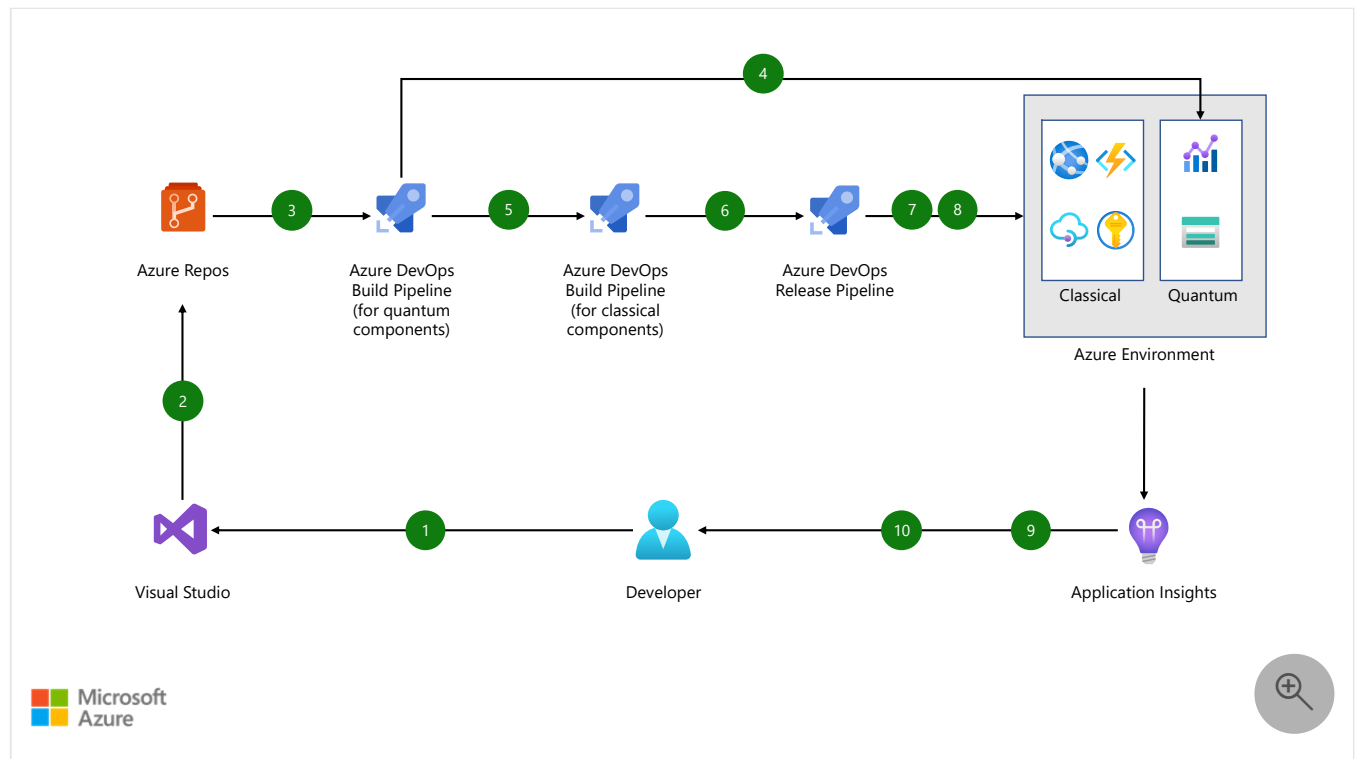
Excelencia operativa

La excelencia operativa abarca los procesos de operaciones que implementan una aplicación y lo mantienen en ejecución en producción. Para obtener más información, vea [Lista de](#)

comprobación de revisión de diseño para la excelencia operativa.

La incorporación de trabajos cuánticos a canalizaciones clásicas de CI/CD se puede realizar mediante Azure DevOps con cambios menores en un diseño típico. El siguiente diseño ilustra un flujo de trabajo de pipeline de DevOps que se puede aplicar a las arquitecturas estrechamente acopladas y débilmente acopladas.

Arquitectura



Descargar un archivo de PowerPoint [de esta arquitectura](#).

Flujo de datos

1. El desarrollador cambia el código fuente de los componentes de la aplicación.
2. Los cambios se confirman en el repositorio de código fuente.
3. Los cambios en el código cuántico desencadenan la canalización de compilación cuántica. La canalización de compilación comprueba el código, lo compila, calcula los recursos necesarios y ejecuta el algoritmo en un simulador.
4. El algoritmo cuántico compilado se envía a un entorno cuántico para realizar pruebas.
5. Los cambios desencadenan una canalización de compilación para los componentes clásicos. La canalización comprueba el código, lo compila y ejecuta pruebas unitarias e de integración.
6. Las pruebas y la compilación correctas desencadenan una canalización de versión. La canalización aprovisiona primero el entorno de Azure mediante la implementación de las

plantillas de Azure Resource Manager que se almacenan en el repositorio (infraestructura como código).

7. Los artefactos de aplicación clásica compilados se implementan en Azure. Los trabajos cuánticos se envían a un área de trabajo cuántica durante el tiempo de ejecución.
8. Application Insights supervisa el comportamiento del entorno de ejecución, el estado, el rendimiento y la información de uso.
9. Los elementos de trabajo pendiente se actualizan según sea necesario, en función de los resultados de la supervisión.
10. El desarrollador usa Application Insights para la optimización y los comentarios de la aplicación.

Componentes

Esta solución usa las siguientes herramientas de DevOps:

- [azure Repos](#) proporciona repositorios git privados ilimitados hospedados en la nube. Aquí se usa para almacenar el código cuántico y clásico y las plantillas de Azure Resource Manager que se usan para aprovisionar el entorno.
- [azure Pipelines](#) le permite compilar, probar e implementar continuamente en la nube. Aquí se usa para implementar CI/CD, incluido el aprovisionamiento del entorno antes de la implementación de código.

Como alternativa, puede usar repositorios de GitHub y acciones de GitHub para implementar los procesos de CI/CD.

La solución usa los siguientes componentes:

- Una aplicación cliente organiza el trabajo cuántico. Puede implementar la integración mediante un [estrechamente acoplado](#) o un enfoque de [acoplado flexiblemente](#).
- [Azure Quantum](#) proporciona una [de área de trabajo](#) para los recursos asociados a la ejecución de aplicaciones de computación cuántica. Los trabajos se ejecutan en simuladores cuánticos o hardware cuántico, en función del proveedor que elija.
- [microsoft Entra ID](#) coordina la autenticación de usuario y protege el acceso al área de trabajo de Azure Quantum.
- [Azure Key Vault](#) protege y mantiene el control de las claves y otros secretos, como el nombre del área de trabajo cuántica.
- [azure Storage](#) contiene los datos de entrada y salida del trabajo cuántico.
- [Application Insights](#) supervisa la aplicación, detecta anomalías de la aplicación, como errores y rendimiento deficientes, y envía telemetría a Azure Portal.

Eficiencia del rendimiento

El rendimiento de la aplicación depende de la disponibilidad y el rendimiento de los destinos de computación cuántica subyacentes. Para obtener información sobre el rendimiento y la escalabilidad de los componentes clásicos, revise el [patrones de diseño típicos para la escalabilidad](#) y la lista de comprobación de eficiencia del rendimiento .

Colaboradores

Microsoft mantiene este artículo. Originalmente fue escrito por el siguiente colaborador.

Autor principal:

- [Holger Sirtl](#)  | Arquitecto técnico sénior en el Centro de tecnología de Microsoft

Para ver perfiles de LinkedIn no públicos, inicie sesión en LinkedIn.

Pasos siguientes

- Para obtener información general sobre Microsoft Quantum, un ecosistema de computación cuántica de pila completa y abierta en la nube, consulte [Microsoft Quantum](#)  y complete la [bases de computación cuántica](#) ruta de aprendizaje.
- Para más información sobre el servicio Azure Quantum, consulte [Azure Quantum](#) .
- Para obtener información general sobre la administración de trabajos de Azure Quantum, consulte [Trabajar con trabajos de Azure Quantum](#).
- Para obtener información sobre cómo ejecutar algoritmos en hardware cuántico, consulte el módulo [Ejecución de algoritmos en hardware cuántico mediante Azure Quantum](#).

Recursos relacionados

- [principios de excelencia operativa](#)
- [patrón asincrónico de Request-Reply](#)
- [de computación cuántica acoplada flexiblemente](#)

Creación de soluciones cuánticas con Azure Quantum Development Kit

Azure Quantum Development Kit(QDK) es un kit de desarrollo de software de código abierto gratuito diseñado específicamente para el desarrollo de programas cuánticos. QDK incluye una extensión para Microsoft's Visual Studio Code (VS Code) y un conjunto de Python bibliotecas. Instale el QDK para comenzar con herramientas de desarrollo cuántico de última generación.

Información general sobre QDK

QDK Ofrece un amplio conjunto de herramientas y características que le ayudarán a destacar en el desarrollo cuántico, como:

- Compatibilidad con varios lenguajes de programación cuántica, incluidos Q#, Qiskity OpenQASM.
- Ejemplos de código cuántico para varios lenguajes y escenarios de desarrollo.
- Soporte para el desarrollo en Jupyter Notebook.
- Herramientas de visualización para circuitos cuánticos y salida de los programas cuánticos.
- Integración completa con Azure Quantum para enviar trabajos que se ejecutan en hardware cuántico real.
- Simuladores locales para probar y ejecutar programas cuánticos en el dispositivo local.

Además de un conjunto sólido de características específicas de computación cuántica, el QDK ofrece herramientas estándar de desarrollo VS Code, como el resaltado de sintaxis, sugerencias de código utilizando IntelliSense, accesos directos CodeLens, comprobación de errores y depuración, y referencias de código.

QDK También incluye una experiencia de agente totalmente integrada AI con Copilot en VS Code. El VS Code agente puede escribir código automáticamente, explicar lo que hace el código, ayudarle a enviar trabajos a Azure Quantum e incluso enseñarle conceptos teóricos en la computación cuántica.

Comienza con QDK

Para instalar QDK, consulte [Configuración de la QDK extensión](#).

Para actualizar el QDK, consulte [actualizar el QDK a la versión más reciente](#).

Características de QDK

Explore la siguiente documentación para obtener más información sobre las características principales de QDK:

- [Microsoft's Q# lenguaje de programación](#)
- [Lenguaje de OpenQASM programación](#)
- [Qiskitbibliotecas y CirqPython](#)
- [Compilación de circuitos con Circuit Editor](#)
- [Copilot integración en VS Code](#)

Last updated on 15/10/2025

Introducción al lenguaje de programación cuántica Q#

Artículo • 08/04/2025

Q# es un lenguaje de programación de código [abierto](#) de alto nivel desarrollado por Microsoft para escribir programas cuánticos. Q# se incluye en el Kit de desarrollo de Quantum (QDK). Para obtener más información, consulte [Configuración del kit](#) de desarrollo de Quantum.

Como lenguaje de programación cuántica, Q# cumple los siguientes requisitos para el lenguaje, el compilador y el entorno de ejecución:

- **Independiente del hardware:** los cúbits de los algoritmos cuánticos no están vinculados a un diseño o hardware cuántico específico. El Q# compilador y el tiempo de ejecución controlan la asignación de cúbits de programa a cúbits físicos, lo que permite que el mismo código se ejecute en diferentes procesadores cuánticos.
- **Integración de la computación cuántica y clásica:** Q# permite la integración de cálculos cuánticos y clásicos, que es esencial para la computación cuántica universal.
- **Administración de cúbits:** Q# proporciona operaciones y funciones integradas para administrar cúbits, incluida la creación de estados de superposición, el entrelazamiento de cúbits y la realización de medidas cuánticas.
- **Respetar las leyes de la física:** Q# y los algoritmos cuánticos deben seguir las reglas de la física cuántica. Por ejemplo, no se puede copiar ni acceder directamente al estado de cúbit en Q#.

Para obtener más información sobre los orígenes de Q#, consulte la entrada del blog [¿Por qué necesitamos Q#?](#).

Estructura de un Q# programa

Antes de empezar a escribir Q# programas, es importante comprender su estructura y componentes. Tenga en cuenta el siguiente Q# programa, denominado Superposición, que crea un estado de **superposición**:

Q#

```
namespace Superposition {
    @EntryPoint()
    operation MeasureOneQubit() : Result {
        // Allocate a qubit. By default, it's in the 0 state.
        use q = Qubit();
        // Apply the Hadamard operation, H, to the state.
        // It now has a 50% chance of being measured as 0 or 1.
    }
}
```

```

        H(q);
        // Measure the qubit in the Z-basis.
        let result = M(q);
        // Reset the qubit before releasing it.
        Reset(q);
        // Return the result of the measurement.
        return result;
    }
}

```

En función de los comentarios (`//`), el Q# programa asigna primero un cúbit, aplica una operación para colocar el cúbit en superposición, mide el estado del cúbit, restablece el cúbit y, por último, devuelve el resultado.

Vamos a dividir este Q# programa en sus componentes.

Espacios de nombres de usuario

Q# Los programas pueden empezar opcionalmente con un espacio de nombres definido por el usuario, como:

```

Q#

namespace Superposition {
    // Your code goes here.
}

```

Los espacios de nombres pueden ayudarle a organizar la funcionalidad relacionada. Los espacios de nombres son opcionales en Q# programas, lo que significa que puede escribir un programa sin definir un espacio de nombres.

Por ejemplo, el programa **Superposición** del ejemplo también se podría escribir sin un espacio de nombres como:

```

Q#

@EntryPoint()
operation MeasureOneQubit() : Result {
    // Allocate a qubit. By default, it's in the 0 state.
    use q = Qubit();
    // Apply the Hadamard operation, H, to the state.
    // It now has a 50% chance of being measured as 0 or 1.
    H(q);
    // Measure the qubit in the Z-basis.
    let result = M(q);
    // Reset the qubit before releasing it.
    Reset(q);
    // Return the result of the measurement.
}

```



```
    return result;
}
```

⚠ Nota

Cada Q# programa solo puede tener un `namespace`. Si no especifica un espacio de nombres, el Q# compilador usa el nombre de archivo como espacio de nombres.

Puntos de entrada

Cada Q# programa debe tener un punto de entrada, que es el punto inicial del programa. De forma predeterminada, el Q# compilador comienza a ejecutar un programa desde la `Main()` operación, si está disponible, que se puede ubicar en cualquier parte del programa. Opcionalmente, puede usar el `@EntryPoint()` atributo para especificar cualquier operación en el programa como punto de ejecución.

Por ejemplo, en el programa **Superposición**, la `MeasureOneQubit()` operación es el punto de entrada del programa porque tiene el `@EntryPoint()` atributo antes de la definición de la operación:

```
Q#

@EntryPoint()
operation MeasureOneQubit() : Result {
    ...
}
```

Sin embargo, el programa también podría escribirse sin el `@EntryPoint()` atributo cambiando el nombre de la `MeasureOneQubit()` operación a `Main()`, como:

```
Q#

// The Q# compiler automatically detects the Main() operation as the entry point.

operation Main() : Result {
    // Allocate a qubit. By default, it's in the 0 state.
    use q = Qubit();
    // Apply the Hadamard operation, H, to the state.
    // It now has a 50% chance of being measured as 0 or 1.
    H(q);
    // Measure the qubit in the Z-basis.
    let result = M(q);
    // Reset the qubit before releasing it.
    Reset(q);
}
```

```
// Return the result of the measurement.  
return result;  
}
```

Tipos

Los tipos son esenciales en cualquier lenguaje de programación porque definen los datos con los que un programa puede trabajar. Q# proporciona tipos incorporados comunes a la mayoría de los lenguajes, incluidos `Int`, `Double`, `Bool` y `String`, y tipos que definen intervalos, matrices y tuplas.

Q# también proporciona tipos específicos de la computación cuántica. Por ejemplo, el `Result` tipo representa el resultado de una medida de cúbit y puede tener dos valores: `Zero` o `One`.

En el programa **Superposición**, la `MeasureOneQubit()` operación devuelve un tipo `Result`, que corresponde al tipo de retorno de la `M` operación. El resultado de la medida se almacena en una nueva variable definida mediante la `let` instrucción :

Q#

```
// The operation definition returns a Result type.  
operation MeasureOneQubit() : Result {  
    ...  
    // Measure the qubit in the Z-basis, returning a Result type.  
    let result = M(q);  
    ...  
}
```

Otro ejemplo de un tipo específico de quantum es el `Qubit` tipo , que representa un bit cuántico.

Q# también permite definir sus propios tipos personalizados. Para obtener más información, vea [Declaraciones de tipo](#).

Asignación de bits cuánticos

En Q#, se asignan qubits mediante la palabra clave `use` y el tipo `Qubit`. Los cúbits siempre se asignan en el estado $|0\rangle$.

Por ejemplo, el programa **Superposición** define un único cúbit y lo almacena en la variable `q`:

Q#

```
// Allocate a qubit.  
use q = Qubit();
```

También puede asignar varios cúbits y acceder a cada uno a través de su índice:

```
Q#  
  
use qubits = Qubit[2]; // Allocate two qubits.  
H(qubits[0]); // Apply H to the first qubit.  
X(qubits[1]); // Apply X to the second qubit.
```

Operaciones cuánticas

Después de asignar un cúbit, puede pasarlo a operaciones y funciones. Un programa Q# tiene como bloques de creación básicos las operaciones. Una Q# operación es una subrutina cuántica o una rutina invocable que contiene operaciones cuánticas que cambian el estado del registro de cúbits.

Para definir una Q# operación, especifique un nombre para la operación, sus entradas y su salida. En el programa **Superposición**, la `MeasureOneQubit()` operación no toma parámetros y devuelve un `Result` tipo:

```
Q#  
  
operation MeasureOneQubit() : Result {  
    ...  
}
```

Este es un ejemplo básico que no toma parámetros y espera ningún valor devuelto. El `Unit` valor es equivalente a `NULL` en otros idiomas:

```
Q#  
  
operation SayHelloQ() : Unit {  
    Message("Hello quantum world!");  
}
```

La Q# biblioteca estándar también proporciona operaciones que puede usar en programas cuánticos, como la operación Hadamard, `H`, en el `Superposition` programa. Dado un cúbit en la base Z, `H` coloca el cúbit en una superposición par, donde tiene una probabilidad del 50 % de medirse como `Zero` o `One`.

Medición de cúbits

Aunque hay muchos tipos de mediciones cuánticas, Q# se centra en las mediciones proyectivas en cúbits individuales, también conocidos como **mediciones** de Pauli.

En Q#, la `Measure` operación mide uno o varios cúbits en la base de Pauli especificada, que puede ser `PauliX`, `PauliY` o `PauliZ`. `Measure` devuelve un tipo `Result`, ya sea `Zero` o `One`.

Para implementar una medición en la base computacional $\{|0\rangle, |1\rangle\}$, puede usar también la operación `M`, que mide un qubit en la base Z de Pauli. Esto equivale `M` a `Measure([PauliZ], [qubit])`.

Por ejemplo, el programa **Superposición** usa la `M` operación :

```
Q#  
  
// Measure the qubit in the Z-basis.  
let result = M(q);
```

Restablecer cúbits

En Q#, los cúbits **deben** estar en el $|0\rangle$ estado cuando se liberan para evitar errores en el hardware cuántico. Puede restablecer un cúbit al $|0\rangle$ estado mediante la `Reset` operación al final del programa. Si no se restablece un cúbit, se produce un error en tiempo de ejecución.

```
Q#  
  
// Reset a qubit.  
Reset(q);
```

Espacios de nombres de biblioteca estándar

La Q# biblioteca estándar tiene espacios de nombres integrados que contienen funciones y operaciones que puede usar en programas cuánticos. Por ejemplo, el `Microsoft.Quantum.Intrinsic` espacio de nombres contiene operaciones y funciones usadas habitualmente, como `M` para medir los resultados y `Message` mostrar mensajes de usuario en cualquier parte del programa.

Para llamar a una función o operación, puede especificar el espacio de nombres completo o usar una `import` instrucción , que hace que todas las funciones y operaciones de ese espacio de nombres estén disponibles y haga que el código sea más legible. En los ejemplos siguientes se llama a la misma operación:

Q#

```
Microsoft.Quantum.Intrinsic.Message("Hello quantum world!");
```

Q#

```
// imports all functions and operations from the Microsoft.Quantum.Intrinsic
namespace.
import Microsoft.Quantum.Intrinsic.*;
Message("Hello quantum world!");

// imports just the `Message` function from the Microsoft.Quantum.Intrinsic
namespace.
import Microsoft.Quantum.Intrinsic.Message;
Message("Hello quantum world!");
```

Q#

```
// namespaces in the standard library may be imported using `Std` instead of
`Microsoft.Quantum`.
import Std.Intrinsic.*;
Message("Hello quantum world!");
```

ⓘ Nota

El programa **Superposición** no tiene instrucciones `import` ni llamadas con espacios de nombres completos. Esto se debe a que el Q# entorno de desarrollo carga automáticamente dos espacios de nombres: `Microsoft.Quantum.Core` y `Microsoft.Quantum.Intrinsic`, que contienen funciones y operaciones usadas habitualmente.

Puede aprovechar el espacio de nombres `Microsoft.Quantum.Measurement` usando la operación `MResetZ` para optimizar el programa de Superposición . `MResetZ` combina las operaciones de medición y restablecimiento en un paso, como en el ejemplo siguiente:

Q#

```
// Import the namespace for the MResetZ operation.
import Microsoft.Quantum.Measurement.*;

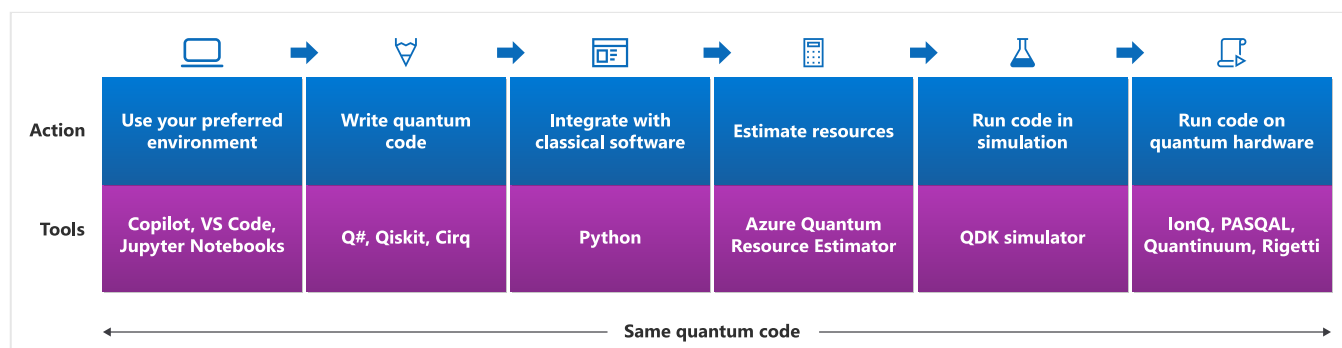
@EntryPoint()
operation MeasureOneQubit() : Result {
    // Allocate a qubit. By default, it's in the 0 state.
    use q = Qubit();
    // Apply the Hadamard operation, H, to the state.
```

```
// It now has a 50% chance of being measured as 0 or 1.
H(q);
// Measure and reset the qubit, and then return the result value.
return MResetZ(q);
}
```

Aprenda a desarrollar programas cuánticos con Q# y Azure Quantum

Q# y Azure Quantum son una combinación eficaz para desarrollar y ejecutar programas cuánticos. Con Q# y Azure Quantum, puede escribir programas cuánticos, simular su comportamiento, calcular los requisitos de recursos y ejecutarlos en hardware cuántico real. Esta integración le permite explorar el potencial de la computación cuántica y desarrollar soluciones innovadoras para problemas complejos. Tanto si es principiante como un desarrollador cuántico experimentado, Q# y Azure Quantum proporciona las herramientas y los recursos que necesita para desbloquear la eficacia de la computación cuántica.

En el diagrama siguiente se muestran las fases a las que pasa un programa cuántico al desarrollarlo con Q# y Azure Quantum. El programa comienza con el entorno de desarrollo y termina con el envío del trabajo al hardware cuántico real.



Vamos a desglosar los pasos del diagrama.

Elección del entorno de desarrollo

Ejecute los programas cuánticos en su entorno de desarrollo preferido. Puede usar el editor de código en línea en el sitio web de Azure Quantum, los cuadernos de Jupyter Notebook hospedados en el área de trabajo de Azure Quantum en Azure Portal o un entorno de desarrollo local con Visual Studio Code. Para obtener más información, consulte [Diferentes formas de ejecutar Q# programas](#).

Escritura del programa cuántico

Puede escribir programas cuánticos en Q# mediante el Kit de desarrollo de Quantum (QDK). Para empezar, consulte [Inicio rápido: Creación del primer Q# programa](#).

Q# Además de , el QDK ofrece compatibilidad con otros lenguajes para la computación cuántica, como [Qiskit](#) y [Cirq](#).

Integración con Python

Puede usarse Q# por sí mismo o junto con Python en varios IDE. Por ejemplo, puede usar un proyecto Q# con un programa host de Python para llamar a operaciones Q# o integrar Q# con Python en Jupyter Notebooks. Para obtener más información, consulte [Integración de Q# y Python](#).

Comando %%qsharp

De forma predeterminada, los Q# programas de Jupyter Notebook usan el `ipykernel` paquete de Python. Para agregar Q# código a una celda del cuaderno, use el `%%qsharp` comando , que está habilitado con el `qsharp` paquete de Python, seguido del Q# código.

Al usar `%%qsharp`, tenga en cuenta lo siguiente:

- Primero debe ejecutar `import qsharp` para habilitar `%%qsharp`.
- `%%qsharp` se refiere a la celda del notebook en la que aparece y cambia el tipo de celda de Python a Q#.
- No se puede colocar una instrucción de Python antes o después de `%%qsharp`.
- Q# el código siguiente `%%qsharp` debe cumplir la Q# sintaxis. Por ejemplo, use `//` en lugar de `#` para indicar comentarios y `;` para finalizar líneas de código.

Estimación de recursos

Antes de ejecutarse en hardware cuántico real, debe averiguar si el programa se puede ejecutar en hardware existente y cuántos recursos consumirá.

El [estimador](#) de recursos de Azure Quantum permite evaluar decisiones arquitectónicas, comparar tecnologías de cúbits y determinar los recursos necesarios para ejecutar un algoritmo cuántico determinado. Puede elegir entre protocolos predefinidos de tolerancia a errores y especificar suposiciones del modelo de cúbit físico subyacente.

Para más información, consulte [Ejecución de la primera estimación](#) de recursos.

El estimador de recursos de Azure Quantum es gratuito y no requiere una cuenta de Azure.

Ejecución del programa en simulación

Al compilar y ejecutar un programa cuántico, el QDK crea una instancia del simulador cuántico y pasa el Q# código a él. El simulador utiliza el código de Q# para crear cúbits (simulaciones de partículas cuánticas) y aplicar transformaciones para modificar su estado. Después, los resultados de las operaciones cuánticas del simulador se devuelven al programa. Al aislar el código de Q# en el simulador, se garantiza que los algoritmos sigan las leyes de la física cuántica y que se puedan ejecutar correctamente en los equipos cuánticos.

Envío del programa al hardware cuántico real

Una vez que haya probado el programa en la simulación, puede ejecutarlo en hardware cuántico real. Al ejecutar un programa cuántico en Azure Quantum, creas y ejecutas una **tarea**. Para enviar un trabajo a los proveedores de Azure Quantum, necesita una cuenta de Azure y un área de trabajo cuántica. Si no tiene un área de trabajo cuántica, consulte [Creación de un área de trabajo](#) de Azure Quantum.

Azure Quantum ofrece algunos de los hardware cuánticos más atractivos y diversos. Vea [Proveedores de computación cuántica](#) para obtener la lista actual de proveedores de hardware compatibles.

Una vez enviado el trabajo, Azure Quantum administra el ciclo de vida del trabajo, incluida la programación, la ejecución y la supervisión del trabajo. Puede realizar un seguimiento del estado del trabajo y ver los resultados en el portal de Azure Quantum.

Contenido relacionado

- [Diferentes formas de ejecutar Q# programas](#)
- [Configuración del kit de desarrollo de Quantum](#)
- [Inicio rápido: Creación del primer Q# programa](#)

Inicio rápido: Creación del primer Q# programa

Artículo • 29/10/2024

Aprenda a escribir un programa básico Q# que muestre el entrelazamiento, un concepto clave de la computación cuántica.

Cuando dos o más [cúbits están entrelazados](#), comparten información cuántica, lo que significa que sucede con un cúbit también ocurre con el otro. En este inicio rápido, creará un estado entrelazado de dos cúbits determinado denominado par Bell. En un par bell, si mide un cúbit en el $|0\rangle$ estado, sabe que el otro cúbit también está en el $|0\rangle$ estado sin medirlo. Para más información, consulte [Entrelazamiento](#) cuántico.

En esta guía de inicio rápido:

- ✓ Cree un archivo Q#.
- ✓ Asigne un par de cúbits.
- ✓ Entrelazar los cúbits.

Requisitos previos

- Versión más reciente de [Visual Studio Code](#).
- Extensión [del](#) Kit de desarrollo de Azure Quantum (QDK). Para obtener más información sobre la instalación, consulte [Configuración del kit](#) de desarrollo de Quantum.

Cree un archivo Q#.

1. Abra Visual Studio Code.
2. Seleccione **Archivo**>nuevo archivo de texto.
3. Guarde el archivo como `Main.qs`. La extensión .qs denota un Q# programa.

Q# Escribir el código

En el `Main.qs` archivo, siga estos pasos para entrelazar y medir un par de cúbits.

Importación de una biblioteca cuántica

El QDK incluye la Q# biblioteca estándar con funciones y operaciones predefinidas para los programas cuánticos. Para usarlos, primero debe importar la biblioteca pertinente.

En el programa, use una `import` instrucción para abrir la `Microsoft.Quantum.Diagnostics` biblioteca. Esto le proporciona acceso a todas sus funciones y operaciones, incluido `DumpMachine()`, que posteriormente se usa para mostrar el estado entrelazado.

Q#

```
import Microsoft.Quantum.Diagnostics.*;
```

Definición de una operación

Después de importar las bibliotecas pertinentes, defina la operación cuántica y sus valores de entrada y salida. Para este inicio rápido, la operación es `Main`. Aquí es donde escribirá el código restante Q# para asignar, manipular y medir dos cúbits.

`Main` no toma parámetros y devuelve dos `Result` valores, o `Zero` `One`, que representan los resultados de las medidas de cúbit:

Q#

```
operation Main() : (Result, Result) {  
    // Your entanglement code goes here.  
}
```

Asignar dos cúbits

La `Main` operación está vacía actualmente, por lo que el siguiente paso es asignar dos cúbits y `q2 q1`. En Q#, se asignan cúbits mediante la `use` palabra clave :

Q#

```
// Allocate two qubits, q1 and q2, in the 0 state.  
use (q1, q2) = (Qubit(), Qubit());
```

⚠ Nota

En Q#, los cúbits siempre se asignan en el $|0\rangle$ estado .

Colocar un cúbit en superposición

Los cúbits `q1` y `q2` están en estado $|0\rangle$. Para preparar los cúbits para el entrelazamiento, debe colocar uno de ellos en una superposición par, donde tiene una probabilidad del 50 % de medirse como $|0\rangle$ o $|1\rangle$.

Para colocar un cúbit en superposición, aplique hadamard, `H`, operación:

Q#

```
// Put q1 into an even superposition.  
H(q1);
```

El estado resultante de `q1` es $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, *que es una superposición par de $|0\rangle$ y $|1\rangle$.*

Entrelazar los cúbits

Ya está listo para entrelazar los cúbits mediante la operación controlled-NOT, `CNOT`, . `CNOT` es una operación de control que toma dos cúbits, una que actúa como control y la otra como destino.

En este inicio rápido, se establece `q1` como cúbit de control y `q2` como cúbit de destino. Esto significa `CNOT` voltear el estado de `q2` cuando el estado de `q1` es $|1\rangle$.

Q#

```
// Entangle q1 and q2, making q2 depend on q1.  
CNOT(q1, q2);
```

El estado resultante de ambos cúbits es el par $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$.

Sugerencia

Si desea obtener información sobre cómo las operaciones Hadamard y CNOT transforman el estado de los cúbits, consulte [Creación de un entrelazamiento con operaciones](#) cuánticas.

Mostrar el estado entrelazado

Antes de medir los cúbits, es importante comprobar que el código anterior los entrelazará correctamente. Puede usar la `DumpMachine` operación, que forma parte de la

Microsoft.Quantum.Diagnostics biblioteca, para generar el estado actual del Q# programa:

Q#

```
// Show the entangled state of the qubits.  
DumpMachine();
```

Medición de los cúbits

Ahora que ha comprobado que los cúbits están entrelazados, puede usar la `M` operación para medirlos. Medir `q1` y `q2` contraer sus estados cuánticos en `Zero` o `One` con probabilidad uniforme.

En Q#, se usa la `let` palabra clave para declarar una nueva variable. Para almacenar los resultados de medición de `q1` y `q2`, declare las variables `m1` y `m2`, respectivamente:

Q#

```
// Measure q1 and q2 and store the results in m1 and m2.  
let (m1, m2) = (M(q1), M(q2));
```

Restablecer los cúbits

Antes de liberarse al final de cada Q# programa, los cúbits deben estar en estado $|0\rangle$. Para ello, use la `Reset` operación :

Q#

```
// Reset q1 and q2 to the 0 state.  
Reset(q1);  
Reset(q2);
```

Devolver los resultados de la medida

Por último, para completar la `Main` operación y observar el estado entrelazado, devuelva los resultados de medición de `m1` y `m2`:

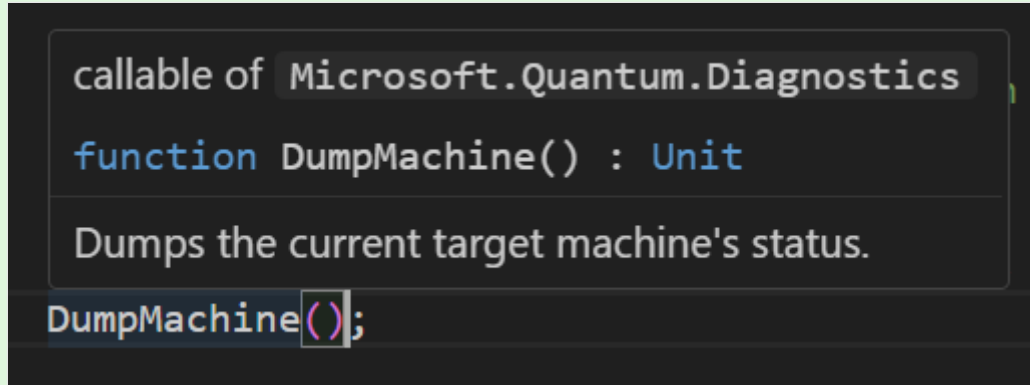
Q#

```
// Return the measurement results.
```

```
return (m1, m2);
```

💡 Sugerencia

Si desea obtener más información sobre una Q# función o una operación, mantenga el puntero sobre ella.



Ejecución del Q# código

Felicidades. Escribió un Q# programa que entrelaza dos cúbits y crea un par de Campanas.

El programa final Q# debe tener este aspecto:

Q#

```
import Microsoft.Quantum.Diagnostics.*;

operation Main() : (Result, Result) {
    // Allocate two qubits, q1 and q2, in the 0 state.
    use (q1, q2) = (Qubit(), Qubit());

    // Put q1 into an even superposition.
    // It now has a 50% chance of being measured as 0 or 1.
    H(q1);

    // Entangle q1 and q2, making q2 depend on q1.
    CNOT(q1, q2);

    // Show the entangled state of the qubits.
    DumpMachine();

    // Measure q1 and q2 and store the results in m1 and m2.
    let (m1, m2) = (M(q1), M(q2));

    // Reset q1 and q2 to the 0 state.
    Reset(q1);
```

```

Reset(q2);

// Return the measurement results.
return (m1, m2);
}

```

Para ejecutar el programa y ver el resultado de ambos cúbits, seleccione **Ejecutar** encima de la **Main** operación o presione **Ctrl+F5**.

```

1  import Microsoft.Quantum.Diagnostics.*;
2
3  Run | Histogram | Estimate | Debug | Circuit
4  operation Main() : (Result, Result) {
5      // Allocate two qubits, q1 and q2, in the 0 state.
6      use (q1, q2) = (Qubit(), Qubit());
7
8      // Put q1 into an even superposition.
9      // It now has a 50% chance of being measured as 0 or 1.
10     H(q1);
11
12     // Entangle q1 and q2, making q2 depend on q1.
13     CNOT(q1, q2);
14
15     // Show the entangled state of the qubits.
16     DumpMachine();
17
18     // Measure q1 and q2 and store the results in m1 and m2.
19     let (m1, m2) = (M(q1), M(q2));
20
21     // Reset q1 and q2 to the 0 state.
22     Reset(q1);

```

Puede ejecutar el programa varias veces, cada uno con un resultado diferente en la consola de depuración. Esto muestra la naturaleza probabilística de las medidas cuánticas y el entrelazamiento de los cúbits.

Por ejemplo, si el resultado es **Zero**, la consola de depuración debe tener este aspecto:

Result

DumpMachine:

Basis	Amplitude	Probability	Phase
00>	0.7071+0.0000i	50.0000%	0.0000
11>	0.7071+0.0000i	50.0000%	0.0000

```
Result: "(Zero, Zero)"
```

Paso siguiente

Para más información sobre el entrelazamiento cuántico con Q#, consulte [Tutorial: Exploración del entrelazamiento cuántico con Q#](#). Este tutorial se expande sobre los conceptos descritos en este inicio rápido y le ayuda a escribir un programa de entrelazamiento más avanzado.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Cómo enviar Q# programas con Visual Studio Code

Aprenda a usar Visual Studio Code (VS Code) para crear y enviar Q# programas al hardware cuántico real. Puede enviar trabajos de computación cuántica a Azure Quantum como un programa independiente Q#, combinarlos Q# con Python en un proyecto Q# y ejecutar un Jupyter Notebook.

Envío de los trabajos de Q# a Azure Quantum

Aprenda a usar VS Code para ejecutar, depurar y enviar un Q# programa a Azure Quantum.

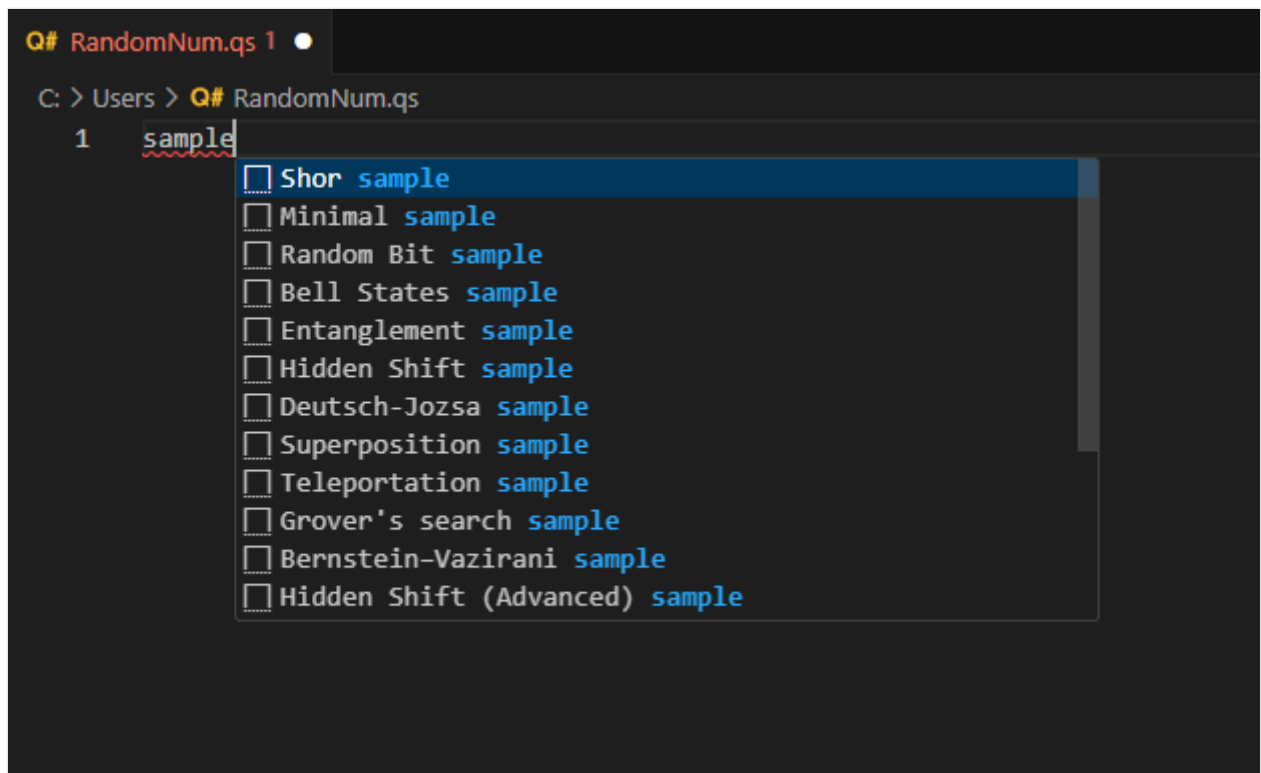
Requisitos previos

Para obtener más información sobre la instalación, consulte [Configuración de la QDK extensión](#).

- Un Azure Quantum área de trabajo en tu suscripción Azure. Para crear un área de trabajo, consulte [Creación de un Azure Quantum área de trabajo](#).
- La versión más reciente de [Visual Studio Code](#) [↗](#) o abre [VS Code en la Web](#) [↗](#).
- La versión más reciente de la [Azure Quantum Development Kit extensión](#) [↗](#).

Carga de un Q# programa de ejemplo

1. En VS Code, seleccione **Archivo > nuevo archivo de texto** y guarde el archivo como **RandomNum.qs**.
2. Abra **RandomNum.qs** y escriba `sample` y, a continuación, seleccione **Ejemplo de bit aleatorio** en la lista de opciones y guarde el archivo.

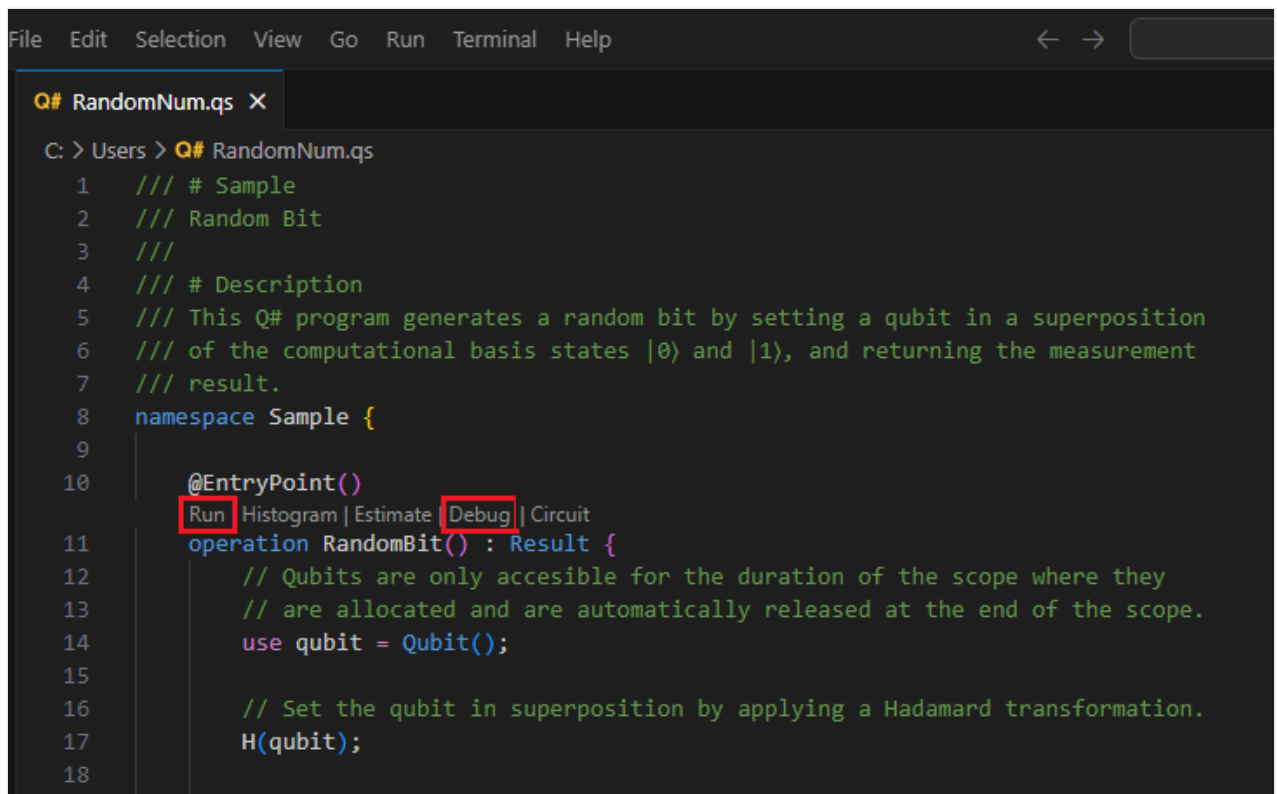


ⓘ Nota:

También puede abrir su propio Q# archivo. Si ejecuta un programa anterior Q# y tiene errores, consulte [Prueba y depuración](#).

Ejecución de un programa de Q#

1. Para probar la ejecución del programa localmente en el simulador integrado, haga clic en **Ejecutar** en la lista de comandos junto a la operación de punto de entrada o presione **Ctrl + F5**. La salida aparecerá en la consola de depuración.
2. Para depurar el programa antes de enviarlo a Azure Quantum, haga clic en **Depurar** en la lista de comandos junto a la operación de punto de entrada o presione **F5**. Use los controles de depuración de la parte superior para pasar, entrar y salir del código. Para obtener más información sobre la depuración de Q# programas, vea [Prueba y depuración](#).



```
File Edit Selection View Go Run Terminal Help

Q# RandomNum.qs X

C: > Users > Q# RandomNum.qs
1  /// # Sample
2  /// Random Bit
3  ///
4  /// # Description
5  /// This Q# program generates a random bit by setting a qubit in a superposition
6  /// of the computational basis states  $|0\rangle$  and  $|1\rangle$ , and returning the measurement
7  /// result.
8  namespace Sample {
9
10     @EntryPoint()
11     operation RandomBit() : Result {
12         // Qubits are only accesible for the duration of the scope where they
13         // are allocated and are automatically released at the end of the scope.
14         use qubit = Qubit();
15
16         // Set the qubit in superposition by applying a Hadamard transformation.
17         H(qubit);
18     }
```

Visualización del histograma de frecuencia

El histograma de frecuencia representa la distribución de resultados obtenidos al ejecutar un programa cuántico varias veces, o "series". Cada barra del histograma corresponde a un posible resultado y su altura representa el número de veces que se observa el resultado. El histograma de frecuencia ayuda a visualizar la distribución de probabilidad de estos resultados.

1. Seleccione **Ver -> Paleta de comandos** y escriba **histograma**, que muestra el comando **QDK: Ejecutar archivo y mostrar histograma**. También puede elegir **Histograma** en la lista de comandos que preceden a la operación de punto de entrada. Seleccione esta opción para abrir la Q# ventana del histograma.

```
File Edit Selection View Go Run Terminal Help

Q# RandomNum.qs X

C: > Users > Q# RandomNum.qs
1  /// # Sample
2  /// Random Bit
3  ///
4  /// # Description
5  /// This Q# program generates a random bit by setting a qubit in a superposition
6  /// of the computational basis states  $|0\rangle$  and  $|1\rangle$ , and returning the measurement
7  /// result.
8  namespace Sample {
9
10     @EntryPoint()
11     operation RandomBit() : Result {
12         // Qubits are only accesible for the duration of the scope where they
13         // are allocated and are automatically released at the end of the scope.
14         use qubit = Qubit();
15
16         // Set the qubit in superposition by applying a Hadamard transformation.
17         H(qubit);
18     }
```

2. Introduzca un número de **intentos** para ejecutar el programa, por ejemplo, 100 intentos, y presione **Intro**. El histograma se mostrará en la ventana del Q# histograma.
3. Haga clic en el icono de **configuración** de la parte superior izquierda para mostrar las opciones.



4. Haga clic en una barra para mostrar el **porcentaje** de ese resultado. En este caso, hay dos resultados posibles, 0 y 1, y el porcentaje de cada resultado está cerca del 50 %.



💡 Sugerencia

Puede acercar el histograma mediante la rueda del ratón o un gesto en el trackpad. Al acercar la vista, puede desplazar el gráfico si pulsa "Alt" mientras se desplaza.

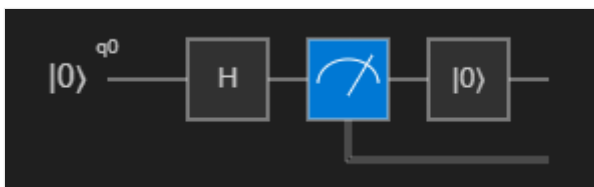
Visualización del circuito cuántico

Los diagramas de circuitos cuánticos son una representación visual de las operaciones cuánticas. Muestran el flujo de cúbits a través del programa cuántico, incluidas las puertas y mediciones aplicadas a ellos. Para obtener más información, consulte [Diagramas de circuitos cuánticos en Visual Studio Code](#).

1. Seleccione **Vista -> Paleta de comandos** y escriba **circuito**, lo que muestra el comando **QDK: Mostrar circuito**. También puede seleccionar **Circuito** en la lista de comandos junto a la operación de punto de entrada.

```
File Edit Selection View Go Run Terminal Help
Q# RandomNum.qs X
C: > Users > Q# RandomNum.qs
1  /// # Sample
2  /// Random Bit
3  ///
4  /// # Description
5  /// This Q# program generates a random bit by setting a qubit in a superposition
6  /// of the computational basis states  $|0\rangle$  and  $|1\rangle$ , and returning the measurement
7  /// result.
8  namespace Sample {
9
10     @EntryPoint()
11     operation RandomBit() : Result {
12         // Qubits are only accesible for the duration of the scope where they
13         // are allocated and are automatically released at the end of the scope.
14         use qubit = Qubit();
15
16         // Set the qubit in superposition by applying a Hadamard transformation.
17         H(qubit);
18     }
```

2. El circuito se muestra en la ventana del Q# circuito. El diagrama de circuitos muestra un registro de cúbits que se inicializa en el estado $|0\rangle$. A continuación, se aplica una puerta Hadamard, **H**, al cúbit, seguida de una **operación** de medición, que se representa mediante un símbolo de medidor. Para más información, consulte [Convenciones](#) de circuitos cuánticos.



Conéctate a Azure Quantum y envía tu trabajo

Puede conectar y enviar trabajos directamente desde VS Code. En este ejemplo, se envía un trabajo al simulador de Rigetti.

1. Seleccione **Ver -> Paleta de comandos** y escriba **QDK: Conectarse a un Azure Quantum área de trabajo**. Presione **ENTRAR**.
2. Seleccione **Azure la cuenta** y siga las indicaciones para conectarse a su directorio, suscripción y área de trabajo preferidos.

❗ Nota:

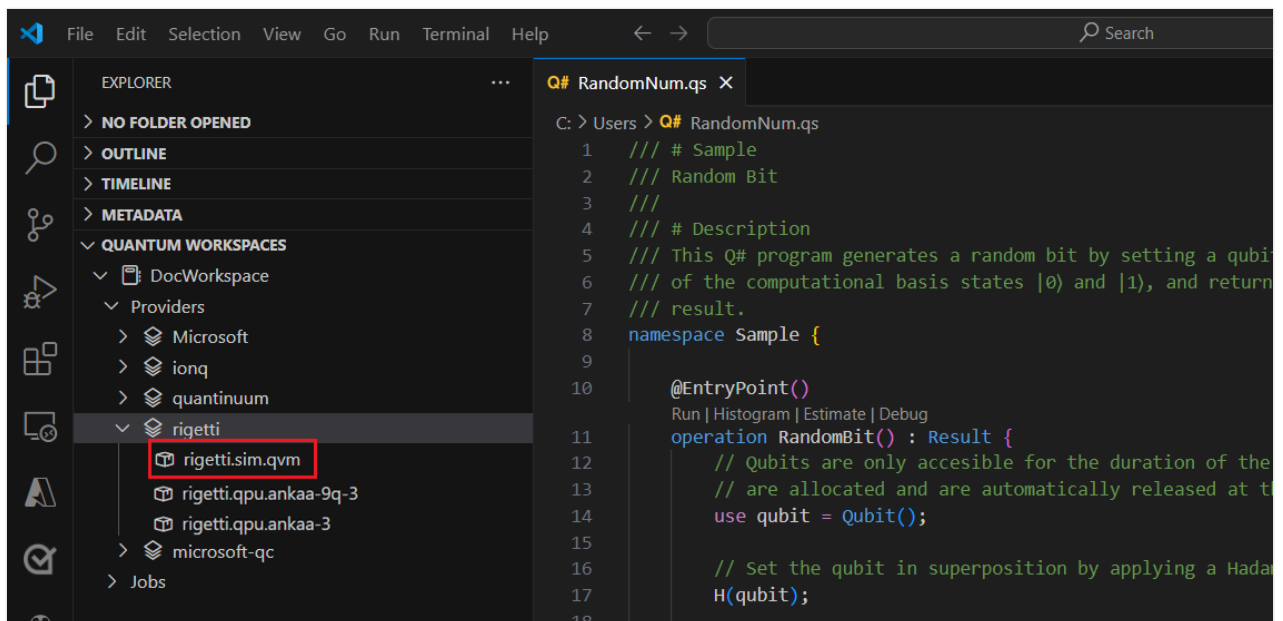
Si tiene una cadena de conexión, puede seleccionar **Cadena de conexión** y pegar la cadena de conexión correspondiente al Azure Quantum espacio de trabajo. Para obtener más información, consulte [Conexión a un área de trabajo de Quantum mediante un cadena de conexión](#).

- Una vez que esté conectado, en el **panel Explorador** , expanda **Áreas de trabajo cuánticas**.
- Expanda el área de trabajo y expanda el proveedor **Rigetti**.

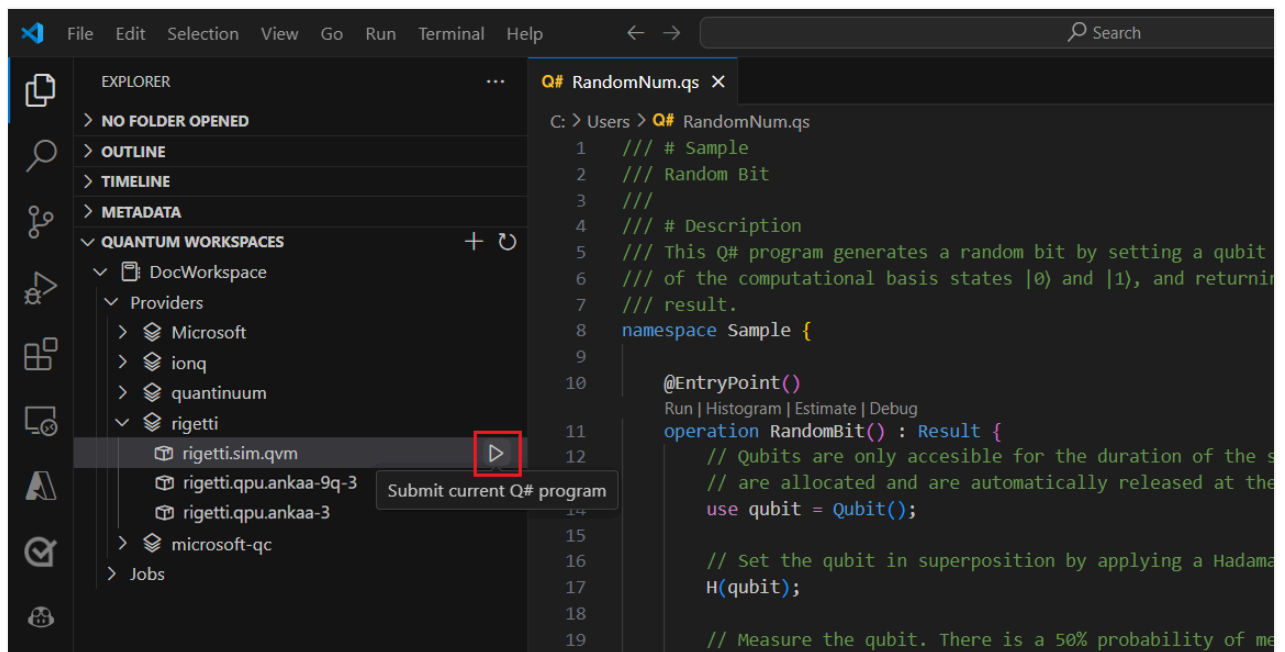
❗ **Nota:**

Si hay un problema al conectarse a Azure Quantum, se muestra un icono de advertencia junto al nombre del área de trabajo. Mantenga el puntero sobre el nombre del área de trabajo para mostrar información de error.

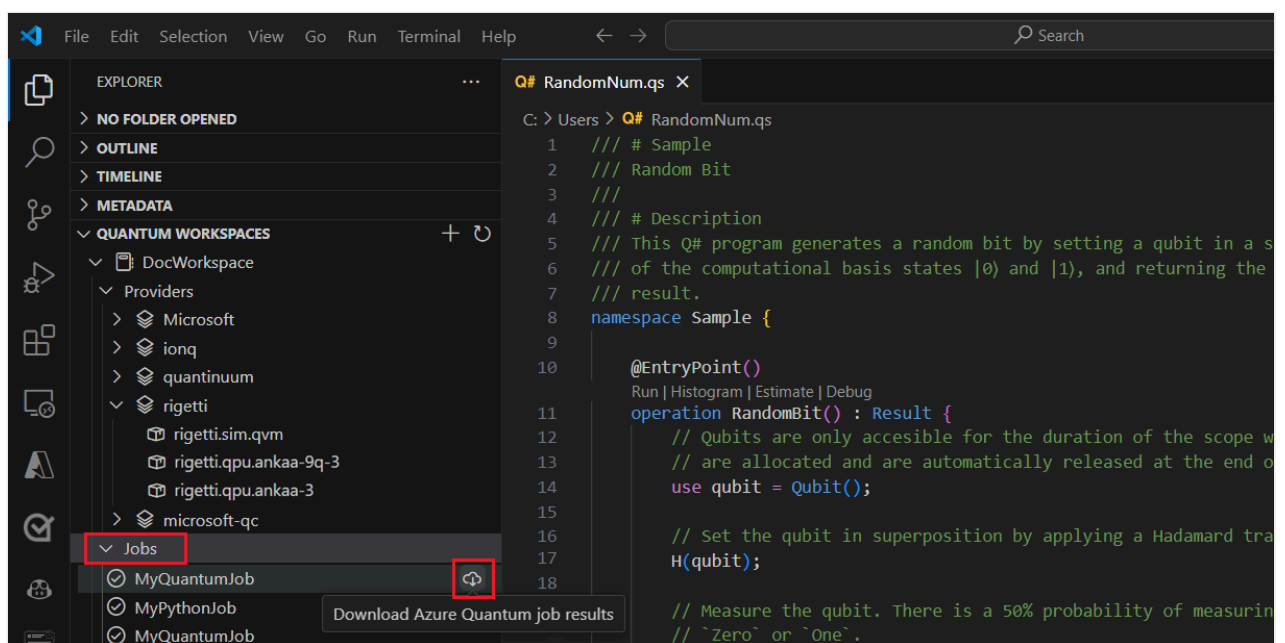
- Seleccione **rigetti.sim.qvm** como su target.



- Seleccione el icono de reproducción situado a la derecha del target nombre para empezar a enviar el programa actual Q# .



7. Agregue un nombre para identificar el trabajo.
8. Agregue el número de tomas o el número de veces que se ejecuta el programa.
9. Presione **Entrar** para enviar el trabajo. El estado del trabajo se mostrará en la parte inferior de la pantalla.
10. Expanda **Trabajos** y mantenga el puntero sobre el trabajo, que muestra las horas y el estado del trabajo.
11. Para ver los resultados, seleccione el icono de nube situado junto al nombre del trabajo para descargar los resultados del almacenamiento del área de trabajo y mostrarlos en VS Code.



Contenido relacionado

- [Trabajar con Azure Quantum tareas](#)
- [Proveedores de computación cuántica](#)

Last updated on 12/11/2025

Creación y administración de Q# proyectos y bibliotecas personalizadas

En este artículo, aprenderá a crear, administrar y compartir Q# proyectos. Un Q# proyecto es una estructura de carpetas con varios Q# archivos que pueden acceder entre sí a las operaciones y funciones del otro. Los proyectos le ayudan a organizar lógicamente el código fuente. También puede usar proyectos como bibliotecas personalizadas a las que se puede acceder desde orígenes externos.

Requisitos previos

- Un Azure Quantum espacio de trabajo en tu suscripción de Azure. Para crear un área de trabajo, consulte [Creación de un Azure Quantum área de trabajo](#).
- Visual Studio Code (VS Code) con las extensiones [Azure Quantum Development Kit](#) y [Python](#) instaladas.
- Una cuenta de GitHub, si planea publicar el proyecto externo en un repositorio público de GitHub.

Para ejecutar Python programas, también necesita lo siguiente:

- Un entorno Python con [Python y Pip](#) instalado.
- La `qdk` Python biblioteca con el complemento opcional `azure`.

Bash

```
python -m pip install --upgrade qdk[azure]
```

Cómo Q# funcionan los proyectos

Un Q# proyecto contiene un Q# archivo de manifiesto, denominado `qsharp.json` y uno o varios `.qs` archivos en una estructura de carpetas especificada. Puede crear un Q# proyecto manualmente o directamente en VS Code.

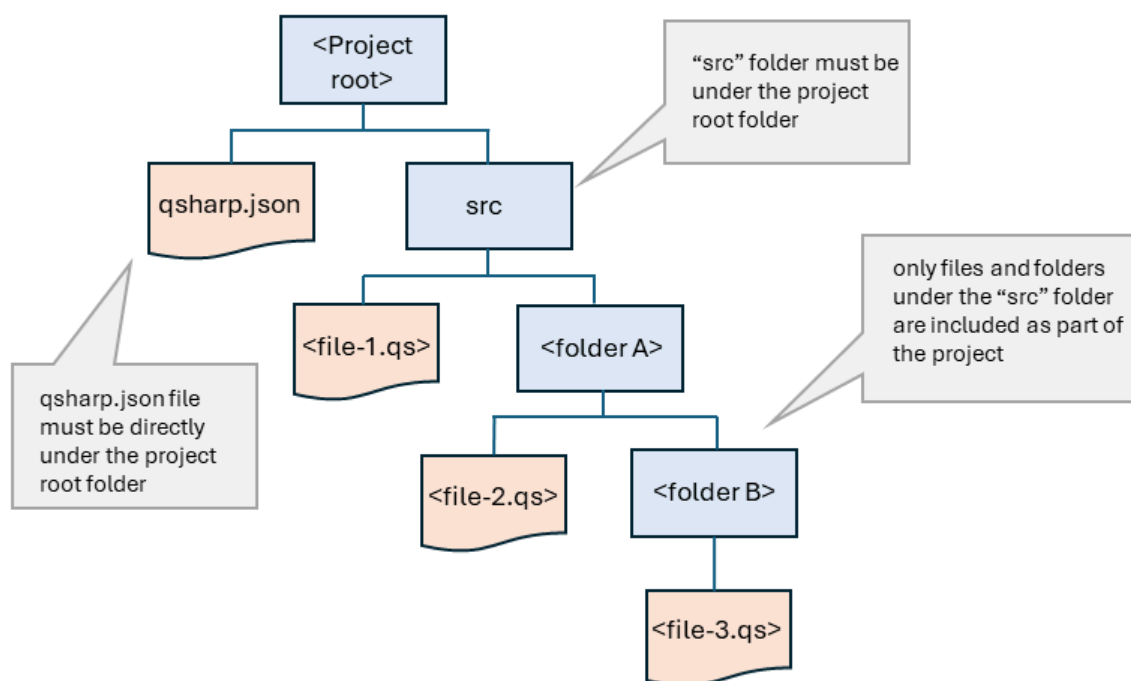
Al abrir un `.qs` archivo en VS Code, el compilador busca en la jerarquía de carpetas circundantes el archivo de manifiesto y determina el ámbito del proyecto. Si no se encuentra ningún archivo de manifiesto, el compilador funciona en un modo de archivo único.

Cuando se establece `project_root` en un archivo Jupyter Notebook o Python, el compilador busca el archivo de manifiesto en la carpeta `project_root`.

Un proyecto externo Q# es un proyecto estándar Q# que reside en otro directorio o en un repositorio público de GitHub y actúa como una biblioteca personalizada. Un proyecto externo usa `export` instrucciones para definir las funciones y las operaciones a las que pueden acceder los programas externos. Los programas definen el proyecto externo como una dependencia en su archivo de manifiesto y usan `import` instrucciones para tener acceso a los elementos del proyecto externo, como operaciones, funciones, estructuras y espacios de nombres. Para obtener más información, consulte [Uso de proyectos como dependencias](#) externas.

Definición de un Q# proyecto

Un Q# proyecto se define mediante la presencia de un archivo de manifiesto, denominado `qsharp.json` y una `src` carpeta, ambos deben estar en la carpeta raíz del proyecto. La `src` carpeta contiene los Q# archivos de origen. En el caso Q# de programas y proyectos externos, el Q# compilador detecta automáticamente la carpeta del proyecto. Para Python programas y Jupyter Notebook archivos, debe [especificar la carpeta del Q# proyecto](#) con una `qsharp.init` invocación. Sin embargo, la estructura de carpetas de un Q# proyecto es la misma para todos los tipos de programas.



Uso de un Q# programa

Definir la carpeta del proyecto (Q# programas)

Al abrir un `.qs` archivo en VS Code, el Q# compilador busca hacia arriba en la estructura de carpetas de un archivo de manifiesto. Si el compilador encuentra un archivo de manifiesto, el compilador incluye todos los Q# archivos del `/src` directorio y todos los subdirectorios. Los elementos definidos en cada archivo están disponibles para todos los demás archivos del proyecto.

Por ejemplo, considere la siguiente estructura de carpetas:

- **Teleportation_project**
 - `qsharp.json`
 - **src**
 - `Main.qs`
 - **TeleportOperations**
 - `TeleportLib.qs`
 - **PrepareState**
 - `PrepareStateLib.qs`

Al abrir el archivo `/src/TeleportOperation/PrepareState/PrepareStateLib.qs`, el Q# compilador hace lo siguiente:

1. Comprueba `/src/TeleportOperation/PrepareState/` para `qsharp.json`.
2. Comprueba `/src/TeleportOperation` para `qsharp.json`.
3. Comprueba `/src` para `qsharp.json`.
4. Comprueba `/*` para `qsharp.json`.
5. Establece `/` como directorio raíz del proyecto e incluye todos los `.qs` archivos en la raíz del proyecto, según la configuración del archivo de manifiesto.

Crear un archivo de manifiesto

Un archivo de manifiesto es un archivo JSON denominado `qsharp.json` que puede incluir opcionalmente campos de **autor**, **licencia** y **lints**. El archivo de manifiesto mínimo viable es la cadena `{}`. Al crear un Q# proyecto en VS Code, se crea un archivo de manifiesto mínimo automáticamente.

JSON

```
{}
```

Ejemplos de archivos de manifiesto

En los ejemplos siguientes se muestra cómo los archivos de manifiesto pueden definir el ámbito del Q# proyecto.

- En este ejemplo, **author** es el único campo especificado, por lo que todos los `.qs` archivos de este directorio y sus subdirectorios se incluyen en el Q# proyecto.

JSON

```
{
  "author": "Microsoft",
  "license": "MIT"
}
```

- Dentro de un Q# proyecto, también puede usar el archivo de manifiesto para ajustar la VS CodeQ# configuración de Linter. De forma predeterminada, las tres reglas de Linter son:
 - `needlessParens`: valor predeterminado = `allow`
 - `divisionByZero`: valor predeterminado = `warn`
 - `redundantSemicolons`: valor predeterminado = `warn`

Puede establecer cada regla en el archivo de manifiesto en `allow`, `warn` o `error`. Por ejemplo:

JSON

```
{
  "author": "Microsoft",
  "lints": [
    {
      "lint": "needlessParens",
      "level": "allow"
    },
    {
      "lint": "redundantSemicolons",
      "level": "warn"
    },
    {
      "lint": "divisionByZero",
      "level": "error"
    }
  ]
}
```

- También puede usar el archivo de manifiesto para definir un proyecto externo Q# como una dependencia y acceder de forma remota a las operaciones y funciones de ese

proyecto externo. Para obtener más información, consulte [Uso de proyectos como dependencias](#) externas.

Q# requisitos y propiedades del proyecto

Los siguientes requisitos y configuraciones se aplican a todos los Q# proyectos.

- Todos los `.qs` archivos que desea incluir en el proyecto deben estar en una carpeta denominada `src`, que debe estar en la carpeta raíz del Q# proyecto. Al crear un Q# proyecto en VS Code, la `/src` carpeta se crea automáticamente.
- El archivo de manifiesto debe estar en el mismo nivel que la `src` carpeta. Al crear un Q# proyecto en VS Code, se crea automáticamente un archivo mínimo.
- Use `import` instrucciones para hacer referencia a operaciones y funciones de otros archivos del proyecto.

Q#

```
import MyMathLib.*; //imports all the callables in the MyMathLib namespace

...

Multiply(x,y);
```

O bien, haga referencia a ellos individualmente con el espacio de nombres .

Q#

```
MyMathLib.Multiply(x,y);
```

Solo para proyectos Q#

- Puede definir una operación de punto de entrada solo en un archivo `.qs` en un proyecto Q#, que es la operación `Main()` por defecto.
- Debe colocar el archivo `.qs` con la definición de punto de entrada en un nivel de directorio del proyecto por debajo del archivo de manifiesto.
- Todas las operaciones y funciones del proyecto Q# que se almacenan en caché a partir de una pantalla `.qs` se muestran como texto predictivo en VS Code.
- Si aún no se importa el espacio de nombres de una operación o función seleccionada, VS Code agrega automáticamente la instrucción necesaria `import` .

Creación de un Q# proyecto

Para crear un Q# proyecto, siga estos pasos:

1. En el VS Code explorador de archivos, vaya a la carpeta que desea usar como carpeta raíz del Q# proyecto.
2. Abra el menú **Ver** y elija **Paleta de comandos**.
3. Escriba **QDK: Crear proyecto Q#** y pulse **Intro**. VS Code crea un archivo de manifiesto mínimo en la carpeta y agrega una `/src` carpeta con un `Main.qs` archivo de plantilla.
4. Edite el archivo de manifiesto del proyecto. Consulte Ejemplos [de archivos](#) de manifiesto.
5. Agregue y organice los Q# archivos de origen en la `/src` carpeta .
6. Si accede al Q# proyecto a través de un Python programa o Jupyter Notebook, establezca la [ruta de acceso de la carpeta raíz](#) utilizando `qsharp.init` . En este ejemplo se supone que el programa está en la `/src` carpeta del Q# proyecto:

Python

```
qsharp.init(project_root = '../Teleportation_project')
```

7. Si solo está utilizando archivos Q# en VS Code, el compilador busca un archivo de manifiesto al abrir un archivo Q#, determina la carpeta raíz del proyecto y luego examina la subcarpeta para archivos `.qs` .

⚠ Nota:

También puede crear manualmente el archivo de manifiesto y la `/src` carpeta .

Proyecto de ejemplo

Este programa de teleportación cuántica es un ejemplo de proyecto que se ejecuta en el simulador local en VS Code. Para ejecutar el programa en Azure Quantum hardware o simuladores de terceros, consulte [Introducción a los programas Q# y VS Code](#) para obtener pasos sobre cómo compilar su programa y conectarse a su espacio de trabajo Azure Quantum.

Este ejemplo tiene la siguiente estructura de directorios:

- **Teleportation_project**
 - `qsharp.json`

- **src**
 - *Main.qs*
 - **TeleportOperations**
 - *TeleportLib.qs*
 - **PrepareState**
 - *PrepareStateLib.qs*

El archivo de manifiesto contiene los **campos de autor y licencia** :

JSON

```
{
  "author": "Microsoft",
  "license": "MIT"
}
```

Q# archivos de origen

El archivo principal, denominado `Main.qs`, contiene el punto de entrada y hace referencia al `TeleportOperations.TeleportLib` espacio de nombres de `TeleportLib.qs`.

Q#

```
import TeleportOperations.TeleportLib.Teleport; // references the Teleport
operation from TeleportLib.qs

operation Main() : Unit {
    use msg = Qubit();
    use target = Qubit();

    H(msg);
    Teleport(msg, target); // calls the Teleport() operation from TeleportLib.qs
    H(target);

    if M(target) == Zero {
        Message("Teleported successfully!");
    }

    Reset(msg);
    Reset(target);
}
```

El `TeleportLib.qs` archivo define la operación `Teleport` y llama a la operación `PrepareBellPair` desde el archivo `PrepareStateLib.qs`.

Q#

```

import TeleportOperations.PrepareState.PrepareStateLib.*; // references the
namespace in PrepareStateLib.qs

operation Teleport(msg : Qubit, target : Qubit) : Unit {
    use here = Qubit();

    PrepareBellPair(here, target); // calls the PrepareBellPair() operation from
PrepareStateLib.qs
    Adjoint PrepareBellPair(msg, here);

    if M(msg) == One { Z(target); }
    if M(here) == One { X(target); }

    Reset(here);
}

```

El `PrepareStateLib.qs` archivo contiene una operación reutilizable estándar para crear un par bell.

Q#

```

operation PrepareBellPair(left : Qubit, right : Qubit) : Unit is Adj + Ctl {
    H(left);
    CNOT(left, right);
}

```

Ejecución de los programas

Elija la pestaña del entorno en el que se ejecuta el programa.

Ejecutar Q# en VS Code

Para ejecutar este programa, abra el `Main.qs` archivo en VS Code y elija **Ejecutar**.

Configurar Q# proyectos como dependencias externas

Puede configurar Q# proyectos como dependencias externas para otros proyectos, de forma similar a una biblioteca. Las funciones y operaciones del proyecto externo Q# están disponibles para varios Q# proyectos. Una dependencia externa puede residir en una unidad compartida o publicarse en un repositorio público de GitHub.

Para usar un Q# proyecto como dependencia externa, debe:

- Agregue el proyecto externo como dependencia en el archivo de manifiesto del proyecto que llama.
- Si el proyecto externo se publica en GitHub, agregue la propiedad **files** al archivo de manifiesto del proyecto externo.
- Agregue instrucciones `export` al proyecto externo.
- Agregue instrucciones `import` al proyecto que llama.

Configurar los archivos de manifiesto

Los proyectos externos Q# pueden residir en un recurso compartido de unidad de red o local, o publicarse en un repositorio público de GitHub.

El archivo de manifiesto del proyecto que llama

Para agregar una dependencia a un proyecto externo en un recurso compartido de unidad, defina la dependencia en el archivo de manifiesto del proyecto que llama.

JSON

```
{
  "author": "Microsoft",
  "license": "MIT",
  "dependencies": {
    "MyDependency": {
      "path": "/path/to/project/folder/on/disk"
    }
  }
}
```

En el archivo de manifiesto anterior, **MyDependency** es una cadena definida por el usuario que identifica el espacio de nombres al llamar a una operación. Por ejemplo, si crea una dependencia denominada `MyMathFunctions`, puede llamar a una función desde esa dependencia con `MyMathFunctions.MyFunction()`.

Para agregar una dependencia a un proyecto publicado en un repositorio público de GitHub, use el siguiente archivo de manifiesto de ejemplo:

JSON

```
{
  "author": "Microsoft",
  "dependencies": {
    "MyDependency": {
      "github": {
        "owner": "GitHubUser",

```

```

    "repo": "GitHubRepoName",
    "ref": "CommitHash",
    "path": "/path/to/dependency"
  }
}
}

```

❗ Nota:

Para las dependencias de GitHub, **ref** hace referencia a una [especificación ref de](#) GitHub. Microsoft recomienda usar siempre un hash de confirmación para que puedas confiar en una versión específica de tu dependencia.

El archivo de manifiesto del proyecto externo

Si el proyecto externo Q# se publica en un repositorio público de GitHub, debe agregar la propiedad **files** al archivo de manifiesto del proyecto externo, incluidos todos los archivos usados en el proyecto.

JSON

```

{
  "author": "Microsoft",
  "license": "MIT",
  "files": [ "src/MyMathFunctions.qs", "src/Strings/MyStringFunctions.qs" ]
}

```

La propiedad **files** es opcional para un proyecto externo que se importa a través de **"path"** (es decir, una importación basada en ruta de archivo local). La propiedad **files** solo es necesaria para los proyectos que se publican en GitHub.

Use la **export** instrucción

Para que las funciones y las operaciones de un proyecto externo sean accesibles para los proyectos que llaman, use la instrucción **export**. Puede exportar cualquiera o todos los elementos invocables en el archivo. No se admite la sintaxis de caracteres comodín, por lo que debe especificar cada invocable que quiera exportar.

Q#

```

operation Operation_A() : Unit {
  ...
}

```

```
operation Operation_B() : Unit {
    ...
}

// makes just Operation_A available to calling programs
export Operation_A;

// makes Operation_A and Operation_B available to calling programs
export Operation_A, Operation_B, etc.;

// makes Operation_A available as 'OpA'
export Operation_A as OpA;
```

Use la `import` instrucción

Para que los elementos de una dependencia externa estén disponibles, use instrucciones `import` del programa de llamada. La `import` instrucción usa el espacio de nombres definido para la dependencia en el archivo de manifiesto.

Por ejemplo, considere la dependencia en el siguiente archivo de manifiesto:

JSON

```
{
  "author": "Microsoft",
  "license": "MIT",
  "dependencies": {
    "MyMathFunctions": {
      "path": "/path/to/project/folder/on/disk"
    }
  }
}
```

Importe los invocables con el código siguiente:

Q#

```
import MyMathFunctions.MyFunction; // imports "MyFunction()" from the namespace
...
```

La `import` instrucción también admite la sintaxis y los alias de comodín.

Q#

```
// imports all items from the "MyMathFunctions" namespace
import MyMathFunctions.*;

// imports the namespace as "Math", all items are accessible via "Math.<callable>"
```

```
import MyMathFunctions as Math;

// imports a single item, available in the local scope as "Add"
import MyMathFunctions.MyFunction as Add;

// imports can be combined on one line
import MyMathFunctions.MyFunction, MyMathFunctions.AnotherFunction as Multiply;
```

ⓘ Nota:

La instrucción usada `open` actualmente en Q#, que se usa para hacer referencia a bibliotecas y espacios de nombres, todavía se admite, pero finalmente quedará en desuso. Mientras tanto, puede actualizar opcionalmente los archivos actuales para usar la `import` instrucción. Por ejemplo, `open Std.Diagnostics;` se puede reemplazar por `import Std.Diagnostics.*;`

Proyecto externo de ejemplo

En este ejemplo, se usa el mismo programa de teletransportación que el ejemplo anterior, pero se separa el programa de llamada y los invocables en proyectos diferentes.

1. Cree dos carpetas en la unidad local, por ejemplo `Project_A` , y `Project_B`.
2. Cree un Q# proyecto en cada carpeta. Para obtener más información, consulte los pasos descritos en [Creación de un Q# proyecto](#).
3. En `Project_A`, el programa que realiza la llamada, copie en el archivo de manifiesto el código siguiente, pero edite la ruta según sea necesario para `Project_B`.

JSON

```
{
  "author": "Microsoft",
  "license": "MIT",
  "dependencies": {
    "MyTeleportLib": {
      "path": "/Project_B"
    }
  }
}
```

4. En `Project_A`, copie el código siguiente en `Main.qs`:

Q#

```

import MyTeleportLib.Teleport; // imports the Teleport operation from the
MyTeleportLib namespace defined in the manifest file

operation Main() : Unit {
    use msg = Qubit();
    use target = Qubit();

    H(msg);
    Teleport(msg, target); // calls the Teleport() operation from the
MyTeleportLib namespace
    H(target);

    if M(target) == Zero {
        Message("Teleported successfully!");
    }

    Reset(msg);
    Reset(target);
}
}

```

5. En `Project_B`, copie el código siguiente en `Main.qs`:

Q#

```

operation Teleport(msg : Qubit, target : Qubit) : Unit {
    use here = Qubit();

    PrepareBellPair(here, target);
    Adjoint PrepareBellPair(msg, here);

    if M(msg) == One { Z(target); }
    if M(here) == One { X(target); }

    Reset(here);
}

operation PrepareBellPair(left : Qubit, right : Qubit) : Unit is Adj + Ctl
{
    H(left);
    CNOT(left, right);
}

export Teleport; // makes the Teleport operation available to
external programs

```

⚠ Nota:

Tenga en cuenta que no es necesario exportar la operación `PrepareBellPair` ya que no se llama directamente desde su programa en `Project_A`. Dado que

`PrepareBellPair` está en el ámbito local de `Project_B`, ya es accesible por la operación `Teleport`.

6. Para ejecutar el programa, abra `/Project_A/Main.qs` en VS Code y elija **Ejecutar**.

Proyectos y espacios de nombres implícitos

En Q# los proyectos, si no se especifica un espacio de nombres en un `.qs` programa, el compilador usa el nombre del archivo como espacio de nombres. A continuación, cuando se hace referencia a un invocable desde una dependencia externa, se usa la sintaxis `<dependencyName>.<namespace>.<callable>`. Sin embargo, si el archivo se denomina `Main.qs`, el compilador asume que el espacio de nombres y la sintaxis de llamada es `<dependencyName>.<callable>`, como en el ejemplo anterior, `import MyTeleportLib.Teleport`.

Dado que es posible que tenga varios archivos de proyecto, debe tener en cuenta la sintaxis correcta al hacer referencia a los invocables. Por ejemplo, considere un proyecto con la siguiente estructura de archivos:

- **/Fuente**
 - `Main.qs`
 - `MathFunctions.qs`

El código siguiente realiza llamadas a la dependencia externa:

Q#

```
import MyTeleportLib.MyFunction;           // "Main" namespace is implied

import MyTeleportLib.MathFunctions.MyFunction; // "Math" namespace must be
explicit
```

Para obtener más información sobre el comportamiento del espacio de nombres, consulte [Espacios de nombres](#) de usuario.

Cómo depurar y probar tu código cuántico

14/10/2025

Al igual que con la programación clásica, es esencial poder comprobar que los programas cuánticos actúan según lo previsto y poder diagnosticar comportamientos incorrectos. En este artículo se describen las herramientas que ofrece el Quantum Development Kit de Azure para probar y depurar programas cuánticos.

Las pruebas y la depuración son tan importantes en la programación cuántica como en la programación clásica. En este artículo se describe cómo depurar y probar los programas cuánticos con Azure Quantum Development Kit (QDK) en Visual Studio Code (VS Code) y Jupyter Notebook.

Depura tu código cuántico

El QDK proporciona varias herramientas para depurar el código. Si usted escribe Q# o programas OpenQASM en VS Code, entonces puede usar el depurador de VS Code para establecer puntos de interrupción en sus programas y analizar su código. El QDK también proporciona un conjunto de funciones de volcado de memoria que puede usar para obtener información en distintos puntos del programa.

Uso del depurador de VS Code

Con la extensión QDK en VS Code, puede usar el depurador para recorrer el código y en cada función o operación, realizar un seguimiento de los valores de las variables locales y seguir los estados cuánticos de los cúbits.

En el ejemplo siguiente se muestra cómo usar el depurador con un programa Q#. Para obtener información completa sobre los depuradores de VS Code, consulte [Depuración](#) en el sitio web de VS Code.

1. En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente:

Q#

```
import Std.Arrays.*;
import Std.Convert.*;

operation Main() : Result {
    use qubit = Qubit();
    H(qubit);
    let result = M(qubit);
    Reset(qubit);
}
```

```
    return result;
}
```

2. En la línea 6, `H(qubit)`, haga clic a la izquierda del número de línea para establecer un punto de interrupción. Aparece un círculo rojo.
3. En la barra del lado principal, elija el icono del depurador para abrir el panel del depurador y, a continuación, elija **Ejecutar y depurar**. Aparece la barra de control del depurador.
4. Presione **F5** para iniciar el depurador y continuar hasta el punto de interrupción. En el menú **Variables** del panel del depurador, expanda la lista desplegable **Estado cuántico** para ver que el cúbit se ha inicializado en el estado $|0\rangle$.
5. Presione **F11** para entrar en la `H` operación. Aparece el código fuente de la `H` operación. Observe que el **estado cuántico** cambia a una superposición al recorrer la `H` operación.
6. Presione **F10** para recorrer paso a paso la `M` operación. Observe que el **estado cuántico** se resuelve en $|0\rangle$ o $|1\rangle$ después de la medición. La `result` variable también aparece en **Variables locales**.
7. Presione **F10** de nuevo para pasar por encima de la `Reset` operación. Observe que el **estado cuántico** se restablece a $|0\rangle$.

Cuando haya terminado de explorar el depurador, presione ***Ctrl + F5** para salir del depurador.

⚠ Nota

El depurador de VS Code solo funciona con Q# archivos (`.qs`) y OpenQASM (`.qasm`). No puede usar el depurador de VS Code en Q# celdas de Jupyter Notebook.

Cómo depurar con el uso de funciones de volcado de QDK

El QDK proporciona varias Q# funciones de Python que vuelcan información sobre el estado actual de tu programa cuando llamas a estas funciones. Use información de estas funciones de volcado de memoria para comprobar si el programa se comporta según lo previsto.

La Q# `DumpMachine` función

`DumpMachine` es una Q# función que permite volcar información sobre el estado actual del sistema de cúbits en la consola a medida que se ejecuta el programa. `DumpMachine` no detiene ni interrumpe el programa durante el tiempo de ejecución.

En el siguiente ejemplo se llama a `DumpMachine` en dos puntos de un programa Q# y se explora la salida.

1. En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente:

Q#

```
import Std.Diagnostics.*;

operation Main() : Unit {
    use qubits = Qubit[2];
    X(qubits[1]);
    H(qubits[1]);
    DumpMachine();

    R1Frac(1, 2, qubits[0]);
    R1Frac(1, 3, qubits[1]);
    DumpMachine();

    ResetAll(qubits);
}
```

2. Presione **Ctrl + Mayús + Y** para abrir la **consola de depuración**.
3. Presione **Ctrl + F5** para ejecutar el programa. La siguiente salida de `DumpMachine` aparece en la **Consola de Depuración**:

Resultados

Basis	Amplitude	Probability	Phase
00>	0.7071+0.0000i	50.0000%	0.0000
01>	-0.7071+0.0000i	50.0000%	-3.1416

Basis	Amplitude	Probability	Phase
00>	0.7071+0.0000i	50.0000%	0.0000
01>	-0.6533-0.2706i	50.0000%	-2.7489

La salida de `DumpMachine` muestra cómo cambia el estado de los sistemas de cúbits después de cada conjunto de puertas.

❗ Nota

La salida de `DumpMachine` usa la ordenación big-endian.

La función de Python `dump_machine`

La `dump_machine` función es una función de la `qsharp` biblioteca de Python. Esta función devuelve la cantidad actual de qubits asignados y un diccionario que contiene las amplitudes de estado escasas del sistema de qubits.

En el ejemplo siguiente se ejecuta el mismo programa que el ejemplo anterior `DumpMachine` , pero en un cuaderno de Jupyter Notebook en lugar de un `.qs` archivo.

1. En VS Code, presione **Ctrl + Mayús + P** para abrir la **paleta de comandos**.
2. Escriba **Create: New Jupyter Notebook (Crear: Nuevo cuaderno de Jupyter Notebook)** y presione **Entrar**. Se abre una nueva pestaña de Jupyter Notebook.
3. En la primera celda, copie y ejecute el código siguiente:

```
Python

import qsharp
```

4. Cree una nueva celda de código y, a continuación, copie y ejecute el código siguiente Q# :

```
Q#

%%qsharp

use qubits = Qubit[2];
X(qubits[0]);
H(qubits[1]);
```

5. Cree una celda de código. Copie y ejecute el siguiente código de Python para ver el estado del cúbit en este punto del programa:

```
Python

dump = qsharp.dump_machine()
dump
```

La `dump_machine` función muestra la siguiente salida:

Resultados

Basis State	Amplitude	Measurement Probability	Phase
$ \psi_1 \dots \psi_n\rangle$			

$ 10\rangle$	$0.7071+0.0000i$	50.0000%	$\uparrow$	0.0000
$ 11\rangle$	$0.7071+0.0000i$	50.0000%	$\uparrow$	0.0000

6. Cree una nueva celda de código y, a continuación, copie y ejecute el código siguiente Q# :

Q#

```
%%qsharp
```

```
R1Frac(1, 2, qubits[0]);
```

```
R1Frac(1, 3, qubits[1]);
```

7. Cree una celda de código. Copie y ejecute el siguiente código de Python para ver el estado del cúbit en este punto del programa:

Python

```
dump = qsharp.dump_machine()
dump
```

La `dump_machine` función muestra la siguiente salida:

Resultados

Basis State

$(\psi_1 \dots \psi_n\rangle)$	Amplitude	Measurement Probability	Phase
$ 10\rangle$	$0.5000+0.5000i$	50.0000%	$\nearrow$ 0.7854
$ 11\rangle$	$0.2706+0.6533i$	50.0000%	$\nearrow$ 1.1781

8. Para imprimir una versión abreviada de la `dump_machine` salida, cree una nueva celda y ejecute el siguiente código de Python:

Python

```
print(dump)
```

9. Para obtener el número total de cúbits en el sistema, cree una nueva celda de código y ejecute el siguiente código de Python:

Python

```
dump.qubit_count
```

10. Puede acceder a las amplitudes de los estados de cúbits individuales que tienen amplitud distinta de cero. Por ejemplo, cree una nueva celda de código y ejecute el siguiente código de Python para obtener las amplitudes individuales de los estados $|10\rangle$ y $|11\rangle$:

Python

```
print(dump[2])  
print(dump[3])
```

La función `dump_operation`

La `dump_operation` función es una función del paquete de `qsharp.utils` Python. Esta función toma dos entradas: una operación o definición de operación como una cadena y el número de cúbits que se usan en la operación. La salida de `dump_operation` es una lista anidada que representa la matriz cuadrada de números complejos que corresponde a la operación cuántica dada. Los valores de la matriz se encuentran en la base computacional y cada sublista representa una fila de la matriz.

En el ejemplo siguiente se usa `dump_operation` para mostrar información de un sistema de 1 cúbit y 2 cúbits.

1. En VS Code, presione **Ctrl + Mayús + P** para abrir la **paleta de comandos**.
2. Escriba **Create: New Jupyter Notebook (Crear: Nuevo cuaderno de Jupyter Notebook)** y presione **Entrar**. Se abre una nueva pestaña de Jupyter Notebook.
3. En la primera celda, copie y ejecute el código siguiente:

Python

```
import qsharp  
from qsharp.utils import dump_operation
```

4. Para mostrar los elementos de matriz de una puerta de un qubit único, llame a `dump_operation` y pase 1 para el número de qubits. Por ejemplo, copie y ejecute el siguiente código de Python en una nueva celda de código para obtener los elementos de matriz de una puerta de identidad y una puerta hadamard:

Python

```
res = dump_operation("qs => ()", 1)  
print("Single-qubit identity gate:\n", res)  
print()
```

```
res = dump_operation("qs => H(qs[0])", 1)
print("Single-qubit Hadamard gate:\n", res)
```

5. También puede llamar a la función `qsharp.eval` y hacer referencia a la operación Q# en `dump_operation` para obtener el mismo resultado. Por ejemplo, cree una nueva celda de código y, a continuación, copie y ejecute el siguiente código de Python para imprimir los elementos de matriz de una puerta Hadamard de un solo cúbit:

Python

```
qsharp.eval("operation SingleH(qs : Qubit[]) : Unit { H(qs[0]) }")

res = dump_operation("SingleH", 1)
print("Single-qubit Hadamard gate:\n", res)
```

6. Para mostrar los elementos de matriz de una puerta de dos cúbits, llame a `dump_operation` y pase 2 como el número de cúbits. Por ejemplo, copie y ejecute el siguiente código de Python en una nueva celda de código para obtener los elementos de matriz de una operación Ry controlada donde el segundo cúbit es el target cúbit:

Python

```
qsharp.eval ("operation ControlRy(qs : Qubit[]) : Unit { Controlled
Ry([qs[0]], (0.5, qs[1])); }")

res = dump_operation("ControlRy", 2)
print("Controlled Ry rotation gate:\n", res)
```

Para obtener más ejemplos de cómo probar y depurar el código con `dump_operation`, consulte [Operaciones de prueba](#) de los ejemplos de QDK.

Prueba del código cuántico

El QDK proporciona varias Q# funciones y operaciones que puede usar para probar el código a medida que se ejecuta. También puede escribir pruebas unitarias para Q# programas.

La expresión `fail`

La `fail` expresión finaliza inmediatamente el programa. Para incorporar pruebas en el código, use las `fail` expresiones dentro de instrucciones condicionales.

En el ejemplo siguiente se usa una `fail` instrucción para probar que una matriz de cúbits contiene exactamente 3 cúbits. El programa finaliza con un mensaje de error cuando no se

supera la prueba.

1. En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente:

Q#

```
operation Main() : Unit {  
    use qs = Qubit[6];  
    let n_qubits = Length(qs);  
  
    if n_qubits != 3 {  
        fail $"The system should have 3 qubits, not {n_qubits}.";  
    }  
}
```

2. Presione **Ctrl + F5** para ejecutar el programa. Se produce un error en el programa y aparece la siguiente salida en la **consola de depuración**:

Resultados

Error: program failed: The system should have 3 qubits, not 6.

3. Edite el código de `Qubit[6]` a `Qubit[3]`, guarde el archivo y presione **Ctrl + F5** para volver a ejecutar el programa. El programa se ejecuta sin error porque la prueba se supera.

La función `Fact`

También puede usar la función Q# `Fact` del espacio de nombres `Std.Diagnostics` para probar el código. La `Fact` función toma una expresión booleana y una cadena de mensaje de error. Si la expresión booleana es true, la prueba pasa y el programa continúa ejecutándose. Si la expresión booleana es false, `Fact` finaliza el programa y muestra el mensaje de error.

Para realizar la misma prueba de longitud de matriz en el código anterior, pero con la `Fact` función, siga estos pasos:

1. En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente:

Q#

```
import Std.Diagnostics.Fact;  
  
operation Main() : Unit {  
    use qs = Qubit[6];  
    let n_qubits = Length(qs);
```

```
Fact(n_qubits == 3, $"The system should have 3 qubits, not {n_qubits}.")
}
```

2. Presione **Ctrl + F5** para ejecutar el programa. La condición de prueba en `Fact` no se supera y el mensaje de error aparece en la **consola de depuración**.
3. Edite el código de `Qubit[6]` a `Qubit[3]`, guarde el archivo y presione **Ctrl + F5** para volver a ejecutar el programa. La condición de prueba en `Fact` supera y el programa se ejecuta sin un error.

Escribe Q# pruebas unitarias con la anotación `@Test()`

En Q# los programas, puede aplicar la `@Test()` anotación a una función o operación invocable para convertir el invocable en una prueba unitaria. Estas pruebas de unidades aparecen en el menú **Pruebas** de VS Code para que pueda aprovechar esta característica de VS Code. Puede convertir un invocable en una prueba unitaria solo cuando el invocable no toma parámetros de entrada.

En el ejemplo siguiente, se ajusta el código de prueba para la longitud de la matriz en una operación y se convierte esa operación en una prueba unitaria.

1. En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente:

```
Q#

import Std.Diagnostics.Fact;

@Test()
operation TestCase() : Unit {
    use qs = Qubit[3];
    let n_qubits = Length(qs);

    Fact(n_qubits == 3, $"The system should have 3 qubits, not {n_qubits}.");
}
```

La `@Test()` anotación en la línea antes de la definición de la `TestCase` operación convierte la operación en un test unitario de VS Code. Aparece una flecha verde en la línea de definición de la operación.

2. Elija la flecha verde para ejecutar `TestCase` y notificar los resultados de la prueba.
3. Para interactuar con tus pruebas unitarias en el Explorador de Pruebas de VS Code, elige el icono **Testing** en la barra lateral principal.

4. Edite el código de `Qubit[3]` a `Qubit[6]` y vuelva a ejecutar la prueba unitaria para ver cómo cambian los resultados de la prueba.

Puede escribir y ejecutar Q# pruebas unitarias en VS Code sin necesidad de una operación de punto de entrada en su programa.

❗ Nota

Los elementos invocables del espacio de nombres `Std.Diagnostics` no son compatibles con la generación de QIR, por lo que solo deberías incluir pruebas unitarias en el código Q# que ejecutas en simuladores. Si desea generar QIR a partir del Q# código, no incluya pruebas unitarias en el código.

Las `CheckZero` y `CheckAllZero` operaciones

Las operaciones `CheckZero` y `CheckAllZero` Q# comprueban si el estado actual de un cúbit o matriz de cúbits es $|0\rangle$. La `CheckZero` operación toma un solo cúbit y solo devuelve `true` cuando el cúbit está en estado $|0\rangle$. Las `CheckAllZero` operaciones toman una matriz de cúbits y solo devuelven `true` cuando todos los cúbits de la matriz están en estado $|0\rangle$. Para usar `CheckZero` y `CheckAllZero`, impórtelos desde el `Std.Diagnostics` espacio de nombres .

En el ejemplo siguiente se usan ambas operaciones. La operación `CheckZero` verifica que la operación `X` invierte el primer cúbit del estado $|0\rangle$ al estado $|1\rangle$, y la operación `CheckAllZero` verifica que ambos cúbits se restablecen al estado $|0\rangle$.

En VS Code, cree y guarde un nuevo `.qs` archivo con el código siguiente, ejecute el programa y examine la salida en la **consola de depuración**.

Q#

```
import Std.Diagnostics.*;

operation Main() : Unit {
    use qs = Qubit[2];
    X(qs[0]);

    if CheckZero(qs[0]) {
        Message("X operation failed");
    }
    else {
        Message("X operation succeeded");
    }

    ResetAll(qs);
}
```



```
if CheckAllZero(qs) {  
    Message("Reset operation succeeded");  
}  
else {  
    Message("Reset operation failed");  
}  
}
```

Visualización de diagramas de circuitos cuánticos con Q#

Los diagramas de circuitos cuánticos son una representación visual de las operaciones cuánticas. Los diagramas de circuitos muestran el flujo de cúbits a través del programa cuántico, incluidas las puertas y medidas que el programa aplica a los cúbits.

En este artículo, aprenderá a representar visualmente algoritmos cuánticos con diagramas de circuitos cuánticos en Azure Quantum Development Kit (QDK) utilizando Visual Studio Code (VS Code) y Jupyter Notebook.

Para más información sobre los diagramas de circuitos cuánticos, consulte [Convenciones de circuito cuántico](#).

Requisitos previos

- La versión más reciente de [VS Code](#) o abra [VS Code para la Web](#).
- La versión más reciente de la [QDK extensión](#), [extensión de Python](#) y [Jupyter extensión](#) instalada en VS Code.
- La versión más reciente de la `qdk` biblioteca de Python con el complemento opcional `jupyter`.

Bash

```
python -m pip install --upgrade qdk[jupyter]
```

Visualización de circuitos cuánticos en VS Code

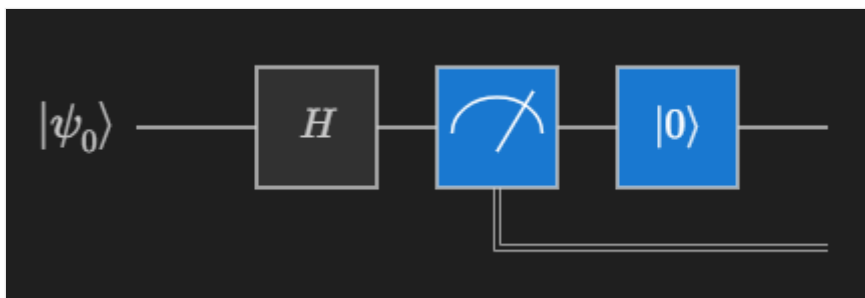
Para visualizar circuitos cuánticos de programas de Q# en VS Code, complete los pasos siguientes.

Visualización de diagramas de circuitos para un programa de Q#

1. Abra un archivo Q# en VS Codeo [cargue uno de los ejemplos cuánticos](#).
2. Elija el comando **Circuit** en el visor de código que precede a la operación de entrada.

```
File Edit Selection View Go Run Terminal Help < >
RandomBit.qs X
RandomBit.qs
1  /// # Summary
2  /// Generates a random bit in the form of `Zero` or `One` result.
3  @EntryPoint()
   Run | Histogram | Estimate | Debug | Circuit
4  operation GenerateRandomBit() : Result {
5      // Allocate a qubit
6      use q = Qubit();
7      // Set the qubit into uniform superposition of  $|0\rangle$  and  $|1\rangle$ 
8      H(q);
9      // Now the qubit has 50% chance of being measured as `One`
10     // and 50% chance of being measured as `Zero`.
11     // Measure and reset the qubit. Return the result.
12     MResetZ(q)
13 }
```

Aparece la ventana del circuito Q# y muestra el diagrama de circuitos del programa. Por ejemplo, el siguiente circuito corresponde a una operación que coloca un cúbit en un estado de superposición y, a continuación, mide el cúbit. El diagrama del circuito muestra un registro de cúbits que se inicializa en el estado $|0\rangle$. A continuación, se aplica una puerta Hadamard al cúbit, seguida de una [operación de medición](#), que se representa mediante un símbolo de medidor. En este caso, el resultado de la medida es cero.



💡 Sugerencia

En los archivos Q# y OpenQASM, seleccione un elemento en el diagrama de circuitos para resaltar el código que crea el elemento de circuito.

Visualización de diagramas de circuitos para operaciones individuales

Para visualizar el circuito cuántico para una operación individual en un archivo Q#, elija el comando **Circuit** en la lente de código que precede a la operación.

```

42
Circuit
43 operation PreparePhiPlus(register : Qubit[]) : Unit {
44     H(register[0]); // |+0>
45     CNOT(register[0], register[1]); // (|00> + |11>)/sqrt(2)
46 }
47
48
Circuit
49 operation PreparePhiMinus(register : Qubit[]) : Unit {
50     H(register[0]); // |+0>
51     Z(register[0]); // |-0>
52     CNOT(register[0], register[1]); // (|00> - |11>)/sqrt(2)
53 }
54

```

Visualización de diagramas de circuitos al depurar

Al usar el VS Code depurador en un programa de Q#, puede visualizar el circuito cuántico en función del estado del programa en el punto de interrupción actual del depurador.

1. Elija el comando **Depurar** en el CodeLens que antecede a su operación de punto de entrada.
2. En el panel **Ejecutar y depurar**, expanda la lista desplegable **Circuito cuántico** en el menú **VARIABLES**. Se abre el panel **QDK Circuito**, que muestra el circuito a medida que recorre el programa.
3. Establezca puntos de interrupción y recorra el código para ver cómo se actualiza el circuito a medida que se ejecuta el programa.

Circuitos cuánticos en Jupyter Notebook

En Jupyter Notebook, puede visualizar circuitos cuánticos con los módulos de Python `qdk.qsharp` y `qdk.widgets`. El `widgets` módulo proporciona un widget que representa un diagrama de circuitos cuánticos como una imagen SVG.

Visualización de diagramas de circuitos para una expresión de entrada

1. En VS Code, abra el menú **Ver** y elija **Paleta de comandos**.
2. Introduzca y seleccione **Crear: Nuevo Jupyter Notebook**.

3. En la primera celda del cuaderno, ejecute el código siguiente para importar el `qsharp` paquete.

Python

```
from qdk import qsharp
```

4. Cree una nueva celda y escriba el código de Q#. Por ejemplo, el código siguiente prepara un estado de Bell.

Q#

```
%%qsharp

// Prepare a Bell State.
operation BellState() : Unit {
    use register = Qubit[2];
    H(register[0]);
    CNOT(register[0], register[1]);
}
```

5. Para mostrar un circuito cuántico simple basado en el estado actual del programa, pase una expresión de punto de entrada a la `qsharp.circuit` función. Por ejemplo, el diagrama de circuito del código anterior muestra dos registros de cúbits que se inicializan en el estado $|0\rangle$. A continuación, se aplica una puerta Hadamard al primer cúbit. Por último, se aplica una puerta CNOT donde el primer cúbit es el control, representado por un punto, y el segundo cúbit es el destino, representado por una X.

Python

```
qsharp.circuit("BellState()")
```

Resultados

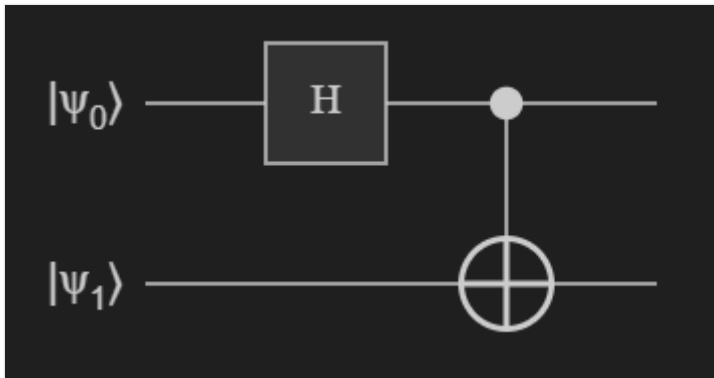
```
q_0    — H — • —
q_1    ————— X —
```

6. Para visualizar un circuito cuántico como una imagen SVG, use el `widgets` módulo. Cree una nueva celda y ejecute el código siguiente para visualizar el circuito que creó en la celda anterior.

Python

```
from qdk.widgets import Circuit
```

```
Circuit(qsharp.circuit("BellState()))
```



Visualización de diagramas de circuitos para operaciones con cúbits

Puede generar diagramas de circuitos de operaciones que toman cúbits o matrices de cúbits, como entrada. El diagrama muestra una conexión para cada cúbit de entrada, junto con cables para cúbits adicionales que se asignan dentro de la operación. Cuando la operación toma una matriz de cúbits (`Qubit[]`), el circuito muestra la matriz como un registro de 2 cúbits.

1. Agregue una nueva celda y, a continuación, copie y ejecute el siguiente código de Q#. Este código prepara un estado cat.

Q#

```
%%qsharp
```

```
operation PrepareCatState(register : Qubit[]) : Unit {  
    H(register[0]);  
    ApplyToEach(CNOT(register[0], _), register[1...]);  
}
```

2. Agregue una nueva celda y ejecute el código siguiente para visualizar el circuito de la `PrepareCatState` operación.

Python

```
Circuit(qsharp.circuit(operation="PrepareCatState"))
```

Diagramas de circuitos para circuitos dinámicos

Los diagramas de circuitos se generan ejecutando la lógica clásica dentro de un programa de Q# y realizando un seguimiento de todas las puertas asignadas y aplicadas. Los bucles y condicionales se admiten cuando solo se ocupan de valores clásicos.

Sin embargo, los programas que contienen bucles y expresiones condicionales que usan resultados de medición de cúbits son más complicados de representar con un diagrama de circuitos. Por ejemplo, considere la siguiente expresión:

Q#

```
if (M(q) == One) {  
    X(q)  
}
```

Esta expresión no se puede representar con un diagrama de circuito sencillo porque las puertas están condicionales a un resultado de medición. Los circuitos con puertas que dependen de los resultados de medición se denominan circuitos dinámicos.

Puede generar diagramas para circuitos dinámicos ejecutando el programa en el simulador cuántico y trazando las puertas a medida que se aplican. Esto se denomina modo de seguimiento porque los cúbits y las puertas se rastrean a medida que se realiza la simulación.

La desventaja de los circuitos rastreados es que solo capturan el resultado de la medición y las aplicaciones de puerta consecuentes para una sola simulación. En el ejemplo anterior, si el resultado de la medición es `Zero`, la `X` puerta no se encuentra en el diagrama. Si vuelve a ejecutar la simulación, es posible que obtenga un circuito diferente.

Realización de experimentos de ejecución prolongada en Azure Quantum

Al enviar un programa cuántico a Azure Quantum, el trabajo se carga en la cuenta de Azure Storage que configuró en el área de trabajo en espera de ejecutarse en el proveedor de QPU que ha seleccionado. Cuando el programa cuántico se encuentra a continuación en la cola del proveedor correspondiente, Azure Quantum descarga el programa y lo envía al proveedor, pasando de *esperar* a *que se ejecute* el estado. Para más información sobre el estado de los trabajos de Azure Quantum, consulte [Supervisión de trabajos](#).

Si el programa cuántico requiere muchas iteraciones, cada una se envía como un único trabajo. Para algunos experimentos cuánticos, como algoritmos de variación (VQE) y algoritmos de optimización (QAOA), es habitual ejecutarse durante horas o días, en particular cuando los tiempos de cola de QPU son altos y los programas requieren muchas iteraciones.

Consulte la sugerencia siguiente para ejecutar experimentos de tiempo de ejecución largo en Azure Quantum.

Programa con pocos bucles

Si el programa no requiere muchos bucles y el tiempo de cola de QPU es alto, puede esperar un par de horas y enviar el trabajo más adelante, o enviar el programa a un proveedor diferente de los disponibles en Azure Quantum.

Sesiones

Una sesión es una agrupación lógica de cualquier combinación de uno o más trabajos en un único target. Las sesiones permiten organizar varios trabajos de computación cuántica con la capacidad de ejecutar código clásico entre trabajos cuánticos. Podrá ejecutar algoritmos complejos para organizar y realizar un seguimiento mejor de los trabajos de computación cuántica individuales.

Cada proveedor de hardware cuántico define su propia heurística para administrar mejor la priorización de trabajos dentro de una sesión. En algunos casos, los trabajos enviados dentro de una sesión tienen prioridad en la cola de ese target.

Para obtener más información, consulte [Introducción a sesiones](#) y [Administración de sesiones](#).

Desarrollo local

La instalación de la extensión kit de desarrollo de Azure Quantum (QDK) para Visual Studio Code en el equipo local proporciona compatibilidad con cuadernos de Jupyter Notebook, Python y Q# archivos. Puede desarrollar aplicaciones de computación cuántica en su IDE y lenguaje preferidos y ejecutarlas en simuladores cuánticos y hardware cuántico mediante el servicio Azure Quantum.

Algunos escenarios en los que puede preferir un entorno local:

- Tiene un entorno personalizado o herramientas preferidas que no están disponibles en línea.
- Necesita el control de código fuente en el proyecto.
- Está trabajando con un proyecto de varios archivos.

Para obtener más información, consulte [Introducción al QDK](#).

Cuadernos de Jupyter Notebook con Google Colab

[Google Colab](#) es una herramienta gratuita basada en la nube que ofrece Google Research que permite a los usuarios escribir y ejecutar código de Python en sus exploradores web. Google Colab se basa en el código abierto de Jupyter, y básicamente le permite crear y compartir archivos sin tener que descargar ni instalar nada.

Para empezar a usar cuadernos de Google Colab para ejecutar los programas cuánticos en Azure Quantum, siga estos pasos:

1. **Agregue** una nueva celda en el cuaderno y **copie** el código siguiente para instalar los paquetes de Python necesarios:

Python

```
!pip install -U azure-quantum
!pip install -U azure-quantum[qiskit]
```

1. Haga clic en Tiempo de ejecución y seleccione **Reiniciar tiempo de ejecución**.
2. Para acceder al área de trabajo de Azure Quantum, debe **autenticarse**. Por ejemplo, recibirá un mensaje como el siguiente:

```
WARNING:azure.identity._internal.interactive:InteractiveBrowserCredential.get_token
failed: Failed to open a browser.
```

Para iniciar sesión, use un **explorador** web para abrir la página <https://microsoft.com/devicelogin> y escriba el código **ET8BM63F3** para autenticarse.

3. Una vez que haya pasado por el proceso de autenticación, debería poder ejecutar todas las celdas como si lo hiciera desde cuadernos hospedados en Azure Quantum o desde un entorno local.

Last updated on 16/10/2025

Tutorial: Implementación de un generador cuántico de números aleatorios en Q#

En este tutorial, aprenderá a escribir un programa cuántico básico en que Q# aproveche la naturaleza de la mecánica cuántica para generar un número aleatorio.

En este tutorial, aprenderá lo siguiente:

- ✓ Cree un Q# programa.
- ✓ Revise los componentes principales de un Q# programa.
- ✓ Definir la lógica de un problema.
- ✓ Combinar operaciones clásicas y cuánticas para resolver un problema.
- ✓ Trabajar con bits cuánticos y con la superposición para crear un generador de números aleatorios cuántico

Sugerencia

Si desea acelerar el recorrido de la computación cuántica, consulte [Código con Azure Quantum](#), una característica única del [sitio web de Microsoft Quantum](#). Aquí puede ejecutar ejemplos integrados Q# o sus propios Q# programas, generar código nuevo Q# a partir de las indicaciones, abrir y ejecutar el código en [VS Code para la Web](#) con un solo clic y formular preguntas de Copilot sobre la computación cuántica.

Requisitos previos

- Para ejecutar el ejemplo de código en [Copilot en Azure Quantum](#):
 - Una cuenta de correo electrónico de Microsoft (MSA).
- Para desarrollar y ejecutar el ejemplo de código en Visual Studio Code:
 - La versión más reciente de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
 - La versión más reciente de la Quantum Development Kit de Azure. Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).
 - Si desea usar Jupyter Notebook, también debe instalar las extensiones [de Python](#) y [Jupyter](#) y la biblioteca de Python más reciente `qdk` con el `jupyter` adicional. Para ello, abra un terminal y ejecute el siguiente comando:

```
Bash
```

```
pip install --upgrade qdk[jupyter]
```

Definir el problema

Los equipos clásicos no producen números aleatorios, sino números pseudoaleatorios. Un generador de números pseudoaleatorios genera una secuencia determinista de números basada en algún valor inicial, denominado semilla. Para aproximarse de mejor forma a los valores aleatorios, este valor de inicialización suele ser la hora actual del reloj de la CPU.

Por otro lado, los equipos cuánticos pueden generar números verdaderamente aleatorios. Esto se debe a que la medición de un cúbit en superposición es un proceso probabilístico. El resultado de la medida es aleatorio y no hay ninguna manera de predecir el resultado. Este es el principio básico de los generadores de números aleatorios cuánticos.

Un cúbit es una unidad de información cuántica que puede estar en superposición. Cuando se mide, un cúbit solo puede estar en el estado 0 o en el estado 1. Sin embargo, antes de la medición, el estado del cúbit representa la probabilidad de leer un 0 o un 1 con una medida.

Para comenzar, tome un cúbit en estado base, por ejemplo, cero. El primer paso del generador de números aleatorios es usar una operación Hadamard para colocar el cúbit en una superposición igual. La medición de este estado da como resultado un cero o uno con una probabilidad del 50 % de cada resultado, un bit realmente aleatorio.

No hay forma de saber qué obtendrá después de la medición del cúbit en superposición y el resultado es un valor diferente cada vez que se invoca el código. ¿Pero cómo puede usar este comportamiento para generar números aleatorios mayores?

Supongamos que repetimos el proceso cuatro veces, generando así esta secuencia de dígitos binarios:

0, 1, 1, 0

Si concatenamos, o combinamos, estos bits en una cadena de bits, podremos formar un número más grande. En este ejemplo, la secuencia de bits 0110 equivale a seis decimal.

$$0110_{\text{binario}} \equiv 6_{\text{decimal}}$$

Si repite este proceso muchas veces, puede combinar varios bits para formar un número grande. Con este método, puede crear un número para usarlo como contraseña segura, ya que puede estar seguro de que ningún hacker podría determinar los resultados de la secuencia de medidas.

Definición de la lógica del generador de números aleatorios

Vamos a describir cuál debe ser la lógica de un generador de números aleatorios:

1. Definir `max` como el número máximo que quiere generar.
2. Definir el número de bits aleatorios que necesita generar. Para ello, se calcula el número de bits, `nBits`, debe expresar enteros hasta `max`.
3. Generar una cadena de bits aleatorios que tenga una longitud de `nBits`.
4. Si la cadena de bits representa un número mayor que `max`, hay que volver al paso tres.
5. Si no, el proceso habrá finalizado. Devolver el número generado como un entero.

Como ejemplo, vamos a establecer `max` en 12. Es decir, 12 es el número más grande que desea usar como contraseña.

Necesita $\lfloor \ln(12)/\ln(2) + 1 \rfloor$ o 4 bits que representen un número entre cero y 12. Podemos usar la función `BitSizeI` integrada, que toma cualquier entero y devuelve el número de bits necesarios para representarlo.

Supongamos que generamos la cadena de bits `1101` *binario*, que equivale a `13` *decimal*. Como 13 es mayor que 12, repetimos el proceso.

Ahora, generamos la cadena de bits `0110` *binario*, que equivale a `6` *decimal*. Dado que 6 es menor que 12, el proceso se ha completado.

El generador de números aleatorios cuánticos devolverá el número 6 como contraseña. En la práctica, establezca un número más grande como máximo, ya que los números inferiores son fáciles de descifrar simplemente probando todas las contraseñas posibles. De hecho, para aumentar la dificultad de adivinar o descifrar la contraseña, puede usar código ASCII para convertir binario en texto y generar una contraseña mediante números, símbolos y letras en mayúsculas y minúsculas mezcladas.

Escritura de un generador de bits aleatorio

El primer paso es escribir una Q# operación que genera un bit aleatorio. Esta operación será uno de los bloques de creación del generador de números aleatorios.

Q#

```
operation GenerateRandomBit() : Result {
    // Allocate a qubit.
    use q = Qubit();

    // Set the qubit into superposition of 0 and 1 using the Hadamard
    H(q);

    // At this point the qubit `q` has 50% chance of being measured in the
    // |0> state and 50% chance of being measured in the |1> state.
```

```
// Measure the qubit value using the `M` operation, and store the
// measurement value in the `result` variable.
let result = M(q);

// Reset qubit to the  $|0\rangle$  state.
// Qubits must be in the  $|0\rangle$  state by the time they are released.
Reset(q);

// Return the result of the measurement.
return result;
}
```

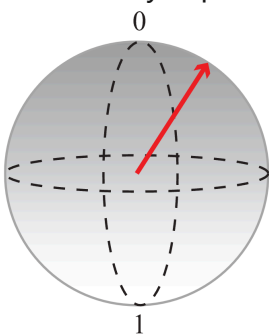
Ahora, eche un vistazo al nuevo código.

- Defina la operación `GenerateRandomBit`, que no toma ninguna entrada y genera un valor de tipo `Result`. El tipo `Result` representa el resultado de una medición y puede tener dos valores posibles: `Zero` o `One`.
- Asigne un solo cúbit con la `use` palabra clave. Cuando se asigna, un cúbit siempre está en el estado $|0\rangle$.
- Use la `H` operación para colocar el cúbit en una superposición igual.
- La operación se usa `M` para medir el cúbit, devolver el valor medido (`Zero` o `One`).
- Use la `Reset` operación para restablecer el cúbit al estado $|0\rangle$.

Al colocar el cúbit en superposición con la operación `H` y medirlo con la operación `M`, el resultado será un valor diferente cada vez que se invoque el código.

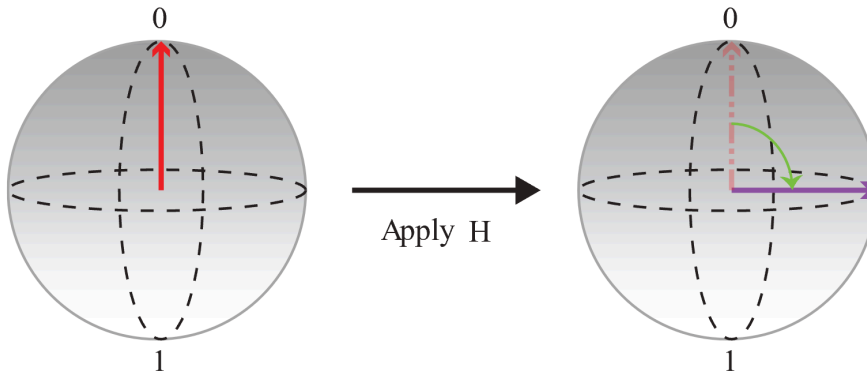
Visualización del Q# código con la esfera bloch

En la esfera Bloch, el polo norte representa el valor clásico 0 y el polo sur representa el valor clásico 1. Cualquier superposición se puede representar mediante un punto en la esfera (representado con una flecha). Cuanto más cerca esté el extremo de la flecha a un polo, mayor será la probabilidad de que el cúbit caiga en el valor clásico asignado a ese polo cuando se mida. Por ejemplo, el estado de cúbit representado por la flecha de la ilustración siguiente tiene una mayor probabilidad de proporcionar el valor 0 si lo mide.

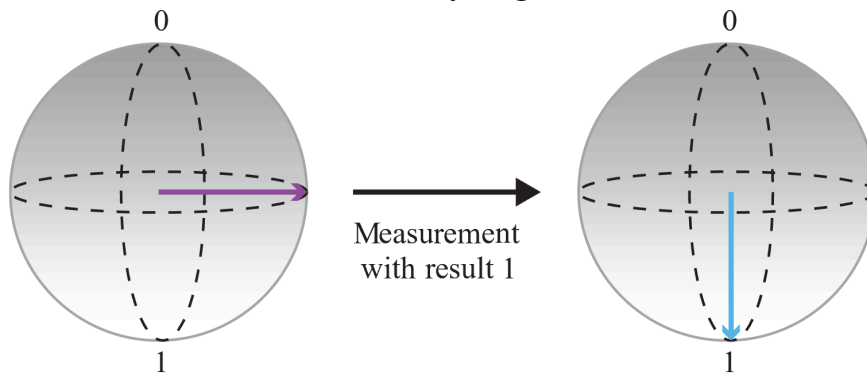


Puede usar esta representación para visualizar lo que está haciendo el código:

1. En primer lugar, comience con un cúbit inicializado en el estado $|0\rangle$ y aplique una H operación para crear una superposición igual en la que las probabilidades de 0 y 1 son las mismas.



2. A continuación, se mide el cúbit y se guarda el resultado:



Dado que el resultado de la medición es aleatorio y las probabilidades de medir 0 y 1 son las mismas, se ha obtenido un bit completamente aleatorio. Puede llamar a esta operación varias veces para crear enteros. Por ejemplo, si llama a la operación tres veces para obtener tres bits aleatorios, puede crear números de 3 bits aleatorios (es decir, un número aleatorio entre 0 y 7).

Escribir un generador de números aleatorios completo

1. En primer lugar, debe importar los espacios de nombres necesarios de la Q# biblioteca estándar al programa. El Q# compilador carga automáticamente muchas funciones y operaciones comunes, pero para el generador de números aleatorios completo, necesita algunas funciones y operaciones adicionales de dos Q# espacios de nombres: `Std.Math` y `Std.Convert`.

Q#

```
import Std.Convert.*;  
import Std.Math.*;
```

2. A continuación, defina la `GenerateRandomNumberInRange` operación. Esta operación llama repetidamente a la operación `GenerateRandomBit` para generar una cadena de bits.

Q#

```
/// Generates a random number between 0 and `max`.
operation GenerateRandomNumberInRange(max : Int) : Int {
    // Determine the number of bits needed to represent `max` and store it
    // in the `nBits` variable. Then generate `nBits` random bits which will
    // represent the generated random number.
    mutable bits = [];
    let nBits = BitSizeI(max);
    for idxBit in 1..nBits {
        bits += [GenerateRandomBit()];
    }
    let sample = ResultArrayAsInt(bits);

    // Return random number if it is within the requested range.
    // Generate it again if it is outside the range.
    return sample > max ? GenerateRandomNumberInRange(max) | sample;
}
```

Paremos un momento a revisar el nuevo código.

- Debe calcular el número de bits necesarios para expresar enteros hasta `max`. La `BitSizeI` función del `Std.Math` espacio de nombres convierte un entero en el número de bits necesarios para representarlo.
- La operación `SampleRandomNumberInRange` usa un bucle `for` para generar números aleatorios hasta llegar a uno que sea igual o menor que `max`. El bucle `for` funciona exactamente igual que un bucle `for` en otros lenguajes de programación.
- La variable `bits` es una variable mutable. Una variable mutable es aquella que puede cambiar durante el cálculo. Para cambiar el valor de una variable mutable, usamos la directiva `set`.
- La `ResultArrayAsInt` función, del espacio de nombres predeterminado `Std.Convert`, convierte la cadena de bits en un entero positivo.

3. Por último, agregue un punto de entrada al programa. De forma predeterminada, el Q# compilador busca una `Main` operación e inicia el procesamiento allí. Llama a la `GenerateRandomNumberInRange` operación para generar un número aleatorio entre 0 y 100.

Q#

```
operation Main() : Int {
    let max = 100;
    Message($"Sampling a random number between 0 and {max}: ");

    // Generate random number in the 0..max range.
    return GenerateRandomNumberInRange(max);
}
```


La directiva `let` declara variables que no cambian durante el cálculo. Aquí se define el valor máximo como 100.

Para obtener más información sobre la `Main` operación, vea [Puntos](#) de entrada.

4. El código completo del generador de números aleatorios es el siguiente:

Q#

```
import Std.Convert.*;
import Std.Math.*;

operation Main() : Int {
    let max = 100;
    Message($"Sampling a random number between 0 and {max}: ");

    // Generate random number in the 0..max range.
    return GenerateRandomNumberInRange(max);
}

/// Generates a random number between 0 and `max`.
operation GenerateRandomNumberInRange(max : Int) : Int {
    // Determine the number of bits needed to represent `max` and store it
    // in the `nBits` variable. Then generate `nBits` random bits which will
    // represent the generated random number.
    mutable bits = [];
    let nBits = BitSizeI(max);
    for idxBit in 1..nBits {
        bits += [GenerateRandomBit()];
    }
    let sample = ResultArrayAsInt(bits);

    // Return random number if it is within the requested range.
    // Generate it again if it is outside the range.
    return sample > max ? GenerateRandomNumberInRange(max) | sample;
}

operation GenerateRandomBit() : Result {
    // Allocate a qubit.
    use q = Qubit();

    // Set the qubit into superposition of 0 and 1 using a Hadamard operation
    H(q);

    // At this point the qubit `q` has 50% chance of being measured in the
    //  $|0\rangle$  state and 50% chance of being measured in the  $|1\rangle$  state.
    // Measure the qubit value using the `M` operation, and store the
    // measurement value in the `result` variable.
    let result = M(q);

    // Reset qubit to the  $|0\rangle$  state.
    // Qubits must be in the  $|0\rangle$  state by the time they are released.
    Reset(q);
}
```

```
// Return the result of the measurement.  
return result;  
}
```

Ejecución del programa generador de números aleatorios

Puede ejecutar el programa en [Copilot en Azure Quantum](#) y en Visual Studio Code como una aplicación independiente Q# o mediante un programa host de Python.

Copilot en Azure Quantum

Puede probar el Q# código con Copilot en Azure Quantum de forma gratuita: todo lo que necesita es una cuenta de correo electrónico de Microsoft (MSA). Para más información sobre Copilot en Azure Quantum, consulte [Exploración de Azure Quantum](#).

1. Abra [Copilot en Azure Quantum](#) en el explorador.
2. Copie y pegue el código siguiente en el editor de código.

Q#

```
import Std.Convert.*;  
import Std.Math.*;  
  
operation Main() : Int {  
    let max = 100;  
    Message($"Sampling a random number between 0 and {max}: ");  
  
    // Generate random number in the 0..max range.  
    return GenerateRandomNumberInRange(max);  
}  
  
/// # Summary  
/// Generates a random number between 0 and `max`.  
operation GenerateRandomNumberInRange(max : Int) : Int {  
    // Determine the number of bits needed to represent `max` and store it  
    // in the `nBits` variable. Then generate `nBits` random bits which  
    will  
    // represent the generated random number.  
    mutable bits = [];  
    let nBits = BitSizeI(max);  
    for idxBit in 1..nBits {  
        bits += [GenerateRandomBit()];  
    }  
    let sample = ResultArrayAsInt(bits);  
}
```

```

// Return random number if it is within the requested range.
// Generate it again if it is outside the range.
return sample > max ? GenerateRandomNumberInRange(max) | sample;
}

/// # Summary
/// Generates a random bit.
operation GenerateRandomBit() : Result {
    // Allocate a qubit.
    use q = Qubit();

    // Set the qubit into superposition of 0 and 1 using the Hadamard
    // operation `H`.
    H(q);

    // At this point the qubit `q` has 50% chance of being measured in the
    //  $|0\rangle$  state and 50% chance of being measured in the  $|1\rangle$  state.
    // Measure the qubit value using the `M` operation, and store the
    // measurement value in the `result` variable.
    let result = M(q);

    // Reset qubit to the  $|0\rangle$  state.
    // Qubits must be in the  $|0\rangle$  state by the time they are released.
    Reset(q);

    // Return the result of the measurement.
    return result;

    // Note that Qubit `q` is automatically released at the end of the
    block.
}

```

3. Seleccione el número de capturas que se van a ejecutar y seleccione **Ejecutar**.
4. Los resultados se muestran en el histograma y en los campos **Resultados**.
5. Seleccione **Explicar código** para preguntar a Copilot para explicarle el código.

💡 Sugerencia

Desde Copilot en Azure Quantum, puede abrir el programa en [VS Code para web](#) seleccionando el botón logotipo de VS Code en la esquina derecha del editor de código.

⚠️ Nota:

Este fragmento de código no se ejecuta actualmente en ningún hardware targetsde Azure Quantum disponible, ya que el invocable `ResultArrayAsInt` requiere una QPU con un [perfil de cálculo completo](#).

Contenido relacionado

Explore otros tutoriales de Q#:

- [El entrelazamiento](#) cuántico muestra cómo escribir un Q# programa que manipula y mide cúbits y muestra los efectos de la superposición y el entrelazamiento.
- [El algoritmo](#) de búsqueda de Grover muestra cómo escribir un Q# programa que usa el algoritmo de búsqueda de Grover.
- [Las transformaciones de Quantum Fourier](#) exploran cómo escribir un Q# programa que direcciona directamente los cúbits específicos.
- [Quantum Katas](#) [↗](#) son tutoriales autodirigido y ejercicios de programación dirigidos a enseñar los elementos de la computación cuántica y Q# la programación al mismo tiempo.

Last updated on 12/11/2025

Tutorial: Exploración del entrelazamiento cuántico con Q#

En este tutorial, escribirá un Q# programa que prepara dos cúbits en un estado cuántico específico, opera en los cúbits para entrelazarlos entre sí y realiza mediciones para demostrar los efectos de superposición y entrelazamiento. Cree el Q# programa por partes para introducir estados de cúbits, operaciones cuánticas y medidas.

Antes de empezar, revise los siguientes conceptos de computación cuántica:

- Los bits clásicos contienen un único valor binario como 0 o 1, pero los cúbits pueden estar en una superposición de los dos estados, 0 y 1. Cada estado de cúbit posible se describe mediante un conjunto de amplitudes de probabilidad.
- Al medir el estado de un cúbit, siempre obtendrá 0 o 1. La probabilidad de cada resultado viene determinada por las amplitudes de probabilidad que definen el estado de superposición cuando se realiza una medición.
- Se pueden entrelazar varios cúbits de forma que no se puedan describir independientemente entre sí. Al medir un cúbit en un par entrelazado, también obtendrá información sobre el otro cúbit sin medirlo.

En este tutorial, aprenderá a:

- ✓ Crear operaciones Q# para inicializar un cúbit en un estado deseado.
- ✓ Coloque un cúbit en un estado de superposición.
- ✓ Entrelazar un par de cúbits.
- ✓ Mida un cúbit y observe los resultados.

Sugerencia

Si desea acelerar el recorrido de la computación cuántica, consulte [Código con Azure Quantum](#), una característica única del [sitio web de Microsoft Quantum](#). Aquí puede ejecutar ejemplos integrados Q# o sus propios Q# programas, generar código nuevo Q# a partir de las indicaciones, abrir y ejecutar el código en [VS Code para la Web](#) con un solo clic y formular preguntas de Copilot sobre la computación cuántica.

Requisitos previos

Para ejecutar el ejemplo de código con [Copilot para Azure Quantum](#), debe tener una cuenta de correo electrónico de Microsoft (MSA).

Para más información sobre Copilot para Azure Quantum, consulte [Exploración de Azure Quantum](#).

Inicialización de un cúbit en un estado conocido

El primer paso es definir una Q# operación que inicializa un cúbit en un estado clásico deseado, ya sea 0 o 1. Esta operación mide un cúbit en un estado cuántico general, que devuelve un Q# `Result` valor de tipo de `Zero` o `One`. Si el resultado de la medida es diferente del estado deseado, la operación cambia el estado para que la operación devuelva el estado deseado 100% del tiempo.

Abra [Copilot para Azure Quantum](#), borre el código predeterminado y, a continuación, copie el código siguiente en la ventana del editor de código. No puede ejecutar este código por sí mismo porque aún no es un programa completo Q#.

Q#

```
operation SetQubitState(desired : Result, target : Qubit) : Unit {  
    if desired != M(target) {  
        X(target);  
    }  
}
```

En el ejemplo de código se presentan dos operaciones estándar Q# y `M` `X`, que transforman el estado de un cúbit.

Esta es una descripción detallada de cómo funciona la `SetQubitState` operación:

1. Toma dos parámetros: un `Result` parámetro de tipo denominado `desired` que representa el estado deseado para que el cúbit esté en (`Zero` o `One`) y un `Qubit` parámetro de tipo.
2. Realiza una operación de medición, `M` que mide el estado del cúbit (`Zero` o `One`) y compara el resultado con el valor que se pasa para `desired`.
3. Si el resultado de la medida no coincide con el valor de `desired`, se aplica una `X` operación al cúbit. Esta operación invierte el estado del cúbit para que se invierten las probabilidades de medición de `Zero` y `One`.

Escribir una operación de prueba para probar el estado de Bell

Para llamar a la operación `SetQubitState` en su programa Q#, cree otra operación denominada `Main`. Esta operación asigna dos cúbits, llama `SetQubitState` a para establecer el primer cúbit

en un estado conocido y, a continuación, mide los cúbits para ver los resultados.

Copie el código siguiente en la ventana del editor de código después de la `SetQubitState` operación.

Q#

```
operation Main() : (Int, Int, Int, Int) {
    mutable numOnesQ1 = 0;
    mutable numOnesQ2 = 0;
    let count = 1000;
    let initial = One;

    // allocate the qubits
    use (q1, q2) = (Qubit(), Qubit());
    for test in 1..count {
        SetQubitState(initial, q1);
        SetQubitState(Zero, q2);

        // measure each qubit
        let resultQ1 = M(q1);
        let resultQ2 = M(q2);

        // Count the number of 'Ones' returned:
        if resultQ1 == One {
            numOnesQ1 += 1;
        }
        if resultQ2 == One {
            numOnesQ2 += 1;
        }
    }

    // reset the qubits
    SetQubitState(Zero, q1);
    SetQubitState(Zero, q2);

    // Display the times that |0> is returned, and times that |1> is returned
    Message($"Q1 - Zeros: {count - numOnesQ1}");
    Message($"Q1 - Ones: {numOnesQ1}");
    Message($"Q2 - Zeros: {count - numOnesQ2}");
    Message($"Q2 - Ones: {numOnesQ2}");
    return (count - numOnesQ1, numOnesQ1, count - numOnesQ2, numOnesQ2 );
}
```

En el código, las variables `count` y `initial` se establecen en `1000` y `One` respectivamente. El primer cúbit se inicializa en `One` y cada cúbit se mide 1000 veces.

La `Main` operación hace lo siguiente:

1. Establece variables para el número de tomas (`count`) y el estado de cúbit inicial (`One`).

2. Llama a la instrucción `use` para inicializar dos cúbits.
3. Recorre en bucle los tiempos del experimento `count` .
4. En el bucle, llama `SetQubitState` a para establecer el valor especificado `initial` en el primer cúbit y, a continuación, llama `SetQubitState` de nuevo para establecer el segundo cúbit en el `Zero` estado.
5. En el bucle, aplica la `M` operación para medir cada cúbit y, a continuación, almacena el número de medidas de cada cúbit que devuelve `One`.
6. Una vez completado el bucle, llama `SetQubitState` de nuevo a para restablecer los cúbits a un estado conocido (`Zero`). Debe restablecer los cúbits que asigne con la sentencia `use`.
7. Llama a la `Message` función para imprimir los resultados en la ventana de salida.

Ejecución del código en Copilot para Azure Quantum

Antes de escribir código para superposición y entrelazamiento, pruebe el programa actual para ver la inicialización y medición de los cúbits.

Para ejecutar el código como un programa independiente, el Q# compilador de Copilot debe saber dónde iniciar el programa. Dado que no especificó un espacio de nombres, el compilador reconoce el punto de entrada predeterminado como la `Main` operación. Para obtener más información, vea [Proyectos y espacios de nombres implícitos](#).

El Q# programa ahora tiene el siguiente aspecto:

Q#

```
operation SetQubitState(desired : Result, target : Qubit) : Unit {
    if desired != M(target) {
        X(target);
    }
}

operation Main() : (Int, Int, Int, Int) {
    mutable numOnesQ1 = 0;
    mutable numOnesQ2 = 0;
    let count = 1000;
    let initial = One;

    // allocate the qubits
    use (q1, q2) = (Qubit(), Qubit());
    for test in 1..count {
        SetQubitState(initial, q1);
        SetQubitState(Zero, q2);

        // measure each qubit
```



```

let resultQ1 = M(q1);
let resultQ2 = M(q2);

// Count the number of 'Ones' returned:
if resultQ1 == One {
    numOnesQ1 += 1;
}
if resultQ2 == One {
    numOnesQ2 += 1;
}
}

// reset the qubits
SetQubitState(Zero, q1);
SetQubitState(Zero, q2);

// Display the times that |0> is returned, and times that |1> is returned
Message($"Q1 - Zeros: {count - numOnesQ1}");
Message($"Q1 - Ones: {numOnesQ1}");
Message($"Q2 - Zeros: {count - numOnesQ2}");
Message($"Q2 - Ones: {numOnesQ2}");
return (count - numOnesQ1, numOnesQ1, count - numOnesQ2, numOnesQ2 );
}

```

Copie y pegue el ejemplo de código completo en la ventana [Código de Copilot para Azure Quantum](#) , establezca el control deslizante para el número de capturas en "1" y, a continuación, elija **Ejecutar**. Los resultados se muestran en el histograma y en los campos **Resultados** .

Resultados

```

Q1 - Zeros: 0
Q1 - Ones: 1000
Q2 - Zeros: 1000
Q2 - Ones: 0

```

El programa aún no modifica los estados de cúbit, por lo que la medición del primer cúbit siempre devuelve **One** y el segundo cúbit siempre devuelve **Zero**.

Si cambia el valor de **initial** a **Zero** y vuelve a ejecutar el programa, entonces el primer cúbit también siempre devuelve **Zero**.

Resultados

```

Q1 - Zeros: 1000
Q1 - Ones: 0
Q2 - Zeros: 1000
Q2 - Ones: 0

```

Colocar un cúbit en un estado de superposición

Actualmente, los cúbits del programa están en un estado clásico, ya sea 1 o 0, al igual que bits en un equipo normal. Para entrelazar los cúbits, primero debe colocar uno de los cúbits en un estado de superposición igual. La medición de un cúbit en un estado de superposición igual tiene un 50% de probabilidad de dar como resultado `Zero` y un 50% de probabilidad de dar como resultado `One`.

Para colocar un cúbit en un estado de superposición, use la operación de Hadamard `Q# H`. La `H` operación convierte un cúbit que está en un estado puro `Zero` o `One` en un tipo de estado medio entre `Zero` y `One`.

Modifique el código de la operación `Main`. Restablezca el valor inicial en `One` e inserte una línea para la `H` operación:

Q#

```
for test in 1..count {  
    use (q1, q2) = (Qubit(), Qubit());  
    for test in 1..count {  
        SetQubitState(initial, q1);  
        SetQubitState(Zero, q2);  
  
        H(q1); // Add the H operation after initialization and before measurement  
  
        // measure each qubit  
        let resultQ1 = M(q1);  
        let resultQ2 = M(q2);  
        ...  
    }  
}
```

Vuelva a ejecutar el programa. Dado que el primer cúbit está en una superposición igual al medirlo, se acerca a un resultado de 50/50 para `Zero` y `One`. Por ejemplo, la salida tiene un aspecto similar al siguiente:

Resultados

```
Q1 - Zeros: 523  
Q1 - Ones: 477  
Q2 - Zeros: 1000  
Q2 - Ones: 0
```

Cada vez que ejecute el programa, los resultados del primer cúbit varían ligeramente, pero están cerca de 50% `One` y 50% `Zero`, mientras que los resultados del segundo cúbit siguen siendo siempre `Zero`.

Inicialice el primer cúbit en `Zero` en vez de `One` y ejecute el programa nuevamente. Obtiene resultados similares porque la operación `H` convierte tanto un estado puro `Zero` como un estado puro `One` en un estado de igual superposición.

ⓘ **Nota:**

Para ver cómo los resultados de la superposición varían en función de la distribución de las tomas, mueva el control deslizante en Copilot para Azure Quantum y aumente el número de tomas.

Entrelazar dos qubits

Los cúbits entrelazados se correlacionan de modo que no se puedan describir independientemente entre sí. Al medir el estado de un cúbit entrelazado, también conoce el estado del otro cúbit sin medirlo. En este tutorial se usa un ejemplo con dos cúbits entrelazados, pero también puede entrelazar tres o más cúbits.

Para crear un estado entrelazado, use la operación `Q# CNOT`, o Controlled-NOT. Cuando se aplica `CNOT` a dos cúbits, un cúbit es el cúbit de control y el otro es el cúbit de destino. Si el estado del cúbit de control es `One`, la `CNOT` operación invierte el estado del cúbit de destino. De lo contrario, `CNOT` no hace nada con los cúbits.

Agregue la operación `CNOT` al programa inmediatamente después de la operación `H`. El programa completo tiene este aspecto:

Q#

```
operation SetQubitState(desired : Result, target : Qubit) : Unit {
    if desired != M(target) {
        X(target);
    }
}

operation Main() : (Int, Int, Int, Int) {
    mutable numOnesQ1 = 0;
    mutable numOnesQ2 = 0;
    let count = 1000;
    let initial = Zero;

    // allocate the qubits
    use (q1, q2) = (Qubit(), Qubit());
    for test in 1..count {
        SetQubitState(initial, q1);
        SetQubitState(Zero, q2);
```

```

H(q1);
CNOT(q1, q2);      // Add the CNOT operation after the H operation

// measure each qubit
let resultQ1 = M(q1);
let resultQ2 = M(q2);

// Count the number of 'Ones' returned:
if resultQ1 == One {
    numOnesQ1 += 1;
}
if resultQ2 == One {
    numOnesQ2 += 1;
}
}

// reset the qubits
SetQubitState(Zero, q1);
SetQubitState(Zero, q2);

// Display the times that |0> is returned, and times that |1> is returned
Message($"Q1 - Zeros: {count - numOnesQ1}");
Message($"Q1 - Ones: {numOnesQ1}");
Message($"Q2 - Zeros: {count - numOnesQ2}");
Message($"Q2 - Ones: {numOnesQ2}");
return (count - numOnesQ1, numOnesQ1, count - numOnesQ2, numOnesQ2 );
}

```

Ejecute el programa y vea la salida. Tus resultados varían ligeramente cada vez que ejecutas el programa.

Resultados

```

Q1 - Zeros: 502
Q1 - Ones: 498
Q2 - Zeros: 502
Q2 - Ones: 498

```

Las estadísticas del primer cúbit siguen mostrando aproximadamente un 50% de posibilidad de medir ambos **One** y **Zero**, pero ahora los resultados de medición del segundo cúbit no siempre son **Zero**. Cada cúbit tiene el mismo número de **Zero** resultados y **One** resultados. El resultado de la medición del segundo cúbit siempre es el mismo que el resultado del primer cúbit porque los dos cúbits están entrelazados. Si el primer cúbit se mide como **zero**, el cúbit entrelazado también debe ser **Zero**. Si el primer cúbit se mide como **One**, el cúbit entrelazado también debe ser **One**.

Contenido relacionado

Explore otros tutoriales de Q#:

- [El algoritmo](#) de búsqueda de Grover muestra cómo escribir un Q# programa que usa el algoritmo de búsqueda de Grover.
- [Quantum Fourier Transform](#) explora cómo escribir un Q# programa que direcciona directamente los cúbits específicos.
- [Quantum Katas](#) [↗](#) son tutoriales autodirigidos y ejercicios de programación que enseñan los elementos de la computación cuántica y Q# programación al mismo tiempo.

Last updated on 03/12/2025

Tutorial: Implementación del algoritmo de búsqueda de Grover en Q#

Artículo • 13/01/2025

En este tutorial, implementará el algoritmo de Grover en para resolver problemas basados en Q# búsquedas. Para obtener una explicación detallada de la teoría subyacente al algoritmo de Grover, consulte la [Teoría del algoritmo de Grover](#).

En este tutorial ha:

- ✓ Definición del algoritmo de Grover para un problema de búsqueda
- ✓ Implementación del algoritmo de Grover en Q#

💡 Sugerencia

Si desea acelerar el recorrido de la computación cuántica, consulte [Código con Azure Quantum](#), una característica única del [sitio web](#) de Azure Quantum. Aquí puede ejecutar ejemplos integrados Q# o sus propios Q# programas, generar código nuevo Q# a partir de las indicaciones, abrir y ejecutar el código en [VS Code para la Web](#) con un solo clic y hacer preguntas a Copilot sobre la computación cuántica.

Requisitos previos

- Para ejecutar el ejemplo de código en [Copilot en Azure Quantum](#):
 - Una cuenta de correo electrónico de Microsoft (MSA).
- Para desarrollar y ejecutar el ejemplo de código en Visual Studio Code:
 - La versión más reciente de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
 - La versión más reciente de la Quantum Development Kit de Azure. Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

Definir el problema

El algoritmo de Grover es uno de los algoritmos más famosos de la computación cuántica. El tipo de problema que resuelve suele denominarse "búsqueda de una base de datos", pero es más preciso pensarlo en términos del *problema* de búsqueda.

Cualquier problema de búsqueda se puede formular matemáticamente con una función abstracta $f(x)$ que acepta elementos de búsqueda x . Si el elemento x es una solución para el problema de búsqueda, entonces $f(x) = 1$. Si el elemento x no es una solución, $f(x) = 0$. El problema de búsqueda consiste en buscar cualquier elemento x_0 de manera que $f(x_0) = 1$.

Por lo tanto, puede formular cualquier problema de búsqueda como: dada una función clásica $f(x) : \{0, 1\}^n \rightarrow \{0, 1\}$, donde n es el tamaño de bits del espacio de búsqueda, busque una entrada x_0 para la que $f(x_0) = 1$.

Para implementar el algoritmo de Grover para resolver un problema de búsqueda, debe:

1. Transformar el problema en una **tarea de Grover**. Por ejemplo, suponga que quiere encontrar los factores de un entero M mediante el algoritmo de Grover. Puede transformar el problema de factorización de enteros en una tarea de Grover mediante la creación de una función

$$f_M(x) = 1[r],$$

en la que $1[r] = 1$ si $r = 0$ y $1[r] = 0$ si $r \neq 0$, y r es el resto de M/x . De este modo, los enteros x_i que hacen $f_M(x_i) = 1$ son los factores de M y ha transformado el problema en una tarea de Grover.

2. Implementar la función de la tarea de Grover como un oráculo cuántico. Para implementar el algoritmo de Grover, debe implementar la función $f(x)$ de la tarea de Grover como un [oráculo cuántico](#).
3. Usar el algoritmo de Grover con el oráculo para resolver la tarea. Una vez que tenga un oráculo cuántico, puede conectarlo a la implementación del algoritmo de Grover para resolver el problema e interpretar la salida.

Algoritmo de Grover

Supongamos que hay $N = 2^n$ elementos aptos para el problema de búsqueda y se indexan asignando a cada elemento un entero de 0 a $N - 1$. Pasos generales del algoritmo

1. Comienza con un registro de n cúbits inicializados en el estado $|0\rangle$.
2. Prepara el registro en una superposición uniforme aplicando H a cada cúbit del registro:

$$|\psi\rangle = \frac{1}{N^{1/2}} \sum_{x=0}^{N-1} |x\rangle$$

3. Aplique las siguientes operaciones al registro N_{optimal} veces:
 - a. El oráculo de fase O_f que aplica un cambio de fase condicional de -1 para los elementos de la solución.
 - b. Aplica H a cada cúbit del registro.
 - c. Aplica $-O_0$, un cambio de fase condicional de -1 a cada estado de base computacional, excepto $|0\rangle$.
 - d. Aplica H a cada cúbit del registro.
4. Mide el registro para obtener el índice de un elemento que es una solución con una probabilidad muy alta.
5. Compruebe el elemento para ver si es una solución válida. Si no es así, vuelve a empezar.

Escribir el código para el algoritmo de Grover en Q#

En esta sección se describe cómo implementar el algoritmo en Q#. Hay algunas cosas que se deben tener en cuenta al implementar el algoritmo de Grover. Debe definir el estado marcado, cómo reflejarlo y cuántas iteraciones ejecutar el algoritmo. También debe definir el oráculo que implementa la función de la tarea de Grover.

Definir el estado marcado

En primer lugar, debe definir qué entrada está intentando encontrar en la búsqueda. Para ello, escriba una operación que aplique los pasos b, c y d del [algoritmo](#) de Grover.

Juntos, estos pasos también se conocen como el **operador** de difusión de Grover $-H^{\otimes n}O_0H^{\otimes n}$.

```
Q#

operation ReflectAboutMarked(inputQubits : Qubit[]) : Unit {
    Message("Reflecting about marked state...");
    use outputQubit = Qubit();
    within {
        // We initialize the outputQubit to (|0> - |1>) / sqrt(2), so that
        // toggling it results in a (-1) phase.
        X(outputQubit);
        H(outputQubit);
        // Flip the outputQubit for marked states.
        // Here, we get the state with alternating 0s and 1s by using the X
        // operation on every other qubit.
        for q in inputQubits[...2...] {
            X(q);
        }
    } apply {
        Controlled X(inputQubits, outputQubit);
    }
}
```

La `ReflectAboutMarked` operación refleja el estado base marcado por ceros y unos. Para ello, aplica el operador de difusión de Grover a los cúbits de entrada. La operación usa un cúbit auxiliar, `outputQubit` que se inicializa en el estado $|-\rangle = \frac{1}{\sqrt{2}}(|-\rangle|0\rangle|1\rangle)$ aplicando las puertas X y H . A continuación, la operación aplica la puerta de X a todos los demás cúbits del registro, que voltean el estado del cúbit. Por último, aplica la puerta X controlada al cúbit auxiliar y a los cúbits de entrada. Esta operación invierte el cúbit auxiliar si y solo si todos los cúbits de entrada están en el estado $|1\rangle$, que es el estado marcado.

Definir el número de iteraciones óptimas

La búsqueda de Grover tiene un número óptimo de iteraciones que produce la mayor probabilidad de medir una salida válida. Si el problema tiene $N = 2^n$ posibles asignaciones de variables y M de ellas son soluciones al problema, el número óptimo de iteraciones es:

$$N_{\text{optimal}} \approx \frac{\pi}{4} \sqrt{\frac{N}{M}}$$

Continuando iterando más allá del número óptimo de iteraciones comienza a reducir esa probabilidad hasta alcanzar una probabilidad de éxito casi cero en la iteración $2N_{\text{optimal}}$. Después de eso, la probabilidad vuelve a crecer hasta $3N_{\text{optimal}}$, y así sucesivamente.

En las aplicaciones prácticas, normalmente no se sabe cuántas soluciones tiene el problema antes de resolverlo. Una estrategia eficaz para controlar este problema es "suponer" el número de soluciones M aumentando progresivamente en potencias de dos (es decir, 1, 2, 4, 8, 16, ..., 2^n). Una de estas suposiciones estará lo suficientemente cerca como para que el algoritmo todavía encuentre la solución con un número promedio $\sqrt{\frac{N}{M}}$.

La función siguiente Q# calcula el número óptimo de iteraciones para un número determinado de cúbits en un registro.

```
Q#

function CalculateOptimalIterations(nQubits : Int) : Int {
    if nQubits > 63 {
        fail "This sample supports at most 63 qubits.";
    }
    let nItems = 1 <<< nQubits; // 2^nQubits
    let angle = ArcSin(1. / Sqrt(IntAsDouble(nItems)));
    let iterations = Round(0.25 * PI() / angle - 0.5);
    return iterations;
}
```

La `CalculateOptimalIterations` función usa la fórmula anterior para calcular el número de iteraciones y, a continuación, la redondea al entero más cercano.

Definición de la operación de Grover

La Q# operación para el algoritmo de búsqueda de Grover tiene tres entradas:

- Número de cúbits, `nQubits : Int`, en el registro de cúbits. Este registro codificará la solución provisional al problema de búsqueda. Después de la operación, se medirá.
- Número de iteraciones óptimas, `iterations : Int`.
- Una operación, `phaseOracle : Qubit[] => Unit` : `Result[]`, que representa el oráculo de fase para la tarea de Grover. Esta operación aplica una transformación unitaria a un registro de cúbit genérico.

```
Q#

operation GroverSearch( nQubits : Int, iterations : Int, phaseOracle : Qubit[] => Unit) : Result[] {

    use qubits = Qubit[nQubits];
    PrepareUniform(qubits);

    for _ in 1..iterations {
        phaseOracle(qubits);
        ReflectAboutUniform(qubits);
    }

    // Measure and return the answer.
    return MResetEachZ(qubits);
}
```

La `GroverSearch` operación inicializa un registro de n cúbits en el estado $|0\rangle$, prepara el registro en una superposición uniforme y, a continuación, aplica el algoritmo de Grover para el número especificado de iteraciones. La propia búsqueda consta de reflejar repetidamente el estado marcado y el estado inicial, que puede escribir en Q# como bucle for. Por último, mide el registro y devuelve el resultado.

El código usa tres operaciones auxiliares: `PrepareUniform`, `ReflectAboutUniform` y `ReflectAboutAllOnes`.

Dado un registro en el estado de todos los ceros, la `PrepareUniform` operación prepara una superposición uniforme en todos los estados de base.

```
Q#

operation PrepareUniform(inputQubits : Qubit[]) : Unit is Adj + Ctl {
    for q in inputQubits {
        H(q);
    }
}
```

La operación "ReflectAboutAllOnes" refleja el estado de todos los usuarios.

Q#

```
operation ReflectAboutAllOnes(inputQubits : Qubit[]) : Unit {  
    Controlled Z(Most(inputQubits), Tail(inputQubits));  
}
```

La operación `ReflectAboutUniform` refleja el estado uniforme de la superposición. En primer lugar, transforma la superposición uniforme en todo cero. A continuación, transforma el estado todo cero en todos los usuarios. Por último, refleja sobre el estado de todos los usuarios. La operación se llama `ReflectAboutUniform` porque se puede interpretar geoméricamente como una reflexión en el espacio vectorial del estado uniforme de la superposición.

Q#

```
operation ReflectAboutUniform(inputQubits : Qubit[]) : Unit {  
    within {  
        Adjoint PrepareUniform(inputQubits);  
        // Transform the all-zero state to all-ones  
        for q in inputQubits {  
            X(q);  
        }  
    } apply {  
        ReflectAboutAllOnes(inputQubits);  
    }  
}
```

Ejecución del código final

Ahora, tiene todos los ingredientes para implementar una instancia determinada del algoritmo de búsqueda de Grover y resolver el problema de factorización. Para finalizar, la `Main` operación configura el problema especificando el número de cúbits y el número de iteraciones.

Q#

```
operation Main() : Result[] {  
    let nQubits = 5;  
    let iterations = CalculateOptimalIterations(nQubits);  
    Message($"Number of iterations: {iterations}");  
  
    // Use Grover's algorithm to find a particular marked state.  
    let results = GroverSearch(nQubits, iterations, ReflectAboutMarked);  
    return results;  
}
```

Ejecución del programa

Seleccione la plataforma deseada para ejecutar el programa.

Copilot en Azure Quantum

Puede probar el Q# código con Copilot en Azure Quantum de forma gratuita: todo lo que necesita es una cuenta de correo electrónico de Microsoft (MSA). Para más información sobre Copilot en Azure Quantum, consulte [Exploración de Azure Quantum](#).

1. Abra [Copilot en Azure Quantum](#) en el explorador.
2. Copie y pegue el código siguiente en el editor de código.

Q#

```
import Microsoft.Quantum.Convert.*;  
import Microsoft.Quantum.Math.*;  
import Microsoft.Quantum.Arrays.*;  
import Microsoft.Quantum.Measurement.*;  
import Microsoft.Quantum.Diagnostics.*;  
  
operation Main() : Result[] {  
    let nQubits = 5;  
    let iterations = CalculateOptimalIterations(nQubits);  
    Message($"Number of iterations: {iterations}");  
}
```

```

// Use Grover's algorithm to find a particular marked state.
let results = GroverSearch(nQubits, iterations, ReflectAboutMarked);
return results;
}

operation GroverSearch(
  nQubits : Int,
  iterations : Int,
  phaseOracle : Qubit[] => Unit) : Result[] {

  use qubits = Qubit[nQubits];

  PrepareUniform(qubits);

  for _ in 1..iterations {
    phaseOracle(qubits);
    ReflectAboutUniform(qubits);
  }

  // Measure and return the answer.
  return MResetEachZ(qubits);
}

function CalculateOptimalIterations(nQubits : Int) : Int {
  if nQubits > 63 {
    fail "This sample supports at most 63 qubits.";
  }
  let nItems = 1 <<< nQubits; // 2^nQubits
  let angle = ArcSin(1. / Sqrt(IntAsDouble(nItems)));
  let iterations = Round(0.25 * PI() / angle - 0.5);
  return iterations;
}

operation ReflectAboutMarked(inputQubits : Qubit[]) : Unit {
  Message("Reflecting about marked state...");
  use outputQubit = Qubit();
  within {
    // We initialize the outputQubit to  $(|0\rangle - |1\rangle) / \sqrt{2}$ , so that
    // toggling it results in a  $(-1)$  phase.
    X(outputQubit);
    H(outputQubit);
    // Flip the outputQubit for marked states.
    // Here, we get the state with alternating 0s and 1s by using the X
    // operation on every other qubit.
    for q in inputQubits[...2...] {
      X(q);
    }
  } apply {
    Controlled X(inputQubits, outputQubit);
  }
}

operation PrepareUniform(inputQubits : Qubit[]) : Unit is Adj + Ctl {
  for q in inputQubits {
    H(q);
  }
}

operation ReflectAboutAllOnes(inputQubits : Qubit[]) : Unit {
  Controlled Z(Most(inputQubits), Tail(inputQubits));
}

operation ReflectAboutUniform(inputQubits : Qubit[]) : Unit {
  within {
    // Transform the uniform superposition to all-zero.
    Adjoint PrepareUniform(inputQubits);
    // Transform the all-zero state to all-ones
    for q in inputQubits {
      X(q);
    }
  } apply {
    // Now that we've transformed the uniform superposition to the
    // all-ones state, reflect about the all-ones state, then let the
    // within/apply block transform us back.
    ReflectAboutAllOnes(inputQubits);
  }
}

```

Desde Copilot en Azure Quantum, puede abrir el programa en [VS Code para web](#) seleccionando el botón logotipo de VS Code en la esquina derecha del editor de código.

Ejecución del programa mediante el simulador en memoria

1. Seleccione **Simulador** en memoria.
2. Seleccione el número de capturas que se van a ejecutar y seleccione **Ejecutar**.
3. Los resultados se muestran en el histograma y en los **campos Resultados**.
4. Seleccione **Explicar código** para preguntar a Copilot para explicarle el código.

Ejecución del programa mediante el emulador de la serie H de Quantinuum

También puede enviar su programa al emulador [gratuito](#) de la serie H-Series quantinuum. El emulador simula un equipo cuántico con 20 cúbits.

1. Seleccione la lista desplegable **Simulador en memoria** y seleccione **Emulador de la serie H de Quantinuum**.
2. Seleccione el número de shots (actualmente limitados a 20) y seleccione **Ejecutar**.

Contenido relacionado

Explore otros tutoriales de Q#:

- [El entrelazamiento](#) cuántico muestra cómo escribir un Q# programa que manipula y mide cúbits y muestra los efectos de la superposición y el entrelazamiento.
- [El generador](#) de números aleatorios cuánticos muestra cómo escribir un Q# programa que genera números aleatorios fuera de cúbits en superposición.
- [Quantum Fourier Transform](#) explora cómo escribir un Q# programa que direcciona directamente los cúbits específicos.
- [Quantum Katas](#) son tutoriales autodirigido y ejercicios de programación dirigidos a enseñar los elementos de la computación cuántica y Q# la programación al mismo tiempo.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Tutorial: Implementación de la transformación de Quantum Fourier en Q#

29/09/2025

En este tutorial, se muestra cómo escribir y simular un programa cuántico que funciona en cúbits individuales.

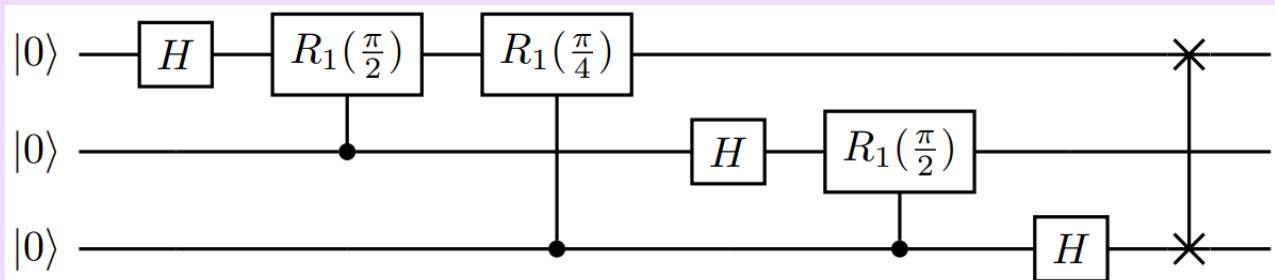
Aunque Q# se creó como un lenguaje de programación genérico para programas cuánticos a gran escala, puede utilizarse también para explorar el nivel inferior de los programas cuánticos, es decir, dirigirse directamente a cúbits específicos. En concreto, en este tutorial vamos a ver más de cerca la [transformación cuántica de Fourier](#) (QFT), una subrutina que forma parte de muchos algoritmos cuánticos de mayor tamaño.

En este tutorial, aprenderá a:

- ✓ Defina las operaciones cuánticas en Q#.
- ✓ Escribir el circuito de transformación Quantum Fourier
- ✓ Simulación de una operación cuántica desde la asignación de cúbits a la salida de medición.
- ✓ Observe cómo evoluciona la función de onda simulada del sistema cuántico a lo largo de la operación.

❗ Nota

Esta vista general del procesamiento cuántico de la información suele describirse en términos de **circuitos cuánticos**, que representan la aplicación secuencial de puertas, u *operaciones*, a cúbits específicos de un sistema. Así, las operaciones de uno y varios cúbits que aplique secuencialmente pueden representarse fácilmente en un diagrama de circuitos. Por ejemplo, la transformación de Fourier cuántico de tres cúbits completa que se usa en este tutorial tiene la siguiente representación como un circuito:



💡 Sugerencia

Si desea acelerar el recorrido de la computación cuántica, consulte [Código con Azure Quantum](#), una característica única del [sitio web](#) de Azure Quantum. Aquí puede ejecutar ejemplos integrados Q# o sus propios Q# programas, generar código nuevo Q# a partir de las indicaciones, abrir y ejecutar el código en [VS Code para la Web](#) con un solo clic y hacer preguntas a Copilot sobre la computación cuántica.

Requisitos previos

- La última versión de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- La versión más reciente de la [extensión \(QDK\) de Azure Quantum Development Kit](#). Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).
- Si desea usar cuadernos de Jupyter Notebook, también debe instalar Python y extensiones de Jupyter [y el paquete de Python más reciente](#). `qsharp` Para ello, abra un terminal y ejecute el siguiente comando:

Bash

```
$ pip install --upgrade qsharp
```

Crear un nuevo Q# archivo

1. En VS Code, seleccione **Archivo nuevo archivo > de texto**.
2. Guarde el archivo como **QFTcircuit.qs**. Este archivo contiene el código Q# del programa.
3. Abra **QFTcircuit.qs**.

Escribir un circuito QFT en Q#

La primera parte de este tutorial consiste en definir la operación de Q# `Main`, que realiza la transformación cuántica de Fourier sobre tres cúbits. La función `DumpMachine` se usa para observar cómo evoluciona la función de onda simulada del sistema de tres cúbits a lo largo de la operación. En la segunda parte del tutorial, agregará funcionalidad de medición y comparará los estados previos y posteriores a la medición de los cúbits.

Construye la operación paso a paso. Copie y pegue el código en las secciones siguientes en el archivo **QFTcircuit.qs**.

Puede ver el Q# completo de esta sección como referencia.

Importación de bibliotecas necesarias Q#

Dentro del Q# archivo, importe los espacios de nombres pertinentes `Microsoft.Quantum.*` .

Q#

```
import Microsoft.Quantum.Diagnostics.*;
import Microsoft.Quantum.Math.*;
import Microsoft.Quantum.Arrays.*;

// operations go here
```

Definición de operaciones con argumentos y retornos

A continuación, se define la operación `Main`:

Q#

```
operation Main() : Unit {
    // do stuff
}
```

La `Main` operación nunca toma argumentos y, por ahora, devuelve un `Unit` objeto , que es análogo a devolver `void` en C# o una tupla vacía, `Tuple[()]` , en Python. Después, modifique la operación para que devuelva una matriz de resultados de medida.

Asignación de cúbits

Dentro de la Q# operación, asigne un registro de tres cúbits con la `use` palabra clave . Con `use` , los cúbits se asignan automáticamente en el estado $|0\rangle$.

Q#

```
use qs = Qubit[3]; // allocate three qubits

Message("Initial state |000>:");
DumpMachine();
```

Al igual que la computación cuántica real, Q# no permite acceder directamente a los estados de los cúbits. Sin embargo, dado que la operación `DumpMachine` imprime el estado actual de la máquina de target, puede proporcionar información valiosa para depuración y aprendizaje cuando se utiliza con el simulador de estado completo.

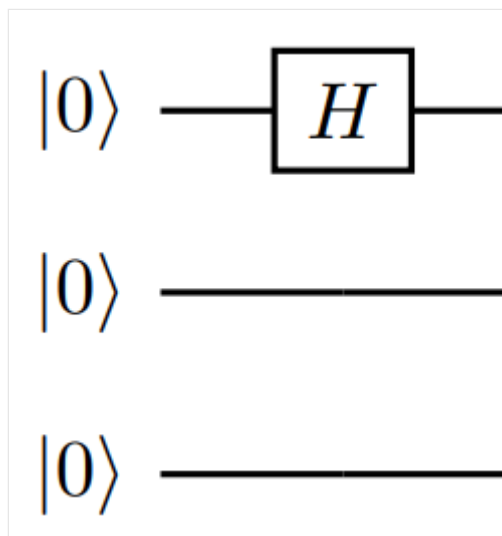
Aplicación de operaciones de un solo cúbit y controladas

A continuación, aplicará las operaciones que componen la `Main` propia operación. Q# ya contiene muchas de estas y otras operaciones cuánticas básicas, en el `Microsoft.Quantum.Intrinsic` espacio de nombres .

ⓘ Nota

Tenga en cuenta que no se importó en el fragmento de código anterior con los otros espacios de nombres, ya que `Microsoft.Quantum.Intrinsic` el compilador lo carga automáticamente para todos los Q# programas.

La primera operación aplicada es la operación `H` (Hadamard) al primer cúbit:



Para aplicar una operación a un cúbit específico de un registro (por ejemplo, un solo `qubit` de una matriz `qubit[]`) use la notación de índice estándar. Así, la aplicación de la operación `H` al primer cúbit del registro `qs` toma la siguiente forma:

```
Q#
```

```
H(qs[0]);
```

Además de aplicar la operación `H` (Hadamard) a los cúbits individuales, el circuito de QFT consiste principalmente en rotaciones de `R1` controladas. Una `R1( $\theta$ , <qubit>)` operación en general deja el componente **Missing argument for \ket** del cúbit sin cambios al aplicar una rotación de $e^{i\theta}$ al componente $|0\rangle$.

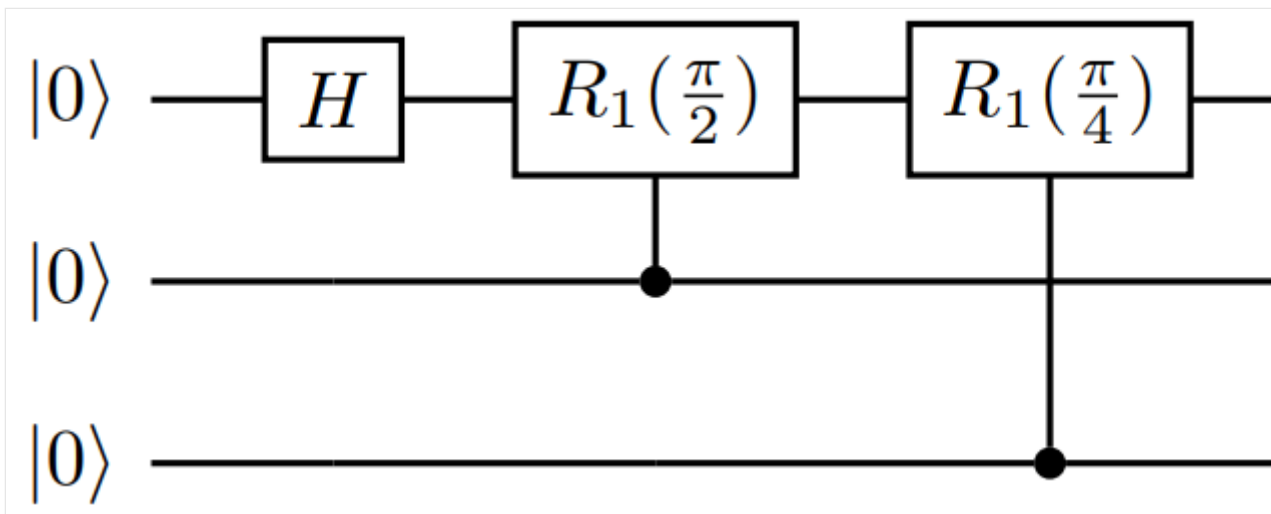
Q# permite condicionar fácilmente la ejecución de una operación a uno o varios cúbits de control. En general, se antepone `Controlled` a la llamada, y los argumentos de la operación

cambian de la siguiente manera:

`Op(<normal args>) → Controlled Op([<control qubits>], (<normal args>))`

Tenga en cuenta que el argumento de los cúbits de control deben proporcionarse como una matriz, aunque se trate de un solo cúbit.

Las operaciones controladas en QFT son las `R1` operaciones que actúan en el primer cúbit (y controladas por los cúbits segundo y tercero):



En el archivo de Q#, llame a estas operaciones con estas instrucciones:

Q#

```
Controlled R1([qs[1]], (PI()/2.0, qs[0]));  
Controlled R1([qs[2]], (PI()/4.0, qs[0]));
```

La función `PI()` se utiliza para definir las rotaciones en términos de radianes pi.

Aplicar operación SWAP

Después de aplicar las operaciones de `H` pertinentes y las rotaciones controladas al segundo y tercer cúbits, el circuito tiene este aspecto:

Q#

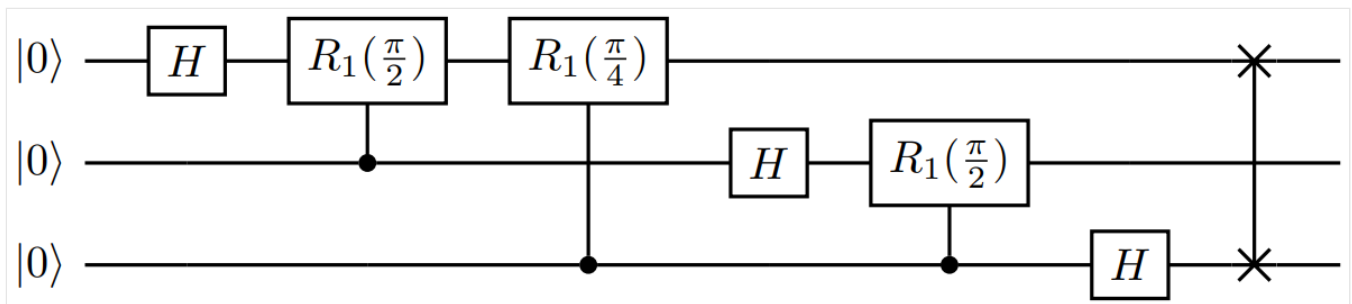
```
//second qubit:  
H(qs[1]);  
Controlled R1([qs[2]], (PI()/2.0, qs[1]));  
  
//third qubit:  
H(qs[2]);
```


Por último, se aplica una `SWAP` operación a los cúbits primero y tercer para completar el circuito. Esta operación es necesaria porque la naturaleza de la transformación cuántica de Fourier hace que los cúbits salgan en orden inverso, por lo que los intercambios permiten una integración perfecta de la subrutina en algoritmos de mayor tamaño.

Q#

```
SWAP(qs[2], qs[0]);
```

Ahora que ha terminado de escribir las operaciones en el nivel de cúbit de la transformación cuántica de Fourier en nuestra operación de Q#:



Desasignación de cúbits

El último paso es llamar de nuevo a `DumpMachine()` para ver el estado después de la operación y desasignar los cúbits. Los cúbits estaban en el estado $|0\rangle$ cuando los asignó y deben restablecerse a su estado inicial mediante la operación `ResetAll`.

Requerir que todos los cúbits se restablezcan explícitamente a $|0\rangle$ es una característica básica de Q#, ya que permite que otras operaciones sepan su estado exactamente cuando comienzan a usar esos mismos cúbits (un recurso insuficiente). Esto también asegura que no se entrelazarán con ningún otro cúbit del sistema. Si el restablecimiento no se realiza al final de un bloque de asignación `use`, puede producirse un error en tiempo de ejecución.

Agregue las siguientes líneas al archivo de Q#:

Q#

```
Message("After:");  
DumpMachine();  
  
ResetAll(qs); // deallocate qubits
```

La operación de QFT completa

El Q# programa se ha completado. Su archivo **QFTcircuit.qs** debería tener este aspecto:

```
Q#

import Microsoft.Quantum.Diagnostics.*;
import Microsoft.Quantum.Math.*;
import Microsoft.Quantum.Arrays.*;

operation Main() : Unit {

    use qs = Qubit[3]; // allocate three qubits

    Message("Initial state |000>:");
    DumpMachine();

    //QFT:
    //first qubit:
    H(qs[0]);
    Controlled R1([qs[1]], (PI()/2.0, qs[0]));
    Controlled R1([qs[2]], (PI()/4.0, qs[0]));

    //second qubit:
    H(qs[1]);
    Controlled R1([qs[2]], (PI()/2.0, qs[1]));

    //third qubit:
    H(qs[2]);

    SWAP(qs[2], qs[0]);

    Message("After:");
    DumpMachine();

    ResetAll(qs); // deallocate qubits

}
```

Ejecución del circuito QFT

Por ahora, la **Main** operación no devuelve ningún valor: la operación devuelve **Unit** el valor . Después, modifique la operación para que devuelva una matriz de resultados de medida (**Result[]**).

Q# programa independiente

1. Para ejecutar el programa, elija **Ejecutar Q# archivo** en el menú que precede **Main** o presione **Ctrl + F5**. El programa ejecuta la **Main** operación en el simulador predeterminado.

2. Las `Message` salidas y `DumpMachine` aparecen en la consola de depuración.

Si tiene curiosidad sobre cómo se ven afectados otros estados de entrada, le recomendamos que pruebe a aplicar operaciones de cúbits antes de la transformación.

Adición de medidas al circuito QFT

La presentación de la función `DumpMachine` mostró los resultados de la operación; lamentablemente, una piedra angular de la mecánica cuántica indica que un sistema cuántico real no puede tener esta función `DumpMachine`. En su lugar, la información se extrae mediante medidas que, por lo general, no solo no proporcionan información sobre el estado cuántico completo, sino que también pueden modificar drásticamente el propio sistema.

Hay muchos tipos de medidas cuánticas, pero el ejemplo se centra en la más básica: la medida de proyección en cúbits únicos. Tras medir en una base determinada (por ejemplo, la base de cálculo $|0\rangle, |1\rangle$), el estado del cúbit se proyecta en el estado base que se midió, lo que destruye cualquier superposición entre los dos.

Modificación de la operación QFT

Para implementar las medidas dentro de un programa de Q#, se usa la operación `M`, que devuelve un tipo `Result`.

En primer lugar, modifique la operación `Main` para devolver una matriz de resultados de medida, `Result[]`, en lugar de `Unit`.

Q#

```
operation Main() : Result[] {
```

Definición e inicialización de la matriz `Result[]`

Antes de asignar cúbits, declare y enlace una matriz de tres elementos (una `Result` para cada cúbit):

Q#

```
mutable resultArray = [Zero, size = 3];
```

La palabra clave `mutable` que precede a `resultArray` permite modificar la variable más adelante en el código; por ejemplo, al agregar los resultados de la medida.

Medida en un bucle `for` y adición de los resultados a la matriz

Después de las operaciones de transformación de QFT, inserte el código siguiente:

```
Q#  
  
for i in IndexRange(qs) {  
    resultArray w/= i <- M(qs[i]);  
}
```

La función `IndexRange` a la que se llama en una matriz (por ejemplo, nuestra matriz de cúbits, `qs`) devuelve un intervalo de índices de la matriz. Aquí, se usa en el bucle `for` para medir secuencialmente cada cúbit con la instrucción `M(qs[i])`. A continuación, cada tipo de `Result` medido (`Zero` o `One`) se agrega a la posición del índice correspondiente en `resultArray` con una instrucción de actualización y reasignación.

⚠ Nota

La sintaxis de esta instrucción es única para Q#, pero corresponde a la reasignación de variables similar `resultArray[i] <- M(qs[i])` que se ve en otros lenguajes, como F# y R.

La palabra clave `set` siempre se usa para reasignar variables enlazadas mediante `mutable`.

Devuelve `resultArray`.

Con los tres bits cúbits medidos y los resultados agregados a `resultArray`, es seguro restablecer y desasignar los cúbits como antes. Para devolver las medidas, inserte:

```
Q#  
  
return resultArray;
```

Ejecutar el circuito QFT con las medidas

Ahora, cambie la ubicación de las funciones `DumpMachine` para generar el estado antes y después de las medidas. El código de Q# final tendrá el aspecto siguiente:

Q#

```
import Microsoft.Quantum.Diagnostics.*;
import Microsoft.Quantum.Math.*;
import Microsoft.Quantum.Arrays.*;

operation Main() : Result[] {

    mutable resultArray = [Zero, size = 3];

    use qs = Qubit[3];

    //QFT:
    //first qubit:
    H(qs[0]);
    Controlled R1([qs[1]], (PI()/2.0, qs[0]));
    Controlled R1([qs[2]], (PI()/4.0, qs[0]));

    //second qubit:
    H(qs[1]);
    Controlled R1([qs[2]], (PI()/2.0, qs[1]));

    //third qubit:
    H(qs[2]);

    SWAP(qs[2], qs[0]);

    Message("Before measurement: ");
    DumpMachine();

    for i in IndexRange(qs) {
        resultArray w/= i <- M(qs[i]);
    }

    Message("After measurement: ");
    DumpMachine();

    ResetAll(qs);
    Message("Post-QFT measurement results [qubit0, qubit1, qubit2]: ");
    return resultArray;
}
```

💡 Sugerencia

Recuerde guardar el archivo cada vez que introduzca un cambio en el código antes de volver a ejecutarlo.

1. Para ejecutar el programa, elija **Ejecutar Q#** en el menú que precede **Main** a o presione **Ctrl + F5**. El programa ejecuta la **Main** operación en el simulador predeterminado.
2. Las **Message** salidas y **DumpMachine** aparecen en la consola de depuración.

La salida se parecerá a esta:

Output

Before measurement:

Basis	Amplitude	Probability	Phase
-----	-----	-----	-----
000>	0.3536+0.0000i	12.5000%	0.0000
001>	0.3536+0.0000i	12.5000%	0.0000
010>	0.3536+0.0000i	12.5000%	0.0000
011>	0.3536+0.0000i	12.5000%	0.0000
100>	0.3536+0.0000i	12.5000%	0.0000
101>	0.3536+0.0000i	12.5000%	0.0000
110>	0.3536+0.0000i	12.5000%	0.0000
111>	0.3536+0.0000i	12.5000%	0.0000

After measurement:

Basis	Amplitude	Probability	Phase
-----	-----	-----	-----
010>	1.0000+0.0000i	100.0000%	0.0000

Post-QFT measurement results [qubit0, qubit1, qubit2]:

[Zero, One, Zero]

Esta salida muestra algunas cosas diferentes:

1. Cuando compare el resultado devuelto con la medida previa **DumpMachine**, *no* se ilustra claramente la superposición posterior de QFT sobre los estados base. Una medida solo devuelve un estado base único, con una probabilidad determinada por la amplitud de ese estado en la función de onda del sistema.
2. A partir de la medida posterior **DumpMachine**, se puede ver que la medida *cambia* el estado en sí y lo proyecta desde la superposición inicial sobre los estados base al estado base único que corresponde al valor medido.

Si repite esta operación muchas veces, las estadísticas de los resultados comienzan a ilustrar la superposición igualmente ponderada del estado posterior a QFT que da lugar a un resultado aleatorio en cada iteración. *Sin embargo*, además de ser un método ineficaz e imperfecto, solo

reproduciría las amplitudes relativas de los estados base, no las fases relativas entre ellos. Esto último no es un problema en este ejemplo, pero vería que aparecen fases relativas si se da una entrada más compleja a QFT que $|000\rangle$.

Uso de las Q# operaciones para simplificar el circuito QFT

Tal y como se mencionó en la introducción, gran parte de la potencia de Q# se debe al hecho de que permite olvidarse de las preocupaciones de tener que tratar con cúbits individuales. De hecho, si desea desarrollar programas cuánticos aplicables a escala completa, tener que preocuparse de si una operación `H` va antes o después de una rotación determinada es un lastre. Azure Quantum proporciona la `ApplyQFT` operación, que puede usar y aplicar para cualquier número de cúbits.

1. Reemplace todo desde la primera `H` operación a la `SWAP` operación, inclusiva, por:

Q#

```
ApplyQFT(qs);
```

2. El código debería tener ahora un aspecto similar al siguiente

Q#

```
import Microsoft.Quantum.Diagnostics.*;
import Microsoft.Quantum.Math.*;
import Microsoft.Quantum.Arrays.*;

operation Main() : Result[] {

    mutable resultArray = [Zero, size = 3];

    use qs = Qubit[3];

    //QFT:
    //first qubit:

    ApplyQFT(qs);

    Message("Before measurement: ");
    DumpMachine();

    for i in IndexRange(qs) {
        resultArray w/= i <- M(qs[i]);
    }

    Message("After measurement: ");
```

```

    DumpMachine();

    ResetAll(qs);
    Message("Post-QFT measurement results [qubit0, qubit1, qubit2]: ");
    return resultArray;
}

```

3. Vuelva a ejecutar el Q# programa y observe que la salida es la misma que antes.
4. Para ver la ventaja real del uso Q# de operaciones, cambie el número de cúbits a algo distinto de 3:

```

Q#

mutable resultArray = [Zero, size = 4];

use qs = Qubit[4];
//...

```

Por lo tanto, puede aplicar el QFT adecuado para cualquier número determinado de cúbits, sin tener que preocuparse de agregar nuevas **H** operaciones y rotaciones en cada cúbit.

Contenido relacionado

Explore otros tutoriales de Q#:

- [El generador](#) de números aleatorios cuánticos muestra cómo escribir un Q# programa que genera números aleatorios fuera de cúbits en superposición.
- [El algoritmo](#) de búsqueda de Grover muestra cómo escribir un Q# programa que usa el algoritmo de búsqueda de Grover.
- [El entrelazamiento](#) cuántico muestra cómo escribir un Q# programa que manipula y mide cúbits y muestra los efectos de la superposición y el entrelazamiento.
- [Quantum Katas](#) [↗](#) son tutoriales autodirigido y ejercicios de programación dirigidos a enseñar los elementos de la computación cuántica y Q# la programación al mismo tiempo.

Referencia: Extensión del Kit de desarrollo de Azure Quantum para Visual Studio Code

29/09/2025

El Kit de desarrollo de Azure Quantum (QDK) usa toda la funcionalidad estándar de Visual Studio Code (VS Code), junto con características específicas de Q#, cuando se trabaja con `.qs` archivos. En esta guía de referencia se describen todos los comandos de QDK a los que puede acceder en la paleta de comandos de VS Code, junto con otras características y vínculos a contenido adicional. Para obtener instrucciones generales de VS Code, consulte la documentación [de VS Code](#).

Sugerencia

La mayoría de los comandos de la extensión QDK son accesibles desde el menú **Ver > paleta de comandos** . Mientras se encuentra en un `.qs` archivo, abra la paleta de comandos y escriba **QDK**: para ver una lista filtrada de comandos.

Comandos

La mayoría de los comandos de QDK están relacionados con la escritura y ejecución de programas de Q#, y solo están disponibles cuando un `.qs` archivo está activo. Otros comandos funcionan en segundo plano y no son accesibles en la paleta de comandos.

En la tabla siguiente se describen los comandos de la extensión QDK que aparecen en la paleta de comandos.

Sugerencia

Puede crear métodos abreviados de teclado personalizados para comandos de VS Code mediante **métodos abreviados de teclado** o **Ctrl + K + S**. Para obtener más información, consulte [Enlaces de claves para VS Code](#) .

 Expandir tabla

Comando	Acción	Notas	Acción de usuario alternativa
QDK: Crear proyecto de Q#	Crea un proyecto de Q# en la carpeta activa, incluido un	Para obtener más información sobre	En Explorador de archivos, haga clic con

Comando	Acción	Notas	Acción de usuario alternativa
	<code>qsharp.json</code> archivo de manifiesto y una <code>src</code> subcarpeta con un archivo predeterminado <code>main.qs</code> .	los proyectos de Q#, vea Trabajar con proyectos de Q#.	el botón derecho en la carpeta de destino y seleccione Crear proyecto de Q#.
QDK: Creación de un cuaderno de Azure Quantum	Abre un cuaderno de Jupyter Notebook de ejemplo que ejecuta un programa de Q# + Python y envía un trabajo a Azure Quantum.	Para más información sobre los cuadernos de Azure Quantum, consulte Uso de cuadernos de Azure Quantum.	N/D
QDK: Conexión a un área de trabajo de Azure Quantum	Conéctese a un área de trabajo de Azure Quantum mediante su cuenta de Azure o una cadena de conexión. Una vez autenticado, el área de trabajo aparece en Explorador de archivos en Áreas de trabajo cuánticas .	Para más información sobre las conexiones de Azure Quantum, consulte Conexión al área de trabajo de Azure Quantum.	En el Explorador de archivos, mantenga el puntero sobre las áreas de trabajo de Quantum y seleccione el icono + .
QDK: Abrir el área de juegos de QDK	Abre una carpeta en línea de programas de ejemplo de Q# en Explorador de archivos. Puede editar y ejecutar los programas en el simulador cuántico local y también establecer puntos de interrupción y recorrer el código con el depurador integrado.	Para obtener más información, vea el archivo LÉAME en la carpeta de ejemplo.	N/D
QDK: Actualización de áreas de trabajo de Azure Quantum	Sincroniza los cambios más recientes de las áreas de trabajo cuánticas conectadas.	Si hay un problema de conexión, aparece un icono de alerta junto al nombre del área de trabajo.	En el Explorador de archivos, mantenga el puntero sobre Las áreas de trabajo cuánticas y seleccione el icono de actualización.
Explorador: Centrar en la vista de Áreas de trabajo cuánticas	Abre Explorador de archivos y se centra en las áreas de trabajo cuánticas conectadas, si está configurada. Si no hay ninguna área de trabajo configurada, se le pedirá que agregue un área de trabajo existente.	Para más información sobre las conexiones de Azure Quantum, consulte Conexión al área de trabajo de Azure Quantum.	N/D

Los siguientes comandos están disponibles cuando un `.qs` archivo está activo.

Comando	Acción	Notas	Acción de usuario alternativa
QDK: Mostrar circuito	Muestra un diagrama de circuitos para el programa Q# antes de que se ejecute.	Para más información, consulte Visualización de diagramas de circuitos cuánticos .	Seleccione la opción de lente de código Circuito en el menú situado junto a la operación de punto de entrada o encima de cada operación definida por el usuario en el programa.
QDK: Ejecutar archivo y mostrar histograma	Ejecuta el programa Q# actual y muestra un histograma de los resultados en un nuevo panel.	Para acceder a las opciones de ordenación y filtro para la visualización del histograma, seleccione el icono de filtro en el panel histograma.	Seleccione la opción de lente de código Histograma en el menú situado junto a la operación de punto de entrada.
QDK: Obtener QIR para el programa QDK actual	Abre el origen de QIR para el código Q# actual en una nueva ventana de edición. El programa debe usar el perfil de destino Base o RI adaptativo para exportar la fuente QIR.	Para obtener más información sobre QIR, consulte Representación intermedia cuántica y el blog para desarrolladores de Q# .	N/D
QDK: Cálculo de estimaciones de recursos	Invoca la versión integrada del estimador de recursos.	Para más información, consulte Introducción al estimador de recursos de Azure Quantum.	N/D
QDK: Ayuda	Una breve introducción a la extensión QDK en VS Code.	Para obtener documentación completa de Azure Quantum, consulte la documentación de Azure Quantum.	N/D
QDK: Ejecutar archivo y mostrar diagrama de circuito	Ejecuta el programa Q# actual y muestra un circuito del programa con salidas.	Para más información, consulte Visualización de diagramas de circuitos cuánticos .	N/D
QDK: Adición del archivo de instrucciones de	Abre un cuadro de diálogo que le ofrece la opción de agregar instrucciones de	Para obtener más información, consulte Configuración del	N/D

Comando	Acción	Notas	Acción de usuario alternativa
Copilot para Q# y OpenQASM	Copilot para ayudarle a escribir y ejecutar código de Q# y OpenQASM.	modo de agente en VS Code para el Kit de desarrollo de Quantum.	
QDK: Mostrar documentación de API	Abre la documentación de api en un panel nuevo. Para buscar en este panel, presione Ctrl + F .	Para más información, consulte la referencia de la API de Azure Quantum .	N/D
QDK: Mostrar registro de cambios	Abre un registro de cambios en una nueva pestaña que muestra las actualizaciones de QDK para las versiones actuales y de todas las versiones anteriores.	El registro de cambios también está disponible en el repositorio de GitHub de QDK de código abierto.	N/D
Depuración: Iniciar depuración	Abre el programa Q# actual en el depurador.	Para más información, consulte Depuración y prueba del código cuántico .	Pulse F5 o seleccione la opción Debug de code lens en el menú ubicado junto a la operación de punto de entrada, o seleccione el icono Ejecutar en la esquina superior derecha y elija Iniciar depuración .
Depuración: Iniciar	Ejecuta el programa Q# actual en el simulador cuántico predeterminado.	Para obtener más información, consulte Introducción a los programas de Q# .	Presione Ctrl + F5 , o seleccione la opción de lente de código Ejecutar en el menú junto a la operación de punto de entrada, o seleccione el icono Ejecutar en la esquina superior derecha y luego elija Ejecutar .

Terminales

Los programas de Q# usan dos ventanas de terminal en VS Code:

 Expandir tabla

Terminal	Acción
Consola de depuración	Muestra la salida de ejecución o la salida de depuración
Problemas	Muestra comprobaciones de errores anteriores a la compilación

Edición del código de Q#

La mayoría de las características comunes de edición de código en VS Code también están disponibles cuando se trabaja con programas de Q#:

- Comprobación de errores de precompilación
- Definición del destino
- Referencias
- Firmas de funciones
- Información de parámetros
- Sugerencias de finalización, incluidas las finalizaciones conscientes del contexto, de miembros del espacio de nombres y de miembros de tipos.
- Linting: En archivos Q#, se configura linting por proyecto en un archivo de manifiesto. Para obtener más información, vea [Trabajar con proyectos](#) de Q#.

Para obtener más información, consulte [IntelliSense](#) en la documentación de VS Code.

Tareas comunes

Trabajar con proyectos y archivos de Q#

 Expandir tabla

Tarea	Acción	Notas
Nuevo archivo Q#	Seleccione Archivo > Nuevo archivo > Archivo de texto . Guarde el archivo con una <code>.qs</code> extensión.	Si establece <code>Files: Default Language = qsharp</code> , un nuevo archivo establece automáticamente el formato de Q#.
Creación de un proyecto de Q#	En una carpeta abierta en Explorador de archivos, seleccione Crear proyecto de Q# en la paleta de comandos o haga clic con el botón derecho en la carpeta en Explorador de archivos y seleccione Crear proyecto de Q# .	Para obtener más información sobre los proyectos de Q#, vea Trabajar con proyectos de Q#.

Tarea	Acción	Notas
Archivos de ejemplo	En un archivo en blanco <code>.qs</code> , escriba ejemplos y, a continuación, seleccione un programa de ejemplo en la lista de opciones.	También puede seleccionar Abrir área de juegos de QDK en la paleta de comandos para abrir una carpeta en línea de programas de ejemplo de Q# en el Explorador de archivos.

Conexión a Azure Quantum

 Expandir tabla

Tarea	Acción	Notas	Acción de usuario alternativa
Conexión a un área de trabajo de Azure Quantum	En el Explorador de archivos, expanda Áreas de trabajo cuánticas y seleccione Agregar un área de trabajo existente . Siga las indicaciones para seleccionar una suscripción y un área de trabajo.	Puede conectarse a varias áreas de trabajo. Seleccione + junto a Áreas de trabajo de Quantum para conectar otra área de trabajo. Las conexiones del área de trabajo persisten entre las sesiones de VS Code.	En la paleta de comandos, seleccione QDK: Conectarse a un área de trabajo de Azure Quantum .
Conexión mediante programación con un programa de Python	Haga clic con el botón derecho en una conexión de área de trabajo existente y seleccione Copiar código de Python para conectarse al área de trabajo . Pegue el código resultante en el programa de Python.	Para más información, consulte Envío de trabajos de Python con Q# a Azure Quantum .	N/D

Ejecución de programas

 Expandir tabla

Tarea	Acción	Notas	Acción de usuario alternativa
Ejecución de un programa de Q# en el simulador cuántico local	En un programa de Q#, seleccione el icono Ejecutar en la esquina superior derecha y seleccione Ejecutar .	Para más información sobre el simulador cuántico, consulte el simulador cuántico disperso.	Presione Ctrl-F5 , elija QDK: Ejecutar archivo y mostrar histograma o QDK: Ejecutar archivo y mostrar diagrama de circuito en la paleta de comandos o elija la opción

Tarea	Acción	Notas	Acción de usuario alternativa
			Ejecutar lente de código Ejecutar encima de la operación de punto de entrada.
Depuración de un programa	Dentro de un programa Q#, seleccione el icono Ejecutar en la esquina superior derecha y luego elija Iniciar depuración .	Para obtener más información sobre el depurador de Q# en VS Code, consulte Depuración y prueba del código cuántico .	Presione F5 o elija la opción Depurar Lens de código en el menú situado encima de la operación de punto de entrada.
Ver proveedores y destinos en tus espacios de trabajo	En el Explorador de archivos, seleccione Áreas de trabajo cuánticas , expanda el área de trabajo y, a continuación, expanda Proveedores para ver los proveedores disponibles en el área de trabajo. Expanda un proveedor individual para ver los objetivos disponibles.	Mantenga el puntero sobre un nombre de destino para ver su Estado y tiempo de cola antes de enviar un trabajo.	N/D
Envío de un trabajo a Azure Quantum	En un programa de Q#, seleccione un área de trabajo, un proveedor y un destino. Para enviar el programa de Q# actual, seleccione la flecha situada junto al destino.	Para más información, consulte Envío de trabajos de Q# a Azure Quantum .	N/D
Visualización de los resultados del trabajo	Expanda el espacio de trabajo y luego expanda Trabajos . Para abrir la salida del trabajo desde Azure Storage, seleccione el icono de nube situado junto al nombre del trabajo.	Los trabajos aparecen de más recientes a más antiguos.	N/D

Ejecución del programa

Artículo • 21/03/2025

En el ejemplo siguiente se da una primera vista de cómo se implementa un programa de Q#:

Q#

```
/// # Sample
/// Bell States
///
/// # Description
/// Bell states or EPR pairs are specific quantum states of two qubits
/// that represent the simplest (and maximal) examples of quantum
/// entanglement.
///
/// This Q# program implements the four different Bell states.

import Std.Diagnostics.*;
import Std.Measurement.*;

operation Main() : (Result, Result)[] {
    // Allocate the two qubits that will be used to create a Bell state.
    use register = Qubit[2];

    // This array contains a label and a preparation operation for each one
    // of the four Bell states.
    let bellStateTuples = [
        ("|Φ+>", PreparePhiPlus),
        ("|Φ->", PreparePhiMinus),
        ("|Ψ+>", PreparePsiPlus),
        ("|Ψ->", PreparePsiMinus)
    ];

    // Prepare all Bell states, show them using the `DumpMachine` operation
    // and measure the Bell state qubits.
    mutable measurements = [];
    for (label, prepare) in bellStateTuples {
        prepare(register);
        Message($"Bell state {label}:");
        DumpMachine();
        measurements += [(MResetZ(register[0]), MResetZ(register[1]))];
    }
    return measurements;
}

operation PreparePhiPlus(register : Qubit[]) : Unit {
    ResetAll(register);           // |00>
    H(register[0]);               // |+0>
    CNOT(register[0], register[1]); // 1/sqrt(2)(|00> + |11>)
}
```



```

operation PreparePhiMinus(register : Qubit[]) : Unit {
    ResetAll(register);           //  $|00\rangle$ 
    H(register[0]);               //  $|+0\rangle$ 
    Z(register[0]);               //  $| -0\rangle$ 
    CNOT(register[0], register[1]); //  $1/\sqrt{2}(|00\rangle - |11\rangle)$ 
}

operation PreparePsiPlus(register : Qubit[]) : Unit {
    ResetAll(register);           //  $|00\rangle$ 
    H(register[0]);               //  $|+0\rangle$ 
    X(register[1]);               //  $|+1\rangle$ 
    CNOT(register[0], register[1]); //  $1/\sqrt{2}(|01\rangle + |10\rangle)$ 
}

operation PreparePsiMinus(register : Qubit[]) : Unit {
    ResetAll(register);           //  $|00\rangle$ 
    H(register[0]);               //  $|+0\rangle$ 
    Z(register[0]);               //  $| -0\rangle$ 
    X(register[1]);               //  $| -1\rangle$ 
    CNOT(register[0], register[1]); //  $1/\sqrt{2}(|01\rangle - |10\rangle)$ 
}

```

Este programa implementa los cuatro estados básicos de Bell del entrelazamiento cuántico y es uno de los programas de ejemplo incluidos la extensión de [Visual Code de Azure Quantum](#).

Puede ejecutar el programa desde el simulador integrado en la extensión QDK de VS Code y obtener la salida estándar.

Output

Message: Bell state $|\Phi+\rangle$:

DumpMachine:

Basis	Amplitude	Probability	Phase
$ 00\rangle$	$0.7071+0.0000i$	50.0000%	0.0000
$ 11\rangle$	$0.7071+0.0000i$	50.0000%	0.0000

Message: Bell state $|\Phi-\rangle$:

DumpMachine:

Basis	Amplitude	Probability	Phase
$ 00\rangle$	$0.7071+0.0000i$	50.0000%	0.0000
$ 11\rangle$	$-0.7071+0.0000i$	50.0000%	-3.1416

Message: Bell state $|\Psi+\rangle$:

DumpMachine:

Basis	Amplitude	Probability	Phase
01⟩	0.7071+0.0000i	50.0000%	0.0000
10⟩	0.7071+0.0000i	50.0000%	0.0000

Message: Bell state $|\Psi\rangle$:

DumpMachine:

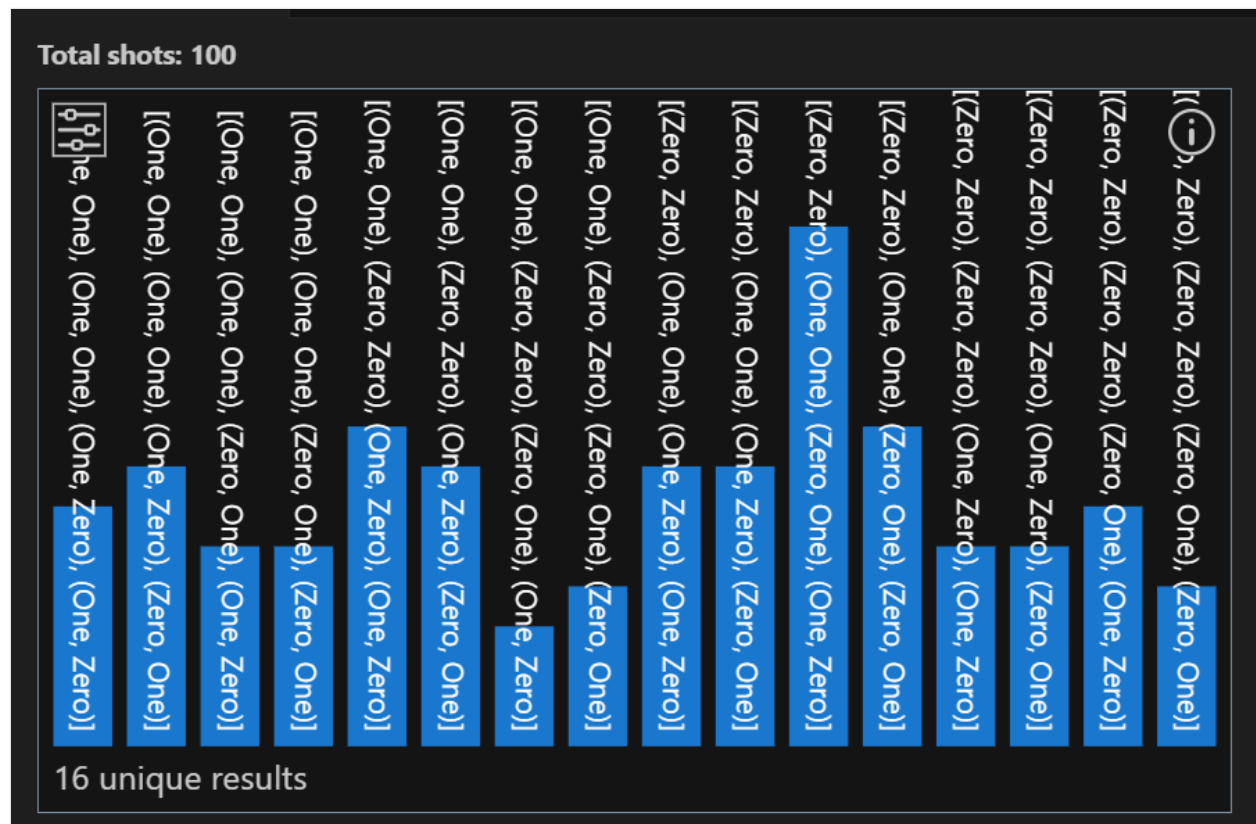
Basis	Amplitude	Probability	Phase
01⟩	0.7071+0.0000i	50.0000%	0.0000
10⟩	-0.7071+0.0000i	50.0000%	-3.1416

Result: "[(One, One), (Zero, Zero), (One, Zero), (Zero, One)]"

Finished shot 1 of 1

Q# simulation completed.

o ejecute el simulador con una salida de histograma



Para ejecutar el programa en hardware cuántico, el programa primero debe compilarse y enviarse a Azure Quantum, todos los cuales se pueden realizar desde VS Code. Para obtener el proceso completo de un extremo a otro, consulte [Introducción a los programas de Q# y Visual Studio Code](#).

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Espacios de nombres

Artículo • 18/02/2025

En su nivel superior, un programa de Q# consta de un conjunto de espacios de nombres. Además de los [comentarios](#), los espacios de nombres son los únicos elementos de nivel superior de un programa de Q# y cualquier otro elemento debe residir dentro de un espacio de nombres. Cada archivo puede contener cero o más espacios de nombres, y cada espacio de nombres puede abarcar varios archivos. Q# no admite espacios de nombres anidados.

❗ Nota

Si no declara explícitamente un espacio de nombres, el Q# compilador usa el nombre de archivo como nombre del espacio de nombres.

Un bloque de espacio de nombres consta de la palabra clave `namespace`, seguido del nombre del espacio de nombres y del contenido del bloque dentro de llaves `{ }`. Los nombres de espacio de nombres constan de una secuencia de uno o varios [símbolos admitidos](#) separados por un punto (`.`). Aunque los nombres de los espacios de nombres pueden contener puntos para mejorar la legibilidad, Q# no admite referencias relativas a espacios de nombres. Por ejemplo, dos espacios de nombres llamados `Foo` y `Foo.Bar` no están relacionados y no hay una noción de jerarquía. Específicamente, para una función `Baz` definida en `Foo.Bar`, no es posible abrir `Foo` y, a continuación, acceder a esa función mediante `Bar.Baz`.

Los bloques de espacio de nombres pueden contener [directivas](#) de importación, [operaciones](#), [funciones](#) y [declaraciones de tipo](#). Estos elementos pueden producirse en cualquier orden y son recursivos de forma predeterminada, lo que significa que se pueden declarar y usar en cualquier orden y pueden llamarse a sí mismos. No es necesario que la declaración de un tipo o que se pueda llamar preceda a su uso.

Directivas de importación

De manera predeterminada, se puede acceder a todo lo declarado en el mismo espacio de nombres sin ninguna calificación adicional. Sin embargo, las declaraciones de otro espacio de nombres solo se pueden usar calificando su nombre con el nombre del espacio de nombres al que pertenecen o abriendo ese espacio de nombres antes de usarlo, como se muestra en el ejemplo siguiente.

Q#

```
namespace Microsoft.Quantum.Samples {  
  
    import Microsoft.Quantum.Arithmetic.*;  
    import Microsoft.Quantum.Arrays.* as Array;  
  
    // ...  
}
```

❗ Nota

En el caso de los espacios de nombres de la biblioteca estándar Q# , la raíz del espacio de nombres se puede simplificar en `Std`. Por ejemplo, el ejemplo anterior podría volver a escribirse como:

Q#

```
import Std.Arithmetic.*;  
import Std.Arrays.* as Array;
```

En el ejemplo se usa una `import` directiva para importar todos los tipos y los invocables declarados en el `Microsoft.Quantum.Arithmetic` espacio de nombres. A continuación, puede hacer referencia a ellos por su nombre no calificado a menos que ese nombre entre en conflicto con una declaración en el bloque de espacio de nombres u otro espacio de nombres abierto.

Para evitar escribir el nombre completo mientras se distingue de dónde proceden determinados elementos, puede definir un nombre alternativo o *alias* para un espacio de nombres determinado. En este caso, puede calificar todos los tipos y invocables declarados en ese espacio de nombres por el nombre corto definido en su lugar. En el ejemplo anterior se usa un alias para el `Microsoft.Quantum.Arrays` espacio de nombres. A continuación, puede usar la función `IndexRange`, declarada en `Microsoft.Quantum.Arrays`, por ejemplo, a través `Array.IndexRange` de dentro de ese bloque de espacio de nombres.

Tanto si va a abrir un espacio de nombres como si define un alias, `import` las directivas son válidas solo en la parte del espacio de nombres de ese archivo.

Comentarios

¿Le ha resultado útil esta página?



[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Declaraciones de tipos

Artículo • 21/03/2025

Q# admite tipos de `struct` definidos por el usuario. `struct` tipos son similares a los tipos de registro en F#; son inmutables, pero admiten una construcción de [de copia y actualización](#).

Tipos de estructura

`struct` tipos solo pueden contener elementos con nombre y no admiten elementos anónimos. Se admite cualquier combinación de elementos con nombre, aunque los elementos no se pueden anidar.

La siguiente declaración, por ejemplo, define una estructura `Complex` que tiene dos elementos con nombre `Real` y `Imaginary`, ambos de tipo `Double`:

Q#

```
struct Complex {  
    Real : Double,  
    Imaginary : Double,  
}
```

Puede acceder a los elementos contenidos a través de su nombre o por deconstrucción (para obtener más información, vea [acceso a elementos](#)). También puede acceder a una tupla de todos los elementos donde la forma coincide con la definida en la declaración a través del operador `unwrap` de .

`struct` tipos son útiles por dos motivos. En primer lugar, siempre que las bibliotecas y programas que usan los tipos definidos accedan a los elementos a través de su nombre en lugar de deconstrucción, el tipo se puede extender para contener elementos adicionales más adelante sin interrumpir ningún código de biblioteca. Debido a esto, no se recomienda acceder a los elementos a través de la deconstrucción.

En segundo lugar, Q# permite transmitir la intención y las expectativas de un tipo de datos específico, ya que no hay ninguna conversión automática entre los valores de dos tipos de `struct`, incluso si sus tipos de elemento son idénticos.

Constructores de estructura

El compilador genera automáticamente constructores para los nuevos tipos de `struct` cuando lee una definición de `struct`. Para la estructura `Complex` en el ejemplo anterior, puede crear una instancia con

Q#

```
let complexPair = Complex(1.4, 2.1);
```

También puede definir instancias con la palabra clave `new`, por ejemplo.

Q#

```
let complexPair = new Complex { Real = 1.4, Imaginary = 2.1 };
```

Puede copiar una estructura existente con la sintaxis `...`

Q#

```
let copyPair = new Complex { ...complexPair };
```

Al copiar, puede especificar campos individuales para cambiar.

Q#

```
let updatedPair = new Complex { ...complexPair, Real = 3.5 };
```

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Declaraciones invocables

Artículo • 21/03/2025

Las declaraciones invocables o *invocables*, declaradas en un ámbito global son visibles públicamente de forma predeterminada; es decir, se pueden usar en cualquier parte del mismo proyecto y en un proyecto que haga referencia al ensamblado en el que se declaran. [modificadores de Access](#) permiten restringir su visibilidad solo al ensamblado actual, de modo que los detalles de implementación se pueden cambiar más adelante sin interrumpir el código que se basa en una biblioteca específica.

Q# admite dos tipos de invocables: operaciones y funciones. El tema [Operations and Functions](#) elabora sobre la distinción entre los dos. Q# también admite la definición de plantillas de ; por ejemplo, implementaciones con parámetros de tipo para un determinado invocable. Para obtener más información, vea [Parámetros de tipo](#).

❗ Nota

Estas implementaciones parametrizadas de tipo no pueden usar ninguna construcción de lenguaje que dependa de propiedades concretas de los argumentos de tipo; actualmente no hay ninguna manera de expresar restricciones de tipo en Q#, o para definir implementaciones especializadas para argumentos de tipo concretos.

Invocables y funtores

Q# permite implementaciones especializadas con fines específicos; Por ejemplo, las operaciones de Q# pueden definir implícita o explícitamente la compatibilidad con determinados funtores de y, junto con ella, las implementaciones especializadas que se invocarán cuando se aplica un functor específico a esa función invocable.

Un functor, en cierto sentido, es una fábrica que define una nueva implementación invocable que tiene una relación específica con el invocable al que se aplicó. Los funtores son más que las funciones de nivel superior tradicionales en las que requieren acceso a los detalles de implementación del invocable al que se han aplicado. En ese sentido, son similares a otras fábricas, como plantillas. También se pueden aplicar a los invocables con parámetros de tipo.

Tenga en cuenta la siguiente operación, `ApplyQFT`:

```
qsharp
```

```
operation ApplyQFT(qs : Qubit[]) : Unit is Adj + Ctl {
  let length = Length(qs);
  Fact(length >= 1, "ApplyQFT: Length(qs) must be at least 1.");
  for i in length - 1..-1..0 {
    H(qs[i]);
    for j in 0..i - 1 {
      Controlled R1Frac([qs[i]], (1, j + 1, qs[i - j - 1]));
    }
  }
}
```

Esta operación toma un argumento de tipo `Qubit[]` y devuelve un valor de tipo `Unit`. La anotación `is Adj + Ctl` en la declaración de `ApplyQFT` indica que la operación admite tanto el `Adjoint` como el functor de `Controlled`. (Para obtener más información, vea [Características de operación](#)). La expresión `Adjoint ApplyQFT` accede a la especialización que implementa el adjacente de `ApplyQFT` y `Controlled ApplyQFT` accede a la especialización que implementa la versión controlada de `ApplyQFT`. Además del argumento de la operación original, la versión controlada de una operación toma una matriz de cúbits de control y aplica la operación original en la condición de que todos estos cúbits de control estén en un estado $|1\rangle$.

En teoría, una operación para la que se puede definir una versión adjacente también debe tener una versión controlada y viceversa. Sin embargo, en la práctica puede ser difícil desarrollar una implementación para una o la otra, especialmente para implementaciones probabilísticas siguiendo un patrón de repetición hasta el éxito. Por ese motivo, Q# le permite declarar compatibilidad para cada functor individualmente. Sin embargo, dado que los dos functors conmutan, una operación que define la compatibilidad para ambos también tiene que tener una implementación (normalmente definida implícitamente, lo que significa que se genera el compilador) para cuando se aplican ambos funtores a la operación.

No hay functors que se puedan aplicar a las funciones. Actualmente, las funciones tienen exactamente una implementación de cuerpo y ninguna especialización adicional. Por ejemplo, la declaración

Q#

```
function Hello (name : String) : String {
  $"Hello, {name}!"
}
```

es equivalente a

Q#

```
function Hello (name : String) : String {  
  body ... {  
    $"Hello, {name}!"  
  }  
}
```

Aquí, `body` especifica que la implementación dada se aplica al cuerpo predeterminado de la función `Hello`, lo que significa que la implementación se invoca cuando no se han aplicado functors u otros mecanismos de fábrica antes de la invocación. Los tres puntos de `body ...` corresponden a una directiva del compilador que indica que los elementos de argumento de la declaración de función deben copiarse y pegarse en este punto.

Los motivos subyacentes indican explícitamente dónde se copian los argumentos de la declaración al que se puede llamar primario y se pegan son dosfolding: uno, no es necesario repetir la declaración de argumento y dos, garantiza que los funtores que requieren argumentos adicionales, como el functor de `Controlled`, se pueden introducir de forma coherente.

Cuando hay exactamente una especialización que define la implementación del cuerpo predeterminado, se puede omitir el ajuste adicional del formulario `body ... {
<implementation> }`.

Recursividad

Q# los invocables pueden ser directamente o indirectamente recursivos y pueden declararse en cualquier orden; una operación o función puede llamarse a sí misma o puede llamar a otra que llame directa o indirectamente al autor de la llamada.

El espacio de pila puede estar limitado al ejecutarse en hardware cuántico y las recursiones que superan ese límite de espacio de pila producen un error en tiempo de ejecución.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Declaraciones de especialización

Artículo • 21/03/2025

Como se explica en la sección sobre [declaraciones invocables](#), actualmente no hay ninguna razón para declarar explícitamente especializaciones para las funciones. Este tema se aplica a las operaciones y elabora sobre cómo declarar las especializaciones necesarias para admitir determinados funtores.

Se trata de un problema bastante común en la computación cuántica para requerir el adyacente de una transformación determinada. Muchos algoritmos cuánticos requieren una operación y su adyacente para realizar un cálculo. Q# emplea cálculos simbólicos que pueden generar automáticamente la implementación adyacente correspondiente para una implementación concreta del cuerpo. Esta generación es posible incluso para implementaciones que combinan libremente cálculos clásicos y cuánticos. Sin embargo, hay algunas restricciones que se aplican en este caso. Por ejemplo, la generación automática no se admite por motivos de rendimiento si la implementación usa variables mutables. Además, cada operación a la que se llama dentro del cuerpo genera las necesidades adyacentes correspondientes para admitir el funtor `Adjoint`.

Aunque no se puedan deshacer fácilmente las medidas en el caso de varios cúbits, es posible combinar medidas para que la transformación aplicada sea unitaria. En este caso, significa que, aunque la implementación del cuerpo contiene medidas que por sí mismas no admiten el funtor de `Adjoint`, el cuerpo en su totalidad es adyacente. No obstante, se produce un error en la generación automática de la implementación adyacente en este caso. Por este motivo, es posible especificar manualmente la implementación. El compilador genera automáticamente implementaciones optimizadas para patrones comunes, como [conjugaciones](#). No obstante, puede ser conveniente definir una implementación más optimizada a mano. Es posible especificar una implementación y cualquier número de implementaciones explícitamente.

ⓘ Nota

El compilador no comprueba la exactitud de esta implementación especificada manualmente.

En el ejemplo siguiente, la declaración de una operación `SWAP`, que intercambia el estado de dos cúbits `q1` y `q2`, declara una especialización explícita para su versión adyacente y su versión controlada. Aunque las implementaciones de `Adjoint SWAP` y `Controlled SWAP` están definidas por el usuario, el compilador sigue necesitando

generar la implementación para la combinación de ambos functors (`Controlled Adjoint SWAP`, que es igual que `Adjoint Controlled SWAP`).

Q#

```
operation SWAP (q1 : Qubit, q2 : Qubit) : Unit
is Adj + Ctl {

    body ... {
        CNOT(q1, q2);
        CNOT(q2, q1);
        CNOT(q1, q2);
    }

    adjoint ... {
        SWAP(q1, q2);
    }

    controlled (cs, ...) {
        CNOT(q1, q2);
        Controlled CNOT(cs, (q2, q1));
        CNOT(q1, q2);
    }
}
```

Directivas de generación automática

Al determinar cómo generar una especialización determinada, el compilador prioriza las implementaciones definidas por el usuario. Esto significa que si una especialización adyacente está definida por el usuario y se genera automáticamente una especialización controlada, la especialización adjunta controlada se genera en función del adyacente definido por el usuario y viceversa. En este caso, ambas especializaciones están definidas por el usuario. Dado que la generación automática de una implementación adyacente está sujeta a más limitaciones, la especialización adjunta controlada tiene como valor predeterminado generar la especialización controlada de la implementación definida explícitamente de la especialización adyacente.

En el caso de la implementación de `SWAP`, la mejor opción es anexar la especialización controlada para evitar condicionar innecesariamente la ejecución de la primera y la última `CNOT` en el estado de los cúbits de control. Agregar una declaración explícita para la versión adyacente controlada que especifica una directiva de generación de *adecuada* obliga al compilador a generar la especialización adjunta controlada basada en la implementación especificada manualmente de la versión controlada en su lugar. Esta declaración explícita de una especialización generada por el compilador adopta la forma

Q#

```
controlled adjoint invert;
```

y se inserta dentro de la declaración de `SWAP`. Por otro lado, insertar la línea

Q#

```
controlled adjoint distribute;
```

fuerza al compilador a generar la especialización en función de la especialización adyacente definida (o generada). Para obtener más información, consulte esta [inferencia parcial](#) propuesta para obtener más detalles.

Para la operación `SWAP`, hay una mejor opción. `SWAP` es *autodjunto*, es decir, es su propio inverso; la implementación definida del adyacente simplemente llama al cuerpo de `SWAP` y se expresa con la directiva .

Q#

```
adjoint self;
```

Declarar la especialización adyacente de esta manera garantiza que la especialización adjunta controlada que el compilador inserte automáticamente simplemente invoca la especialización controlada.

Existen las siguientes directivas de generación y son válidas:

[Expandir tabla](#)

Especialización	Directivas
<code>body</code> especialización:	-
<code>adjoint</code> especialización:	<code>self</code> , <code>invert</code>
<code>controlled</code> especialización:	<code>distribute</code>
<code>controlled adjoint</code> especialización:	<code>self</code> , <code>invert</code> , <code>distribute</code>

Que todas las directivas de generación son válidas para una especialización adjunta controlada no es una coincidencia; siempre que los functors conmuten, el conjunto de directivas de generación válidas para implementar la especialización para una

combinación de functors siempre es la unión del conjunto de generadores válidos para cada uno.

Además de las directivas enumeradas anteriormente, la directiva `auto` es válida para todas las especializaciones excepto `body`; indica que el compilador debe elegir automáticamente una directiva de generación adecuada. Declaración

Q#

```
operation DoNothing() : Unit {  
    body ... { }  
    adjoint auto;  
    controlled auto;  
    controlled adjoint auto;  
}
```

es equivalente a

Q#

```
operation DoNothing() : Unit  
is Adj + Ctl { }
```

La anotación `is Adj + Ctl` en este ejemplo especifica las características de operación , que contienen la información sobre qué functors admite una operación determinada.

Aunque por motivos de legibilidad, se recomienda anotar cada operación con una descripción completa de sus características, el compilador inserta o completa automáticamente la anotación en función de especializaciones declaradas explícitamente. Por el contrario, el compilador también genera especializaciones que no se declaran explícitamente, pero que necesitan existir en función de las características anotadas. Decimos que la anotación dada *declara implícitamente* estas especializaciones. El compilador genera automáticamente las especializaciones necesarias si puede, seleccionando una directiva adecuada. Q# admite así la inferencia de las características de operación y las especializaciones existentes basadas en anotaciones (parciales) y especializaciones definidas explícitamente.

En cierto sentido, las especializaciones son similares a las sobrecargas individuales para el mismo invocable, con la advertencia de que ciertas restricciones se aplican a las sobrecargas que se pueden declarar.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Comentarios

Artículo • 21/03/2025

Los comentarios comienzan con dos barras diagonales (`//`) y continúan hasta el final de la línea. Estos comentarios de fin de línea pueden aparecer en cualquier parte del código fuente. Q# no admite actualmente comentarios en bloque.

Comentarios de documentación

Los comentarios que comienzan con tres barras diagonales, `///`, se tratan especialmente mediante el compilador cuando aparecen antes de una declaración de tipo o invocable. En ese caso, su contenido se toma como documentación para el tipo definido o invocable, como para otros lenguajes .NET.

En `///` comentarios, el texto que aparece como parte de la documentación de api tiene el formato [Markdown](#), con diferentes partes de la documentación indicadas por encabezados con nombre especial. Como extensión a Markdown, se pueden incluir referencias cruzadas a operaciones, funciones y tipos de estructura en Q# mediante `@<ref target>`, donde se reemplaza `<ref target>` por el nombre completo del objeto de código al que se hace referencia. Opcionalmente, un motor de documentación también puede admitir extensiones adicionales de Markdown.

Por ejemplo:

Q#

```
/// # Summary
/// Given an operation and a target for that operation,
/// applies the given operation twice.
///
/// # Input
/// ## op
/// The operation to be applied.
/// ## target
/// The target to which the operation is to be applied.
///
/// # Type Parameters
/// ## 'T
/// The type expected by the given operation as its input.
///
/// # Example
/// ```Q#
/// // Should be equivalent to the identity.
/// ApplyTwice(H, qubit);
/// ```
```

```

///
/// # See Also
/// - Microsoft.Quantum.Intrinsic.H
operation ApplyTwice<'T>(op : ('T => Unit), target : 'T) : Unit {
    op(target);
    op(target);
}

```

Q# reconoce los siguientes nombres como encabezados de comentario de documentación.

- **Resumen:** un breve resumen del comportamiento de una función o operación o el propósito de un tipo. El primer párrafo del resumen se usa para obtener información sobre el puntero. Debe ser texto sin formato.
- **Descripción:** una descripción del comportamiento de una función o operación o el propósito de un tipo. El resumen y la descripción se concatenan para formar el archivo de documentación generado para la función, la operación o el tipo. La descripción puede contener símbolos y ecuaciones con formato LaTeX en línea.
- **entrada:** una descripción de la tupla de entrada para una operación o función. Puede contener subsecciones de Markdown adicionales que indican cada elemento de la tupla de entrada.
- **salida:** una descripción de la tupla devuelta por una operación o función.
- **Parámetros de tipo:** una sección vacía que contiene una subsección adicional para cada parámetro de tipo genérico.
- **Elementos con nombre:** una descripción de los elementos con nombre en un tipo de estructura. Puede contener subsecciones adicionales de Markdown con la descripción de cada elemento con nombre.
- **Ejemplo:** un breve ejemplo de la operación, función o tipo en uso.
- **Comentarios:** prosas varias que describen algún aspecto de la operación, función o tipo.
- **Vea también:** una lista de nombres completos que indican funciones, operaciones o tipos de estructura relacionados.
- **Referencias:** una lista de referencias y citas para el elemento documentado.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Instrucciones

Artículo • 21/03/2025

Q# distingue entre instrucciones y expresiones. Q# programas constan de una mezcla de cálculos clásicos y cuánticos, y la implementación es muy similar a cualquier otro lenguaje de programación clásico. Algunas instrucciones, como los enlaces de `let` y `mutable`, son conocidas por los lenguajes clásicos, mientras que otras, tales asignaciones de cúbits, son únicas para el dominio cuántico.

Las siguientes instrucciones están disponibles actualmente en Q#:

- **instrucción Expression de**

Contiene una expresión Q# que se va a ejecutar, como una llamada a una operación. Si la última instrucción de un bloque es una instrucción de expresión, puede que tenga su punto y coma final omitido para dar al bloque el valor evaluado de la expresión contenida.

- **de declaración variable**

Define una o varias variables locales que son válidas para el resto del ámbito actual y las enlaza a los valores especificados. Las variables se pueden enlazar o declarar permanentemente para que se vuelvan a asignar más adelante. Consulte [declaraciones y reasignaciones de variables](#) para obtener más información.

- **de asignación de cúbits**

Crea instancias e inicializa cúbits, o matrices de cúbits, y los enlaza a las variables declaradas. Opcionalmente, la instrucción se puede usar con un bloque de código especificado, en el que las asignaciones de cúbits son válidas. De lo contrario, las asignaciones son válidas para el ámbito envolvente. Los cúbits se liberan automáticamente al final del ámbito adecuado. Consulte de administración de memoria cuántica para obtener más información.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Visibilidad de variables locales

Artículo • 21/03/2025

En general, los enlaces de símbolos en Q# se vuelven inoperativos al final del bloque de instrucciones en el que se producen. Sin embargo, hay algunas excepciones a esta regla.

Visibilidad de variables locales

 Expandir tabla

Ámbito	Visibilidad
Variables de bucle	Los enlaces de variables de bucle en un bucle <code>for</code> solo se definen para el cuerpo del bucle. Son inoperativos fuera del bucle.
Instrucciones <code>use</code> y <code>borrow</code>	Los enlaces de cúbits asignados en <code>use</code> e instrucciones <code>borrow</code> se definen para el cuerpo de la asignación y son inoperativos después de que finalice la instrucción. Esta acción solo se aplica a las instrucciones <code>use</code> y <code>borrow</code> que tienen un bloque de instrucciones asociado.
Expresiones <code>repeat</code>	En <code>repeat</code> expresiones, tanto los bloques como la condición se tratan como un único ámbito, es decir, los símbolos enlazados en el cuerpo son accesibles tanto en la condición como en el bloque <code>fixup</code> .
Bucles	Cada iteración de un bucle se ejecuta en su propio ámbito y todas las variables definidas se enlazan de nuevo para cada iteración.

Los enlaces de los bloques externos son visibles y se definen en bloques internos. Un símbolo solo se puede enlazar varias veces por bloque en cuyo caso el último enlace del ámbito es el que se usa para la lógica. Este escenario se conoce como "sombreado".

Las secuencias siguientes son válidas:

```
Q#

if a == b {
    ...
    let n = 5;
    ...           // n is 5
}
let n = 8;
...             // n is 8
```

y

Q#

```
if a == b {  
    ...  
    let n = 5;  
    ...           // n is 5  
} else {  
    ...  
    let n = 8;  
    ...           // n is 8  
}  
...              // n is not bound to a value
```

Q#

```
let n = 5;  
...           // n is 5  
let n = 8;  
...           // n is 8
```

y

Q#

```
let n = 8;  
if a == b {  
    ...           // n is 8  
    let n = 5;  
    ...           // n is 5  
}  
...              // n is 8 again
```

Para obtener más información, vea [declaraciones de variables y reasignaciones](#).

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Declaraciones y reasignaciones de variables

Artículo • 21/03/2025

Los valores se pueden enlazar a símbolos mediante las instrucciones `let` y `mutable`. Estos tipos de enlaces proporcionan una manera cómoda de acceder a un valor a través del identificador definido. A pesar de la terminología engañosa tomada de otros lenguajes, los identificadores declarados en un ámbito local y los valores que contienen se denominan variables. Este nombre puede ser engañoso porque las instrucciones `let` definen *identificadores de asignación única*, que son identificadores que permanecen enlazados al mismo valor durante la duración de su validez. Las variables que se pueden volver a enlazar a valores diferentes en distintos puntos del código deben declararse explícitamente como tal y se especifican mediante la instrucción `mutable`.

Q#

```
let var1 = 3;  
mutable var2 = 3;  
var2 = var2 + 1;
```

En este ejemplo, la instrucción `let` declara una variable denominada `var1` que no se puede reasignar y siempre contiene el valor `3`. La instrucción `mutable` define una variable `var2` que está enlazada temporalmente al valor `3`, pero que se puede reasignar a un valor diferente más adelante mediante una expresión de asignación, como se muestra en la última línea. Puede expresar la misma instrucción con la versión más corta `var2 += 1;`, como es habitual en otros idiomas. Para obtener más información, vea [Evaluar y reasignar instrucciones](#).

Para resumir:

- `let` se usa para crear un enlace inmutable.
- `mutable` se usa para crear un enlace mutable.
- `=` sin un `let` se usa para cambiar el valor de un enlace mutable.

Para las tres instrucciones, el lado izquierdo consta de un símbolo o una tupla de símbolos. Si el lado derecho del enlace es una tupla, esa tupla puede ser totalmente o parcialmente deconstruida tras la asignación. El único requisito de deconstrucción es que la forma de la tupla en el lado derecho coincide con la forma de la tupla de símbolos en el lado izquierdo. La tupla de símbolos puede contener tuplas anidadas o símbolos omitidos, o ambos, indicados por un carácter de subrayado. Por ejemplo:

Q#

```
let (a, (_, b)) = (1, (2, 3)); // a is bound to 1, b is bound to 3
mutable (x, y) = ((1, 2), [3, 4]); // x is bound to (1, 2), y is bound to
[3, 4]
(x, _, y) = ((5, 6), 7, [8]); // x is re-bound to (5,6), y is re-bound to
[8]
```

Para obtener más información sobre la deconstrucción mediante el operador unwrap (!), vea [Acceso a elementos para tipos de estructura](#).

Todas las asignaciones de Q# obedecer las mismas reglas de deconstrucción, incluidas, por ejemplo, asignaciones de cúbits y asignaciones de variables de bucle.

Para ambos tipos de enlaces, los tipos de las variables se deducen del lado derecho del enlace. El tipo de una variable siempre sigue siendo el mismo y una expresión de asignación no puede cambiarla. Las variables locales se pueden declarar como mutables o inmutables. Hay algunas excepciones, como las variables de bucle en `for` bucles, donde el comportamiento está predefinido y no se puede especificar. Los argumentos de función y operación siempre están enlazados inmutablemente. En combinación con la falta de tipos de referencia, como se describe en el tema [Inmutabilidad](#), estar enlazado inmutablemente significa que una función o operación llamada nunca puede cambiar ningún valor en el lado del autor de la llamada.

Dado que los estados de `Qubit` valores no se definen ni son observables desde dentro de Q#, esto no impide la acumulación de efectos secundarios cuánticos, que solo son observables a través de medidas. Para obtener más información, consulte [tipos de datos cuánticos](#).

Independientemente de cómo se enlaza un valor, los propios valores son inmutables. En concreto, esto es cierto para matrices y elementos de matriz. A diferencia de los lenguajes clásicos populares en los que las matrices suelen ser tipos de referencia, matrices en Q#, como todos los tipos, son tipos de valor e inmutables siempre; es decir, no se pueden modificar después de la inicialización. El cambio de los valores a los que acceden las variables de tipo matriz requiere construir explícitamente una nueva matriz y reasignarla al mismo símbolo. Para obtener más información, vea de inmutabilidad y [expresiones de copia y actualización](#).

Instrucciones Evaluate-and-reassign

Las instrucciones de la forma `intValue += 1;` son comunes en muchos otros lenguajes. Aquí, `intValue` debe ser una variable enlazada inmutablemente de tipo `Int`. Estas

instrucciones proporcionan una forma cómoda de concatenación si el lado derecho consiste en aplicar un operador binario y el resultado se vuelve a enlazar al argumento izquierdo del operador. Por ejemplo, este segmento de código

Q#

```
mutable counter = 0;
for i in 1 .. 2 .. 10 {
    counter += 1;
    // ...
}
```

incrementa el valor del contador `counter` en cada iteración del bucle `for` y es equivalente a

Q#

```
mutable counter = 0;
for i in 1 .. 2 .. 10 {
    counter = counter + 1;
    // ...
}
```

Existen instrucciones similares para una amplia gama de operadores de `.`. Estas expresiones evaluate-and-reassign existen para todos los operadores en los que el tipo de subexpresión de la izquierda coincide con el tipo de expresión. De forma más precisa, están disponibles para operadores lógicos y bit a bit binarios, como desplazamiento derecho e izquierdo, expresiones aritméticas, como exponente y módulo, y concatenaciones, así como expresiones de copia y actualización [expresiones de copia y actualización](#).

En el ejemplo de función siguiente se calcula la suma de una matriz de números de [Complex](#):

Q#

```
function ComplexSum(values : Complex[]) : Complex {
    mutable res = Complex(0., 0.);
    for complex in values {
        res = new Complex { Re = res.Re + complex.Re, Im = res.Im +
complex.Im };
    }
    return res;
}
```


De forma similar, la función siguiente multiplica cada elemento de una matriz con el factor especificado:

Q#

```
function Multiplied(factor : Double, array : Double[]) : Double[] {  
    mutable res = new Double[Length(array)];  
    for i in IndexRange(res) {  
        res w/= i <- factor * array[i];  
    }  
    return res;  
}
```

Para obtener más información, vea [expresiones contextuales](#), que contiene otros ejemplos en los que se pueden omitir expresiones en un contexto específico cuando el compilador puede inferir una expresión adecuada.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Administración de memoria cuántica

Artículo • 21/03/2025

Un programa siempre se inicia sin cúbits, lo que significa que no se pueden pasar valores de tipo `Qubit` como argumentos de punto de entrada. Esta restricción es intencionada, ya que el propósito de Q# es expresar y razonar sobre un programa en su totalidad. En su lugar, un programa asigna y libera cúbits, o memoria cuántica, tal como va. En este sentido, Q# modela el equipo cuántico como un montón de cúbits.

En lugar de admitir *independientes*, *asigne instrucciones* y *release* para la memoria cuántica, Q# admite la asignación de memoria cuántica en forma de instrucciones de bloque, donde solo se puede acceder a la memoria dentro del ámbito de esa instrucción de bloque. El bloque de instrucciones se puede definir implícitamente al asignar cúbits durante el ámbito actual, como se describe con más detalle en las secciones sobre las instrucciones `use` y `borrow`. Al intentar acceder a los cúbits asignados una vez finalizada la instrucción, se produce una excepción en tiempo de ejecución.

Q# tiene dos instrucciones, `use` y `borrow`, que crean instancias de valores de cúbit, matrices de cúbits o cualquier combinación de estos. Solo puede usar estas instrucciones dentro de las operaciones. Recopilan los valores de cúbit con instancias, los enlazan a las variables especificadas en la instrucción y, a continuación, ejecutan un bloque de instrucciones. Al final del bloque, las variables enlazadas salen del ámbito y ya no se definen.

Q# distingue entre la asignación de limpia y *cúbits de sucios*. Los cúbits limpios no están entrelazados y no se usan en otra parte del cálculo. Los cúbits sucios son cúbits cuyo estado es desconocido e incluso se pueden entrelazar con otras partes de la memoria del procesador cuántico.

Use (instrucción)

La instrucción `use` asigna cúbits limpios.

- La instrucción consta de la palabra clave `use` seguida de un enlace y un bloque de instrucciones opcionales.
- Si hay un bloque de instrucciones presente, los cúbits solo están disponibles dentro de ese bloque. De lo contrario, los cúbits están disponibles hasta el final del ámbito actual.

- El enlace sigue el mismo patrón que las instrucciones `let`: un solo símbolo o una tupla de símbolos, seguido de un signo igual `=` y una sola tupla o una tupla coincidente de *inicializadores*.

Los inicializadores están disponibles para un solo cúbit, indicado como `Qubit()` o una matriz de cúbits, `Qubit[n]`, donde `n` es una expresión `Int`. Por ejemplo

Q#

```
use qubit = Qubit();  
// ...  
  
use (aux, register) = (Qubit(), Qubit[5]);  
// ...  
  
use qubit = Qubit() {  
    // ...  
}  
  
use (aux, register) = (Qubit(), Qubit[5]) {  
    // ...  
}
```

Se garantiza que los cúbits estén en un estado $|0\rangle$ tras la asignación. Se liberan al final del ámbito y deben estar en un estado $|0\rangle$ al liberarse. Este requisito no se aplica al compilador, ya que esto requeriría una evaluación simbólica que resulta prohibitivamente costosa. Al ejecutarse en simuladores, se puede aplicar el requisito en tiempo de ejecución. En los procesadores cuánticos, el requisito no se puede aplicar en tiempo de ejecución; Un cúbit no seguro se puede restablecer a $|0\rangle$ a través de la transformación unitaria. Si no lo hace, se produce un comportamiento incorrecto.

La instrucción `use` asigna los cúbits del montón de bits cuánticos libres del procesador cuántico y los devuelve al montón no más adelante del final del ámbito en el que están enlazados los cúbits.

Instrucción Borrow

La instrucción `borrow` concede acceso a los cúbits que ya están asignados, pero que actualmente no están en uso. Estos cúbits pueden estar en un estado arbitrario y deben estar en el mismo estado de nuevo cuando finaliza la instrucción `borrow`. Algunos algoritmos cuánticos pueden usar cúbits sin depender de su estado exacto y sin necesidad de que no estén entrelazados con el resto del sistema. Es decir, requieren cúbits adicionales temporalmente, pero pueden asegurarse de que esos cúbits se devuelven exactamente a su estado original, independientemente del estado que era.

Si hay cúbits que están en uso pero que no se tocan durante las partes de una subrutina, esos cúbits se pueden tomar prestados para su uso por este tipo de algoritmo en lugar de asignar memoria cuántica adicional. El préstamo en lugar de asignar puede reducir significativamente los requisitos generales de memoria cuántica de un algoritmo y es un ejemplo cuántico de un equilibrio de tiempo espacial típico.

Una instrucción `borrow` sigue el mismo patrón descrito anteriormente para la instrucción `use`, con los mismos inicializadores disponibles. Por ejemplo

```
Q#  
  
borrow qubit = Qubit();  
// ...  
  
borrow (aux, register) = (Qubit(), Qubit[5]);  
// ...  
  
borrow qubit = Qubit() {  
    // ...  
}  
  
borrow (aux, register) = (Qubit(), Qubit[5]) {  
    // ...  
}
```

Los cúbits prestados están en un estado desconocido y salen del ámbito al final del bloque de instrucciones. El prestatario se compromete a abandonar los cúbits en el mismo estado que cuando fueron prestados; es decir, se espera que su estado al principio y el final del bloque de instrucciones sea el mismo.

La instrucción `borrow` recupera cúbits en uso garantizados que el programa no va a usar desde el momento en que el cúbit está enlazado hasta el último uso de ese cúbit. Si no hay suficientes cúbits disponibles para tomar prestado, los cúbits se asignan y se devuelven al montón como una instrucción `use`.

ⓘ Nota

Entre los casos de uso conocidos de cúbits sucios se implementan puertas CNOT multicontroladas que requieren muy pocos cúbits e implementaciones de incrementos. Este [documento sobre la factorización con cúbits](#) [↗] proporciona un ejemplo de un algoritmo que utiliza cúbits prestados.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones

Artículo • 21/03/2025

En el núcleo, las expresiones de Q# son literales de valor o identificadores de , donde los identificadores pueden hacer referencia a variables declaradas localmente o a los invocables declarados globalmente (actualmente no hay constantes globales en Q#). Los operadores, los combinadores y los modificadores se pueden usar para combinar estos identificadores en una variedad más amplia de expresiones.

- *Operadores* en un sentido no son más que sintaxis dedicada para determinados invocables.

Aunque Q# aún no sea lo suficientemente expresivo como para capturar formalmente las funcionalidades de cada operador en forma de declaración invocable de respaldo, esto debe corregirse en el futuro.

- *modificadores* solo se pueden aplicar a determinadas expresiones. Uno o varios modificadores se pueden aplicar a expresiones que son
 - Identificadores
 - expresiones de acceso de elementos de matriz
 - expresiones de acceso a elementos con nombre
 - una expresión entre paréntesis que es la misma que una tupla de un solo elemento. Para obtener más información, consulte [equivalencia de tupla singleton](#)). Pueden preceder a la expresión (prefijo) o seguir (postfijo) la expresión. Por lo tanto, son operadores unarios especiales que enlazan más estrecha que las llamadas de función o operación, pero menos estrechas que cualquier tipo de acceso a elementos. Concretamente, [functors](#) son modificadores de prefijo, mientras que el operador [unwrap](#) es un modificador postfijo.
- Las llamadas de acceso a funciones, operaciones y elementos se pueden ver como un tipo especial de operador, similar a los modificadores. todos están sujetos a las mismas restricciones en lo que respecta a dónde se pueden aplicar; nos referimos a ellos como *combinadores*.

La sección sobre [precedencia y asociatividad](#) contiene una lista completa [de todos los operadores](#) así como una lista completa de [de todos los modificadores y combinadores](#).

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Precedencia y asociatividad

Artículo • 21/03/2025

La precedencia y la asociatividad definen el orden en el que se aplican los operadores. Los operadores con mayor prioridad están enlazados primero a sus argumentos (operandos), mientras que los operadores con la misma prioridad se enlazan en la dirección de su asociatividad. Por ejemplo, la expresión `1+2*3` según la precedencia de suma y multiplicación es equivalente a `1+(2*3)` y `2^3^4` es igual a `2^(3^4)`, ya que la exponenciación es asociativa a la derecha.

Operadores

En la tabla siguiente se enumeran los operadores disponibles en Q#, así como su precedencia y asociatividad. También se enumeran modificadores de [adicionales y combinadores](#) y enlazan más estrictos que cualquiera de estos operadores.

 Expandir tabla

Descripción	Sintaxis	Operador	asociatividad	Precedencia
operador copy-and-update	<code>w/ <-</code>	ternario	Izquierda	1
operador de intervalo	<code>..</code>	infijo	Izquierda	2
operador condicional	<code>? \ </code>	ternario	Derecha	3
or lógico	<code>or</code>	infijo	Izquierda	4
and lógico	<code>and</code>	infijo	Izquierda	5
OR bit a bit	<code>\ \ \ </code>	infijo	Izquierda	6
XOR bit a bit	<code>^^^</code>	infijo	Izquierda	7
AND bit a bit	<code>&&&</code>	infijo	Izquierda	8
de igualdad	<code>==</code>	infijo	Izquierda	9
de desigualdad	<code>!=</code>	infijo	Izquierda	9
menor o igual que	<code><=</code>	infijo	Izquierda	10
menos de	<code><</code>	infijo	Izquierda	11
mayor o igual que	<code>>=</code>	infijo	Izquierda	11

Descripción	Sintaxis	Operador	asociatividad	Precedencia
mayor que	>	infijo	Izquierda	11
de desplazamiento a la derecha	>>>	infijo	Izquierda	12
de desplazamiento izquierdo	<<<	infijo	Izquierda	12
adición de o de concatenación	+	infijo	Izquierda	13
de resta	-	infijo	Izquierda	13
de multiplicación	*	infijo	Izquierda	14
division	/	infijo	Izquierda	14
módulo	%	infijo	Izquierda	14
de exponente	^	infijo	Derecha	15
NOT bit a bit	~~~	prefix	Derecha	16
lógico NOT	not	prefix	Derecha	16
negativa	-	prefix	Derecha	16

Las expresiones de copia y actualización deben tener necesariamente la prioridad más baja para garantizar un comportamiento coherente de la instrucción evaluate-and-reassign correspondiente . Las consideraciones similares se mantienen para el operador de intervalo para garantizar un comportamiento coherente de la expresión contextual de [correspondiente](#).

Modificadores y combinaciones

Los modificadores se pueden ver como operadores especiales que solo se pueden aplicar a determinadas expresiones. Se les puede asignar una prioridad artificial para capturar su comportamiento.

Para obtener más información, vea expresiones de .

Esta precedencia artificial se muestra en la tabla siguiente, junto con la forma en que la precedencia de los operadores y modificadores se relaciona con la forma en que se enlazan los combinadores de acceso a elementos estrictos ([,] y :: respectivamente) y los combinadores de llamadas ((,)).

Descripción	Sintaxis	Operador	asociatividad	Precedencia
del combinador de llamadas	()	n/a	Izquierda	17
functor contiguo	Adjoint	prefix	Derecha	18
functor controlado	Controlled	prefix	Derecha	18
de aplicación Unwrap	!	Postfix	Izquierda	19
de acceso a elementos con nombre	.	n/a	Izquierda	20
de acceso a elementos de Array	[]	n/a	Izquierda	20
lambda de Function	->	n/a	Derecha	21
lambda operation	=>	n/a	Derecha	21

Para ilustrar las implicaciones de las precedencias asignadas, supongamos que tiene una operación unitaria `DoNothing` (tal como se define en [declaraciones de especialización](#)), un `GetStatePrep` invocable que devuelve una operación unitaria y una matriz `algorithms` que contiene elementos de tipo `Algorithm` definidos como se indica a continuación.

```
Q#
struct Algorithm {
    Register : Qubit[],
    Initialize : Transformation,
    Apply : Transformation,
}
```

Las expresiones siguientes, a continuación, son válidas:

```
Q#
GetStatePrep()(arg)
Adjoint DoNothing()
Controlled Adjoint DoNothing(cs, ())
Controlled algorithms[0].Apply!(cs, _)
algorithms[0].Register![i]
```

Si examina las precedencias definidas en la tabla anterior, puede ver que los paréntesis alrededor de `(Transformation(GetStatePrep()))` son necesarios para que el operador `unwrap` posterior se aplique al valor de `Transformation` en lugar de a la operación devuelta. Sin embargo, los paréntesis no son necesarios en `GetStatePrep()(arg)`; Las funciones se aplican de izquierda a derecha, por lo que esta expresión es equivalente a

`(GetStatePrep())(arg)`. Las aplicaciones de functor tampoco requieren paréntesis alrededor de ellas para invocar la especialización correspondiente, ni matrices ni expresiones de acceso a elementos con nombre. Por lo tanto, la expresión `arr2D[i][j]` es perfectamente válida, como es `algorithms[0]::Register![i]`.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones de copia y actualización

Artículo • 22/12/2024

Para reducir la necesidad de enlaces mutables, Q# admite expresiones de copia y actualización para matrices, lo que permite acceder a los elementos a través de un índice o intervalo de índices.

Las expresiones copy-and-update crean una instancia de una nueva matriz con todos los elementos establecidos en el valor correspondiente de la matriz original, excepto los determinados elementos especificados, que se establecen en los definidos en el lado derecho de la expresión. Se construyen utilizando un operador ternario `w/ <-`; la sintaxis `w/` debe leerse como la notación corta que se usa habitualmente para "with":

Q#

```
original w/ itemAccess <- modification
```

donde `original` es una expresión de matriz, `itemAccess` es cualquier expresión válida para la segmentación de matriz y `modification` es el nuevo valor o valores.

Concretamente, la expresión `itemAccess` puede ser de tipo `Int` o `Range`. Si `itemAccess` es un valor de tipo `Int`, el tipo de `modification` debe coincidir con el tipo de elemento de la matriz. Si `itemAccess` es un valor de tipo `Range`, el tipo de `modification` debe ser el mismo que el tipo de matriz.

Por ejemplo, si `arr` contiene una matriz `[0, 1, 2, 3]`,

- `arr w/ 0 <- 10` es la matriz `[10, 1, 2, 3]`.
- `arr w/ 2 <- 10` es la matriz `[0, 1, 10, 3]`.
- `arr w/ 0..2..3 <- [10, 12]` es la matriz `[10, 1, 12, 3]`.

En términos de precedencia, el operador copy-and-update es asociativo a la izquierda y tiene prioridad más baja, y, en particular, menor prioridad que el operador de intervalo (`..`) o el operador condicional ternario (`? |`). La asociatividad izquierda elegida permite encadenar fácilmente expresiones de copia y actualización:

Q#

```
let model = ArrayConstructor()  
  w/ 1 <- alpha  
  w/ 3 <- gamma  
  w/ 5 <- epsilon;
```

En cuanto a cualquier operador que construya una expresión del mismo tipo que la expresión de la izquierda implicada, está disponible la instrucción `evaluate-and-reassign` correspondiente [instrucción evaluate-and-reassign](#). Las dos instrucciones siguientes, por ejemplo, logran lo siguiente: la primera instrucción declara una variable mutable `arr` y la enlaza al valor predeterminado de una matriz de enteros. A continuación, la segunda instrucción compila una nueva matriz con el primer elemento (con el índice 0) establecido en 10 y lo reasigna a `arr`.

Q#

```
mutable arr = [0, size = 3]; // arr contains [0, 0, 0]
arr w/= 0 <- 10;           // arr contains [10, 0, 0]
```

La segunda instrucción es simplemente breve para la sintaxis más detallada `arr = arr w/ 0 <- 10;`.

Las expresiones de copia y actualización permiten la creación eficaz de nuevas matrices basadas en las existentes. La implementación de expresiones de copia y actualización evita copiar toda la matriz duplicando solo las partes necesarias para lograr el comportamiento deseado y realiza una modificación local si es posible. Por lo tanto, las inicializaciones de matriz no incurren en sobrecarga adicional debido a la inmutabilidad.

El espacio de nombres `Std.Arrays` proporciona un arsenal de herramientas cómodas para la creación y manipulación de matrices.

Las expresiones de copia y actualización son una manera cómoda de construir nuevas matrices sobre la marcha; La siguiente expresión, por ejemplo, se evalúa como una matriz con todos los elementos establecidos en `PauliI`, excepto el elemento en el índice `i`, que se establece en `PauliZ`:

Q#

```
[PauliI, size = n] w/ i <- PauliZ
```

Comentarios

¿Le ha resultado útil esta página?

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Expresiones condicionales

Artículo • 21/03/2025

Las expresiones condicionales constan de tres subexpresiones, donde la subexpresión de la izquierda es de tipo `Bool` y determina cuál de las otras dos subexpresiones se evalúa. Son de la forma

Q#

```
cond ? ifTrue | ifFalse
```

En concreto, si `cond` se evalúa como `true`, la expresión condicional se evalúa como la expresión `ifTrue`; de lo contrario, se evalúa como la expresión `ifFalse`. La otra expresión (la expresión `ifFalse` y `ifTrue`, respectivamente) nunca se evalúa, al igual que las ramas de una instrucción `if`. Por ejemplo, en una expresión `a == b ? C(qs) | D(qs)`, si `a` es igual a `b`, se invoca el `C` invocable. De lo contrario, se invoca `D`.

Los tipos del `ifTrue` y la expresión de `ifFalse` tienen que tener un tipo base común. Independientemente de cuál produzca en última instancia el valor al que se evalúa la expresión, su tipo siempre coincide con el tipo base determinado.

Por ejemplo, si

- `Op1` es de tipo `Qubit[] => Unit is Adj`
- `Op2` es de tipo `Qubit[] => Unit is Ctl`
- `Op3` es de tipo `Qubit[] => Unit is Adj + Ctl`

y luego

- `cond ? Op1 | Op2` es de tipo `Qubit[] => Unit`
- `cond ? Op1 | Op3` es de tipo `Qubit[] => Unit is Adj`
- `cond ? Op2 | Op3` es de tipo `Qubit[] => Unit is Ctl`

Para obtener más información, consulte [subtipado](#).

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones comparativas

Artículo • 05/03/2024

Comparaciones de igualdad

Actualmente, las *comparaciones de igualdad* (`==`) y las *comparaciones de desigualdad* (`!=`) se limitan a los siguientes tipos de datos: `Int`, `BigInt`, `Double`, `String`, `Bool`, `Result`, `Pauli` y `Qubit`. Actualmente no se admiten comparaciones de igualdad de tipos y llamadas definidos por el usuario.

La comparación de igualdad de valores de tipo `Qubit` evalúa si las dos expresiones identifican el mismo cúbit. No hay ninguna noción de un estado cuántico en Q#; la comparación de igualdad en particular *no* tiene acceso, mide ni modifica el estado cuántico de los cúbits.

Las comparaciones de igualdad de valores `Double` pueden ser engañosas debido a los efectos de redondeo. Por ejemplo, la siguiente comparación se evalúa como `false` debido a errores de redondeo: `49.0 * (1.0/49.0) == 1.0`.

La comparación de igualdad de matrices y tuplas son compatibles con comparaciones de sus elementos y solo se admiten si todos sus tipos anidados admiten la comparación de igualdad.

Se admite la comparación de igualdad de los intervalos cerrados y se consideran iguales dos intervalos si producen la misma secuencia de enteros. Por ejemplo, los dos intervalos siguientes

Q#

```
let r1 = 0..2..5; // generates the sequence 0,2,4
let r2 = 0..2..4; // generates the sequence 0,2,4
```

se consideran iguales. No se admite la comparación de igualdad de intervalos abiertos.

Comparación cuantitativa

Los operadores *menor que* (`<`), *menor o igual que* (`<=`), *mayor que* (`>`) y *mayor o igual que* (`>=`) definen comparaciones cuantitativas. Solo se pueden aplicar a los tipos de datos que admiten dichas comparaciones, es decir, los mismos tipos de datos que también admiten [expresiones aritméticas](#).

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones lógicas

Artículo • 21/03/2025

Los operadores lógicos se expresan como palabras clave. Q# admite los operadores lógicos estándar *AND* (`and`), *OR* (`or`) y *NOT* (`not`). Actualmente, no hay un operador para un *XOR* lógico. Todos estos operadores actúan en operandos de tipo `Bool` y dan como resultado una expresión de tipo `Bool`. Como es habitual en la mayoría de los lenguajes, la evaluación de *AND* y *OR* cortocircuitos, lo que significa si la primera expresión de *OR* se evalúa como `true`, la segunda expresión no se evalúa y la misma contiene si la primera expresión de *AND* se evalúa como `false`. El comportamiento de las expresiones condicionales en un sentido es similar, en que solo se evalúa la condición y una de las dos expresiones.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones bit a bit

Artículo • 21/03/2025

Los operadores bit a bit se expresan como tres caracteres que no son de letra. Además de las versiones bit a bit para *AND* (`&&&`), *OR* (`|||`) y *NOT* (`~~~`), también existe un *XOR bit a bit* (`^^^`). Esperan operandos de tipo `Int` o `BigInt`, y para operadores binarios, el tipo de ambos operandos tiene que coincidir. El tipo de toda la expresión es igual al tipo de los operandos.

Además, existen operadores de desplazamiento de izquierda y derecha (`<<<` y `>>>` respectivamente), multiplicando o dividiendo la expresión de lado izquierdo (lhs) dada por poderes de dos. La expresión `lhs <<< 3` desplaza la representación de bits de `lhs` por tres, lo que significa que `lhs` se multiplica por 2^3 , siempre que todavía esté dentro del intervalo válido para el tipo de datos de `lhs`. Las lhs pueden ser de tipo `Int` o `BigInt`. La expresión del lado derecho siempre debe ser de tipo `Int`. La expresión resultante es del mismo tipo que el operando lhs.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones aritméticas

Artículo • 21/03/2025

Los operadores aritméticos son suma (+), resta (-), multiplicación (*), división (/), negación (-) y exponenciación (^). Se pueden aplicar a operandos de tipo `Int`, `BigInt` o `Double`. Además, para los tipos enteros (`Int` y `BigInt`), hay disponible un operador que calcula el módulo (%).

Para los operadores binarios, el tipo de ambos operandos debe coincidir, excepto para la exponenciación; un exponente para un valor de tipo `BigInt` debe ser de tipo `Int`. El tipo de toda la expresión coincide con el tipo del operando izquierdo. Para la exponenciación de `Int` y `BigInt`, el comportamiento no está definido si el exponente es negativo o requiere más de 32 bits para representar (es decir, si es mayor que 2147483647).

Division and modulus for values of type `Int` and `BigInt` follow the following behavior for negative numbers:

[Expandir tabla](#)

A	B	A / B	A % B
5	2	2	1
5	-2	-2	1
-5	2	-2	-1
-5	-2	2	-1

Es decir, `a % b` siempre tiene el mismo signo que `a` y `b * (a / b) + a % b` siempre es igual a `a`.

Q# no admite conversiones automáticas entre tipos de datos aritméticos ni ningún otro tipo de datos para ese asunto. Esto es importante especialmente para el tipo de datos `Result` y facilita la restricción de cómo se puede propagar la información en tiempo de ejecución. Tiene la ventaja de evitar errores accidentales, como los relacionados con la pérdida de precisión.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Concatenación

Artículo • 21/03/2025

Las concatenaciones se admiten para los valores de tipo `String` y matrices. En ambos casos, se expresan a través del operador `+`. Por ejemplo, `"Hello " + "world!"` se evalúa como `"Hello world!"` y `[1, 2, 3] + [4, 5, 6]` se evalúa como `[1, 2, 3, 4, 5, 6]`.

La concatenación de dos matrices requiere que ambas matrices sean del mismo tipo. Este requisito difiere de la construcción de un literal de matriz donde el compilador determina un tipo base común para todos los elementos de matriz. Esta diferencia se debe a que las matrices se tratan como invariables. El tipo de toda la expresión coincide con el tipo de los operandos.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Aplicación de funtor

Artículo • 21/03/2025

Los funtores son fábricas que permiten acceder a implementaciones de especialización concretas de un invocable. Q# actualmente admite dos funtores; el `Adjoint` y el `Controlled`, ambos se pueden aplicar a las operaciones que proporcionan las especializaciones necesarias.

Los funtores `Controlled` y `Adjoint` conmutan; Si `ApplyUnitary` es una operación que admite ambos funtores, no hay ninguna diferencia entre `Controlled Adjoint ApplyUnitary` y `Adjoint Controlled ApplyUnitary`. Ambos tienen el mismo tipo y, tras la invocación, ejecute la implementación definida para la especialización de `controlled adjoint`.

Functor adyacente

Si la operación `ApplyUnitary` define una transformación unitaria U del estado cuántico, `Adjoint ApplyUnitary` accede a la implementación de $U^\dagger$. El functor `Adjoint` es su propio inverso, ya que $(U^\dagger)^\dagger = U$ por definición. Por ejemplo, `Adjoint Adjoint ApplyUnitary` es igual que `ApplyUnitary`.

La expresión `Adjoint ApplyUnitary` es una operación del mismo tipo que `ApplyUnitary`; tiene el mismo argumento y tipo de valor devuelto y admite los mismos funtores. Al igual que cualquier operación, se puede invocar con un argumento de tipo adecuado. La expresión siguiente aplica la especialización adyacente de `ApplyUnitary` a un argumento `arg`:

```
Q#
```

```
Adjoint ApplyUnitary(arg)
```

Functor controlado

Para una operación `ApplyUnitary` que define una transformación unitaria U del estado cuántico, `Controlled ApplyUnitary` accede a la implementación que aplica U condicional en todos los cúbits de una matriz de cúbits de control que se encuentra en el estado $|1\rangle$.

La expresión `Controlled ApplyUnitary` es una operación con el mismo tipo de valor devuelto y [características de operación](#) que `ApplyUnitary`, lo que significa que admite

los mismos funtores. Toma un argumento de tipo `(Qubit[], <TIn>)`, donde se debe reemplazar `<TIn>` por el tipo de argumento de `ApplyUnitary`, teniendo en cuenta [equivalencia de tupla singleton](#).

[Expandir tabla](#)

Operación	Tipo de argumento	Tipo de argumento controlado
X	<code>Qubit</code>	<code>(Qubit[], Qubit)</code>
INTERCAMBIO	<code>(Qubit, Qubit)</code>	<code>(Qubit[], (Qubit, Qubit))</code>

Concretamente, si `cs` contiene una matriz de cúbits, `q1` y `q2` son dos cúbits, y la operación `SWAP` es la definida [aquí](#), la expresión siguiente intercambia el estado de `q1` y `q2` si todos los cúbits de `cs` están en el estado $|1\rangle$:

Q#

```
Controlled SWAP(cs, (q1, q2))
```

⚠ Nota

La aplicación condicional de una operación basada en los cúbits de control que se encuentren en un estado distinto del $|1\rangle$ se puede lograr aplicando la transformación adyacente adecuada a los cúbits de control antes de la invocación y aplicando los inversos después. Al condicionar la transformación en todos los cúbits de control que se encuentra en el estado $|0\rangle$, por ejemplo, se puede lograr aplicando la operación de `X` antes y después. Esto se puede expresar convenientemente mediante una conjugación. Sin embargo, el detalle de dicha construcción puede merecer compatibilidad adicional con una sintaxis más compacta en el futuro.

Comentarios

¿Le ha resultado útil esta página?

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Acceso a elementos

Artículo • 21/03/2025

Q# admite el acceso de elementos para los elementos de matriz y para los tipos de `struct`. En ambos casos, el acceso es de solo lectura; El valor no se puede cambiar sin crear una nueva instancia mediante una expresión copy-and-update .

Acceso a elementos de matriz y segmentación de matrices

Dada una expresión de matriz y una expresión de tipo `Int` o `Range`, se puede formar una nueva expresión mediante el operador de acceso de elemento de matriz que consta de `[` y `]`.

Si la expresión dentro de los corchetes es de tipo `Int`, la nueva expresión contiene el elemento de matriz en ese índice. Por ejemplo, si `arr` es de tipo `Double[]` y contiene cinco o más elementos, `arr[4]` es una expresión de tipo `Double`.

Si la expresión dentro de los corchetes es de tipo `Range`, la nueva expresión contiene una matriz de todos los elementos indizado por el `Range` especificado. Si el `Range` está vacío, la matriz resultante está vacía. Por ejemplo

Q#

```
let arr = [10, 11, 36, 49];
let ten = arr[0]; // contains the value 10
let odds = arr[1..2..4]; // contains the value [11, 49]
let reverse = arr[...-1...]; // contains the value [49, 36, 11, 10]
```

En la última línea del ejemplo, se ha omitido el valor inicial y final del intervalo para mayor comodidad. Para obtener más información, vea [expresiones contextuales](#).

Si la expresión de matriz no es un identificador simple, debe incluirse entre paréntesis para extraer un elemento o un segmento. Por ejemplo, si `arr1` y `arr2` son matrices de enteros, un elemento de la concatenación se expresaría como `(arr1 + arr2)[13]`. Para obtener más información, vea [Precedencia y asociatividad](#).

Todas las matrices de Q# se basan en cero, es decir, el primer elemento de una matriz `arr` siempre es `arr[0]`. Se produce una excepción en tiempo de ejecución si el índice o uno de los índices usados para la segmentación está fuera de los límites de la matriz, por ejemplo, si es menor que cero o mayor o igual que la longitud de la matriz.

Acceso a elementos para tipos de estructura

Se puede acceder a los elementos contenidos de una `struct` de dos maneras:

- Por referencia, mediante un punto (`.`)
- Por deconstrucción, mediante el operador `unwrap` (`!`).

Por ejemplo, dada la estructura `Complex`

Q#

```
struct Complex {  
    Real : Double,  
    Imaginary : Double,  
}  
  
// create an instance of type Complex  
let complex = Complex(1., 0.);  
  
// item access via deconstruction  
let (re, _) = complex!;  
  
// item access via name reference  
let im = complex.Imaginary;
```

El acceso a través de la deconstrucción usa el operador `unwrap` (`!`). El operador `unwrap` devuelve una tupla de todos los elementos contenidos, donde la forma de la tupla coincide con la definida en la declaración y una tupla de un solo elemento es equivalente al propio elemento (vea [esta sección](#)).

El operador `!` tiene una prioridad [inferior](#) que ambos operadores de acceso a elementos, pero mayor prioridad que cualquier otro operador. Para obtener una lista completa de las precedencias, consulte [precedencia y asociatividad](#).

Para obtener más información sobre la sintaxis de deconstrucción, vea [Declaraciones y reasignaciones de variables](#).

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Expresiones de llamada

Artículo • 21/03/2025

Las expresiones de llamada son una parte importante de cualquier lenguaje de programación. Las llamadas de operación y función, al igual que [aplicaciones parciales](#), se pueden usar como expresión en cualquier lugar, siempre y cuando el valor devuelto sea de un tipo adecuado.

La utilidad de las operaciones de llamada en este formato reside principalmente en la depuración, y estas llamadas de operación son una de las construcciones más comunes en cualquier programa de Q#. Al mismo tiempo, solo se puede llamar a las operaciones desde otras operaciones y no desde dentro de las funciones. Para obtener más información, consulte también [qubits](#).

Con los invocables siendo valores de primera clase, las expresiones de llamada son una forma genérica de admitir patrones que no son lo suficientemente comunes como para merecer su propia construcción de lenguaje dedicado, o la sintaxis dedicada aún no se ha introducido por otras razones. Algunos ejemplos de métodos de biblioteca que sirven para ese propósito son `ApplyIf`, que invoca una operación condicional en un bit clásico que se establece; `ApplyToEach`, que aplica una operación determinada a cada elemento de una matriz; y `ApplyWithInputTransformation`, como se muestra en el ejemplo siguiente.

Q#

```
operation ApplyWithInputTransformation<'TArg, 'TIn>(
    fn : 'TIn -> 'TArg,
    op : 'TArg => Unit,
    input : 'TIn
) : Unit {

    op(fn(input));
}
```

`ApplyWithInputTransformation` toma una función `fn`, una operación `op` y un valor de `input` como argumentos y, a continuación, aplica la función dada a la entrada antes de invocar la operación dada con el valor devuelto por la función.

Para que el compilador genere automáticamente las especializaciones para admitir [funtores concretos](#), normalmente requiere que las operaciones llamadas también admitan esos funtores. Las dos excepciones son llamadas en bloques externos de [conjugaciones](#), que siempre necesitan admitir el functor de `Adjoint`, pero nunca

necesitan admitir el functor de `Controlled` y las operaciones autodjuntas, que admiten el functor de `Adjoint` sin imponer requisitos adicionales en las llamadas individuales.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Devoluciones y finalización

Artículo • 21/03/2025

Hay dos expresiones disponibles que concluyen la ejecución de la subrutina actual o el programa; `return` y las expresiones de `fail`. Por lo general, los invocables pueden finalizar su ejecución antes de ejecutar todas sus instrucciones con una expresión `return` o `fail`. Una expresión `return` acaba de finalizar la ejecución del actual invocable, mientras que un `fail` finaliza la ejecución de todo el programa y produce un error en tiempo de ejecución.

Expresión de devolución

La expresión `return` sale del actual invocable y devuelve el control al destinatario. Cambia el contexto de la ejecución al extraer un marco de pila.

La expresión siempre devuelve un valor al contexto del destinatario; consta de la palabra clave `return`, seguida de una expresión del tipo adecuado. El valor devuelto se evalúa antes de realizar cualquier acción de terminación y se devuelve el control. Las acciones de terminación incluyen, por ejemplo, limpiar y liberar cúbits asignados dentro del contexto del invocable. Cuando se ejecuta en un simulador o validador, las acciones de terminación suelen incluir comprobaciones relacionadas con el estado de esos cúbits. Por ejemplo, pueden comprobar si están adecuadamente desenredados de todos los cúbits que permanecen activos.

La expresión `return` al final de un invocable que devuelve un valor de `Unit` se puede omitir. En ese caso, el control se devuelve automáticamente cuando se completan todas las instrucciones y todas las acciones de terminación realizadas. Los invocables pueden contener varias expresiones de `return`, aunque no se puede generar automáticamente la implementación adyacente para las operaciones que contienen varias expresiones de `return`.

Por ejemplo

```
Q#
```

```
return 1;
```

o

```
Q#
```

```
return ();
```

Expresión fail

La expresión `fail` finaliza completamente el cálculo. Corresponde a un error irrecuperable que anula el programa.

Consta de la palabra clave `fail`, seguida de una expresión de tipo `String`. El `String` debe proporcionar información sobre el error detectado.

Por ejemplo

Q#

```
fail "Impossible state reached";
```

o bien, mediante una cadena interpolada ,

Q#

```
fail $"Syndrome {syn} is incorrect";
```

Además del `String` dado, una expresión `fail` recopila y permite la recuperación de información sobre el estado del programa. Esto facilita el diagnóstico y la solución del origen del error y requiere compatibilidad con el entorno de ejecución y el firmware en ejecución, lo que puede variar entre distintos destinos.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Iteraciones

Artículo • 21/03/2025

Los bucles que recorren en iteración una secuencia de valores se expresan como bucles `for` en Q#. Un bucle `for` en Q# no se interrumpe en función de una condición, sino que corresponde a una iteración, o lo que a menudo se expresa como `foreach` o `iter` en otros lenguajes. Actualmente hay dos tipos de datos en Q# que admiten iteración: matrices y intervalos.

La expresión consta de la palabra clave `for`, seguida de un símbolo o tupla de símbolos, la palabra clave `in`, una expresión de matriz o `Range` tipo y un bloque de instrucciones.

El bloque de instrucciones (el cuerpo del bucle) se ejecuta repetidamente, con una o varias variables de bucle enlazadas a cada valor del intervalo o matriz. Las mismas reglas de deconstrucción se aplican a las variables de bucle definidas que a cualquier otra asignación de variables, como enlaces en `let`, `mutable`, `use` e instrucciones `borrow`. Las propias variables de bucle están enlazadas inmutablemente, no se pueden reasignar dentro del cuerpo del bucle y salir del ámbito cuando finaliza el bucle. Expresión sobre la que se recorre en iteración el bucle antes de entrar en el bucle y no cambia mientras se ejecuta el bucle.

Este escenario se muestra en el ejemplo siguiente. Supongamos que `qubits` es un valor de tipo `Qubit[]` y, a continuación,

Q#

```
for qubit in qubits {
    H(qubit);
}

mutable results : (Int, Result)[] = [];
for index in 0 .. Length(qubits) - 1 {
    results += [(index, M(qubits[index]))];
}

mutable accumulated = 0;
for (index, measured) in results {
    if measured == One {
        accumulated += 1 <<< index;
    }
}
```

Restricciones específicas del destino

Dado que no hay ningún `break` o `continue` primitivos en Q#, la longitud del bucle se conoce en cuanto se conoce el valor de iteración. Por lo tanto, los bucles `for` se pueden ejecutar en todo el hardware cuántico.

Comentarios

¿Le ha resultado útil esta página?



[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Bucles condicionales

Artículo • 21/03/2025

Al igual que la mayoría de los lenguajes de programación clásicos, Q# admite bucles que se interrumpen en función de una condición: bucles para los que se desconoce el número de iteraciones y pueden variar de la ejecución a la ejecución. Dado que la secuencia de instrucciones se desconoce en tiempo de compilación, el compilador controla estos bucles condicionales de una manera concreta en un entorno de ejecución cuántico.

📘 Importante

restricciones de hardware de Quantum

Los bucles que se interrumpen en función de una condición son difíciles de procesar en hardware cuántico *si la condición depende de los resultados de medición*, ya que la longitud de la secuencia de instrucciones que se va a ejecutar no se conoce con antelación.

A pesar de su presencia común en determinadas clases de algoritmos cuánticos, el hardware actual aún no proporciona compatibilidad nativa con estos tipos de construcciones de flujo de control. La ejecución de estos tipos de bucles en hardware cuántico puede admitirse en el futuro mediante la imposición de un número máximo de iteraciones o a medida que esté disponible otra compatibilidad con hardware. Sin embargo, los simuladores cuánticos ejecutan bucles basados en medidas.

Compilación de bucles

Siempre que la condición no dependa de las medidas cuánticas, los bucles condicionales se procesan con una compilación Just-In-Time antes de enviar la secuencia de instrucciones al procesador cuántico. En concreto, el uso de bucles condicionales dentro de las funciones no es problemático, ya que el código dentro de las funciones siempre se puede ejecutar en hardware convencional (no cuántico). Q#, por lo tanto, admite el uso de bucles `while` tradicionales dentro de las funciones.

Repetir expresión

Al ejecutar programas en simuladores cuánticos, Q# permite expresar el flujo de control que depende de los resultados de las medidas cuánticas. Esta funcionalidad permite implementaciones probabilísticas que pueden reducir significativamente los costos computacionales. Un ejemplo común es el *patrón de repetición hasta el éxito*, que repite un cálculo hasta una determinada condición, que normalmente depende de una medida, se cumple. Estos `repeat` bucles se usan ampliamente en clases concretas de algoritmos cuánticos. Q# por lo tanto, tiene una construcción de lenguaje dedicada para expresarlas, a pesar de que siguen planteando un desafío para la ejecución en hardware cuántico.

La expresión `repeat` tiene la siguiente forma

Q#

```
repeat {  
    // ...  
}  
until condition  
fixup {  
    // ...  
}
```

o alternativamente

Q#

```
repeat {  
    // ...  
}  
until condition;
```

donde `condition` es una expresión arbitraria de tipo `Bool`.

El bucle `repeat` ejecuta un bloque de instrucciones antes de evaluar una condición. Si la condición se evalúa como true, el bucle se cierra. Si la condición se evalúa como false, se ejecuta un bloque adicional de instrucciones definidas como parte de un bloque opcional de `fixup`, si está presente, antes de escribir la siguiente iteración del bucle.

Bucle While

Un bucle más conocido para los cálculos clásicos es el bucle `while`, que consta de la palabra clave `while`, una expresión de tipo `Bool` y un bloque de instrucciones. Por ejemplo, si `arr` es una matriz de enteros positivos,

Q#

```
mutable (item, index) = (-1, 0);  
while index < Length(arr) && item < 0 {  
    item = arr[index];  
    index += 1;  
}
```

El bloque de instrucciones se ejecuta siempre que la condición se evalúe como `true`.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Ramificación condicional.

Artículo • 21/03/2025

La bifurcación condicional se expresa en forma de expresiones `if`. Una expresión `if` consta de una cláusula `if`, seguida de cero o más cláusulas `elif` y, opcionalmente, un bloque `else`. Cada cláusula sigue el patrón

Q#

```
keyword condition {  
    <statements>  
}
```

donde `keyword` se reemplaza por `if` o `elif` respectivamente, `condition` es una expresión de tipo `Bool` y `<statements>` se va a reemplazar por cero o más instrucciones. El `else`-block opcional consta de la palabra clave `else` seguido de cero o más instrucciones entre llaves, `{ }`.

Primer bloque para el que se ejecutará el `condition` se evalúa como `true`. El bloque `else`, si está presente, se ejecuta si ninguna de las condiciones se evalúa como `true`. El bloque se ejecuta en su propio ámbito, lo que significa que los enlaces realizados como parte del bloque no son visibles después de que finalice el bloque.

Por ejemplo, supongamos que `qubits` es el valor de tipo `Qubit[]` y `r1` y `r2` son de tipo `Result`,

Q#

```
if r1 == One {  
    let q = qubits[0];  
    H(q);  
}  
elif r2 == One {  
    let q = qubits[1];  
    H(q);  
}  
else {  
    H(qubits[2]);  
}
```

También puede expresar la bifurcación simple en forma de una expresión condicional .

Restricciones específicas del destino

La estrecha integración entre las construcciones de flujo de control y los cálculos cuánticos supone un desafío para el hardware cuántico actual. Algunos procesadores cuánticos no admiten la bifurcación en función de los resultados de la medición. Por lo tanto, la comparación de valores de tipo `Result` siempre producirá un error de compilación para Q# programas destinados a ejecutarse en dicho hardware.

Otros procesadores cuánticos admiten tipos específicos de bifurcación en función de los resultados de la medición. Las expresiones de `if` más generales admitidas en Q# se compilan en instrucciones adecuadas que se pueden ejecutar en dichos procesadores. Las restricciones impuestas son que los valores de tipo `Result` solo se pueden comparar como parte de la condición dentro de `if` expresiones en operaciones. Además, los bloques de ejecución condicional no pueden contener expresiones `return` ni actualizar variables mutables declaradas fuera de ese bloque.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Conjugaciones

Artículo • 21/03/2025

Las conjugaciones son comunes en los cálculos cuánticos. En términos matemáticos, son patrones de la forma $U^\dagger V U$ para dos transformaciones unitarias U y V . Ese patrón es relevante debido a las peculiaridades de la memoria cuántica: los cálculos crean correlaciones cuánticas o *entrelazamiento*, para usar los recursos únicos de quantum. Sin embargo, esto también significa que una vez que una subrutina ya no necesita sus cúbits, esos cúbits no se pueden restablecer y liberar fácilmente, ya que observar su estado afectaría al resto del sistema. Por ese motivo, el efecto de un cálculo anterior normalmente debe invertirse antes de liberar y reutilizar la memoria cuántica.

Q# por lo tanto, tiene una construcción dedicada para expresar cálculos que requieren dicha limpieza. Las expresiones constan de dos bloques de código, uno que contiene la implementación de U y otro que contiene la implementación de V . El *de* sin calcular (es decir, la aplicación de $U^\dagger$) se realiza automáticamente como parte de la expresión.

La expresión toma la forma

Q#

```
within {  
    <statements>  
}  
apply {  
    <statements>  
}
```

donde `<statements>` se reemplaza por cualquier número de instrucciones que definen la implementación de U y V respectivamente. Ambos bloques pueden contener cálculos clásicos arbitrarios, aparte de las restricciones habituales para generar automáticamente versiones adyacentes que se aplican al bloque `within`. Es posible que las variables enlazadas inmutablemente usadas como parte del bloque `within` no se vuelvan a asignar como parte del bloque `apply`.

Comentarios

¿Le ha resultado útil esta página?

☐ Sí

☐ No

[Proporcionar comentarios sobre el producto](#) [↗](#) | [Obtener ayuda en Microsoft Q&A](#)

Cierres

Artículo • 21/03/2025

Los cierres son invocables que capturan variables del entorno envolvente. Se pueden crear cierres de función y operación. Se puede crear un cierre de operación dentro de una función, pero solo se puede aplicar en una operación.

Q# tiene dos mecanismos para crear cierres: expresiones lambda y aplicación parcial.

Expresiones lambda

Una expresión lambda crea una función o operación anónimas. La sintaxis básica es una tupla de símbolos para enlazar los parámetros, una flecha (`->` para una función y `=>` para una operación) y una expresión que se va a evaluar cuando se aplica.

Q#

```
// Function that captures 'x':  
y -> x + y  
  
// Operation that captures 'qubit':  
deg => Rx(deg * PI() / 180.0, qubit)  
  
// Function that captures nothing:  
(x, y) -> x + y
```

Parámetros

Los parámetros se enlazan mediante una tupla de símbolos idéntica al lado izquierdo de una instrucción de declaración de variable de . El tipo de la tupla del parámetro es implícito. No se admiten anotaciones de tipo; Si se produce un error en la inferencia de tipos, es posible que tenga que crear una declaración invocable de nivel superior y usar la aplicación parcial en su lugar.

Variables de captura mutable

No se pueden capturar variables mutables. Si solo necesita capturar el valor de una variable mutable al instante en que se crea la expresión lambda, puede crear una copia inmutable:

Q#

```
// ERROR: 'variable' cannot be captured.
mutable variable = 1;
let f = () -> variable;

// OK.
let value = variable;
let g = () -> value;
```

Características

Las características de una operación anónima se deducen en función de las aplicaciones de la expresión lambda. Si la expresión lambda se usa con una aplicación de functor o en un contexto que espera una característica, se deduce que la expresión lambda tiene esa característica. Por ejemplo:

```
Q#

operation NoOp(q : Qubit) : Unit is Adj {}
operation Main() : Unit {
    use q = Qubit();
    let foo = () => NoOp(q);
    foo(); // Has type Unit => Unit with no characteristics

    let bar = () => NoOp(q);
    Adjoint bar(); // Has type Unit => Unit is Adj
}
```

Si necesita características diferentes para una expresión lambda de operación a la que se infirió, deberá crear en su lugar una declaración de operación de nivel superior.

Aplicación parcial

La aplicación parcial es una abreviatura conveniente para aplicar algunos argumentos, pero no todos, a los que se puede llamar. La sintaxis es la misma que una expresión de llamada, pero los argumentos sin aplicar se reemplazan por `_`. Conceptualmente, la aplicación parcial es equivalente a una expresión lambda que captura los argumentos aplicados y toma los argumentos no aplicados como parámetros.

Por ejemplo, dado que `f` es una función y `o` es una operación, y la variable capturada `x` es inmutable:

Aplicación parcial	Expresión lambda
<code>f(x, _)</code>	<code>a -> f(x, a)</code>
<code>o(x, _)</code>	<code>a => o(x, a)</code>
<code>f(_, (1, _))</code>	<code>(a, b) -> f(a, (1, b))^[1]</code>
<code>f((_, _, x), (1, _))</code>	<code>((a, b), c) -> f((a, b, x), (1, c))</code>

Variables de captura mutable

A diferencia de las expresiones lambda, la aplicación parcial puede capturar automáticamente una copia del valor de una variable mutable:

Q#

```
mutable variable = 1;
let f = Foo(variable, _);
```

Esto equivale a la siguiente expresión lambda:

Q#

```
mutable variable = 1;
let value = variable;
let f = x -> Foo(value, x);
```

[^1]: La tupla de parámetro se escribe estrictamente `(a, (b))`, pero `(b)` es equivalente a `b`.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Expresiones contextuales y omitidas

Artículo • 21/03/2025

Las expresiones contextuales son expresiones que solo son válidas en determinados contextos, como el uso de nombres de elemento en [expresiones de copia y actualización](#) sin tener que calificarlas.

Las expresiones se pueden *omitir* cuando el compilador puede inferirlos e insertarlos automáticamente, como [instrucciones evaluate-and-reassign](#).

Los intervalos abiertos son otro ejemplo que se aplica a expresiones contextuales y omitidas. Solo son válidos dentro de un contexto determinado y el compilador los traduce en expresiones normales `Range` durante la compilación mediante la inferencia de límites adecuados.

Un valor de tipo `Range` genera una secuencia de enteros, especificado por un valor inicial, un valor de paso (opcional) y un valor final. Por ejemplo, la expresión literal `Range 1..3` genera la secuencia 1,2,3. Del mismo modo, la expresión `3..-1..1` genera la secuencia 3,2,1. También puede usar intervalos para crear una nueva matriz a partir de una existente mediante la segmentación, por ejemplo:

Q#

```
let arr = [1,2,3,4];
let slice1 = arr[1..2..4]; // contains [2,4]
let slice2 = arr[2..-1..0]; // contains [3,2,1]
```

No se puede definir un intervalo infinito en Q#; Los valores inicial y final siempre deben especificarse. La única excepción es cuando se usa un `Range` para segmentar una matriz. En ese caso, el compilador puede deducir razonablemente los valores inicial o final del intervalo.

En los ejemplos de segmentación de matriz anteriores, es razonable que el compilador suponga que el extremo del intervalo previsto debe ser el índice del último elemento de la matriz si el tamaño del paso es positivo. Si el tamaño del paso es negativo, es probable que el final del intervalo sea el índice del primer elemento de la matriz, `0`. El contrario se mantiene para el inicio del intervalo.

Para resumir, si se omite el valor de inicio del intervalo, el valor inicial inferido

- es cero si no se especifica ningún paso o el paso especificado es positivo.
- es la longitud de la matriz menos una si el paso especificado es negativo.

Si omite el valor final del intervalo, el valor final inferido

- es la longitud de la matriz menos una si no se especifica ningún paso o el paso especificado es positivo.
- es cero si el paso especificado es negativo.

Q# por lo tanto, permite el uso de intervalos abiertos dentro de expresiones de segmentación de matriz, por ejemplo:

Q#

```
let arr = [1,2,3,4,5,6];
let slice1 = arr[3...];           // slice1 is [4,5,6];
let slice2 = arr[0..2...];        // slice2 is [1,3,5];
let slice3 = arr[...2];           // slice3 is [1,2,3];
let slice4 = arr[...2..3];        // slice4 is [1,3];
let slice5 = arr[...2...];        // slice5 is [1,3,5];
let slice7 = arr[4..-2...];       // slice7 is [5,3,1];
let slice8 = arr[...-1..3];       // slice8 is [6,5,4];
let slice9 = arr[...-1...];       // slice9 is [6,5,4,3,2,1];
let slice10 = arr[...];           // slice10 is [1,2,3,4,5,6];
```

Dado que la determinación de si el paso de intervalo es positivo o negativo se produce en tiempo de ejecución, el compilador inserta una expresión adecuada que se evalúa en tiempo de ejecución. Para los valores finales omitidos, la expresión insertada es `step < 0 ? 0 | Length(arr)-1` y para los valores iniciales omitidos se `step < 0 ? Length(arr)-1 | 0`, donde `step` es la expresión dada para el paso de intervalo o `1` si no se especifica ningún paso.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Literales

Artículo • 21/03/2025

Literal de unidad

El único literal existente para el tipo de `Unit` es el valor `()`.

El valor de `Unit` se usa normalmente como argumento para los invocables, ya sea porque no es necesario pasar ningún otro argumento o retrasar la ejecución. También se usa como valor devuelto cuando no es necesario devolver ningún otro valor, que es el caso de las operaciones unitarias, es decir, operaciones que admiten el `Adjoint` o el functor de `Controlled`.

Literales int

Los literales de valor del tipo `Int` se pueden expresar en representación binaria, octal, decimal o hexadecimal. Los literales expresados en binario tienen como prefijo `0b`, con `0o` para octal y con `0x` para hexadecimal. No hay ningún prefijo para la representación decimal usada habitualmente.

 Expandir tabla

Representación	Literal de valor
Binario	<code>0b101010</code>
Octal	<code>0o52</code>
Decimal	<code>42</code>
Hexadecimal	<code>0x2a</code>

Literales BigInt

Los literales de valor para el tipo de `BigInt` siempre se colocan postfijos con `L` y se pueden expresar en representación binaria, octal, decimal o hexadecimal. Los literales expresados en binario tienen como prefijo `0b`, con `0o` para octal y con `0x` para hexadecimal. No hay ningún prefijo para la representación decimal usada habitualmente.

Representación	Literal de valor
Binario	<code>0b101010L</code>
Octal	<code>0o52L</code>
Decimal	<code>42L</code>
Hexadecimal	<code>0x2aL</code>

Literales dobles

Los literales de valor del tipo `Double` se pueden expresar en notación estándar o científica.

Representación	Literal de valor
Estándar	<code>0.1973269804</code>
Científico	<code>1.973269804e-1</code>

Si no se sigue nada después del separador decimal, se puede omitir el dígito después del separador decimal. Por ejemplo, `1.` es un literal de `Double` válido y el mismo que `1.0`.

Literales bool

Los literales existentes del tipo de `Bool` son `true` y `false`.

Literales de cadena

Un literal de valor para el tipo de `String` es una secuencia de longitud arbitraria de caracteres Unicode entre comillas dobles. Dentro de una cadena, el carácter de barra diagonal inversa `\` se puede usar para escapar de un carácter de comilla doble y para insertar una nueva línea como `\n`, un retorno de carro como `\r` y una pestaña como `\t`.

A continuación se muestran ejemplos de literales de cadena válidos:

```
Q#
```

```
"This is a simple string."  
"\This is a more complex string.\", she said.\n"
```

Q# también admite cadenas interpoladas . Una cadena interpolada es un literal de cadena que puede contener cualquier número de expresiones de interpolación. Estas expresiones pueden ser de tipos arbitrarios. Tras la construcción, las expresiones se evalúan y su representación `String` se inserta en la ubicación correspondiente dentro del literal definido. La interpolación está habilitada al preponer el carácter especial `$` directamente antes de la comilla inicial, sin espacio en blanco entre ellos.

Por ejemplo, si `res` es una expresión que se evalúa como `1`, la segunda frase del siguiente `String` literal muestra "El resultado era 1":

```
Q#  
  
$"This is an interpolated string. The result was {res}."
```

Literales de cúbit

No existen literales para el tipo de `Qubit`, ya que el tiempo de ejecución administra la memoria cuántica. Los valores de tipo `Qubit` solo se pueden obtener a través de [asignación](#).

Los valores de tipo `Qubit` representan un identificador opaco por el que se puede solucionar un bit cuántico o cúbit. El único operador que admite es [comparación de igualdad](#). Para obtener más información sobre el tipo de datos de `Qubit`, consulte [qubits](#).

Literales de resultado

Los literales existentes del tipo de `Result` son `Zero` y `One`.

Los valores de tipo `Result` representan el resultado de una medida cuántica binaria. `Zero` indica una proyección en el espacio propio +1, `One` indica una proyección en el espacio propio -1.

Literales de Pauli

Los literales existentes para el tipo de `Pauli` son `PauliI`, `PauliX`, `PauliY` y `PauliZ`.

Los valores de tipo `Pauli` representan una de las cuatro matrices de un solo cúbit [pauli](#), con `PauliI` que representan la identidad. Los valores de tipo `Pauli` se usan normalmente para indicar el eje para las rotaciones y especificar con respecto a la base que se va a medir.

Literales de rango

Los literales de valor del tipo `Range` son expresiones del formulario `start..step..stop`, donde `start`, `step` y `end` son expresiones de tipo `Int`. Si el tamaño del paso es uno, se puede omitir. Por ejemplo, `start..stop` es un literal de `Range` válido y el mismo que `start..1..stop`.

Los valores de tipo `Range` representan una secuencia de enteros, donde el primer elemento de la secuencia es `start` y los elementos posteriores se obtienen agregando `step` al anterior, hasta que se pasa `stop`. `Range` valores son inclusivos en ambos extremos, es decir, el último elemento del intervalo es `stop` si la diferencia entre `start` y `stop` es un múltiplo de `step`. Un intervalo puede estar vacío si, por ejemplo, `step` es positivo y `stop < start`.

A continuación se muestran ejemplos de literales de `Range` válidos:

- `1..3` es el intervalo 1, 2, 3.
- `2..2..5` es el intervalo 2, 4.
- `2..2..6` es el intervalo 2, 4, 6.
- `6..-2..2` es el intervalo 6, 4, 2.
- `2..-2..1` es el intervalo 2.
- `2..1` es el intervalo vacío.

Para obtener más información, vea [expresiones contextuales](#).

Literales de matriz

Una matriz literal es una secuencia de cero o más expresiones, separadas por comas y entre corchetes `[` y `]`; por ejemplo, `[1,2,3]`. Todas las expresiones deben tener un [tipo base común](#), que es el tipo de elemento de la matriz. Si se especifica una matriz vacía con `[]`, es posible que se necesite una anotación de tipo para que el compilador determine el tipo adecuado de la expresión.

Las matrices de longitud arbitraria se pueden crear mediante una expresión de matriz de tamaño. Esta expresión tiene el formato `[expr, size = s]`, donde `s` puede ser cualquier

expresión de tipo `Int` y `expr` se evalúa como un valor que será los elementos de la matriz repetidas `s` veces. Por ejemplo, `[1.2, size = 3]` crea la misma matriz que `[1.2, 1.2, 1.2]`.

Literales de tupla

Un **tupla** literal es una secuencia de una o varias expresiones de cualquier tipo, separadas por comas y entre paréntesis `(` y `)`. El tipo de la tupla incluye la información sobre cada tipo de elemento.

 Expandir tabla

Literal de valor	Tipo
<code>("Id", 0, 1.)</code>	<code>(String, Int, Double)</code>
<code>(PauliX,(3,1))</code>	<code>(Pauli, (Int, Int))</code>

Las tuplas que contienen un solo elemento se tratan como idénticas al propio elemento, tanto en el tipo como en el valor, que se denomina **equivalencia de tupla singleton**.

Las tuplas se usan para agrupar valores en un solo valor, lo que facilita su paso. Esto permite que cada llamada tome exactamente una entrada y devuelva exactamente una salida.

Literales para tipos de estructura

Los valores de un tipo de estructura se asignan al crear una nueva instancia, mediante la instrucción `new` o el constructor creado por el compilador.

Por ejemplo, si `IntPair` es un `struct` definido como

Q#

```
struct IntPair {  
    num1 : Int,  
    num2 : Int,  
}
```

a continuación, se crea una nueva instancia y se asignan valores mediante

Q#

```
let MyPair = new IntPair { num1 = 5, num2 = 7 };
```

O

Q#

```
let MyPair = IntPair(5, 7);
```

Literales de operación y función

Las operaciones y funciones anónimas se pueden crear mediante una expresión lambda

.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Sistema de tipos

Artículo • 21/03/2025

Con el objetivo de que el algoritmo cuántico sea más hacia lo que se debe lograr en lugar de en una representación de problemas en términos de estructuras de datos, tomar una perspectiva más funcional sobre el diseño del lenguaje es una opción natural. Al mismo tiempo, el sistema de tipos es un mecanismo eficaz que se puede usar para el análisis de programas y otras comprobaciones en tiempo de compilación que facilitan la formulación de código sólido.

En general, el sistema de tipos de Q# es bastante minimalista, en el sentido de que no hay una noción explícita de clases o interfaces, ya que se puede usar para desde lenguajes clásicos como C# o Java. También adoptamos un enfoque algo pragmático que hace un progreso incremental, de modo que ciertas construcciones aún no están totalmente integradas en el sistema de tipos. Un ejemplo de esto son functors, que se pueden usar dentro de expresiones, pero aún no tienen una representación en el sistema de tipos. En consecuencia, no se pueden asignar ni pasar actualmente como argumentos, de forma similar a la del tipo parametrized callables. Esperamos realizar un progreso incremental en la ampliación del sistema de tipos para que sea más completo y equilibrar las necesidades inmediatas con planes a largo plazo.

Tipos disponibles

Todos los tipos de Q# son inmutables .

 Expandir tabla

Tipo	Descripción
Unit	Representa un tipo singleton cuyo único valor es <code>()</code> .
Int	Representa un entero de 64 bits con signo. Los valores van de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807.
BigInt	Representa valores enteros con signo de cualquier tamaño.
Double	Representa un número de punto flotante de 64 bits de precisión doble. Valores intervalo de -1.79769313486232e308 a 1.79769313486232e308, así como NaN (no un número).
Bool	Representa valores de booleanos. Los valores posibles son <code>true</code> o <code>false</code> .
String	Representa texto como valores que constan de una secuencia de unidades de código

Tipo	Descripción
	UTF-16.
Qubit	Representa un identificador opaco por el que se puede abordar la memoria cuántica virtual. los valores de tipo <code>Qubit</code> se crean instancias a través de la asignación.
Result	Representa el resultado de una medida projectiva en los espacios propios de un operador cuántico con valores propios ± 1 . Los valores posibles son <code>Zero</code> o <code>One</code> .
Pauli	Representa una matriz de Pauli de un solo cúbit. Los valores posibles son <code>PauliI</code> , <code>PauliX</code> , <code>PauliY</code> o <code>PauliZ</code> .
Range	Representa una secuencia ordenada de valores <code>Int</code> espaciados por igual. Los valores pueden representar secuencias en orden ascendente o descendente.
Array	Representa valores que contienen una secuencia de valores del mismo tipo.
Tuple	Representa valores que cada uno contiene un número fijo de elementos de diferentes tipos. Las tuplas que contienen un solo elemento son equivalentes al elemento que contienen.
struct	Representa un tipo definido por el usuario que consta de elementos con nombre de diferentes tipos. se crean instancias de valores al declarar una nueva instancia.
Operación	Representa un argumento de entrada no determinista invocable que toma un argumento de entrada (posiblemente con valores de tupla) y devuelve una salida (posiblemente con valores de tupla). Las llamadas a la operación valores pueden tener efectos secundarios y la salida puede variar para cada llamada, incluso cuando se invoca con el mismo argumento.
Función	Representa un argumento de entrada determinista invocable que toma un argumento de entrada (posiblemente con valores de tupla) y devuelve una salida (posiblemente con valores de tupla). Las llamadas a función valores no tienen efectos secundarios y la salida siempre será la misma dada la misma entrada.

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Tipos de datos específicos de Quantum

Artículo • 21/03/2025

En este tema se describe el tipo de `Qubit`, junto con otros dos tipos que son algo específicos del dominio cuántico: `Pauli` y `Result`.

Cúbit

Q# trata los cúbits como elementos opacos que se pueden pasar a funciones y operaciones, pero solo se pueden interactuar pasando a instrucciones nativas del procesador cuántico de destino. Estas instrucciones siempre se definen en forma de operaciones, ya que su intención es modificar el estado cuántico. La restricción que las funciones no pueden modificar el estado cuántico, a pesar del hecho de que los cúbits se pueden pasar como argumentos de entrada, se aplica mediante la necesidad de que las funciones solo puedan llamar a otras funciones y no puedan llamar a operaciones.

Las bibliotecas de Q# se compilan en un conjunto estándar de operaciones intrínsecas, lo que significa que las operaciones que no tienen ninguna definición para su implementación en el lenguaje. Tras el destino, el compilador vincula las implementaciones que las expresan en términos de las instrucciones que son nativas del destino de ejecución. Un programa de Q# combina estas operaciones tal como se define en una máquina de destino para crear nuevas operaciones de nivel superior para expresar el cálculo cuántico. De este modo, Q# facilita la expresión de la lógica subyacente a los algoritmos cuánticos e híbridos clásicos cuánticos, mientras que también es muy general con respecto a la estructura de una máquina de destino y su realización del estado cuántico.

Dentro de Q# sí mismo, no hay ningún tipo ni construcción en Q# que represente el estado cuántico. En su lugar, un cúbit representa la unidad física direccionable más pequeña en un equipo cuántico. Por lo tanto, un cúbit es un elemento de larga duración, por lo que Q# no necesita tipos lineales. Por lo tanto, no hacemos referencia explícitamente al estado dentro de Q#, sino que se describe cómo el programa transforma el estado, por ejemplo, a través de la aplicación de operaciones como `X` y `H`. De forma similar a cómo un programa de sombreador de gráficos acumula una descripción de las transformaciones a cada vértice, un programa cuántico en Q# acumula transformaciones a estados cuánticos, representados como referencias completamente opacas a la estructura interna de una máquina de destino.

Un programa de Q# no tiene capacidad para introspección en el estado de un cúbit y, por tanto, es totalmente independiente de lo que es un estado cuántico o de cómo se

realiza. En su lugar, un programa puede llamar a operaciones como `Measure` para obtener información sobre el estado cuántico del cálculo.

Pauli

Los valores de tipo `Pauli` especificar un operador Pauli de un solo cúbit; las posibilidades son `PauliI`, `PauliX`, `PauliY` y `PauliZ`. `Pauli` valores se usan principalmente para especificar la base de una medida cuántica.

Resultado

El tipo `Result` especifica el resultado de una medida cuántica. Q# refleja la mayoría del hardware cuántico proporcionando medidas en productos de operadores pauli de un solo cúbit; un `Result` de `Zero` indica que se midió el valor propio +1 y un `Result` de `One` indica que se midió el valor propio -1. Es decir, Q# representa valores propios por la potencia a la que se eleva -1. Esta convención es más común en la comunidad de algoritmos cuánticos, ya que se asigna más estrechamente a los bits clásicos.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Inmutabilidad

Artículo • 21/03/2025

Todos los tipos de Q# son tipos de valor . Q# no tiene un concepto de referencia o puntero. En su lugar, permite reasignar un nuevo valor a una variable declarada previamente a través de una expresión de asignación. Por ejemplo, no hay distinción en el comportamiento entre reasignaciones para variables de tipo `Int` o variables de tipo `Int[]`. Tenga en cuenta la siguiente secuencia de instrucciones:

Q#

```
mutable arr1 = new Int[3];  
let arr2 = arr1;  
arr1 w/= 0 <- 3;
```

La primera instrucción crea una instancia de una nueva matriz de enteros `[0,0,0]` y la asigna a `arr1`. La siguiente instrucción asigna ese valor a una variable con el nombre `arr2`. La última instrucción crea una nueva instancia de matriz basada en `arr1` con los mismos valores, excepto el valor en el índice 0, que se establece en 3. A continuación, la matriz recién creada se asigna a la variable `arr1`. La última línea usa la sintaxis abreviada para [instrucciones evaluate-and-reassign](#), y podría haberse escrito de forma equivalente como `arr1 = arr1 w/ 0 <- 1;`.

Después de ejecutar las tres instrucciones, `arr1` contendrá el valor `[3,0,0]` mientras `arr2` permanece sin cambios y contiene el valor `[0,0,0]`.

Q# distingue claramente la mutabilidad de un identificador y el comportamiento de un tipo. La mutabilidad dentro de Q# es un concepto que se aplica a un símbolo de en lugar de un tipo o valor; se aplica al identificador que le permite acceder a un valor en lugar del propio valor. No se representado en el sistema de tipos, implícita o explícitamente.

Por supuesto, esto es simplemente una descripción del comportamiento formalmente definido; en segundo plano, la implementación real usa un esquema de recuento de referencias para evitar copiar la memoria tanto como sea posible. La modificación se realiza específicamente siempre que solo haya un identificador válido actualmente que tenga acceso a un valor determinado.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Operaciones y funciones

Artículo • 21/03/2025

Como se explica con más detalle en la descripción del tipo de datos de [cúbit](#), los cálculos cuánticos se ejecutan en forma de efectos secundarios de las operaciones que se admiten de forma nativa en el procesador cuántico de destino. Estos son, de hecho, los únicos efectos secundarios en Q#. Dado que todos los tipos son inmutables, no hay efectos secundarios que afecten a un valor que se representa explícitamente en Q#. Por lo tanto, siempre que una implementación de un determinado invocable no llame directa o indirectamente a ninguna de estas operaciones implementadas de forma nativa, su ejecución siempre genera la misma salida, dada la misma entrada.

Q# permite dividir explícitamente estos cálculos puramente deterministas en funciones. Dado que el conjunto de instrucciones admitidas de forma nativa no es fijo ni está integrado en el propio lenguaje, sino totalmente configurable y expresado como una biblioteca de Q#, se garantiza el determinismo al exigir que las funciones solo puedan llamar a otras funciones y no puedan llamar a ninguna operación. Además, las instrucciones nativas que no son deterministas, es decir, porque afectan al estado cuántico, se representan como operaciones. Con estas dos restricciones, las funciones se pueden evaluar en cuanto se conoce su valor de entrada y, en principio, nunca es necesario evaluar más de una vez para la misma entrada.

Q#, por lo tanto, distingue entre dos tipos de [invocables](#): operaciones y funciones. Todos los invocables toman un único argumento (potencialmente con valores de tupla) como entrada y generan un único valor (tupla) como salida. Sintácticamente, el tipo de operación se expresa como `<TIn> => <TOut> is <Char>`, donde `<TIn>` se va a reemplazar por el tipo de argumento, `<TOut>` se reemplazará por el tipo de valor devuelto y `<Char>` se reemplazará por las características de operación de . Si no es necesario especificar ninguna característica, la sintaxis simplifica `<TIn> => <TOut>`. Del mismo modo, los tipos de función se expresan como `<TIn> -> <TOut>`.

Aparte de esta garantía de determinismo, hay poca diferencia entre las operaciones y las funciones. Ambos son valores de primera clase que se pueden pasar libremente; se pueden usar como valores devueltos o argumentos para otros invocables, como se muestra en el ejemplo siguiente:

Q#

```
function Pow<'T>(op : 'T => Unit, pow : Int) : 'T => Unit {  
    return PowImpl(op, pow, _);  
}
```

Ambos se pueden crear instancias en función de una definición parametrizada de tipo, por ejemplo, la función de [parametrizada](#) función `Pow` anterior y se pueden [aplicar parcialmente](#) como se hace en la instrucción `return` del ejemplo.

Características de la operación

Además de la información sobre el tipo de entrada y salida, el tipo de operación contiene información sobre las características de una operación. Esta información, por ejemplo, describe qué funtores son compatibles con la operación. Además, la representación interna también contiene información relevante para la optimización que el compilador deduce.

Las características de una operación son un conjunto de etiquetas predefinidas e integradas. Se expresan en forma de una expresión especial que forma parte de la firma de tipo. La expresión consta de uno de los conjuntos predefinidos de etiquetas o de una combinación de expresiones de características a través de un operador binario admitido.

Hay dos conjuntos predefinidos, `Adj` y `Ctl`.

- `Adj` es el conjunto que contiene una sola etiqueta que indica que una operación es adyacente, lo que significa que admite el functor [Adjoint](#) y la transformación cuántica aplicada puede "deshacerse", es decir, se puede invertir.
- `ctl` es el conjunto que contiene una sola etiqueta que indica que una operación es controlable, lo que significa que admite la [Controlled functor](#) y su ejecución puede estar condicionada en el estado de otros cúbits.

Los dos operadores que se admiten como parte de las expresiones de características son la unión establecida `+` y la intersección del conjunto `*`. En EBNF (formulario Backus–Naur extendido),

```
predefined = "Adj" | "Ctl";
characteristics = predefined
    | "(" , characteristics , ")"
    | characteristics ("+"|"*") characteristics;
```

Como cabría esperar, `*` tiene mayor prioridad que `+` y ambos son asociativos a la izquierda. El tipo de una operación unitaria, por ejemplo, se expresa como `<TIn> => <TOut>` is `Adj + Ctl`, donde `<TIn>` debe reemplazarse por el tipo del argumento operation y `<TOut>` reemplazado por el tipo del valor devuelto.

ⓘ Nota

La indicación de las características de una operación en este formato tiene dos principales ventajas; para una, se pueden introducir nuevas etiquetas sin tener exponencialmente muchas palabras clave de lenguaje para todas las combinaciones de etiquetas. Más importante, el uso de expresiones para indicar las características de una operación también admite parametrizaciones sobre las características de operación en el futuro.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Equivalencia de tupla singleton

Artículo • 21/03/2025

Para evitar cualquier ambigüedad entre tuplas y paréntesis que agrupan subexpresiones, una tupla con un solo elemento se considera equivalente al elemento contenido, incluido su tipo. Por ejemplo, los tipos `Int`, `(Int)` y `((Int))` se tratan como idénticos. Lo mismo se mantiene true para los valores `5`, `(5)` y `((5))`, o para `(5, (6))` y `(5, 6)`. Esta equivalencia se aplica para todos los fines, incluida la asignación. Dado que no hay ningún envío dinámico o reflexión en Q# y todos los tipos de Q# se pueden resolver en tiempo de compilación, la equivalencia de tupla singleton se puede implementar fácilmente durante la compilación.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Subtipos y varianza

Artículo • 21/03/2025

Q# admite solo algunos mecanismos de conversión. Las conversiones implícitas solo pueden producirse al aplicar operadores binarios, evaluar expresiones condicionales o construir un literal de matriz. En estos casos, se determina un supertipo común y las conversiones necesarias se realizan automáticamente. Además de estas conversiones implícitas, las conversiones explícitas a través de llamadas de función son posibles y a menudo necesarias.

Actualmente, la única relación de subtyping que existe se aplica a las operaciones. Intuitivamente, tiene sentido que se debe permitir que se sustituya una operación que admita más del conjunto necesario de funtores. Concretamente, para dos tipos concretos `TIn` y `TOut`, la relación de subtyping es

```
(TIn => TOut) :>  
(TIn => TOut is Adj), (TIn => TOut is Ctl) :>  
(TIn => TOut is Adj + Ctl)
```

donde `A :> B` indica que `B` es un subtipo de `A`. Frase diferente, `B` es más restrictivo que `A` de modo que se pueda usar un valor de tipo `B` siempre que se requiera un valor de tipo `A`. Si un invocable se basa en un argumento (elemento) de ser de tipo `A`, se puede sustituir un argumento de tipo `B` de forma segura, ya que si proporciona todas las funcionalidades necesarias.

Este tipo de polimorfismo se extiende a las tuplas en que una tupla de tipo `B` es un subtipo de un tipo de tupla `A` si contiene el mismo número de elementos y el tipo de cada elemento es un subtipo del tipo de elemento correspondiente en `A`. Esto se conoce como subtipado de profundidad. Actualmente no hay compatibilidad con *subtipado de ancho*, es decir, no hay ninguna relación de subtipo entre dos tipos `struct` o un tipo de `struct` y cualquier tipo integrado. La existencia del operador `unwrap`, que permite extraer una tupla que contiene todos los elementos con nombre, evita esto.

❗ Nota

En lo que respecta a los invocables, si un invocable procesa un argumento de tipo `A`, también es capaz de procesar un argumento de tipo `B`. Si un invocable se pasa como argumento a otro invocable, debe ser capaz de procesar todo lo que pueda

requerir la firma de tipo. Esto significa que si el invocable necesita poder procesar un argumento de tipo B , cualquier invocable capaz de procesar un argumento más general de tipo A se puede pasar de forma segura. Por el contrario, esperamos que si se requiere que el objeto al que se llama devuelva un valor de tipo A , la promesa de devolver un valor de tipo B es suficiente, ya que ese valor proporcionará todas las funcionalidades necesarias.

El tipo de función o operación se contravariante en su tipo de argumento y covariante en su tipo de valor devuelto. $A \rightarrow B$ por lo tanto implica que para cualquier tipo concreto $T1$,

$$(B \rightarrow T1) \rightarrow (A \rightarrow T1), \text{ and} \\ (T1 \rightarrow A) \rightarrow (T1 \rightarrow B)$$

donde $\rightarrow$ aquí puede significar una función o una operación, y se omiten las anotaciones de las características. Sustituir A con $(B \rightarrow T2)$ y $(T2 \rightarrow A)$ respectivamente, y sustituir B por $(A \rightarrow T2)$ y $(T2 \rightarrow B)$ respectivamente, conduce a la conclusión de que, para cualquier tipo concreto $T2$,

$$((A \rightarrow T2) \rightarrow T1) \rightarrow ((B \rightarrow T2) \rightarrow T1), \text{ and} \\ ((T2 \rightarrow B) \rightarrow T1) \rightarrow ((T2 \rightarrow A) \rightarrow T1), \text{ and} \\ (T1 \rightarrow (B \rightarrow T2)) \rightarrow (T1 \rightarrow (A \rightarrow T2)), \text{ and} \\ (T1 \rightarrow (T2 \rightarrow A)) \rightarrow (T1 \rightarrow (T2 \rightarrow B))$$

Por inducción, sigue que cada direccionamiento indirecto adicional invierte la varianza del tipo de argumento y deja sin cambios la varianza del tipo devuelto.

❗ Nota

Esto también hace claro cuál debe ser el comportamiento de varianza de las matrices; recuperar elementos a través de un operador de acceso a elementos corresponde a invocar una función de tipo $(Int \rightarrow TItem)$, donde $TItem$ es el tipo de los elementos de la matriz. Dado que esta función se pasa implícitamente al pasar una matriz, sigue que las matrices deben ser covariantes en su tipo de elemento. Las mismas consideraciones también tienen en cuenta las tuplas, que son inmutables y, por tanto, covariante con respecto a cada tipo de elemento. Si las matrices no eran inmutables, la existencia de una construcción que le permitiría

establecer elementos en una matriz y, por tanto, tomar un argumento de tipo `TItem`, implicaría que las matrices también deben ser contravariantes. La única opción para los tipos de datos que admiten la obtención y configuración de elementos es, por lo tanto, ser *invariable*, lo que significa que no hay ninguna relación de subtipado; `B[]` no se es un subtipo de `A[]` incluso si `B` es un subtipo de `A`. A pesar de que las matrices de Q# son inmutables, son invariables en lugar de covariante. Esto significa, por ejemplo, que un valor de tipo `(Qubit => Unit is Adj) []` no se puede pasar a un invocable que requiere un argumento de tipo `(Qubit => Unit)[]`. Mantener las matrices invariables permite una mayor flexibilidad relacionada con cómo se controlan y optimizan las matrices en tiempo de ejecución, pero puede ser posible revisarlas en el futuro.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Parametrizaciones de tipos

Artículo • 21/03/2025

Q# admite operaciones y funciones con parámetros de tipo. Las bibliotecas estándar de Q# hacen un uso intensivo de los invocables de tipo parametrizado para proporcionar una serie de abstracciones útiles, incluidas funciones como `Mapped` y `Fold` que son familiares de los lenguajes funcionales.

Para motivar el concepto de parametrizaciones de tipo, considere el ejemplo de la función `Mapped`, que aplica una función determinada a cada valor de una matriz y devuelve una nueva matriz con los valores calculados. Esta funcionalidad se puede describir perfectamente sin especificar los tipos de elemento de las matrices de entrada y salida. Dado que los tipos exactos no cambian la implementación de la función `Mapped`, tiene sentido que debería ser posible definir esta implementación para tipos de elementos arbitrarios; queremos definir un *de fábrica* de o *plantilla* que, dados los tipos concretos de los elementos de la matriz de entrada y salida, devuelve la implementación de función correspondiente. Esta noción se formaliza en forma de parámetros de tipo.

Concretización

Cualquier declaración de operación o función puede especificar uno o varios parámetros de tipo que se pueden usar como tipos, o parte de los tipos, de la entrada o salida invocables, o ambos. Las excepciones son puntos de entrada, que deben ser concretos y no pueden ser parametrizados de tipo. Los nombres de parámetros de tipo comienzan con un tic (') y pueden aparecer varias veces en los tipos de entrada y salida. Todos los argumentos que corresponden al mismo parámetro de tipo de la firma invocable deben ser del mismo tipo.

Es necesario que se concrete un parámetro de tipo parametrizado, es decir, debe proporcionarse con los argumentos de tipo necesarios para poder asignarlos o pasarlos como argumento, de modo que todos los parámetros de tipo se puedan reemplazar por tipos concretos. Un tipo se considera concreto si es uno de los tipos integrados, un tipo `struct` o si es concreto dentro del ámbito actual. En el ejemplo siguiente se muestra lo que significa que un tipo sea concreto dentro del ámbito actual y se explica con más detalle a continuación:

Q#

```
function Mapped<'T1, 'T2> (  
    mapper : 'T1 -> 'T2,  
    array : 'T1[]
```

```

) : 'T2[] {

    mutable mapped = new 'T2[Length(array)];
    for (i in IndexRange(array)) {
        mapped w/= i <- mapper(array[i]);
    }
    return mapped;
}

function AllCControlled<'T3> (
    ops : ('T3 => Unit)[]
) : ((Bool, 'T3) => Unit)[] {

    return Mapped(CControlled<'T3>, ops);
}

```

La función `CControlled` se define en el espacio de nombres `Microsoft.Quantum.Canon`. Toma una operación `op` de tipo `'TIn => Unit` como argumento y devuelve una nueva operación de tipo `(Bool, 'TIn) => Unit` que aplica la operación original, siempre que un bit clásico (de tipo `Bool`) se establezca en *true*; esto se conoce a menudo como la versión controlada clásicamente de `op`.

La función `Mapped` toma una matriz de un tipo de elemento arbitrario `'T1` como argumento, aplica la función `mapper` dada a cada elemento y devuelve una nueva matriz de tipo `'T2[]` que contiene los elementos asignados. Se define en el espacio de nombres `Microsoft.Quantum.Array`. Para el ejemplo, los parámetros de tipo se numeran para evitar que la discusión sea más confusa al proporcionar los parámetros de tipo en ambas funciones el mismo nombre. Esto no es necesario; los parámetros de tipo para diferentes invocables pueden tener el mismo nombre y el nombre elegido solo es visible y relevante dentro de la definición de ese invocable.

La función `AllCControlled` toma una matriz de operaciones y devuelve una nueva matriz que contiene las versiones controladas clásicamente de estas operaciones. La llamada de `Mapped` resuelve su parámetro de tipo `'T1` para `'T3 => Unit` y su parámetro de tipo `'T2` para `(Bool, 'T3) => Unit`. El compilador deduce los argumentos de tipo de resolución en función del tipo del argumento especificado. Decimos que se *implícitamente* definidos por el argumento de la expresión de llamada. Los argumentos de tipo también se pueden especificar explícitamente, como se hace para `CControlled` en la misma línea. El `CControlled<'T3>` de concretización explícita es necesario cuando no se pueden deducir los argumentos de tipo.

El tipo `'T3` es concreto en el contexto de `AllCControlled`, ya que se conoce por cada *invocación* de `AllCControlled`. Esto significa que tan pronto como se conoce el punto de entrada del programa (que no se puede parametrizar de tipo), por lo que es el tipo

concreto `'T3` para cada llamada a `AllControlled`, de modo que se pueda generar una implementación adecuada para esa resolución de tipos en particular. Una vez conocido el punto de entrada a un programa, se pueden eliminar todos los usos de parámetros de tipo en tiempo de compilación. Nos referimos a este proceso como *monomorfización*.

Se necesitan algunas restricciones para asegurarse de que esto se puede hacer en tiempo de compilación, en lugar de solo en tiempo de ejecución.

Restricciones

Considere el ejemplo siguiente:

Q#

```
operation Foo<'TArg> (  
    op : 'TArg => Unit,  
    arg : 'TArg  
) : Unit {  
  
    let cbit = RandomInt(2) == 0;  
    Foo(CControlled(op), (cbit, arg));  
}
```

Al ignorar que una invocación de `Foo` da como resultado un bucle infinito, sirve para la ilustración. `Foo` se invoca con la versión controlada clásicamente de la operación original `op` que se ha pasado, así como una tupla que contiene un bit clásico aleatorio además del argumento original.

Para cada iteración de la recursividad, el parámetro type `'TArg` de la siguiente llamada se resuelve en `(Bool, 'TArg)`, donde `'TArg` es el parámetro type de la llamada actual. En concreto, supongamos que se invoca `Foo` con la operación `H` y un argumento `arg` de tipo `Qubit`. `Foo` luego se invoca con un argumento de tipo `(Bool, Qubit)`, que luego invoca `Foo` con un argumento de tipo `(Bool, (Bool, Qubit))`, etc. Claramente, en este caso `Foo` no se puede monomorfizar en tiempo de compilación.

Se aplican restricciones adicionales a los ciclos del gráfico de llamadas que implican solo los invocables de tipo parametrizado. Cada invocable debe invocarse con el mismo conjunto de argumentos de tipo después de atravesar el ciclo.

❗ Nota

Sería posible ser menos restrictivo y requerir que para cada invocable en el ciclo, hay un número finito de ciclos después del cual se invoca con el conjunto original de argumentos de tipo, como el caso de la siguiente función:

Q#

```
function Bar<'T1,'T2,'T3>(a1:'T1, a2:'T2, a3:'T3) : Unit{  
    Bar<'T2,'T3,'T1>(a2, a3, a1);  
}
```

Para simplificar, se aplica el requisito más restrictivo. Tenga en cuenta que, en el caso de los ciclos que implican al menos un concreto invocable sin ningún parámetro de tipo, este tipo garantiza que los invocables parametrizados de tipo dentro de ese ciclo siempre se llamen con un conjunto fijo de argumentos de tipo.

Comentarios

¿Le ha resultado útil esta página?



[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Inferencia de tipo

Artículo • 21/03/2025

Q# algoritmo de inferencia de tipos se basa en algoritmos de inferencia diseñados para el sistema de tipos de Hindley-Milner. Aunque los invocables de nivel superior deben declararse con anotaciones de tipo explícitas, la mayoría de los tipos usados dentro de un invocable se pueden inferir. Por ejemplo, dados estos invocables:

Q#

```
function Length<'a>(xs : 'a[]) : Int
function Mapped<'a, 'b>(f : 'a -> 'b, xs : 'a[]) : 'b[]
```

y esta expresión:

Q#

```
Mapped(Length, [], ["a"], ["b", "c"])
```

a continuación, el argumento de tipo para `Length` se deduce como `Length<String[]>` y los argumentos de tipo para `Mapped` se deducen como `Mapped<String[], Int>`. No es necesario escribir estos tipos explícitamente.

Tipos ambiguos

A veces no hay un único tipo de entidad de seguridad que se pueda deducir para una variable de tipo. En estos casos, la inferencia de tipos produce un error que hace referencia a un tipo ambiguo. Por ejemplo, cambie ligeramente el ejemplo anterior:

Q#

```
Mapped(Length, [[]])
```

¿Cuál es el tipo de `[[]]`? En algunos sistemas de tipos, es posible asignarle el tipo `∀a. a[][]`, pero esto no se admite en Q#. Se requiere un tipo concreto, pero hay un número infinito de tipos que funcionan: `String[][]`, `(Int, Int)[][]`, `Double[][][]`, etc. Debe decir explícitamente qué tipo quería decir.

Hay varias maneras de hacerlo, dependiendo de la situación. Por ejemplo:

1. Llame a `Length` con un argumento de tipo.

Q#

```
Mapped<Length<String>, [[]]>
```

2. Llame a `Mapped` con su primer argumento de tipo. (El `_` para su segundo argumento de tipo significa que todavía debe inferirse).

Q#

```
Mapped<String[], _>(Length, [[]])
```

3. Reemplace el literal de matriz vacía por una llamada de tipo explícita a una función de biblioteca.

Q#

```
Mapped<String[], _>(Length, [EmptyArray<String>()])
```

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Escribir y ejecutar OpenQASM código en Quantum Development Kit

22/07/2025

Quantum Development Kit (QDK) proporciona un entorno de desarrollo para programas de OpenQASM con integración de Azure Quantum. En este artículo, aprenderá a escribir, depurar y ejecutar código de OpenQASM en QDK.

Las siguientes características de QDK están disponibles en programas OpenQASM:

- Finalización de código
- Lentes de código (**Run**, **Histogram**, **Estimate**, **Debug** y **Circuit**)
- Resaltado de sintaxis y características básicas de sintaxis, como la coincidencia de llaves de apertura
- Comprobación de errores en los archivos de origen OpenQASM
- Depuración de puntos de interrupción y ejecución de scripts para archivos de origen de OpenQASM
- Depuración con generación de circuitos en directo
- Integración con Azure Quantum para el envío de trabajos cuanto

Trabajar con OpenQASM en Visual Studio Code y Azure Quantum

Para escribir OpenQASM código en Visual Studio Code, abra un proyecto de Q# y cree un archivo OpenQASM con la extensión `.qasm`. QDK reconoce automáticamente el archivo `.qasm` como OpenQASM y las características de QDK para OpenQASM están disponibles a medida que escribe el código de OpenQASM en VS Code.

Para ejecutar el OpenQASM programa desde VS Code, puede usar un simulador cuántico local o puede enviar el código como un trabajo a Azure Quantum. El simulador cuántico local es una característica de QDK y no tiene ningún costo. Azure Quantum los trabajos tienen un costo según el proveedor y plan que elija para su envío de trabajo.

Ejecución local del código en un simulador cuántico

Para ejecutar OpenQASM el código en un simulador cuántico local en VS Code, abra y seleccione un `.qasm` archivo y elija uno de los métodos siguientes:

- Presione **Ctrl + F5**.

- Presione **Ctrl + Mayús + P** (o **Cmd + Mayús + P**) para abrir la paleta de comandos y, a continuación, escriba **Depurar: Ejecutar**.
- Seleccione el CodeLens **Ejecutar**.
- Elija la acción **ejecutar** del editor del botón, representada por un icono de reproducción.

La salida de su código aparece en la pestaña **CONSOLA DE DEPURACIÓN** del panel de salida.

ⓘ Nota

Si no elige un perfil de Q# target adecuado para la simulación, obtendrá errores del compilador al ejecutar el OpenQASM código. Para obtener más información sobre los target perfiles, vea [Diferentes tipos de target perfiles en Azure Quantum](#).

Programas de ejemplo integrados

QDK incluye varios ejemplos de algoritmo cuántico integrados para OpenQASM. Para probar los algoritmos de ejemplo, escriba `sample` en un archivo vacío `.qasm`. Aparece una lista de finalización que contiene los algoritmos de ejemplo. Elija un ejemplo y ejecute el código generado.

Depuración local en VS Code

QDK incluye un depurador para OpenQASM. El depurador controla la depuración típica al estilo de punto de interrupción (entrar, sobrepasar, salir, etc.) y puede mostrar el estado del programa.

Para iniciar el depurador, elija uno de los métodos siguientes:

- Elija la lista desplegable en el botón **Ejecutar** y, a continuación, seleccione **Iniciar depuración**.
- Elija la lente de código **Depurar**.

En el modo de depuración, puede establecer puntos de interrupción en el código para ayudarle a depurar. El estado del programa está visible en la vista **Ejecutar y depurar** de la barra lateral. Por ejemplo, puede ver los estados de las variables locales y los estados de cúbit.

Ejecute su código en Azure Quantum

Para obtener instrucciones sobre cómo enviar un trabajo a Azure Quantum desde VS Code, consulte [Conexión a Azure Quantum y envío del trabajo](#).

Trabajar con OpenQASM en Python y Jupyter Notebook

QDK incluye dos paquetes de Python, `qsharp` y `qsharp-widgets`, que le permiten trabajar con código Q# y OpenQASM en un entorno de Python en Jupyter Notebook. Escribes tu código OpenQASM como una cadena de Python y luego usas esa cadena para ejecutar, importar o compilar tu código OpenQASM en el entorno de Python.

Prerrequisitos

Para ejecutar código de OpenQASM en un Jupyter Notebook con VS Code, configure VS Code con extensiones para QDK, Python y Jupyter Notebook. Para obtener más información sobre cómo instalar estas extensiones y ejecutar cuadernos de Jupyter con VS Code, vea [Tutorial: Q# en Jupyter Notebooks y VS Code](#).

Ejecuta OpenQASM directamente en una celda del notebook de Python

Para ejecutar código de OpenQASM directamente en una celda de Python, use la función `run()` de la biblioteca `qsharp.openqasm`. Por ejemplo, la siguiente celda llama a la función `run()` con una cadena de código OpenQASM, junto con argumentos para el número de tomas y el ruido:

Python

```
import qsharp
from qsharp import BitFlipNoise, TargetProfile
from qsharp.openqasm import run

qsharp.init(target_profile=TargetProfile.Base)

results = run(
    """
    include "stdgates.inc";
    qubit[2] q;
    reset q;
    h q[0];
    cx q[0], q[1];
    bit c = measure q[1];
    """,
    shots=10,
    noise=BitFlipNoise(0.1),
)
print(results)
```

Llamar a un OpenQASM programa desde Python

Puede usar la `import_openqasm()` función para almacenar un OpenQASM programa como un objeto de Python y, a continuación, llamar a ese objeto más adelante en el código de Python. Por ejemplo, la celda siguiente almacena un OpenQASM circuito en un objeto de Python denominado `bell`:

Python

```
import qsharp
from qsharp import init, TargetProfile
from qsharp.openqasm import import_openqasm

qsharp.init(target_profile=TargetProfile.Base)

import_openqasm(
    """
    include "stdgates.inc";
    qubit[2] q;
    reset q;
    h q[0];
    cx q[0], q[1];
    bit c = measure q[1];
    """,
    name="bell",
)
```

Esta celda siguiente importa y llama al `bell` circuito:

Python

```
from qsharp.code import bell
bell()
```

Después `bell` de formar parte de su entorno, puede ejecutarlo directamente desde una celda de Q#:

Q#

```
%%qsharp
bell()
```

También puede usar toda la funcionalidad del QDK paquete con `bell`, como simulación ruidosa, representación de circuitos y generación de código.

Programas con parámetros OpenQASM

La celda siguiente crea un OpenQASM circuito que toma un ángulo `theta` como entrada y, a continuación, llama al circuito como una función de Python:

Python

```
import_openqasm(
    """
    include "stdgates.inc";
    input float theta;
    qubit[2] q;
    rx(theta) q[0];
    rx(-theta) q[1];
    bit[2] c;
    c = measure q;
    """,
    name="parameterized_circuit",
)

from qsharp.code import parameterized_circuit

parameterized_circuit(1.57)
```

Compilar programas OpenQASM en QIR

Puede compilar código de OpenQASM en Quantum Intermediate Representation (QIR) con la función `compile()`.

Por ejemplo, la celda siguiente compila código para un circuito con parámetros OpenQASM en QIR:

Python

```
import_openqasm(
    """
    include "stdgates.inc";
    input float theta;
    qubit[2] q;
    rx(theta) q[0];
    rx(-theta) q[1];
    bit[2] c;
    c = measure q;
    """,
    name="parameterized_circuit",
)

from qsharp.code import parameterized_circuit

bound_compilation = qsharp.compile(parameterized_circuit, 1.57)
print(bound_compilation)
```


ⓘ Nota

La API de Python es necesaria para compilar OpenQASM programas que toman variables de entrada.

OpenQASM características de lenguaje que no se admiten

La compatibilidad con OpenQASM en QDK está en curso.

Para obtener una lista de las limitaciones y características de OpenQASM lenguaje que están en desarrollo en QDK, vea [Características de lenguaje no admitidas o no implementadasOpenQASM](#) .

Qiskit e Cirq interoperabilidad con Quantum Development Kit

08/10/2025

Azure Quantum Development Kit (QDK) admite el desarrollo en varios marcos de programación cuántica, incluidos Qiskit y Cirq.

Qiskit y Cirq son Python bibliotecas que ofrecen herramientas para crear y visualizar circuitos cuánticos y escribir programas cuánticos. QDK admite cierta interoperabilidad con tanto Qiskit como Cirq. Puede escribir los programas cuánticos en estos marcos y, a continuación, ejecutarlos en el simulador local o enviarlos como trabajos a Azure Quantum con QDK.

Comienza con Qiskit y Cirq en QDK

Instale la biblioteca Qiskit y Cirq, y ejecute Qiskit y Cirq en los circuitos de Azure Quantum, mediante `azure-quantum` PythonSDK. Puede ejecutar circuitos Qiskit también en el simulador local QDK's.

Para empezar a trabajar con Qiskit, consulte [Envío de un circuito con Qiskit a Azure Quantum](#).

Para empezar a trabajar con Cirq, consulte [Envío de un circuito con Cirq a Azure Quantum](#).

Envío de un circuito con Qiskit a Azure Quantum

Aprenda a enviar un circuito cuántico qiskit mediante el `azure-quantum` Python paquete. Puede enviar circuitos Qiskit a Azure Quantum mediante el Kit de desarrollo de Azure Quantum (QDK) y Jupyter Notebook en Visual Studio Code (VS Code). También puede probar los circuitos mediante el simulador disperso local.

Para más información, consulte [Circuitos cuánticos](#).

Pasos siguientes

- [Inicio rápido: Envío de un circuito con Cirq to Azure Quantum](#).
- [Inicio rápido: Envío de un circuito con un formato específico del proveedor a Azure Quantum](#).

Last updated on 12/11/2025

Cómo enviar un circuito con Cirq a Azure Quantum

Aprenda a enviar un circuito cuántico cirq mediante el `azure-quantum` Python paquete. Puede enviar circuitos Cirq a Azure Quantum mediante el Kit de desarrollo de Azure Quantum (QDK) y Jupyter Notebook en Visual Studio Code (VS Code) desde la máquina local.

Para más información, consulte [Circuitos cuánticos](#).

Requisitos previos

Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

- Un área de trabajo de Azure Quantum en la suscripción de Azure. Para crear un área de trabajo, consulte [Creación de un área de trabajo de Azure Quantum](#).
- Un entorno Python con [Python y Pip](#) [↗] instalado.
- VS Code con las [extensiones de Azure Quantum Development Kit](#) [↗], [Python](#) [↗] y [Jupyter](#) [↗] instaladas.
- El paquete de Azure Quantum `azure-quantum` con la etiqueta `[cirq]` y los paquetes `qsharp` y `ipykernel`.

Bash

```
python -m pip install --upgrade azure-quantum[cirq] qsharp ipykernel
```

⚠ Nota

Si no se detecta el kernel Python de Jupyter `ipykernel`, VS Code le pedirá que lo instale.

Crear un nuevo cuaderno de Jupyter Notebook

1. En VS Code, seleccione **Ver > Paleta de comandos** y seleccione **Crear: Nuevo Jupyter Notebook**.
2. En la parte superior derecha, VS Code detectará y mostrará la versión de Python y el entorno virtual Python seleccionado para el cuaderno. Si tiene varios Python entornos, es

posible que tenga que seleccionar un núcleo mediante el selector de núcleo ubicado en la parte superior derecha. Si no se detectó ningún entorno, consulte [Cuadernos de Jupyter en VS Code](#) para más información sobre la configuración.

Carga de las importaciones necesarias

En la primera celda del cuaderno, ejecute el código siguiente para cargar las importaciones necesarias:

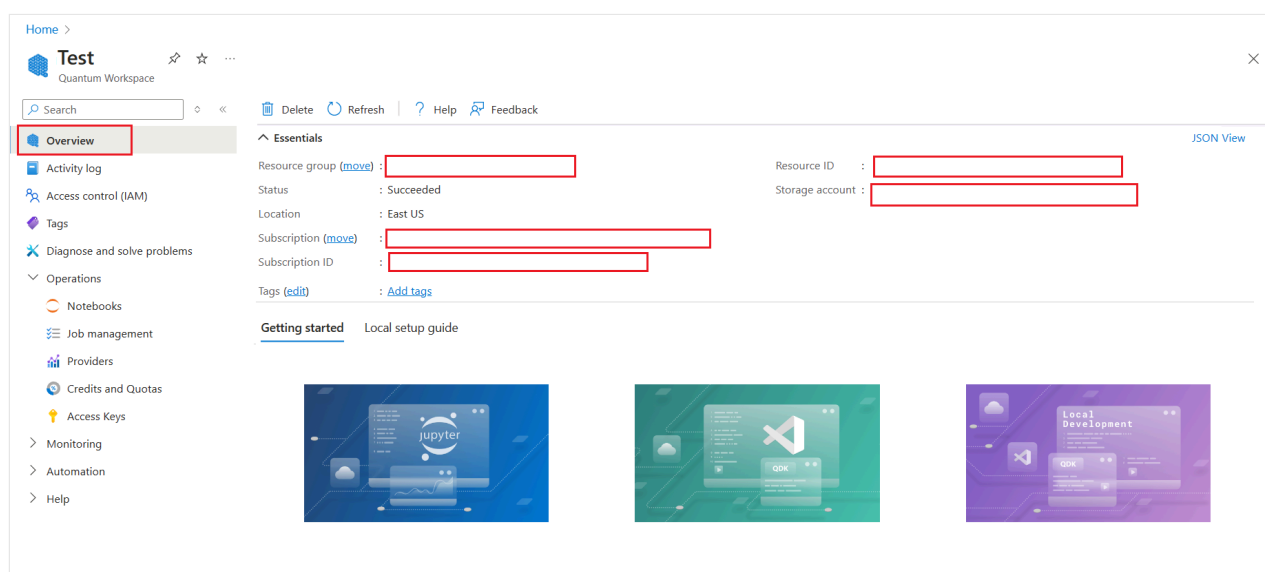
Python

```
import azure.quantum
from azure.quantum.cirq import AzureQuantumService
```

Conexión al servicio Azure Quantum

Para conectarse al servicio Azure Quantum, el programa necesitará el identificador de recurso y la ubicación del área de trabajo de Azure Quantum.

1. Inicie sesión en su cuenta de Azure, <https://portal.azure.com>,
2. Seleccione el área de trabajo de Azure Quantum y vaya a **Información general**.
3. Copie los parámetros en los campos.



Agregue una nueva celda y use la información de su cuenta para crear objetos `Workspace` y `AzureQuantumService` para conectarse a su área de trabajo de Azure Quantum.

Python

```
workspace = Workspace(
    resource_id = "", # Add the resourceID of your workspace
    location = "" # Add the location of your workspace (for example "westus")
)

service = AzureQuantumService(workspace)
```

Enumerar todos los elementos targets

Usa el método `targets()` para enumerar todos los targets de tu área de trabajo que pueden ejecutar tu circuito, incluido el tiempo de cola actual y la disponibilidad.

ⓘ Nota

Es posible que no aparezcan todos los targets en su área de trabajo; aquí solo se mostrarán los targets que puedan aceptar un circuito de Cirq o OpenQASM.

Python

```
print(service.targets())
```

Resultados

```
[<Target name="quantinuum.qpu.h2-1", avg. queue time=0 s, Degraded>,
<Target name="quantinuum.sim.h2-1sc", avg. queue time=1 s, Available>,
<Target name="quantinuum.sim.h2-1e", avg. queue time=40 s, Available>,
<Target name="ionq.simulator", avg. queue time=3 s, Available>,
<Target name="ionq.qpu.aria-1", avg. queue time=1136774 s, Available>]
```

Creación de un circuito simple

A continuación, cree un circuito de Cirq simple para ejecutarlo. Este circuito usa la raíz cuadrada de la puerta X, nativa del sistema de hardware de IonQ.

Python

```
import cirq

q0, q1 = cirq.LineQubit.range(2)
circuit = cirq.Circuit(
    cirq.X(q0)**0.5,          # Square root of X
    cirq.CX(q0, q1),         # CNOT
    cirq.measure(q0, q1, key='b') # Measure both qubits
```

```
)  
print(circuit)
```

Resultados

```
0: —X^0.5—@—M—  
          |  |  
1: —————X—M—
```

Selecciona un target para ejecutar tu programa

IonQ

Ejecución en el simulador de IonQ

Ahora puede ejecutar el programa a través del servicio Azure Quantum y obtener el resultado. La celda siguiente envía un trabajo (al simulador de IonQ predeterminado) que ejecuta el circuito con 100 tomas, espera hasta que se complete el trabajo y devuelve los resultados.

Python

```
result = service.run(program=circuit, repetitions=100, target="ionq.simulator")
```

Esto devuelve un objeto `cirq.Result`.

Python

```
print(result)
```

Resultados

```
b=100110010110000100001101110100001101010001011110001100100010010001000000111001  
0010101110110000011010,  
10011001011000010000110111010000110101000101111000110010001001000100000011100100  
10101110110000011010
```

Estimación del costo del trabajo

Antes de ejecutar un trabajo en la QPU, debería calcular cuánto costará ejecutarlo.

Para obtener los precios más actuales, consulte los [precios de IonQ](#) o vaya a su área de trabajo y consulte las opciones de precios en la hoja "Proveedores" en: aka.ms/aq/myworkspaces.

Ejecución en la QPU de IonQ

El trabajo anterior se ejecutó en el simulador predeterminado, `"ionq.simulator"`. Sin embargo, también puede ejecutarlo en el procesador de hardware de IonQ, una [unidad de procesador cuántico](#) (QPU). Para ejecutarlo en la QPU de IonQ, proporcione `"ionq.qpu.aria-1"` como argumento `target`:

Python

```
result = service.run(
    program=circuit,
    repetitions=100,
    target="ionq.qpu.aria-1",
    timeout_seconds=500 # Set timeout to accommodate queue time on QPU
)
```

De nuevo, esto devuelve un objeto `cirq.Result`.

Python

```
print(result)
```

Resultados

```
b=010101101101111110000101110110101101111010001000000001111011100010010011011010
1100110001001111101111,
01010110110111111000010111011010110111101000100000000111101110001001001101101011
00110001001111101111
```

Modelo asíncrono con trabajos

En el caso de los circuitos de ejecución prolongada, puede ser útil ejecutarlos de forma asíncrona. El método `service.create_job` devuelve un objeto `Job`, que puede usar para obtener los resultados después de que el trabajo se haya ejecutado correctamente.

Python

```
job = service.create_job(
    program=circuit,
    repetitions=100,
```



```
target="ionq.simulator"  
)
```

Para comprobar el estado del trabajo, use `job.status()`:

Python

```
print(job.status())
```

Resultados

```
'completed'
```

Para esperar a que se complete el trabajo y, a continuación, obtener los resultados, use la llamada de bloqueo `job.results()`:

Python

```
result = job.results()  
print(result)
```

Resultados

```
00: 0.5  
11: 0.5
```

Tenga en cuenta que no devuelve un objeto `cirq.Result`. En su lugar, devuelve un objeto de resultado específico del simulador de IonQ y usa probabilidades de estado en lugar de datos de toma.

Python

```
type(result)
```

Resultados

```
cirq_ionq.results.SimulatorResult
```

Para convertirlo en un objeto `cirq.Result`, use `result.to_cirq_result()`:

Python

```
print(result.to_cirq_result())
```

Resultados

```
b=11101011111111011100001110101111100110001000000101101110100111100111100110110  
0111010000001100011100,  
1110101111111101110000111010111110011000100000010110111010011110011110011011001  
11010000001100011100
```

Last updated on 17/10/2025

Envío de circuitos con formato específicos a Azure Quantum

Artículo • 26/02/2025

Aprenda a usar el `azure-quantum` Python paquete para enviar circuitos en formatos específicos al servicio Azure Quantum. En este artículo se muestra cómo enviar circuitos en los siguientes formatos:

- [Enviar circuitos QIR](#)
- [Envío de circuitos específicos del proveedor](#)

Para más información, consulte [Circuitos cuánticos](#).

Requisitos previos

Para ejecutar los circuitos en un cuaderno en Azure Portal, necesita lo siguiente:

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Para más información, consulte [Creación de un área de trabajo de Azure Quantum](#).

Para desarrollar y ejecutar los circuitos en Visual Studio Code, también necesita lo siguiente:

- Un Python entorno con [Python y Pip](#) instalado.
- VS Code con las [extensiones de Azure Quantum Development Kit](#), [Python](#) y [Jupyter](#) instaladas.
- Los paquetes de Azure Quantum `qsharp`, `azure-quantum` y `ipykernel`.

Bash

```
python -m pip install --upgrade qsharp azure-quantum ipykernel
```

Crear un nuevo cuaderno de Jupyter Notebook

Puede crear un cuaderno en VS Code o directamente en el portal de Azure Quantum.

1. Inicie sesión en [Azure Portal](#) y seleccione el área de trabajo del paso anterior.
2. En la hoja izquierda, seleccione **Cuadernos**.
3. Haga clic en **Mis cuadernos** y, luego, en **Agregar nuevo**.
4. En **Tipo de kernel**, seleccione **IPython**.
5. Escriba un nombre para el archivo y haga clic en **Crear archivo**.

Cuando se abre el nuevo cuaderno, se crea automáticamente el código de la primera celda, en función de la información de la suscripción y del área de trabajo.

Python

```
from azure.quantum import Workspace
workspace = Workspace (
    resource_id = "", # Your resource_id
    location = "" # Your workspace location (for example, "westus")
)
```

Enviar circuitos con formato QIR

La representación intermedia cuántica (QIR) es una representación intermedia que actúa como una interfaz común entre lenguajes de programación cuántica o marcos y plataformas de cálculo cuántico de destino. Para obtener más información, consulte [Quantum Intermediate Representation](#).

1. Cree el circuito QIR. Por ejemplo, el código siguiente crea un circuito de entrelazamiento simple.

Python

```
QIR_routine = """%Result = type opaque
%Qubit = type opaque

define void @ENTRYPOINT__main() #0 {
    call void @__quantum__qis__h__body(%Qubit* inttoptr (i64 0 to %Qubit*))
    call void @__quantum__qis__cx__body(%Qubit* inttoptr (i64 0 to %Qubit*), %Qubit* inttoptr (i64 1 to %Qubit*))
    call void @__quantum__qis__h__body(%Qubit* inttoptr (i64 2 to %Qubit*))
    call void @__quantum__qis__cz__body(%Qubit* inttoptr (i64 2 to %Qubit*), %Qubit* inttoptr (i64 0 to %Qubit*))
    call void @__quantum__qis__h__body(%Qubit* inttoptr (i64 2 to
```

```

%Qubit*))
    call void @__quantum__qis__h__body(%Qubit* inttoptr (i64 3 to
%Qubit*))
    call void @__quantum__qis__cz__body(%Qubit* inttoptr (i64 3 to
%Qubit*), %Qubit* inttoptr (i64 1 to %Qubit*))
    call void @__quantum__qis__h__body(%Qubit* inttoptr (i64 3 to
%Qubit*))
    call void @__quantum__qis__mz__body(%Qubit* inttoptr (i64 2 to
%Qubit*), %Result* inttoptr (i64 0 to %Result*)) #1
    call void @__quantum__qis__mz__body(%Qubit* inttoptr (i64 3 to
%Qubit*), %Result* inttoptr (i64 1 to %Result*)) #1
    call void @__quantum__rt__tuple_record_output(i64 2, i8* null)
    call void @__quantum__rt__result_record_output(%Result* inttoptr (i64
0 to %Result*), i8* null)
    call void @__quantum__rt__result_record_output(%Result* inttoptr (i64
1 to %Result*), i8* null)
    ret void
}

declare void @__quantum__qis__ccx__body(%Qubit*, %Qubit*, %Qubit*)
declare void @__quantum__qis__cx__body(%Qubit*, %Qubit*)
declare void @__quantum__qis__cy__body(%Qubit*, %Qubit*)
declare void @__quantum__qis__cz__body(%Qubit*, %Qubit*)
declare void @__quantum__qis__rx__body(double, %Qubit*)
declare void @__quantum__qis__rx__body(double, %Qubit*, %Qubit*)
declare void @__quantum__qis__ry__body(double, %Qubit*)
declare void @__quantum__qis__ryy__body(double, %Qubit*, %Qubit*)
declare void @__quantum__qis__rz__body(double, %Qubit*)
declare void @__quantum__qis__rzz__body(double, %Qubit*, %Qubit*)
declare void @__quantum__qis__h__body(%Qubit*)
declare void @__quantum__qis__s__body(%Qubit*)
declare void @__quantum__qis__s__adj(%Qubit*)
declare void @__quantum__qis__t__body(%Qubit*)
declare void @__quantum__qis__t__adj(%Qubit*)
declare void @__quantum__qis__x__body(%Qubit*)
declare void @__quantum__qis__y__body(%Qubit*)
declare void @__quantum__qis__z__body(%Qubit*)
declare void @__quantum__qis__swap__body(%Qubit*, %Qubit*)
declare void @__quantum__qis__mz__body(%Qubit*, %Result* writeonly) #1
declare void @__quantum__rt__result_record_output(%Result*, i8*)
declare void @__quantum__rt__array_record_output(i64, i8*)
declare void @__quantum__rt__tuple_record_output(i64, i8*)

attributes #0 = { "entry_point" "output_labeling_schema"
"qir_profiles"="base_profile" "required_num_qubits"="4"
"required_num_results"="2" }
attributes #1 = { "irreversible" }

; module flags

!llvm.module.flags = !{!0, !1, !2, !3}

!0 = !{i32 1, !"qir_major_version", i32 1}
!1 = !{i32 7, !"qir_minor_version", i32 0}
!2 = !{i32 1, !"dynamic_qubit_management", i1 false}

```

```
!3 = !{i32 1, !"dynamic_result_management", i1 false}
""
```

2. Cree una `submit_qir_job` función auxiliar para enviar el circuito QIR a `.target`. Tenga en cuenta que los formatos de datos de entrada y salida se especifican como `qir.v1` y `microsoft.quantum-results.v1`, respectivamente.

Python

```
# Submit the job with proper input and output data formats
def submit_qir_job(target, input, name, count=100):
    job = target.submit(
        input_data=input,
        input_data_format="qir.v1",
        output_data_format="microsoft.quantum-results.v1",
        name=name,
        input_params = {
            "entryPoint": "ENTRYPOINT__main",
            "arguments": [],
            "count": count
        }
    )

    print(f"Queued job: {job.id}")
    job.wait_until_completed()
    print(f"Job completed with state: {job.details.status}")
    #if job.details.status == "Succeeded":
    result = job.get_results()

    return result
```

3. Seleccione y target envíe el circuito QIR a Azure Quantum. Por ejemplo, para enviar el circuito QIR al simulador target de IonQ:

Python

```
target = workspace.get_targets(name="ionq.simulator")
result = submit_qir_job(target, QIR_routine, "QIR routine")
result
```

Output

```
{'Histogram': ['(0, 0)', 0.5, '(1, 1)', 0.5]}
```

Envío de un circuito con un formato específico del proveedor a Azure Quantum

Además de los lenguajes QIR, como Q# o Qiskit, puede enviar circuitos cuánticos en formatos específicos del proveedor a Azure Quantum. Cada proveedor tiene su propio formato para representar circuitos cuánticos.

- [IonQ](#)
- [PASQAL](#)
- [Quantinuum](#)
- [Rigetti](#)

Envío de un circuito a IonQ con formato JSON

1. Cree un circuito cuántico mediante el formato JSON independiente del lenguaje compatible con [IonQ targets](#), tal como se describe en la [documentación](#) de IonQ API. El ejemplo siguiente crea una superposición entre tres cúbits:

Python

```
circuit = {
    "qubits": 3,
    "circuit": [
        {
            "gate": "h",
            "target": 0
        },
        {
            "gate": "cnot",
            "control": 0,
            "target": 1
        },
        {
            "gate": "cnot",
            "control": 0,
            "target": 2
        },
    ]
}
```

2. Envíe el circuito a IonQ target. En el ejemplo siguiente se usa el simulador de IonQ, que devuelve un objeto `Job`.

Python

```
target = workspace.get_targets(name="ionq.simulator")
job = target.submit(circuit)
```

3. Espere hasta que se complete el trabajo y obtenga los resultados.

Python

```
results = job.get_results()
print(results)
```

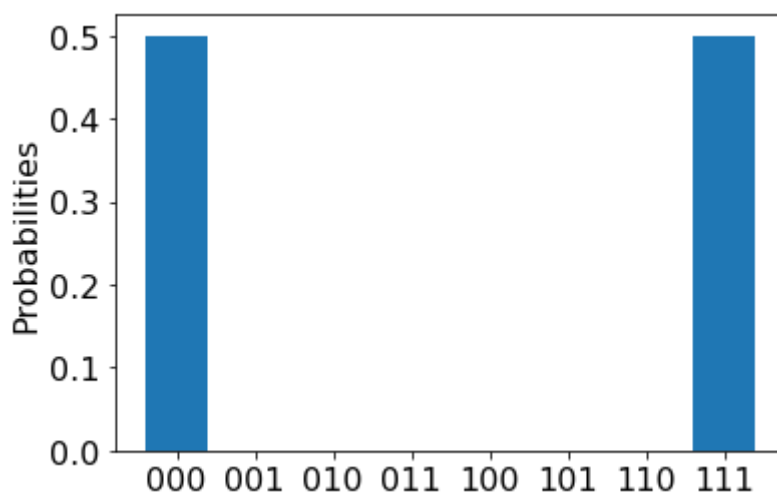
Output

```
.....
{'duration': 8240356, 'histogram': {'0': 0.5, '7': 0.5}}
```

4. A continuación, podemos visualizar los resultados mediante [Matplotlib](#).

Python

```
import pylab as pl
pl.rcParams["font.size"] = 16
hist = {format(n, "03b"): 0 for n in range(8)}
hist.update({format(int(k), "03b"): v for k, v in
results["histogram"].items()})
pl.bar(hist.keys(), hist.values())
pl.ylabel("Probabilities")
```



5. Antes de ejecutar un trabajo en la QPU, debe calcular cuánto costará ejecutarse.

ⓘ Nota

Para obtener los precios más actuales, consulte los [precios de IonQ](#) o vaya a su área de trabajo y consulte las opciones de precios en la hoja "Proveedores" en: aka.ms/aq/myworkspaces.

Envío de un circuito a PASQAL mediante el SDK de Pulser

Para enviar un circuito a PASQAL, puede usar el SDK de Pulser para crear secuencias de pulso y enviarlos al PASQAL target.

Instalación del SDK de Pulser

[Pulser](#) es un marco para crear, simular y ejecutar secuencias de pulso para dispositivos cuánticos neutros atom. Está diseñado por PASQAL como un paso a través para enviar experimentos cuánticos a sus procesadores cuánticos. Para obtener más información, consulte [la documentación](#) de Pulser.

Para enviar las secuencias de pulso, instale primero los paquetes del SDK de Pulser:

Python

```
try:
    import pulser
    import pulser_pasqal
except ImportError:
    !pip -q install pulser pulser-pasqal --index-url https://pypi.org/simple
```

Creación de un registro cuántico

Debe definir un registro y un diseño antes de continuar. El registro especifica dónde se organizarán los átomos, mientras que el diseño especifica el posicionamiento de capturas necesarios para capturar y estructurar estos átomos dentro del registro.

Para obtener más información sobre los diseños, consulte la documentación [de](#) Pulser.

- En primer lugar, se crea un objeto "devices" para importar el equipo targetcuántico PASQAL , [Fresnel](#).

Python

```
from pulser_pasqal import PasqalCloud
```

```
devices = PasqalCloud().fetch_available_devices()
QPU = devices["FRESNEL"]
```

Diseños calibrados previamente

El dispositivo define una lista de [diseños](#) pre calibrados. Puede crear el registro fuera de uno de estos diseños.

Esta es la opción recomendada porque mejorará el rendimiento de la QPU.

- Opción 1: Definir el registro mediante diseños calibrados previamente

Inspeccione los diseños disponibles en Fresnel y defina el registro de este diseño. Consulte la documentación de pulser para obtener más información sobre cómo hacerlo.

Ejemplo:

Python

```
# let's say we are interested in the first layout available on the
device
layout = QPU.pre_calibrated_layouts[0]
# Select traps 1, 3 and 5 of the layout to define the register
traps = [1,3,5]
reg = layout.define_register(*traps)
# You can draw the resulting register to verify it matches your
expectations
reg.draw()
```

Diseños arbitrarios

Si los diseños pre calibrados no satisfacen los requisitos del experimento, puede crear un diseño personalizado.

Para cualquier registro arbitrario dado, una QPU de átomo neutro colocará trampas según el diseño, que después debe someterse a calibración. Dado que cada calibración requiere tiempo, generalmente es aconsejable reutilizar un diseño calibrado existente siempre que sea posible.

- Opción 2: Derivar automáticamente un diseño del registro definido

Esta opción permite la generación automática de un diseño basado en un registro especificado. Sin embargo, en el caso de los registros grandes, este proceso puede

producir soluciones sub-óptimas debido a limitaciones en el algoritmo usado para crear el diseño.

Python

```
from pulser import Register
qubits = {
    "q0": (0, 0),
    "q1": (0, 10),
    "q2": (8, 2),
    "q3": (1, 15),
    "q4": (-10, -3),
    "q5": (-8, 5),
}

reg = Register(qubits).with_automatic_layout(device)
```

- Opción 3: Definir el registro mediante un diseño definido manualmente
 - Crear un diseño arbitrario con 20 capturas colocadas aleatoriamente en un plano 2D

Python

```
import numpy as np
from pulser.register.register_layout import RegisterLayout

# Generating random coordinates
np.random.seed(301122) # Keeps results consistent between runs
traps = np.random.randint(0, 30, size=(20, 2))
traps = traps - np.mean(traps, axis=0)
# Creating the layout
layout = RegisterLayout(traps, slug="random_20")
```

- Definición del registro con identificadores de captura específicos

Python

```
trap_ids = [4, 8, 19, 0]
reg = layout.define_register(*trap_ids, qubit_ids=["a", "b", "c", "d"])
reg.draw()
```

Escribir una secuencia de pulsos

Los átomos neutros se controlan con pulsos láser. El SDK de Pulser permite crear secuencias de pulso para aplicar al registro cuántico.

1. En primer lugar, se definen los atributos de secuencia de pulsos declarando los canales que se usarán para controlar los átomos. Para crear un `Sequence`, debe proporcionar una `Register` instancia junto con el dispositivo donde se ejecutará la secuencia. Por ejemplo, el código siguiente declara un canal: `ch0`.

⚠ Nota

Puede usar el `QPU = devices["FRESNEL"]` dispositivo o importar un dispositivo virtual desde Pulser para obtener más flexibilidad. El uso de un `VirtualDevice` permite la creación de secuencias que está menos restringida por las especificaciones del dispositivo, lo que hace que sea adecuado para la ejecución en un emulador. Para obtener más información, consulte [la documentación](#) [↗] de Pulser.

Python

```
from pulser import Sequence

seq = Sequence(reg, QPU)
# print the available channels for your sequence
print(seq.available_channels)
# Declare a channel. In this example we will be using `rydberg_global`
seq.declare_channel("ch0", "rydberg_global")
```

2. Agregue pulsos a la secuencia. Para ello, cree y agregue pulsos a los canales que declaró. Por ejemplo, el código siguiente crea un pulso y lo agrega al canal `ch0`.

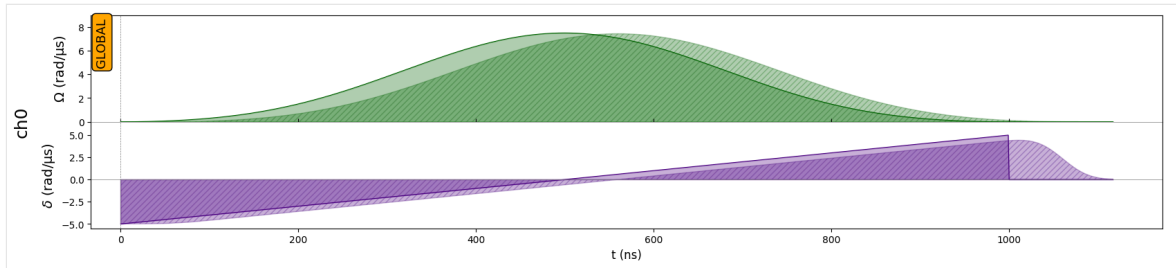
Python

```
from pulser import Pulse
from pulser.waveforms import RampWaveform, BlackmanWaveform
import numpy as np

amp_wf = BlackmanWaveform(1000, np.pi)
det_wf = RampWaveform(1000, -5, 5)
pulse = Pulse(amp_wf, det_wf, 0)
seq.add(pulse, "ch0")

seq.draw()
```

En la imagen siguiente se muestra la secuencia de pulsos.



Conversión de la secuencia en una cadena JSON

Para enviar las secuencias de pulso, debe convertir los objetos Pulser en una cadena JSON que se puede usar como datos de entrada.

Python

```
import json

# Convert the sequence to a JSON string
def prepare_input_data(seq):
    input_data = {}
    input_data["sequence_builder"] = json.loads(seq.to_abstract_repr())
    to_send = json.dumps(input_data)
    return to_send
```

Enviar la secuencia de pulso a PASQAL target

1. En primer lugar, debe establecer los formatos de datos de entrada y salida adecuados. Por ejemplo, el código siguiente establece el formato `pasqal.pulser.v1` de datos de entrada en y el formato de datos de salida en `pasqal.pulser-results.v1`.

Python

```
# Submit the job with proper input and output data formats
def submit_job(target, seq, shots):
    job = target.submit(
        input_data=prepare_input_data(seq), # Take the JSON string
        previously defined as input data
        input_data_format="pasqal.pulser.v1",
        output_data_format="pasqal.pulser-results.v1",
        name="PASQAL sequence",
        shots=shots # Number of shots
    )

    print(f"Queued job: {job.id}")
    return job
```

⚠ Nota

El tiempo necesario para ejecutar un trabajo en la QPU depende de los tiempos de cola actuales. Para ver el tiempo medio de cola de un target , seleccione la **hoja Proveedores** del área de trabajo.

2. Envíe el programa a PASQAL. Antes de enviar el código al hardware cuántico real, puede probar el código mediante el emulador `pasqal.sim.emu-tn` como `.target`

Python

```
target = workspace.get_targets(name="pasqal.sim.emu-tn") # Change to
"pasqal.qpu.fresnel" to use Fresnel QPU
job = submit_job(target, seq, 10)

job.wait_until_completed()
print(f"Job completed with state: {job.details.status}")
result = job.get_results()
print(result)
```

Output

```
{
  "1000000": 3,
  "0010000": 1,
  "0010101": 1
}
```

Envío de un circuito a Quantinuum mediante OpenQASM

1. Cree un circuito cuántico en la representación de [OpenQASM](#) [↗]. En el ejemplo siguiente, se crea un circuito de teletransporte:

py

```
circuit = """OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg c0[3];
h q[0];
cx q[0], q[1];
cx q[1], q[2];
measure q[0] -> c0[0];
measure q[1] -> c0[1];
```

```
measure q[2] -> c0[2];
"""
```

También puede cargar el circuito desde un archivo:

```
py

with open("my_teleport.qasm", "r") as f:
    circuit = f.read()
```

- Envíe el circuito al Quantinuum target. El siguiente ejemplo utiliza el validador de API de Quantinuum, que devuelve un objeto `Job`.

```
Python

target = workspace.get_targets(name="quantinuum.sim.h1-1sc")
job = target.submit(circuit, shots=500)
```

- Espere hasta que se complete el trabajo y obtenga los resultados.

```
Python

results = job.get_results()
print(results)
```

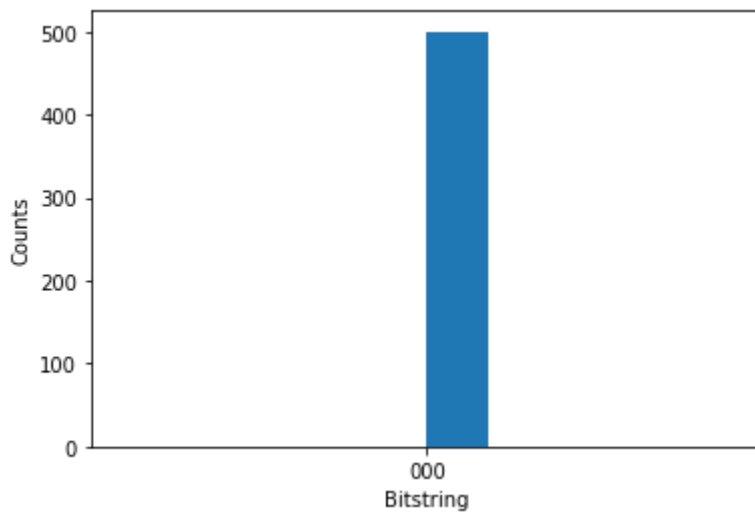
Output

```
.....
{'c0': ['000',
'000',
'000',
'000',
'000',
'000',
'000',
...
]}
```

- A continuación, podemos visualizar los resultados mediante [Matplotlib](#).

```
Python

import pylab as pl
pl.hist(results["c0"])
pl.ylabel("Counts")
pl.xlabel("Bitstring")
```



Si observa el histograma, puede observar que el generador de números aleatorios devuelve 0 cada vez, lo que no es muy aleatorio. Esto se debe a que, aunque el validador de API garantiza que el código se ejecutará correctamente en el hardware de Quantinuum, también devuelve 0 para cada medida cuántica. Para que el generador de números aleatorios sea verdadero, debe ejecutar el circuito en hardware cuántico.

5. Antes de ejecutar un trabajo en la QPU, debe calcular cuánto costará ejecutarse.

⚠ Nota

Para obtener los detalles de precios más actuales, consulte [Precios](#) de Azure Quantum o busque el área de trabajo y vea las opciones de precios en la pestaña "Proveedor" del área de trabajo a través de: aka.ms/aq/myworkspaces.

Envío de un circuito a Rigetti mediante Quil

La manera más fácil de enviar trabajos de Quil es usar el [paquete pyquil-for-azure-quantum](#), ya que permite usar las herramientas y la documentación de la [biblioteca pyQuil](#). Sin este paquete, pyQuil se puede usar para *construir* programas de Quil, pero no para enviarlos a Azure Quantum.

También puede construir programas de Quil manualmente y enviarlos directamente mediante el paquete `azure-quantum`.

Uso de pyquil-for-azure-quantum

1. En primer lugar, cargue las importaciones necesarias.

Python

```
from pyquil.gates import CNOT, MEASURE, H
from pyquil.quil import Program
from pyquil.quilbase import Declare
from pyquil_for_azure_quantum import get_qpu, get_qvm
```

2. Use la `get_qvm` función o `get_qpu` para obtener una conexión a QVM o QPU.

Python

```
qc = get_qvm() # For simulation
# qc = get_qpu("Ankaa-3") for submitting to a QPU
```

3. Cree un programa Quil. Se acepta cualquier programa Quil válido, pero el readout **debe** denominarse `ro`.

Python

```
program = Program(
    Declare("ro", "BIT", 2),
    H(0),
    CNOT(0, 1),
    MEASURE(0, ("ro", 0)),
    MEASURE(1, ("ro", 1)),
).wrap_in_numshots_loop(5)

# Optionally pass to_native_gates=False to .compile() to skip the
# compilation stage

result = qc.run(qc.compile(program))
data_per_shot = result.readout_data["ro"]
```

4. Aquí, `data_per_shot` es una `numpy` matriz, por lo que puede usar `numpy` métodos.

Python

```
assert data_per_shot.shape == (5, 2)
ro_data_first_shot = data_per_shot[0]
assert ro_data_first_shot[0] == 1 or ro_data_first_shot[0] == 0
```

5. Imprima todos los datos.

Python

```
print("Data from 'ro' register:")
for i, shot in enumerate(data_per_shot):
    print(f"Shot {i}: {shot}")
```

Importante

Actualmente, no se admite el envío de varios circuitos en un solo trabajo. Como solución alternativa, puede llamar al método `backend.run` para enviar cada circuito de forma asincrónica y, a continuación, capturar los resultados de cada trabajo. Por ejemplo:

Python

```
jobs = []
for circuit in circuits:
    jobs.append(backend.run(circuit, shots=N))

results = []
for job in jobs:
    results.append(job.result())
```

Contenido relacionado

- [Envíe un circuito con Qiskit a Azure Quantum.](#)
- [Envíe un circuito con Cirq to Azure Quantum.](#)

Comentarios

¿Le ha resultado útil esta página?

 Sí

 No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Creación y visualización de circuitos con Circuit Editor

18/06/2025

Circuit Editor es una característica del Kit de desarrollo de Quantum (QDK) que proporciona una interfaz gráfica donde puede crear, editar y visualizar diagramas de circuitos cuánticos dentro de los Q# proyectos. Puede usar los circuitos que construya directamente en sus Q# programas como operaciones invocables.

Cómo acceder a Circuit Editor

Para empezar a trabajar con Circuit Editor, siga estos pasos:

1. Cree un nuevo Q# proyecto en VS Code o abra un proyecto existente.
2. Cree un nuevo archivo de circuito con la extensión `.qsc`.
3. Abra el archivo de circuito en VS Code.

Circuit Editor es la vista predeterminada para `.qsc` archivos.

Funciones de Circuit Editor

Esto es lo que puede hacer con el Circuit Editor:

 Expandir tabla

Característica	Cómo usar
Insertar un elemento de circuito	Haga clic y arrastre el elemento desde el cuadro de herramientas al diagrama del circuito.
Quitar un elemento de circuito	Haga clic y arrastre el elemento fuera del diagrama del circuito. O bien, haga clic con el botón derecho en el elemento y elija Eliminar en el menú contextual.
Mover un elemento de circuito	Haga clic y arrastre el elemento a una nueva ubicación en el diagrama del circuito.
Copiar un elemento de circuito	Mantenga presionada la tecla Ctrl mientras hace clic y arrastra el elemento a una nueva ubicación en el diagrama del circuito.
Adición de un cúbit	Coloque un elemento del cuadro de herramientas en un nuevo cable en el diagrama de circuitos. O bien, mueva un elemento existente a una nueva conexión en el diagrama.

Característica	Cómo usar
Quitar un cúbit	Haga clic y arrastre el icono de cúbit fuera del diagrama del circuito. El cúbit inferior se elimina automáticamente al quitar todos los elementos del circuito de ese cúbit.
Reorganizar el orden de cúbits	Haga clic y arrastre los iconos de cúbit en el diagrama del circuito.
Agregar un control a una puerta	Haga clic con el botón derecho en el icono de puerta, elija Agregar control en el menú contextual y, a continuación, elija el cable de cúbit de control.
Quitar un control de una puerta	Haga clic con el botón derecho en el icono de puerta, elija Quitar control en el menú contextual y, a continuación, seleccione el icono del control que desea quitar. También puede hacer clic con el botón derecho en el icono de control directamente y elegir Quitar control o arrastrar el icono de control fuera del diagrama.
Convertir una puerta a su adyacente	Haga clic con el botón derecho en el icono de la puerta y elija Alternar adyacente en el menú contextual.
Definición de un argumento para una puerta	Cuando se coloca una puerta que requiere un argumento, aparece un cuadro de mensaje. Escriba un número o expresión en el cuadro de solicitud. Para actualizar el argumento, haga clic con el botón derecho en la puerta y elija Editar argumento . O bien, haga clic con el botón izquierdo en el texto del argumento en el icono de la puerta.

Cómo utilizar circuitos de Circuit Editor en proyectos de Q#

Los archivos de circuito con extensión `.qsc` definen operaciones a las que puedes hacer referencia desde el código Q# en el mismo proyecto Q#. Los circuitos de los archivos `.qsc` aparecen como cualquier otra operación Q# y son compatibles con el servicio de lenguaje de Q# con características como finalizaciones, ayuda de firma y definición de go-to.

El archivo de circuito define un espacio de nombres que procede del nombre de archivo, de manera similar a cómo funcionan los archivos Q#. El circuito en tu archivo `.qsc` define una operación Q# que tiene el mismo nombre que el archivo dentro de ese mismo espacio de nombres. Por ejemplo, para hacer referencia a un circuito desde el archivo `Foo.qsc` en su código de Q#, utilice la siguiente instrucción de importación: `import Foo.Foo;`

El siguiente Q# programa importa un circuito desde el archivo `JointMeasurement.qsc` y, a continuación, llama a la `JointMeasurement()` operación para aplicar el circuito a un sistema de cúbits.

Q#

```
import JointMeasurement.JointMeasurement;

operation Main() : Result[] {
    use qs = Qubit[4];
    ApplyToEach(H, qs[0..2]);
    let results = JointMeasurement(qs);
    ResetAll(qs);
    results
}
```

Contenido relacionado

- [Visualización de diagramas de circuitos cuánticos con Q#](#)

Configuración del modo de agente en VS Code para el Kit de desarrollo de Quantum

18/06/2025

Utilice el modo de agente de VS Code, con tecnología de GitHubCopilot, para mejorar la experiencia del creador con la extensión del Kit de desarrollo de Quantum (QDK).

El modo de agente es una experiencia de desarrollo asistida por IA que le ayuda a escribir y depurar código y a completar otras tareas de desarrollo en VS Code. Aunque puede usar el modo de agente para QDK sin ninguna configuración, siga estas sugerencias para sacar el máximo partido al modo de agente en los QDK proyectos:

1. [Adición Copilot de instrucciones para Q# y OpenQASM](#)
2. [Activación de las herramientas para el QDK modo de agente](#)
3. [Prueba de diferentes modelos de agente](#)

Agregue instrucciones Copilot para Q# y OpenQASM

QDK incluye un conjunto de instrucciones para Copilot que incluyen procedimientos recomendados para construir proyectos con Q#, OpenQASM y Azure Quantum.

Para agregar las instrucciones QDKCopilot a su configuración, siga estos pasos:

1. Presione **Ctrl+Mayús+P** para abrir la paleta de comandos.
2. Ingrese y elija **QDK: Añadir el archivo de instrucciones de Copilot para Q# y OpenQASM**.
3. En el cuadro de confirmación que se abre, elija **Sí**.

Las instrucciones de QDK se agregan a su contexto al usar archivos de GitHubCopilot Chat en Q# y OpenQASM. Puede configurar manualmente esta opción en

`chat.instructionsFilesLocations`.

Activa las herramientas para el QDK modo de agente

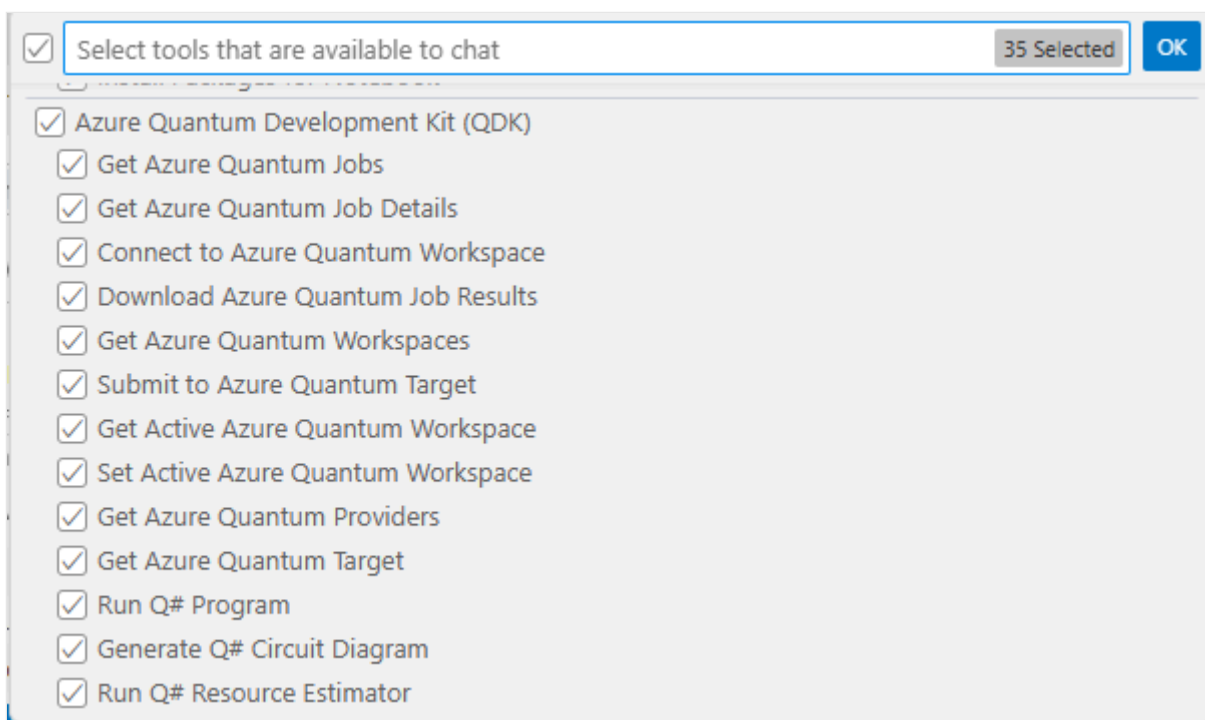
La QDK extensión incluye un conjunto de herramientas que adapta Copilot Chat para trabajar con Q# y Azure Quantum. Por ejemplo, puede simular Q# código, visualizar diagramas de circuitos cuánticos y realizar la estimación de recursos directamente desde la Copilot vista

Chat. También puede hacer cosas como conectarse al área de trabajo de Azure Quantum, enviar trabajos y ver el estado del trabajo y los resultados del trabajo.

Para activar las herramientas para el QDK modo de agente, siga estos pasos:

1. Abra Copilot Chat en VS Code.
2. Si el chat está en modo **Pregunta** o modo **de edición**, luego seleccione el menú desplegable de modo al final del cuadro de texto del chat y elija **Agente**.
3. Seleccione el icono Herramientas en la parte inferior del cuadro de entrada del chat.

Aparece una lista que contiene todas las herramientas del agente disponibles. Seleccione todas las herramientas bajo **Kit de desarrollo de Azure Quantum (QDK)** para activar las herramientas de QDK.



Prueba de diferentes modelos de agente

Copilot permite elegir entre un conjunto de diferentes modelos de lenguaje que se usarán en el modo de agente. Los diferentes modelos tienen diferentes puntos fuertes, por lo que el mejor modelo para usted depende de su caso de uso.

Para explorar diferentes modelos, seleccione el menú desplegable modelo en la parte inferior del cuadro de entrada de chat y elija uno de los modelos disponibles de la lista.

Pruebe algunas indicaciones

Ahora que está listo para sacar provecho de la IA en los proyectos de Q#, pruebe algunos mensajes en Copilot Chat.

Para empezar, aquí tienes algunas indicaciones de ejemplo:

Simule este programa para 1000 tomas y muéstrame un histograma.

Envíe este Q# programa a Azure Quantum.

Agregue pruebas para este Q# código.

Muéstrame mis trabajos recientes en Azure Quantum.

He estado pensando en aprender un poco sobre la computación cuántica, pero no tengo pistas de dónde empezar. ¿Puede ayudarme a escribir un programa muy sencillo, entenderlo e incluso ejecutarlo en un equipo cuántico real?

¿Qué es la computación cuántica híbrida?

Artículo • 07/08/2024

La *computación* cuántica híbrida hace referencia a los procesos y la arquitectura de un equipo clásico y un equipo cuántico que trabaja conjuntamente para resolver un problema. Con la última generación de arquitectura de computación cuántica híbrida disponible en Azure Quantum, puede empezar a programar equipos cuánticos mediante la combinación de instrucciones clásicas y cuánticas.

Azure Quantum representa una visión de futuro para la computación cuántica híbrida, donde ciertas arquitecturas ya están operativas, mientras que otras se están desarrollando activamente. En este artículo se describen los distintos enfoques de la computación cuántica híbrida y cómo se pueden usar para optimizar determinados problemas.

Agrupación de circuitos con computación cuántica por lotes

La computación cuántica por lotes permite enviar varios circuitos cuánticos como un único trabajo al hardware cuántico.

Normalmente, los circuitos cuánticos se envían de uno en uno como trabajos únicos a un destino de hardware cuántico. Cuando el cliente recibe el resultado de un circuito, el siguiente circuito se agrega como un nuevo trabajo a la cola. Sin embargo, el procesamiento por lotes de varios circuitos en un trabajo elimina la espera entre envíos de trabajos, lo que le permite ejecutar varios trabajos más rápido. Algunos ejemplos de problemas que pueden aprovechar la computación cuántica por lotes incluyen el algoritmo de Shor y la estimación de fase cuántica simple.

Con el modelo de computación por lotes, también puede procesar por lotes varios circuitos predefinidos en un trabajo. Los circuitos se envían al hardware cuántico tan pronto como se complete el circuito anterior, lo que reduce la espera entre envíos de trabajos.

En esta arquitectura, el estado de los cúbits se pierde entre cada envío de circuito.

ⓘ **Nota**

Actualmente, Azure Quantum no admite la computación cuántica por lotes.

Agrupación de trabajos con sesiones

Las sesiones permiten organizar varios trabajos de computación cuántica con la capacidad de ejecutar código clásico entre trabajos cuánticos. Podrá ejecutar algoritmos complejos para organizar y realizar un seguimiento mejor de los trabajos de computación cuántica individuales. Además, los trabajos agrupados en sesiones tienen prioridad sobre los trabajos que no son de sesión.

En este modelo, el recurso de proceso de cliente se mueve a la nube, lo que da lugar a una menor latencia y a la ejecución repetida del circuito cuántico con distintos parámetros. Aunque las sesiones permiten tiempos de cola más cortos y problemas de ejecución más largos, los estados de cúbit no se conservan entre cada iteración. Algunos ejemplos de problemas que pueden usar este enfoque son Variational Quantum Eigensolvers (VQE) y Quantum Approximate Optimization Algorithms (QAOA).

Para obtener más información, consulte [Introducción a las sesiones](#).

Ejecución de la computación cuántica híbrida

Con la computación cuántica híbrida, las arquitecturas clásicas y cuánticas están estrechamente acopladas, lo que permite realizar cálculos clásicos mientras los cúbits físicos son coherentes. Aunque limitado por la vida de cúbits y la corrección de errores, esto permite que los programas cuánticos se muevan de solo circuitos. Los programas ahora pueden usar construcciones de programación comunes para realizar mediciones de circuito medio, optimizar y reutilizar cúbits, y adaptarse en tiempo real a la QPU. Algunos ejemplos de escenarios que pueden aprovechar este modelo son la estimación de fase adaptable y el aprendizaje automático.

Para más información, consulte [Computación cuántica integrada](#).

Ejecución de la computación cuántica distribuida

En esta arquitectura, el cálculo clásico funciona junto con cúbits lógicos. Con el control clásico totalmente integrado y los cúbits lógicos de larga duración, el modelo de computación cuántica distribuida permite cálculos en tiempo real en recursos cuánticos y distribuidos. Los controles clásicos ya no se limitan a los bucles y permiten escenarios

como el modelado de materiales complejos o la evaluación de reacciones catalíticas completas.

ⓘ **Nota**

Actualmente, Azure Quantum no admite la computación cuántica distribuida.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Introducción a las sesiones

Las sesiones son una característica clave de la computación cuántica híbrida que permite agrupar varios trabajos de computación cuántica. Una sesión es una agrupación lógica de uno o varios trabajos enviados a un único target. Cada sesión tiene un identificador único asociado a cada trabajo de esa sesión. Las sesiones son útiles cuando desea ejecutar varios trabajos de computación cuántica en secuencia, con la capacidad de ejecutar código clásico entre trabajos cuánticos.

En este artículo se explica la arquitectura de las sesiones en la computación cuántica híbrida y cómo crear una nueva sesión.

Requisitos previos

Para crear una sesión, necesita los siguientes requisitos previos:

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de Azure Quantum trabajo. Para obtener más información, consulte [Creación de un Azure Quantum área de trabajo](#).
- Un entorno Python con [Python y Pip](#) instalado.
- El paquete `azure-quantum` Python. Si desea usar Qiskit o Cirq, debe instalar el paquete `azure-quantum` con los extras `qiskit` o `cirq`.

Bash

```
pip install --upgrade azure-quantum[qiskit]
```

o

Bash

```
pip install --upgrade azure-quantum[cirq]
```

¿Qué es una sesión?

En las sesiones, el recurso de proceso de cliente se puede mover a la nube, lo que da lugar a una menor latencia y a la capacidad de repetir la ejecución del circuito cuántico con distintos parámetros. Los trabajos se pueden agrupar lógicamente en una sesión y los trabajos de esa sesión se pueden priorizar en los trabajos que no son de sesión. Aunque los estados de cúbit

no se conservan entre trabajos, una sesión permite tiempos de cola más cortos para trabajos y problemas de ejecución más largos.

Las sesiones permiten organizar varios trabajos de computación cuántica con la capacidad de ejecutar código clásico entre trabajos cuánticos. Podrá ejecutar algoritmos complejos para organizar y realizar un seguimiento mejor de los trabajos de computación cuántica individuales.

Un escenario de usuario clave en el que es posible que quiera combinar trabajos en una sesión es *algoritmos cuánticos con parámetros*, donde la salida de un trabajo de computación cuántica informa a los parámetros del siguiente trabajo de computación cuántica. Los ejemplos más comunes de este tipo de algoritmo son Variational Quantum Eigensolvers (VQE) y Quantum Approximate Optimization Algorithms (QAOA).

Hardware admitido

Las sesiones se admiten en todos los proveedores de hardware de computación cuántica. En algunos casos, los trabajos que se envían dentro de una sesión se les da prioridad en la cola de ese target. Para obtener más información, consulte [Comportamiento de destino](#).

Creación de una sesión

Para crear una sesión, siga estos pasos:

Q# + Python

En este ejemplo se muestra cómo crear una sesión con Q# código insertado mediante un Jupyter Notebook en Visual Studio Code (VS Code). También puede crear sesiones mediante un [Python programa](#) que invoque un programa adyacente Q#.

ⓘ Nota:

Las sesiones se administran con Python, incluso cuando se está ejecutando código en línea Q#.

1. En VS Code, abra el menú **Ver** y elija **Paleta de comandos**.
2. Introduzca y seleccione **Crear: Nuevo Jupyter Notebook**.
3. En la parte superior derecha, VS Code detectará y mostrará la versión de Python y el entorno virtual Python seleccionado para el cuaderno. Si tiene varios Python

entornos, es posible que tenga que seleccionar un núcleo mediante el selector de núcleo ubicado en la parte superior derecha. Si no se detectó un entorno, consulte [Jupyter Notebook en VS Code](#) para detalles sobre la configuración.

4. En la primera celda del cuaderno, ejecute el código siguiente:

Python

```
from qdk.azure import Workspace

workspace = Workspace(
    resource_id = "", # add your resource ID
    location = "", # add your location, for example "westus"
)
```

5. Agregue una nueva celda en el cuaderno e importe el `qsharp` Python paquete:

Python

```
from qdk import qsharp
```

6. Elija su [cuántico target](#). En este ejemplo, target es el [simulador de IonQ](#).

Python

```
target = workspace.get_targets("ionq.simulator")
```

7. Seleccione las configuraciones del [target perfil](#), ya sea `Base`, `Adaptive_RI` o `Unrestricted`.

Python

```
qsharp.init(target_profile=qsharp.TargetProfile.Base) # or
qsharp.TargetProfile.Adaptive_RI, qsharp.TargetProfile.Unrestricted
```

⚠ **Nota:**

`Adaptive_RI` target actualmente se admiten trabajos de perfil en Quantinuum targets. Para más información, consulte [Computación](#) cuántica híbrida integrada.

8. Escriba el Q# programa. Por ejemplo, el siguiente Q# programa genera un bit aleatorio. Para ilustrar el uso de argumentos de entrada, este programa toma un entero, `n` y una matriz de ángulos, `angle`, como entrada.

Q#

```
%%qsharp
import Std.Measurement.*;
import Std.Arrays.*;

operation GenerateRandomBits(n: Int, angle: Double[]) : Result[] {
    use qubits = Qubit[n]; // n parameter as the size of the qubit array
    for q in qubits {
        H(q);
    }
    R(PauliZ, angle[0], qubits[0]); // arrays as entry-points parameters
    R(PauliZ, angle[1], qubits[1]);
    let results = MeasureEachZ(qubits);
    ResetAll(qubits);
    return results;
}
```

9. A continuación, creará una sesión. Supongamos que quiere ejecutar la `GenerateRandomBit` operación tres veces, por lo que se usa `target.submit` para enviar la Q# operación con los `target` datos y repetir el código tres veces: en un escenario real, puede que desee enviar diferentes programas en lugar del mismo código.

Python

```
angle = [0.0, 0.0]
with target.open_session(name="Q# session of three jobs") as session:
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle}")), name="Job 1", shots=100) # First job submission
    angle[0] += 1
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle}")), name="Job 2", shots=100) # Second job submission
    angle[1] += 1
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle}")), name="Job 3", shots=100) # Third job submission

session_jobs = session.list_jobs()
[session_job.details.name for session_job in session_jobs]
```

Importante

Cuando pasas argumentos como parámetros a la tarea, los argumentos están formateados en la expresión Q# cuando se llama a `qsharp.compile`. Esto significa que debe dar formato a los argumentos como Q# objetos. En este ejemplo, dado que los arrays en Python ya se muestran utilizando el formato `[item0, item1, ...]`, los argumentos de entrada son compatibles con el formato Q#. En el caso de otras Python estructuras de datos, pueda necesitar

más manejos para obtener los valores de cadena insertados en Q# de manera compatible.

10. Una vez creada una sesión, puede usar `workspace.list_session_jobs` para recuperar una lista de todos los trabajos de la sesión. Para obtener más información, consulte [Administración de sesiones](#).

Comportamiento objetivo

Cada proveedor de hardware cuántico define su propia heurística para administrar mejor la priorización de trabajos dentro de una sesión.

Quantinuum

Si decide enviar trabajos dentro de su sesión a [Quantinuum target](#), su sesión tiene acceso exclusivo al hardware siempre que los trabajos se pongan en cola con un intervalo de un minuto entre cada uno. Después, los trabajos se aceptan y gestionan con la lógica de la puesta en cola y priorización estándar.

Contenido relacionado

- [Administración de las sesiones](#)
- [Ejecución de la computación cuántica híbrida](#)

Last updated on 12/11/2025

Administración de las sesiones

En Azure Quantum, puede agrupar varios trabajos en un único destino para administrar eficazmente sus trabajos. Esto se denomina sesión. Para obtener más información, consulte [Introducción a las sesiones](#).

En este artículo, aprenderá a usar sesiones para gestionar los trabajos de forma manual. También obtendrá información sobre las políticas de errores del trabajo y cómo evitar los tiempos de espera en las sesiones.

Prerrequisitos

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de Azure Quantum trabajo. Para obtener más información, consulte [Creación de un Azure Quantum área de trabajo](#).
- Un entorno Python con [Python y Pip](#) instalado.
- El paquete `azure-quantum` Python. Si desea usar Qiskit o Cirq, debe instalar el paquete `azure-quantum` con el extra `qiskit` o `cirq`.

Bash

```
pip install --upgrade azure-quantum[qiskit,cirq]
```

ⓘ Nota:

Las sesiones se gestionan con Python, incluso cuando se ejecuta código en línea con Q#.

Supervisión de las sesiones

Puede usar la hoja **Administración de trabajos** en el área de trabajo de Quantum para ver todos los elementos enviados de nivel superior, incluidas las sesiones y los trabajos individuales que no están asociados a ninguna sesión.

1. Seleccione la sección **Administración de trabajos** en el área de trabajo de Quantum.
2. Identifique los trabajos de tipo **Session**. En esta vista, puede encontrar el identificador único de una sesión en la columna **Id** y supervisar el **estado** de la sesión. Las sesiones pueden tener los siguientes estados:

- **En espera:** los trabajos dentro de la sesión se están ejecutando actualmente.
- **Correcto:** la sesión finalizó correctamente.
- **Tiempo de espera:** si no se envía ningún nuevo trabajo dentro de la sesión durante 10 minutos, esa sesión agota el tiempo de espera. Para obtener más información, consulte [Tiempos de espera de sesión](#).
- **Error:** si se produce un error en un trabajo dentro de una sesión, esa sesión finaliza e informa del estado **Error**. Para obtener más información, consulte [Política de fallo de trabajos en sesiones](#).

3. Haga clic en el nombre de una sesión para obtener más detalles.

4. Puede ver la lista de **Todos los trabajos** dentro de la sesión y supervisar sus estados.

Recuperar y enumerar sesiones

En la tabla siguiente se muestran los Python comandos para obtener la lista de todas las sesiones y todos los trabajos de una sesión determinada.

 Expandir tabla

Command	Description
<code>workspace.list_sessions()</code> o <code>session.list_sessions()</code>	Recupere una lista de todas las sesiones de un área de trabajo de Quantum.
<code>workspace.get_session(sessionId)</code> o <code>session.get_session(sessionId)</code>	Recupere la sesión con el identificador <code>sessionId</code> . Cada sesión tiene un identificador único.
<code>workspace.list_session_jobs(sessionId)</code> o <code>session.list_session_jobs(sessionId)</code>	Recupere una lista de todos los trabajos de la sesión con el identificador <code>sessionId</code> . Cada sesión tiene un identificador único.

Por ejemplo, el código siguiente define una función que obtiene una sesión con un número mínimo de tareas. A continuación, para esa sesión, enumera todos los trabajos, el número total de trabajos y los primeros 10 trabajos.

Python

```
def get_a_session_with_jobs(min_jobs):
    all_sessions = workspace.list_sessions() # list of all sessions
    for session in all_sessions:
        if len(workspace.list_session_jobs(session.id)) >= min_jobs:
            return session

session = get_a_session_with_jobs(min_jobs=3) # Get a Session with at least 3 jobs

session_jobs = workspace.list_session_jobs(session.id) # List of all jobs within Session ID
```

```
print(f"Job count: {len(session_jobs)} \n")
print(f"First 10 jobs for session {session.id}:")
for job in session_jobs[0:10]:
    print(f"Id: {job.id}, Name={job.details.name}")
```

Abrir o cerrar sesiones manualmente

Para crear una nueva sesión, se recomienda seguir los pasos descritos en [Introducción a las sesiones](#). Si quiere crear sesiones manualmente en su lugar, elija la pestaña del entorno de desarrollo.

Q# + Python

1. Cree un objeto Session.

Python

```
from azure.quantum.job.session import Session, SessionDetails,
SessionJobFailurePolicy
import uuid

session = Session(
    workspace=workspace, # required
    id=f"{uuid.uuid1()}", # optional, if not passed will use uuid.uuid1()
    name="", # optional, will be blank if not passed
    provider_id="ionq", # optional, if not passed will try to parse from
the target
    target="ionq.simulator", # required
    job_failure_policy=SessionJobFailurePolicy.ABORT # optional, defaults
to abort
)

print(f"Session status: {session.details.status}")
```

⚠ Nota:

En este momento, la sesión solo existe en el cliente y puede ver que el estado es **Ninguno**. Para ver el estado de la sesión, primero debe crear la sesión en el servicio.

2. Para crear una sesión en el servicio, puede usar `workspace.open_session(session)` o `session.open()`.

3. Puede actualizar el **estado** y los detalles de la sesión con `session.refresh()` o obteniendo un nuevo objeto de sesión de un identificador de sesión.

Python

```
same_session = workspace.get_session(session.id)
print(f"Session: {session.details} \n")
print(f"Session: {same_session.details} \n")
```

4. Puede **cerrar** una sesión con `session.close()` o `workspace.close_session(session)`.
5. Para **adjuntar la sesión** a un destino, puede usar `target.latest_session`.
6. Puede **esperar a que** se complete una sesión:

Python

```
session_jobs = session.list_jobs()
[session_job.id for session_job in session_jobs]

import time
while (session.details.status != "Succeeded" and session.details.status !=
"Failed" and session.details.status != "TimedOut"):
    session.refresh()
    time.sleep(5)
```

Pasar argumentos a Q#

Si la Q# operación toma argumentos de entrada, esos argumentos se pasan durante el envío del trabajo, que es Python código. Esto significa que debe dar formato a los argumentos como Q# objetos.

Cuando pasas argumentos como parámetros al trabajo, los argumentos se formatean como Q# de código cuando se llama a `qsharp.compile`, por lo que los valores de Python deben formatearse en una cadena como sintaxis válida de Q#.

Tenga en cuenta el siguiente Q# programa, que toma un entero, `n`, y una matriz de ángulos, `angle`, como entrada.

Q#

```
import Std.Measurement.*;
import Std.Arrays.*;

operation GenerateRandomBits(n: Int, angle: Double[]) : Result[] {
```

```

use qubits = Qubit[n]; // n parameter as the size of the qubit array
for q in qubits {
    H(q);
}
R(PauliZ, angle[0], qubits[0]); // arrays as entry-points parameters
R(PauliZ, angle[1], qubits[1]);
let results = MeasureEachZ(qubits);
ResetAll(qubits);
return results;
}

```

Quiere ejecutar la `GenerateRandomBits` operación tres veces con `n=2` y diferentes ángulos. Puede usar el código siguiente Python para enviar tres trabajos con distintos ángulos.

Python

```

angle = [0.0, 0.0]
with target.open_session(name="Q# session of three jobs") as session:
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle})"),
name="Job 1", shots=100) # First job submission
    angle[0] += 1
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle})"),
name="Job 2", shots=100) # Second job submission
    angle[1] += 1
    target.submit(input_data=qsharp.compile(f"GenerateRandomBits(2, {angle})"),
name="Job 3", shots=100) # Third job submission

session_jobs = session.list_jobs()
[session_job.details.name for session_job in session_jobs]

```

En este ejemplo, dado que las matrices de Python ya se imprimen como `[item0, item1, ...]`, los argumentos de entrada coinciden con el Q# formato. Para otras Python estructuras de datos, podría necesitar más manipulación para insertar los valores de cadena en Q# de manera compatible. Por ejemplo, una Q# tupla debe estar entre paréntesis con valores separados por comas.

Tiempos de espera de sesión

Una sesión expira si no se envía ningún nuevo trabajo en 10 minutos. La sesión informa que el estado es **TimedOut**. Para evitar esta situación, agregue un `with` bloque mediante `backend.open_session(name="Name")`, por lo que el servicio invoca la sesión `close()` al final del bloque de código.

⚠ **Nota:**

Si hay errores o errores en el programa, puede tardar más de 10 minutos en enviar un nuevo trabajo después de que se completen los trabajos anteriores en la sesión.

Los fragmentos de código siguientes muestran un ejemplo de una sesión que agota el tiempo de espera después de 10 minutos porque no se envían nuevos trabajos. Para evitarlo, el siguiente fragmento de código muestra cómo usar un `with` bloque para crear una sesión.

Python

```
#Example of a session that times out
```

```
session = backend.open_session(name="Qiskit circuit session") # Session times out because only contains one job
backend.run(circuit=circuit, shots=100, job_name="Job 1")
```

Python

```
#Example of a session that includes a with block to avoid timeout
```

```
with backend.open_session(name="Qiskit circuit session") as session: # Use a with block to submit multiple jobs within a session
    job1 = backend.run(circuit=circuit, shots=100, job_name="Job 1") # First job submission
    job1.wait_for_final_state()
    job2 = backend.run(circuit=circuit, shots=100, job_name="Job 2") # Second job submission
    job2.wait_for_final_state()
    job3 = backend.run(circuit=circuit, shots=100, job_name="Job 3") # Third job submission
    job3.wait_for_final_state()
```

Directiva de fallo de tarea en sesiones

La directiva predeterminada para una sesión cuando se produce un error en un trabajo es finalizar esa sesión. Si envía un trabajo adicional dentro de la misma sesión, el servicio lo rechaza y la sesión notifica un estado De **error**. Los trabajos en curso se cancelan.

Sin embargo, este comportamiento se puede cambiar especificando una directiva de error de trabajo de `job_failure_policy=SessionJobFailurePolicy.CONTINUE`, en lugar del valor predeterminado `SessionJobFailurePolicy.ABORT`, al crear la sesión. Cuando la directiva de error del trabajo es `CONTINUE`, el servicio sigue aceptando trabajos. La sesión notifica un estado de **Errores** en este caso, que cambiará a **Error** una vez cerrada la sesión.

Si la sesión nunca se cierra y expira el tiempo de espera, el estado es **TimedOut**, incluso si los trabajos han fallado.

Por ejemplo, el siguiente programa crea una sesión con tres trabajos. Se produce un error en el primer trabajo porque especifica "garbage" como datos de entrada. Para evitar el final de la sesión en este momento, el programa muestra cómo agregar `job_failure_policy=SessionJobFailurePolicy.CONTINUE` al crear la sesión.

Python

```
#Example of a session that does not close but reports Failure(s) when a jobs fails

with target.open_session(name="JobFailurePolicy Continue",
job_failure_policy=SessionJobFailurePolicy.CONTINUE) as session:
    target.submit(input_data="garbage", name="Job 1") #Input data is missing, this
job fails
    target.submit(input_data=quil_program, name="Job 2") #Subsequent jobs are
accepted because of CONTINUE policy
    target.submit(input_data=quil_program, name="Job 3")
```

Contenido relacionado

- [Ejecución de la computación cuántica híbrida](#)
- [Introducción a las sesiones](#)

Last updated on 12/11/2025

Ejecución de trabajos de computación cuántica híbrida con perfil adaptable target

La informática híbrida combina procesos de computación clásica y cuántica para resolver problemas complejos.

En la informática híbrida, el código clásico controla la ejecución de operaciones cuánticas basadas en medidas de circuito medio mientras los cúbits físicos permanecen activos. Puede usar técnicas de programación comunes, como condicionales anidados, bucles y llamadas de función, en un único programa cuántico para ejecutar problemas complejos, lo que reduce el número de tomas necesarias. Con las técnicas de reutilización de cúbits, los programas más grandes se pueden ejecutar en máquinas con un número menor de cúbits.

En este artículo se explica cómo enviar trabajos híbridos a Azure Quantum mediante el [Adaptive Rtarget](#) perfil. El Adaptive Rtarget perfil ofrece compatibilidad con medidas de circuito medio, flujo de control basado en medidas, restablecimiento de cúbits y cálculo de enteros clásico.

Requisitos previos

- Una cuenta de Azure con una suscripción activa. Si no tiene una cuenta de Azure, regístrese gratuitamente y regístrese para obtener una [suscripción](#) de pago por uso.
- Un área de trabajo de Azure Quantum. Para más información, consulte [Creación de un área de trabajo de Azure Quantum](#).
- Si desea enviar Q# programas independientes, necesita los siguientes requisitos previos:
 - [Visual Studio Code](#) con la Quantum Development Kit de Azure instalada.
 - La versión más reciente de la Quantum Development Kit de Azure.
- Si quiere enviar programas y Q# Python, necesita los siguientes requisitos previos:
 - Un entorno de Python con [Python y Pip](#) instalados.
 - Azure Quantum `azure-quantum` y `qsharp` los paquetes.

Bash

```
pip install --upgrade azure-quantum qsharp
```

Se ha agregado compatibilidad con targets.

Para ejecutar trabajos de computación cuántica híbrida, debe seleccionar un proveedor cuántico que admita el [Adaptive Rtarget](#) perfil.

Actualmente, el perfil adaptable target en Azure Quantum se admite en [Quantinuum](#)targets.

Envío de trabajos de RI adaptables

Para enviar trabajos de computación cuántica híbrida, debe configurar el [target perfil](#) como **Adaptive RI**, donde RI significa "qubit Reset y Integer computations".

El Adaptive RI target perfil ofrece compatibilidad con medidas de circuito medio, flujo de control basado en medidas, restablecimiento de cúbits y cálculo de enteros clásico.

Puede enviar trabajos cuánticos híbridos a Azure Quantum como Q# programas independientes o python + Q# programas. Para configurar el target perfil para trabajos cuánticos híbridos, consulte las secciones siguientes.

Q# en Visual Studio Code

Para configurar el target perfil para trabajos híbridos en Visual Studio Code, elija una de las siguientes opciones:

- Si su Q# archivo no forma parte de un Q# proyecto, abra el archivo y escriba `@EntryPoint(Adaptive_RI)` antes de la operación de punto de entrada del programa.
- Si el Q# archivo forma parte de un Q# proyecto, agregue lo siguiente al archivo del `qsharp.json` proyecto:

JSON

```
{
  "targetProfile": "adaptive_ri"
}
```

Al configurar el perfil target en **Adaptive RI**, puede enviar el programa Q# como un trabajo cuántico híbrido a Quantinuum. Para ello, siga estos pasos:

1. Abra el menú **Ver** y elija **Paleta de comandos**, escriba **QDK: Conéctese a un área de trabajo de Azure Quantum** y presione **Entrar**.
2. Seleccione **Cuenta** de Azure y siga las indicaciones para conectarse a su directorio, suscripción y área de trabajo preferidos.
3. Después de conectarse, en el panel **Explorador**, expanda **Áreas de trabajo cuánticas**.
4. Expanda el área de trabajo y expanda el **proveedor quantinuum**.
5. Seleccione cualquier Quantinuum disponible target, por ejemplo **quantinuum.sim.h2-1e**.

6. Seleccione el icono de reproducción situado a la derecha del target nombre para empezar a enviar el programa actual Q# .
7. Escriba un nombre para identificar el trabajo y el número de tomas.
8. Presione **Entrar** para enviar el trabajo. El estado del trabajo se muestra en la parte inferior de la pantalla.
9. Expanda **Trabajos** y mantenga el puntero sobre el trabajo, que muestra las horas y el estado del trabajo.

Características admitidas

En la tabla siguiente se enumeran las características admitidas para la computación cuántica híbrida con Quantinuum en Azure Quantum.

 Expandir tabla

Característica compatible	Notas
Valores de Dynamics	Bools e enteros cuyo valor depende de un resultado de medición
Bucles	Bucles delimitados clásicamente solo
Flujo de control arbitrario	Uso de bifurcación if/else
Medición del circuito medio	Utiliza recursos de registro clásicos
Reutilización de cúbits	Compatible
Proceso clásico en tiempo real	Aritmética de enteros con signo de 64 bits Utiliza recursos de registro clásicos

El QDK proporciona targetcomentarios específicos cuando Q# no se admiten características de lenguaje para el objeto seleccionado target. Si el Q# programa contiene características no admitidas al ejecutar trabajos cuánticos híbridos, recibirá un mensaje de error durante el tiempo de desarrollo. Para obtener más información, consulte la [QIR página wiki](#) .

Nota

Debe seleccionar el perfil adecuado **Adaptive R**target para obtener la retroalimentación apropiada cuando use Q# funciones que target no admite.

Para ver las características admitidas en acción, copie el código siguiente en un Q# archivo y agregue los fragmentos de código siguientes.

Q#

```
import Microsoft.Quantum.Measurement.*;
import Microsoft.Quantum.Math.*;
import Microsoft.Quantum.Convert.*;

operation Main() : Result {
    use (q0, q1) = (Qubit(), Qubit());
    H(q0);
    let r0 = MResetZ(q0);

    // Copy here the code snippets below to see the supported features
    // in action.
    // Supported features include dynamic values, classically-bounded loops,
    // arbitrary control flow, and mid-circuit measurement.

    r0
}
```

Quantinuum admite booles y enteros dinámicos, lo que significa booles e enteros que dependen de los resultados de la medición. Tenga en cuenta que `r0` es un `Result` tipo que se puede usar para generar valores bool e enteros dinámicos.

Q#

```
let dynamicBool = r0 != Zero;
let dynamicBool = ResultAsBool(r0);
let dynamicInt = dynamicBool ? 0 | 1;
```

Quantinuum admite booles dinámicos e enteros, pero no admite valores dinámicos para otros tipos de datos, como double. Copie el código siguiente para ver comentarios sobre las limitaciones de los valores dinámicos.

Q#

```
let dynamicDouble = r0 == One ? 1. | 0.; // cannot use a dynamic double
value
let dynamicInt = r0 == One ? 1 | 0;
let dynamicDouble = IntAsDouble(dynamicInt); // cannot use a dynamic double
value
let dynamicRoot = Sqrt(dynamicDouble); // cannot use a dynamic double value
```

Aunque no se admiten valores dinámicos para algunos tipos de datos, esos tipos de datos todavía se pueden usar con valores estáticos.

Q#

```
let staticRoot = Sqrt(4.0);
let staticBigInt = IntAsBigInt(2);
```

Incluso los valores dinámicos de los tipos admitidos no se pueden usar en determinadas situaciones. Por ejemplo, Quantinuum no admite matrices dinámicas, es decir, matrices cuyo tamaño depende de un resultado de medición. Quantinuum tampoco admite bucles enlazados dinámicamente. Copie el código siguiente para ver las limitaciones de los valores dinámicos.

Q#

```
let dynamicInt = r0 == Zero ? 2 | 4;
let dynamicallySizedArray = [0, size = dynamicInt]; // cannot use a
dynamically-sized array
let staticallySizedArray = [0, size = 10];
// Loops with a dynamic condition are not supported by Quantinuum.
for _ in 0..dynamicInt {
    Rx(PI(), q1);
}

// Loops with a static condition are supported.
let staticInt = 3;
for _ in 0..staticInt {
    Rx(PI(), q1);
}
```

Quantinuum admite el flujo de control, incluida `if/else` la bifurcación, mediante condiciones estáticas y dinámicas. La bifurcación en condiciones dinámicas también se conoce como bifurcación basada en los resultados de la medición.

Q#

```
let dynamicInt = r0 == Zero ? 0 | 1;
if dynamicInt > 0 {
    X(q1);
}
let staticInt = 1;
if staticInt > 5 {
    Y(q1);
} else {
    Z(q1);
}
```

Quantinuum admite bucles con condiciones clásicas y expresiones incluidas `if`.

Q#

```
for idx in 0..3 {
    if idx % 2 == 0 {
        Rx(ArcSin(1.), q0);
        Rz(IntAsDouble(idx) * PI(), q1)
    } else {
        Ry(ArcCos(-1.), q1);
        Rz(IntAsDouble(idx) * PI(), q1)
    }
}
```

```
}  
}
```

Quantinuum admite la medición del circuito medio, es decir, la bifurcación basada en los resultados de la medición.

Q#

```
if r0 == One {  
    X(q1);  
}  
let r1 = MResetZ(q1);  
if r0 != r1 {  
    let angle = PI() + PI() + PI()* Sin(PI()/2.0);  
    Rxx(angle, q0, q1);  
} else {  
    Rxx(PI() + PI() + 2.0 * PI() * Sin(PI()/2.0), q1, q0);  
}
```

Estimación del costo de un trabajo de computación cuántica híbrida

Puede calcular el costo de ejecutar un trabajo de computación cuántica híbrida en hardware Quantinuum ejecutándolo primero en un emulador.

Después de una ejecución correcta en el emulador:

1. En el área de trabajo de Azure Quantum, seleccione **Administración de trabajos**.
2. Seleccione el trabajo que envió.
3. En el **menú emergente Detalles del trabajo**, seleccione **Estimación** de costos para ver cuántos créditos del emulador de Quantinuum se usaron. Este número se traduce directamente en el número de HQC (créditos cuánticos de Quantinuum) necesarios para ejecutar el trabajo en hardware Quantinuum.

Job details
Quantum Workspace

Cancel job Refresh

Essentials

Name
IterativePhaseEstimation

Id
338214f9-a487-430c-a93f-37add00c599b

Provider
quantinuum

Target
quantinuum.sim.h1-1e

Cost estimate
--

Input/Output **Cost Estimation**

The table below shows the usage generated by this job, and an estimate of the job cost that will appear on your Azure bill. [Learn more](#)

Dimension	Unit Price	Consumed Units	Billed Units	Cost Estimate
EHQC	\$0 / hqc	11.31	11.31	\$0
				Total: \$0

ⓘ Nota

Quantinuum anula la inscripción del circuito completo y calcula el costo en todas las rutas de acceso de código, tanto si se ejecutan condicionalmente como si no.

Ejemplos de computación cuántica híbrida

Los ejemplos siguientes se pueden encontrar en el [Q# repositorio](#) de ejemplos de código. Muestran el conjunto de características actual para la computación cuántica híbrida.

Código de repetición de tres cúbits

En este ejemplo se muestra cómo crear un [código](#) de repetición de tres cúbits que se puede usar para detectar y corregir errores de volteo de bits.

Aprovecha las características de computación híbrida integradas para contar el número de veces que se realizó la corrección de errores mientras el estado de un registro de cúbit lógico es coherente.

Puede encontrar el ejemplo de código [aquí](#).

Estimación de fase iterativa

En este programa de ejemplo se muestra una estimación de fase iterativa dentro de Q#. Usa la estimación de fase iterativa para calcular un producto interno entre dos vectores 2 dimensionales codificados en un target cúbit y un cúbit ancilla. También se inicializa un cúbit de control adicional, que es el único cúbit utilizado para la medición.

El circuito comienza codificando el par de vectores en el target cúbit y el cúbit ancilla. A continuación, aplica un operador de Oracle a todo el registro, controlado fuera del cúbit de control, que se configura en el estado $|+\rangle$. El operador de Oracle controlado genera una fase en el estado $|1\rangle$ del cúbit de control. A continuación, se puede leer aplicando una puerta H al cúbit de control para que la fase sea observable al medir.

Puede encontrar el ejemplo de código [aquí](#).

ⓘ Nota

Este código de ejemplo fue escrito por miembros del equipo cuántico de [KPMG](#) en Australia y se encuentra bajo una licencia MIT. Muestra las funcionalidades expandidas de Adaptive Rtargets y hace uso de bucles delimitados, llamadas a funciones clásicas en tiempo de ejecución, instrucciones condicionales anidadas, medidas de circuito medio y reutilización de cúbits.

Contenido relacionado

- [Introducción a las sesiones](#)

Last updated on 27/10/2025

Conceptos de computación cuántica híbrida

Artículo • 16/09/2024

Toda la computación cuántica es híbrida en algún sentido. Los primeros sistemas cuánticos usaron procesos clásicos para enviar instrucciones al equipo cuántico y recibir y procesar los datos resultantes. Las arquitecturas híbridas más avanzadas aportan una integración más estrecha y enriquecida entre el cálculo clásico y cuántico, acelerando el tiempo de ejecución y abriendo la puerta a una nueva generación de algoritmos. Por ejemplo, la computación clásica suele ser más rápida y eficaz que la computación cuántica para determinadas tareas, como el procesamiento y el análisis de datos. Sin embargo, la computación cuántica funciona mejor para determinados tipos de problemas de optimización y simulación.

Registros clásicos y cuánticos

En la computación cuántica, aunque los procesadores clásicos y cuánticos están estrechamente integrados, siguen siendo entidades físicas independientes y los programas de computación cuántica híbrida pueden aprovechar cada una de sus capacidades.

Un registro clásico usa la arquitectura familiar basada en chip de silicio y es más adecuado para operaciones como enviar instrucciones al procesador cuántico, capturar resultados de medición y usar esos resultados para determinar el siguiente conjunto de instrucciones.

Un registro cuántico es un sistema formado por varios cúbits. Los equipos cuánticos se destacan al realizar cálculos complejos manipulando sus cúbits dentro de un registro cuántico.

Medición del circuito medio

La medición del circuito medio es el proceso de realizar mediciones de estado cuántico en varios puntos durante la ejecución del programa, en lugar de solo al final. Estas medidas se usan para obtener información sobre los estados intermedios del sistema y se usan en el código clásico para tomar decisiones en tiempo real sobre el flujo del programa. Las medidas del circuito medio también actúan como correcciones de errores "comprobaciones de integridad" comprobando el estado del circuito en un determinado

punto antes de continuar y están estrechamente relacionados con la reutilización de cúbits.

Reutilización de cúbits

Los equipos cuánticos actuales pueden admitir un número creciente de cúbits. Sin embargo, todavía estamos lejos de poder admitir los millones de cúbits necesarios para ejecutar cálculos totalmente tolerantes a errores. Para obtener más información sobre la escala de los equipos cuánticos necesarios, consulte La introducción a la [estimación](#) de recursos. Además, es conveniente usar tan pocos cúbits como sea posible.

La reutilización de cúbits es la práctica de diseñar circuitos para usar el mismo cúbit varias veces en un cálculo cuántico para minimizar el total de cúbits necesarios para ejecutar el programa. Por ejemplo, después de realizar una [medición](#) del circuito medio y procesar el resultado, ese cúbit se puede restablecer y reutilizar para otro cálculo en lugar de asignar un nuevo cúbit. Hay varias técnicas para volver a usar cúbits en la computación cuántica, como la teletransportación cuántica, la corrección de errores cuánticos y la computación cuántica basada en medidas.

Mitigación de errores

Para que el hardware cuántico actual sea más sólido frente a errores y ruido, *se pueden usar cúbits lógicos*. Los cúbits lógicos se crean con varios cúbits físicos para codificar y proteger la información cuántica. Sin embargo, dado que se deben usar varios cúbits para crear un cúbit lógico, se reduce el número total de cúbits que se pueden usar para los cálculos. A medida que aumenta la capacidad de hardware para admitir más cúbits físicos, por lo que las funcionalidades de tolerancia a errores.

Además de usar cúbits lógicos, los errores en los cálculos cuánticos se pueden mitigar mediante técnicas como varias medidas, el diseño de algoritmos que reducen el número de operaciones necesarias o el ajuste de parámetros en puertas cuánticas para reducir el impacto del ruido.

Corrección de errores y tolerancia a errores

La corrección de errores y la tolerancia a errores son aspectos críticos de la computación cuántica, ya que los cúbits son más propensos a errores que los bits clásicos debido a la naturaleza delicada de los estados cuánticos y se necesitan sistemas altamente tolerantes a errores para lograr las ventajas completas de [la computación](#) cuántica híbrida distribuida. En la computación clásica, los errores se pueden corregir agregando

redundancia al cálculo y usando códigos de corrección de errores. Sin embargo, las técnicas tradicionales de corrección de errores no son directamente aplicables a la computación cuántica, ya que dependen de poder repetir el cálculo varias veces, lo que no es posible en la [computación cuántica debido al teorema](#) de no clonación.

Los *códigos de floquet* son una nueva clase de códigos de corrección de errores que responden dinámicamente al ruido y a los errores, en lugar de códigos de corrección tradicionales que protegen contra errores estáticos. Para obtener más información, consulte [Corrección de errores con códigos](#) [↗](#) floquet.

Algoritmos híbridos

- **Variational Quantum Eigensolver (VQE)** - un algoritmo cuántico para la química cuántica, simulaciones cuánticas y problemas de optimización y se usa para encontrar el estado de tierra de un sistema físico determinado. El equipo clásico se usa para definir circuitos cuánticos con determinados parámetros. Una vez medido el estado cuántico, el equipo clásico evalúa cómo mejorar los parámetros y, a continuación, vuelve a enviar los circuitos. Los VQE suelen ser programas de larga duración que pueden beneficiarse de la integración más estrecha de la computación cuántica híbrida.
- **Quantum Approximate Optimization Algorithm (QAOA)** - también un algoritmo cuántico variante, se usa para buscar soluciones aproximadas a problemas de optimización combinatoria: problemas en los que el número de soluciones posibles crece extremadamente grande con el tamaño del problema. Es un campo activo de investigación para identificar su aplicabilidad para aplicaciones como el control del tráfico aéreo, las rutas de envío o entrega, o las optimizaciones financieras.
- **Estimación de fase iterativa:** es otro método para realizar *la estimación* de fase cuántica, que calcula la fase de un operador unitario y se usa en muchos otros algoritmos cuánticos. Ambos métodos usan una serie de puertas de rotación para determinar la fase, pero la estimación de fase iterativa utiliza el registro clásico para almacenar información y cálculos sobre las medidas de puerta. Esto reduce el número de cúbits necesarios y minimiza el ruido y los errores.

Para obtener más información sobre los algoritmos cuánticos, visite el [zoológico](#) [↗](#) de algoritmos cuánticos.

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

¿Qué es el estimador de recursos de Azure Quantum?

Artículo • 02/07/2024

El estimador de recursos de Azure Quantum es una [herramienta de código abierto](#) que permite calcular los recursos necesarios para ejecutar un programa cuántico en un equipo cuántico tolerante a errores.

El estimador de recursos calcula el número total de cúbits físicos y lógicos, el tiempo de ejecución y los detalles de las fórmulas y los valores usados para cada estimación. Con el estimador de recursos de Azure Quantum, puede comparar tecnologías de cúbits, esquemas de corrección de errores cuánticos y otras características de hardware para comprender cómo afectan a los recursos necesarios para ejecutar un programa cuántico.

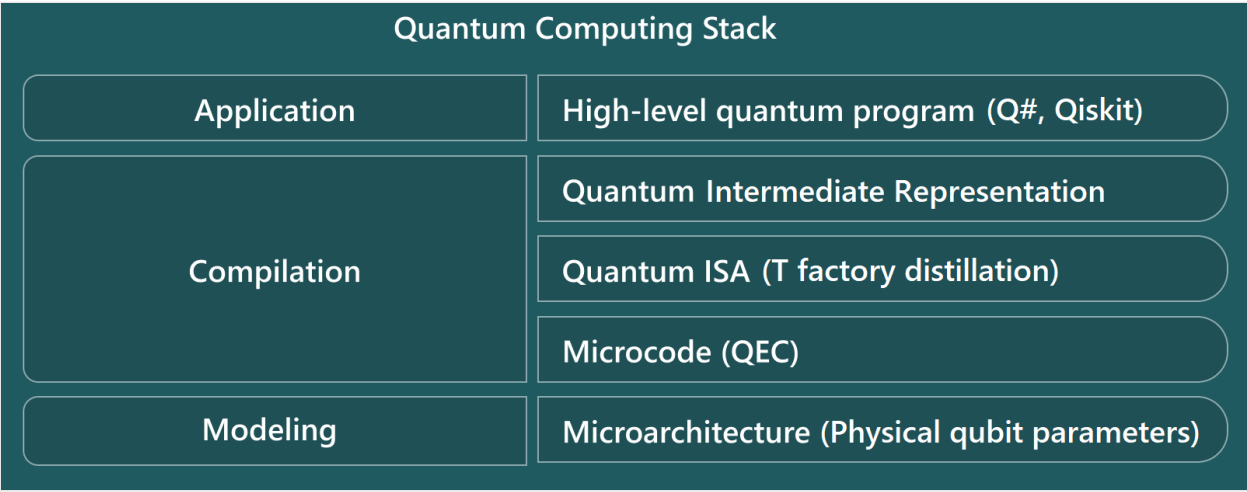
Sugerencia

El estimador de recursos de Azure Quantum es **gratuito y no requiere una cuenta de Azure**.

¿Qué características hacen que el estimador de recursos sea único?

El estimador de recursos es una herramienta eficaz que implica todos los niveles de pila de computación cuántica. La pila de computación cuántica se puede dividir en tres niveles: el nivel de aplicación, la programación cuántica o el nivel de compilación, y el nivel de hardware o modelado.

El estimador de recursos permite personalizar los parámetros de cada nivel y analizar cómo afectan a los recursos generales necesarios para ejecutar un programa cuántico.



Personalización

El estimador de recursos tiene una API de extensibilidad para modelar cualquier arquitectura cuántica y modificar todas las suposiciones. Puede adaptar el estimador de recursos y especificar las características del sistema cuántico.

Puede usar parámetros cuánticos predefinidos y esquemas de corrección de errores cuánticos (QEC) o configurar opciones únicas en una amplia gama de características de la máquina. Para obtener más información, consulte [Personalización de los target parámetros del estimador](#) de recursos.

 Expandir tabla

Parámetros de destino	Describir el sistema
Modelo de cúbit físico	Por ejemplo, especifique el conjunto de instrucciones, el tiempo de medición de cúbits, las tasas de error o los tiempos de puerta.
Esquema de corrección de errores cuánticos	Por ejemplo, especifique el número de cúbits físicos por cúbit lógico, el tiempo del ciclo lógico o el umbral de corrección de errores.
Presupuesto de errores	Por ejemplo, especifique el presupuesto de errores para implementar cúbits lógicos, destilación de estados T y síntesis de las puertas de rotación.
Unidades de destilación	Por ejemplo, especifique el número de estados T necesarios para el proceso de destilación, el número de estados T producidos como salida del proceso de destilación o la probabilidad de error del proceso de destilación.
Restricciones	Por ejemplo, especifique el número máximo de cúbits físicos, el tiempo de ejecución máximo o el número máximo de copias de fábrica de T .

ⓘ Nota

Con el estimador de recursos, puede modelar cualquier arquitectura cuántica. Por ejemplo, el inicio [Alice & Bob](#) [↗] usa el Estimador de recursos para evaluar su arquitectura, que usa cúbits cat y código de corrección de errores de repetición. Para obtener más información, consulte esta [entrada en el blog de Q#](#) [↗]

Flexibilidad

Puede traer su propio código y herramientas de compilación al Estimador de recursos. El estimador de recursos admite Q# y Qiskit como entrada. Consulte [Diferentes formas de ejecutar el estimador](#) de recursos.

Procesamiento por lotes de varias estimaciones

El estimador de recursos permite calcular los recursos necesarios para ejecutar el mismo algoritmo cuántico para [diferentes configuraciones de target parámetros](#) y comparar los resultados. De este modo, puede comprender cómo afecta la arquitectura de cúbits, el esquema QEC y el resto de los target parámetros a los recursos generales.

Optimization

Puede reducir el tiempo de ejecución del estimador de recursos mediante la incorporación de algunas estimaciones en el costo total. Por ejemplo, si está trabajando con un programa grande, puede calcular y [almacenar en caché el costo de las subrutinas](#), o si ya [conoce las estimaciones de una operación](#), puede pasarlas al Estimador de recursos.

Visualización de recursos

Puede visualizar los inconvenientes entre el número de cúbits físicos y el tiempo de ejecución del algoritmo mediante el [diagrama](#) de tiempo espaciado, lo que le permite encontrar la combinación óptima de {número de cúbits, pares runtime}.

También puede inspeccionar la distribución de cúbits físicos usados para el algoritmo y las [factorías](#) de T mediante el diagrama [de](#) espacio.

Introducción al estimador de recursos

El estimador de recursos forma parte del Kit de desarrollo de Quantum (QDK) y no requiere una cuenta de Azure. Para empezar, consulte [Ejecución de la primera estimación](#) de recursos.

En la tabla siguiente se muestran diferentes escenarios de usuario y los artículos recomendados para empezar con el estimador de recursos.

 Expandir tabla

Escenario de usuario	Si quiere...
Estoy desarrollando códigos QEC	Puede usar el estimador de recursos para personalizar los códigos QEC y comparar diferentes combinaciones de parámetros. Consulte Personalización de los esquemas de QEC.
Estoy desarrollando algoritmos cuánticos	Al analizar el impacto de diferentes configuraciones de perfiles de hardware y software en los requisitos de recursos, puede obtener información sobre cómo el algoritmo cuántico puede funcionar en diferentes condiciones de hardware y error. Esta información puede ayudarle a optimizar el algoritmo para velocidades de error o hardware cuántico específicos. Consulte Ejecución de varias configuraciones de target parámetros .
Quiero mejorar el rendimiento de los programas cuánticos	Para obtener información sobre cómo aprovechar la eficacia del estimador de recursos, consulte Ejecución de programas grandes y Uso de estimaciones conocidas.
Me interesa la computación cuántica a gran escala	Puede usar el Estimador de recursos para analizar los recursos de los problemas reales que se espera que resuelvan los equipos cuánticos tolerantes a errores a gran escala. Consulte cómo en Estimación de recursos para la computación cuántica a gran escala.
Estoy desarrollando criptografía segura para quantum	Puede usar el Estimador de recursos para comparar el rendimiento de diferentes algoritmos de cifrado, puntos fuertes de clave, tipos de cúbits y tasas de error, y su resistencia a los ataques cuánticos. Consulte Estimación de recursos y criptografía .

Estimación de recursos para la computación cuántica a gran escala

Si desea desarrollar algoritmos cuánticos para equipos cuánticos a gran escala, consulte el [tutorial Estimación de los recursos de un problema](#) de química cuántica.

Este tutorial representa un primer paso para integrar la estimación de recursos de soluciones cuánticas a problemas de estructura electrónica. Una de las aplicaciones más importantes de los equipos cuánticos escalados es resolver problemas de química cuántica. La simulación de sistemas mecánicos cuánticos complejos tiene el potencial de desbloquear avances en áreas como la captura de carbono, la inseguridad alimentaria y el diseño de mejores combustibles y materiales.

Por ejemplo, uno de los hamiltonianos usados en este tutorial, el *nitrogenase_54orbital*, describe la enzima nitrogenasa. Si podría simular con precisión cómo funciona esta enzima en un nivel cuántico, podría ayudarnos a comprender cómo producirla a escala. Podrías reemplazar el proceso muy intensivo de energía que se utiliza para producir suficiente fertilizante para alimentar el planeta. Esto tiene el potencial de reducir la huella de carbono mundial y también para ayudar a abordar las preocupaciones relativas a la inseguridad alimentaria en una población creciente.

¿Por qué es importante la estimación de recursos en el desarrollo de la computación cuántica?

Aunque los equipos cuánticos prometen resolver problemas científicos y comerciales importantes, lograr la viabilidad comercial requerirá equipos cuánticos a gran escala y tolerantes a errores que tengan un gran número de cúbits en superposición y tasas de errores físicos por debajo de un umbral determinado. La viabilidad comercial y científica requerirá también esquemas de QEC para lograr la tolerancia a errores. QEC requiere tanto tiempo como esfuerzo, lo que hace que sea necesario un mayor tiempo de ejecución para las operaciones de nivel lógico o de algoritmo, así como cúbits físicos adicionales para almacenar y calcular información.

Con el Estimador de recursos, puede comprender el impacto de las opciones de diseño arquitectónico y los esquemas de corrección de errores cuánticos. El estimador de recursos le ayudará a comprender cuántos cúbits se necesitan para ejecutar una aplicación, cuánto tiempo tardará en ejecutarse y qué tecnologías de cúbits son más adecuadas para resolver un problema específico. Comprender estos requisitos le permitirá preparar y refinar las soluciones cuánticas para que se ejecuten en máquinas cuánticas futuras y escaladas.

Comentarios

¿Le ha resultado útil esta página?



[Proporcionar comentarios sobre el producto](#) 

| [Obtener ayuda en Microsoft Q&A](#)

Inicio rápido: Ejecución de la primera estimación de recursos

Artículo • 13/01/2025

En este inicio rápido, aprenderá a usar el estimador de recursos de Azure Quantum para calcular los recursos de un programa de Q#.

Requisitos previos

- La última versión de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- La versión más reciente de la extensión Quantum Development Kit [QDK](#). Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

Sugerencia

No es necesario tener una cuenta de Azure para ejecutar el estimador de recursos.

Carga de un programa de ejemplo de Q#

1. En VS Code, seleccione **Archivo** > **nuevo archivo** y guarde el archivo como **RandomNum.qs**.
2. Abra **RandomNum.qs** y escriba `sample`, a continuación, seleccione **Ejemplo de bits aleatorios** y guarde el archivo.

Ejecución del estimador de recursos

El estimador de recursos ofrece [seis parámetros de cúbit predefinidos](#), cuatro de los cuales tienen conjuntos de instrucciones basados en puertas y dos que tienen un conjunto de instrucciones Majorana. También ofrece dos `floquet_code`

En este ejemplo, ejecutará el estimador de recursos mediante el parámetro de cúbit `qubit_gate_us_e3` y el código de corrección de errores cuánticos `surface_code`. Para obtener más información, consulte [Target parámetros](#) para el estimador de recursos.

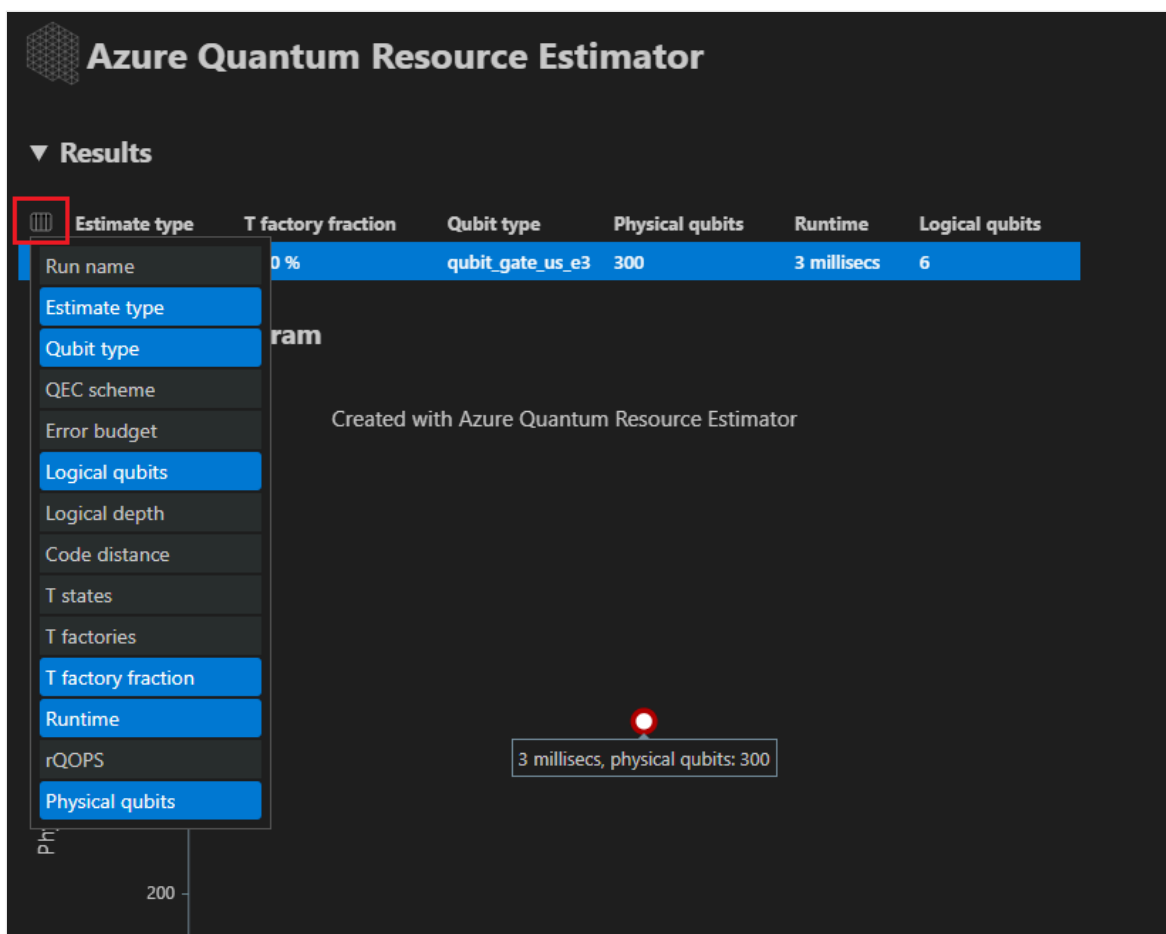
1. Seleccione **Ver** -> **Paleta de comandos**, o presione **Ctrl+Mayús+P** y escriba "recurso" que debe abrir la opción **Q#: Calculate Resource Estimates (Calcular**

- estimaciones** de recursos). Seleccione esta opción para abrir la ventana del estimador de recursos.
2. Puede seleccionar uno o varios tipos de **parámetros de cúbit y códigos de corrección de errores** para calcular los recursos. En este ejemplo, seleccione **qubit_gate_us_e3** y haga clic en **Aceptar**.
 3. Especifique el **presupuesto de errores** o acepte el valor predeterminado 0,001. En este ejemplo, deje el valor predeterminado y presione **Entrar**.
 4. Presione **Entrar** para aceptar el nombre de resultado predeterminado en función del nombre de archivo, en este caso, **RandomNum**.

Ver resultados

El estimador de recursos proporciona varias estimaciones para el mismo algoritmo, cada una de las cuales muestra inconvenientes entre el número de cúbits y el tiempo de ejecución. Comprender el equilibrio entre el tiempo de ejecución y la escala del sistema es uno de los aspectos más importantes de la estimación de recursos.

1. El resultado de la estimación de recursos se muestra en la ventana **Estimación de Q#**.
2. En la pestaña **Resultados**, se muestra un resumen de la estimación de recursos. **Haga clic en el icono** situado junto a la primera fila para seleccionar las columnas que desee mostrar. Puede seleccionar entre el nombre de ejecución, el tipo de estimación, el tipo de cúbit, el esquema qec, el presupuesto de errores, los cúbits lógicos, la profundidad lógica, la distancia del código, los estados de T, los generadores de T, la fracción de factoría de T, el tiempo de ejecución, rQOPS y los cúbits físicos.

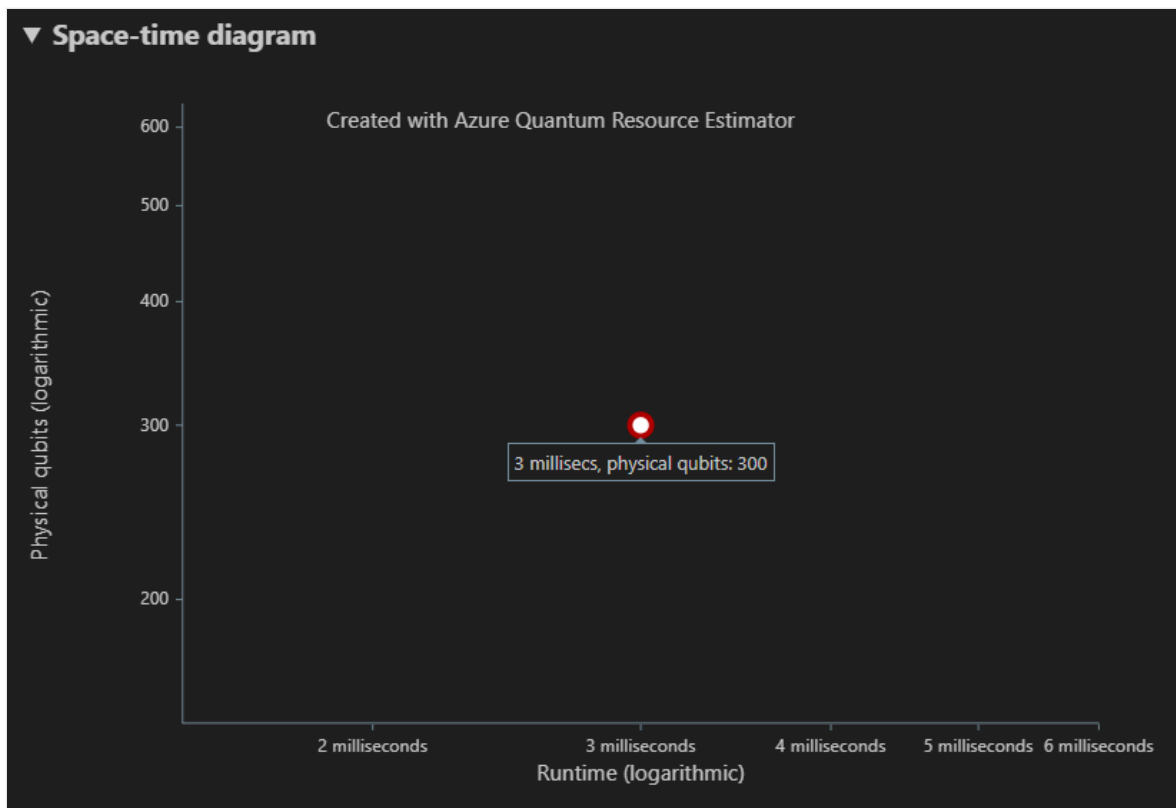


En la **columna Tipo estimado** de la tabla de resultados, puede ver el número de combinaciones óptimas de $\{\text{número de cúbits}, \text{runtime}\}$ para el algoritmo. Estas combinaciones se pueden ver en el diagrama de tiempo espacial.

⚠ Nota

Si selecciona más de un parámetro de cúbit y código de corrección de errores en la configuración, los resultados se muestran en filas diferentes en la pestaña **Resultados**. Al hacer clic en un resultado de la tabla, se abre el diagrama de espacio correspondiente y los datos del informe.

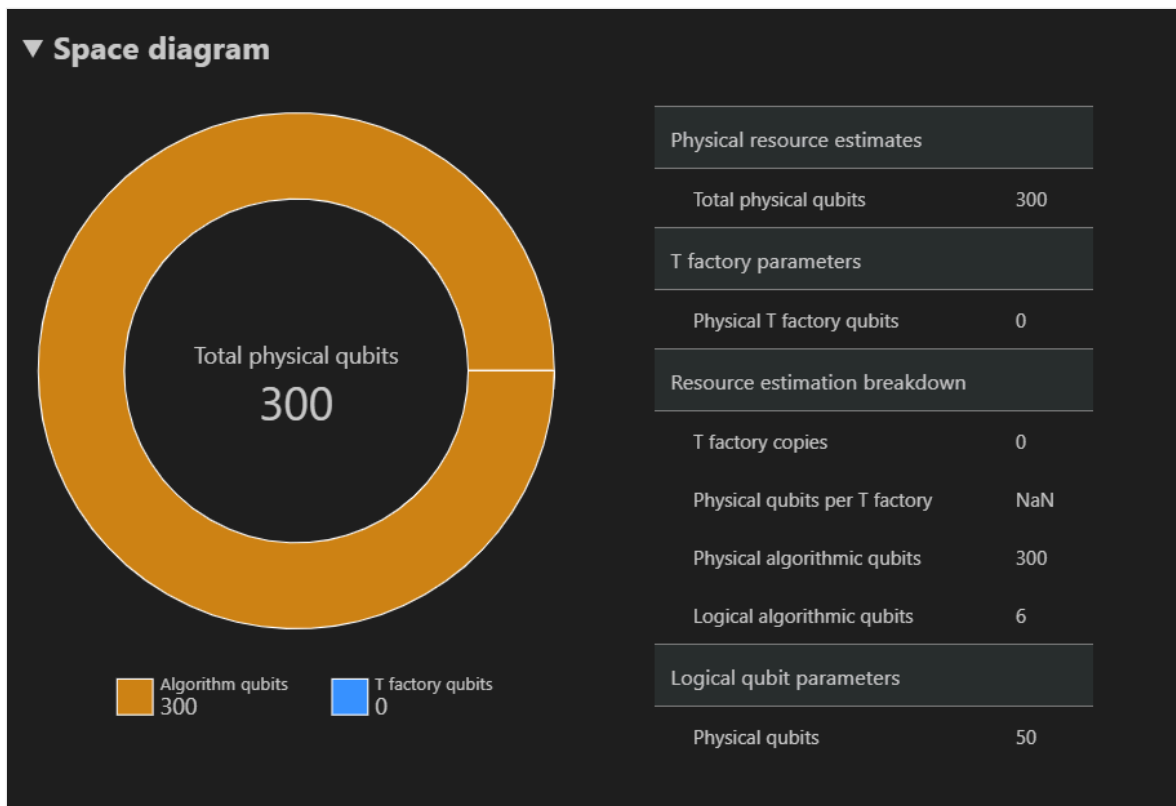
3. En el **diagrama** space-time se muestran los inconvenientes entre el número de cúbits físicos y el tiempo de ejecución del algoritmo. En este caso, el estimador de recursos encuentra una combinación óptima de muchos miles posibles. Puede mantener el puntero sobre cada $\{\text{número de cúbits}, \text{tiempo de ejecución}\}$ para ver los detalles de la estimación de recursos en ese momento. Para obtener más información, vea [Diagrama de tiempo espaciador](#).



ⓘ Nota

Debe hacer clic en un punto del diagrama de tiempo espaciado, es decir, un {número de cúbits, un par runtime}, para ver el diagrama de espacio y los detalles de la estimación de recursos correspondiente a ese punto.

4. En la pestaña **Diagrama de espacio**, se muestra la distribución de cúbits físicos usados para el algoritmo y los [generadores de T](#). En este ejemplo, los cúbits del algoritmo y los cúbits totales son los mismos porque el algoritmo no usa ninguna copia de fábrica de T. Para obtener más información, vea [Diagramas de tiempo espaciador](#).



5. Por último, en la pestaña **Estimaciones de recursos** se muestra la lista completa de datos de salida del estimador de recursos. Se pueden inspeccionar los detalles de los costos contrayendo los grupos, que contienen más información. Por ejemplo, contraiga el **grupo Parámetros cuánticos lógicos**. Para obtener más información, consulte [los datos del informe del estimador](#) de recursos para el estimador de recursos.

[Expandir tabla](#)

Parámetro de cúbit lógico	Value
Esquema de QEC	surface_code
Distancia del código	5
Cúbits físicos	50
Tiempo de ciclo lógico	3 milisegundos
Tasa de errores de cúbit lógico	3,00E-5
Factor previo de cruce	0,03
Umbral de corrección de errores	0,01
Fórmula de tiempo de ciclo lógico	$(4 * \text{twoQubitGateTime} + 2 * \text{oneQubitMeasurementTime}) * \text{codeDistance}$

Parámetro de cúbit lógico	Value
Fórmula de cúbits físicos	$2 * \text{codeDistance} * \text{codeDistance}$

Sugerencia

Haga clic en **Mostrar filas de detalles** para mostrar la descripción de cada salida de los datos del informe.

La funcionalidad completa del estimador de recursos está fuera del ámbito de este inicio rápido. Para obtener más información, consulte [Uso del estimador de recursos con distintos SDK e IDE](#).

Nota

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Pasos siguientes

- [Descripción de los resultados del estimador de recursos](#)
- [Personalización de estimaciones de recursos en las características de la máquina](#)
- [Control de programas grandes con el estimador de recursos](#)
- [Tutorial: Estimación de los recursos de un problema de química cuántica](#)

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Diferentes formas de ejecutar el estimador de recursos

Artículo • 31/12/2024

En este artículo, aprenderá a trabajar con el [estimador](#) de recursos de Azure Quantum. El estimador de recursos le ayuda a calcular los recursos necesarios para ejecutar un programa cuántico en un equipo cuántico. Puede usar el estimador de recursos para calcular el número de cúbits, el número de ejecuciones y la profundidad del circuito necesario para ejecutar un programa cuántico.

El estimador de recursos está disponible en Visual Studio Code con la extensión Quantum Development Kit. Para obtener más información, consulte [Instalación del Kit de Desarrollo Cuántico](#).

Advertencia

El estimador de recursos de Azure Portal está en desuso. Recomendamos que realice la transición al Estimador de Recursos local en Visual Studio Code proporcionado en el Quantum Development Kit [↗](#).

Requisitos previos para VS Code

- La versión más reciente de [Visual Studio Code](#) [↗](#) o abrir [VS Code en la Web](#) [↗](#).
- La versión más reciente de [Extensión de kit de desarrollo de Quantum](#) [↗](#). Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

Sugerencia

No es necesario tener una cuenta de Azure para ejecutar el estimador de recursos.

Creación de un nuevo archivo Q#

1. Abra Visual Studio Code y seleccione **Archivo > Nuevo archivo de texto** para crear un nuevo archivo.
2. Guarde el archivo como `ShorRE.qs`. Este archivo contendrá el código Q# del programa.

Creación del algoritmo cuántico

Copie el código siguiente en el archivo `ShorRE.qs`:

Q#

```
import Std.Arrays.*;
import Std.Canon.*;
import Std.Convert.*;
import Std.Diagnostics.*;
import Std.Math.*;
import Std.Measurement.*;
import Microsoft.Quantum.Unstable.Arithmetic.*;
import Std.ResourceEstimation.*;

operation Main() : Unit {
    let bitsize = 31;

    // When choosing parameters for `EstimateFrequency`, make sure that
    // generator and modulus are not co-prime
    let _ = EstimateFrequency(11, 2^bitsize - 1, bitsize);
}

// In this sample we concentrate on costing the `EstimateFrequency`
// operation, which is the core quantum operation in Shors algorithm,
and
// we omit the classical pre- and post-processing.

/// # Summary
/// Estimates the frequency of a generator
/// in the residue ring  $\mathbb{Z} \bmod \text{modulus}$ .
///
/// # Input
/// ## generator
/// The unsigned integer multiplicative order (period)
/// of which is being estimated. Must be co-prime to `modulus`.
/// ## modulus
/// The modulus which defines the residue ring  $\mathbb{Z} \bmod \text{modulus}$ 
/// in which the multiplicative order of `generator` is being estimated.
/// ## bitsize
/// Number of bits needed to represent the modulus.
///
/// # Output
/// The numerator  $k$  of dyadic fraction  $k/2^{\text{bitsPrecision}}$ 
/// approximating  $s/r$ .
operation EstimateFrequency(
    generator : Int,
    modulus : Int,
    bitsize : Int
)
: Int {
    mutable frequencyEstimate = 0;
```

```

let bitsPrecision = 2 * bitsize + 1;

// Allocate qubits for the superposition of eigenstates of
// the oracle that is used in period finding.
use eigenstateRegister = Qubit[bitsize];

// Initialize eigenstateRegister to 1, which is a superposition of
// the eigenstates we are estimating the phases of.
// We first interpret the register as encoding an unsigned integer
// in little endian encoding.
ApplyXorInPlace(1, eigenstateRegister);
let oracle = ApplyOrderFindingOracle(generator, modulus, _, _);

// Use phase estimation with a semiclassical Fourier transform to
// estimate the frequency.
use c = Qubit();
for idx in bitsPrecision - 1..-1..0 {
    within {
        H(c);
    } apply {
        // `BeginEstimateCaching` and `EndEstimateCaching` are the
operations
        // exposed by Azure Quantum Resource Estimator. These will
instruct
        // resource counting such that the if-block will be executed
        // only once, its resources will be cached, and appended in
        // every other iteration.
        if BeginEstimateCaching("ControlledOracle", SingleVariant())
{
            Controlled oracle([c], (1 <<< idx, eigenstateRegister));
            EndEstimateCaching();
        }
        R1Frac(frequencyEstimate, bitsPrecision - 1 - idx, c);
    }
    if MResetZ(c) == One {
        frequencyEstimate += 1 <<< (bitsPrecision - 1 - idx);
    }
}

// Return all the qubits used for oracles eigenstate back to 0 state
// using Microsoft.Quantum.Intrinsic.ResetAll.
ResetAll(eigenstateRegister);

return frequencyEstimate;
}

/// # Summary
/// Interprets `target` as encoding unsigned little-endian integer k
/// and performs transformation  $|k\rangle \mapsto |g^p \cdot k \bmod N\rangle$  where
/// p is `power`, g is `generator` and N is `modulus`.
///
/// # Input
/// ## generator
/// The unsigned integer multiplicative order ( period )
/// of which is being estimated. Must be co-prime to `modulus`.

```

```

    /// ## modulus
    /// The modulus which defines the residue ring  $\mathbb{Z} \bmod \text{`modulus`}$ 
    /// in which the multiplicative order of  $\text{`generator`}$  is being estimated.
    /// ## power
    /// Power of  $\text{`generator`}$  by which  $\text{`target`}$  is multiplied.
    /// ## target
    /// Register interpreted as little endian encoded which is multiplied by
    /// given power of the generator. The multiplication is performed modulo
    ///  $\text{`modulus`}$ .
    internal operation ApplyOrderFindingOracle(
        generator : Int, modulus : Int, power : Int, target : Qubit[]
    )
    : Unit
    is Adj + Ctl {
        // The oracle we use for order finding implements  $|x\rangle \mapsto |x \cdot a \bmod N\rangle$ .
        // also use  $\text{`ExpModI`}$  to compute  $a$  by which  $x$  must be multiplied.
        // note that we interpret target as unsigned integer in little-
        // endian
        // encoding.
        ModularMultiplyByConstant(modulus,
                                   ExpModI(generator, power, modulus),
                                   target);
    }

    /// # Summary
    /// Performs modular in-place multiplication by a classical constant.
    ///
    /// # Description
    /// Given the classical constants  $\text{`c`}$  and  $\text{`modulus`}$ , and an input
    /// quantum register  $|y\rangle$ , this operation
    /// computes  $\text{`(c*x) \% modulus`}$  into  $|y\rangle$ .
    ///
    /// # Input
    /// ## modulus
    /// Modulus to use for modular multiplication
    /// ## c
    /// Constant by which to multiply  $|y\rangle$ 
    /// ## y
    /// Quantum register of target
    internal operation ModularMultiplyByConstant(modulus : Int, c : Int, y :
    Qubit[])
    : Unit is Adj + Ctl {
        use qs = Qubit[Length(y)];
        for (idx, yq) in Enumerated(y) {
            let shiftedC = (c <<< idx) % modulus;
            Controlled ModularAddConstant([yq], (modulus, shiftedC, qs));
        }
        ApplyToEachCA(SWAP, Zipped(y, qs));
        let invC = InverseModI(c, modulus);
        for (idx, yq) in Enumerated(y) {
            let shiftedC = (invC <<< idx) % modulus;
            Controlled ModularAddConstant([yq], (modulus, modulus -
            shiftedC, qs));
        }
    }

```

```

    }
}

/// # Summary
/// Performs modular in-place addition of a classical constant into a
/// quantum register.
///
/// # Description
/// Given the classical constants `c` and `modulus`, and an input
/// quantum register  $|y\rangle$ , this operation
/// computes  $(x+c) \% \text{modulus}$  into  $|y\rangle$ .
///
/// # Input
/// ## modulus
/// Modulus to use for modular addition
/// ## c
/// Constant to add to  $|y\rangle$ 
/// ## y
/// Quantum register of target
internal operation ModularAddConstant(modulus : Int, c : Int, y :
Qubit[])
: Unit is Adj + Ctl {
    body (...) {
        Controlled ModularAddConstant([], (modulus, c, y));
    }
    controlled (ctrls, ...) {
        // We apply a custom strategy to control this operation instead
of
        // letting the compiler create the controlled variant for us in
which
        // the `Controlled` functor would be distributed over each
operation
        // in the body.
        //
        // Here we can use some scratch memory to save ensure that at
most one
        // control qubit is used for costly operations such as
`AddConstant`
        // and `CompareGreaterThanOrEqualConstant`.
        if Length(ctrls) >= 2 {
            use control = Qubit();
            within {
                Controlled X(ctrls, control);
            } apply {
                Controlled ModularAddConstant([control], (modulus, c,
y));
            }
        } else {
            use carry = Qubit();
            Controlled AddConstant(ctrls, (c, y + [carry]));
            Controlled Adjoint AddConstant(ctrls, (modulus, y +
[carry]));
            Controlled AddConstant([carry], (modulus, y));
            Controlled CompareGreaterThanOrEqualConstant(ctrls, (c, y,
carry));

```

```

    }
  }
}

/// # Summary
/// Performs in-place addition of a constant into a quantum register.
///
/// # Description
/// Given a non-empty quantum register  $|y\rangle$  of length  $n+1$  and a positive
/// constant  $c < 2^n$ , computes  $|y + c\rangle$  into  $|y\rangle$ .
///
/// # Input
/// ## c
/// Constant number to add to  $|y\rangle$ .
/// ## y
/// Quantum register of second summand and target; must not be empty.
internal operation AddConstant(c : Int, y : Qubit[]) : Unit is Adj + Ctl
{
    // We are using this version instead of the library version that is
based
    // on Fourier angles to show an advantage of sparse simulation in
this sample.

    let n = Length(y);
    Fact(n > 0, "Bit width must be at least 1");

    Fact(c >= 0, "constant must not be negative");
    Fact(c < 2 ^ n, $"constant must be smaller than {2L ^ n}");

    if c != 0 {
        // If c has j trailing zeroes than the j least significant bits
        // of y won't be affected by the addition and can therefore be
        // ignored by applying the addition only to the other qubits and
        // shifting c accordingly.
        let j = NTrailingZeroes(c);
        use x = Qubit[n - j];
        within {
            ApplyXorInPlace(c >>> j, x);
        } apply {
            IncByLE(x, y[j...]);
        }
    }
}

/// # Summary
/// Performs greater-than-or-equals comparison to a constant.
///
/// # Description
/// Toggles output qubit `target` if and only if input register `x`
/// is greater than or equal to `c`.
///
/// # Input
/// ## c
/// Constant value for comparison.
/// ## x

```

```

    /// Quantum register to compare against.
    /// ## target
    /// Target qubit for comparison result.
    ///
    /// # Reference
    /// This construction is described in [Lemma 3, arXiv:2201.10200]
    internal operation CompareGreaterThanOrEqualConstant(c : Int, x :
Qubit[], target : Qubit)
    : Unit is Adj+Ctl {
        let bitWidth = Length(x);

        if c == 0 {
            X(target);
        } elif c >= 2 ^ bitWidth {
            // do nothing
        } elif c == 2 ^ (bitWidth - 1) {
            ApplyLowTCNOT(Tail(x), target);
        } else {
            // normalize constant
            let l = NTrailingZeroes(c);

            let cNormalized = c >>> l;
            let xNormalized = x[l...];
            let bitWidthNormalized = Length(xNormalized);
            let gates = Rest(IntAsBoolArray(cNormalized,
bitWidthNormalized));

            use qs = Qubit[bitWidthNormalized - 1];
            let cs1 = [Head(xNormalized)] + Most(qs);
            let cs2 = Rest(xNormalized);

            within {
                for i in IndexRange(gates) {
                    (gates[i] ? ApplyAnd | ApplyOr)(cs1[i], cs2[i], qs[i]);
                }
            } apply {
                ApplyLowTCNOT(Tail(qs), target);
            }
        }
    }

    /// # Summary
    /// Internal operation used in the implementation of
    GreaterThanOrEqualConstant.
    internal operation ApplyOr(control1 : Qubit, control2 : Qubit, target :
Qubit) : Unit is Adj {
        within {
            ApplyToEachA(X, [control1, control2]);
        } apply {
            ApplyAnd(control1, control2, target);
            X(target);
        }
    }

    internal operation ApplyAnd(control1 : Qubit, control2 : Qubit, target :

```

```

Qubit)
: Unit is Adj {
  body (...) {
    CCNOT(control1, control2, target);
  }
  adjoint (...) {
    H(target);
    if (M(target) == One) {
      X(target);
      CZ(control1, control2);
    }
  }
}

/// # Summary
/// Returns the number of trailing zeroes of a number
///
/// ## Example
/// ```qsharp
/// let zeroes = NTrailingZeroes(21); // = NTrailingZeroes(0b1101) = 0
/// let zeroes = NTrailingZeroes(20); // = NTrailingZeroes(0b1100) = 2
/// ```
internal function NTrailingZeroes(number : Int) : Int {
  mutable nZeroes = 0;
  mutable copy = number;
  while (copy % 2 == 0) {
    nZeroes += 1;
    copy /= 2;
  }
  return nZeroes;
}

/// # Summary
/// An implementation for `CNOT` that when controlled using a single
control uses
/// a helper qubit and uses `ApplyAnd` to reduce the T-count to 4
instead of 7.
internal operation ApplyLowTCNOT(a : Qubit, b : Qubit) : Unit is Adj+Ctl
{
  body (...) {
    CNOT(a, b);
  }

  adjoint self;

  controlled (ctls, ...) {
    // In this application this operation is used in a way that
    // it is controlled by at most one qubit.
    Fact(Length(ctls) <= 1, "At most one control line allowed");

    if IsEmpty(ctls) {
      CNOT(a, b);
    } else {
      use q = Qubit();

```

```

        within {
            ApplyAnd(Head(ctls), a, q);
        } apply {
            CNOT(q, b);
        }
    }
}

controlled adjoint self;
}

```

Ejecución del estimador de recursos

El estimador de recursos ofrece [seis parámetros de cúbit predefinidos](#), cuatro de los cuales tienen conjuntos de instrucciones basados en puertas y dos que tienen un conjunto de instrucciones Majorana. También ofrece dos

En este ejemplo, ejecutará el estimador de recursos mediante el parámetro de cúbit `qubit_gate_us_e3` y el código de corrección de errores cuánticos `surface_code`.

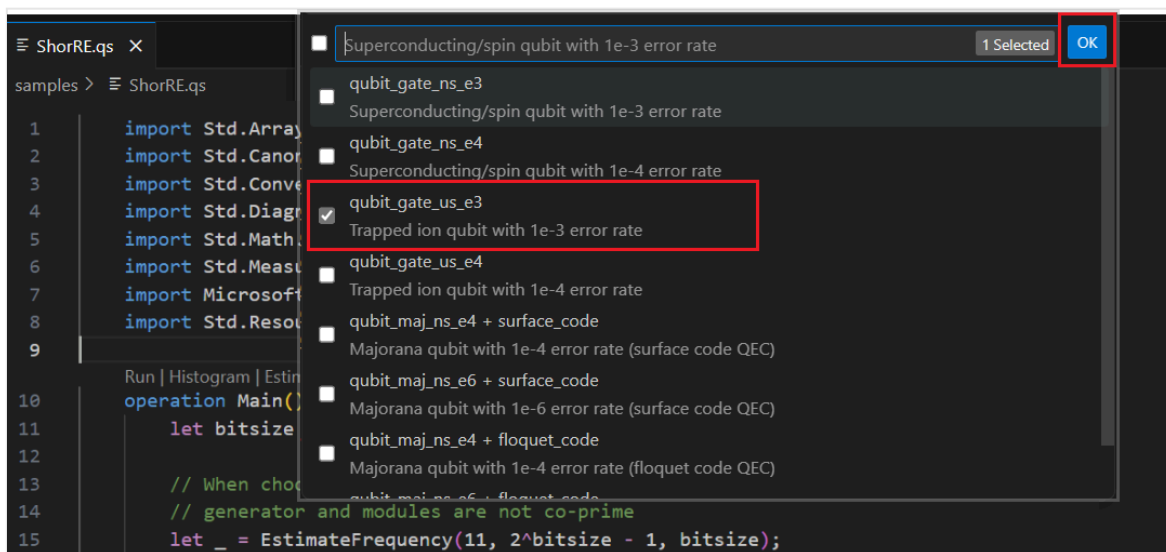
1. Seleccione **Ver** -> **Paleta de comandos** y escriba "recurso" que debe abrir la opción **Q#: Calculate Resource Estimates (Calcular estimaciones de recursos)**. También puede hacer clic en **Estimación** en la lista de comandos que se muestran justo antes de la **Main** operación. Seleccione esta opción para abrir la ventana del estimador de recursos.

```

5      import Std.Math.*;
6      import Std.Measurement.*;
7      import Microsoft.Quantum.Unstable.Arithmetic.*;
8      import Std.ResourceEstimation.*;
9
10     Run | Histogram | Estimate | Debug | Circuit
11     operation Main() : Unit {
12         let bitsize = 31;
13
14         // When choosing parameters for `EstimateFrequency`, make sure that
15         // generator and modules are not co-prime
16         let _ = EstimateFrequency(11, 2^bitsize - 1, bitsize);
17     }

```

2. Puede seleccionar uno o varios tipos de **parámetros de cúbit y códigos de corrección de errores** para calcular los recursos. En este ejemplo, seleccione `qubit_gate_us_e3` y haga clic en **Aceptar**.



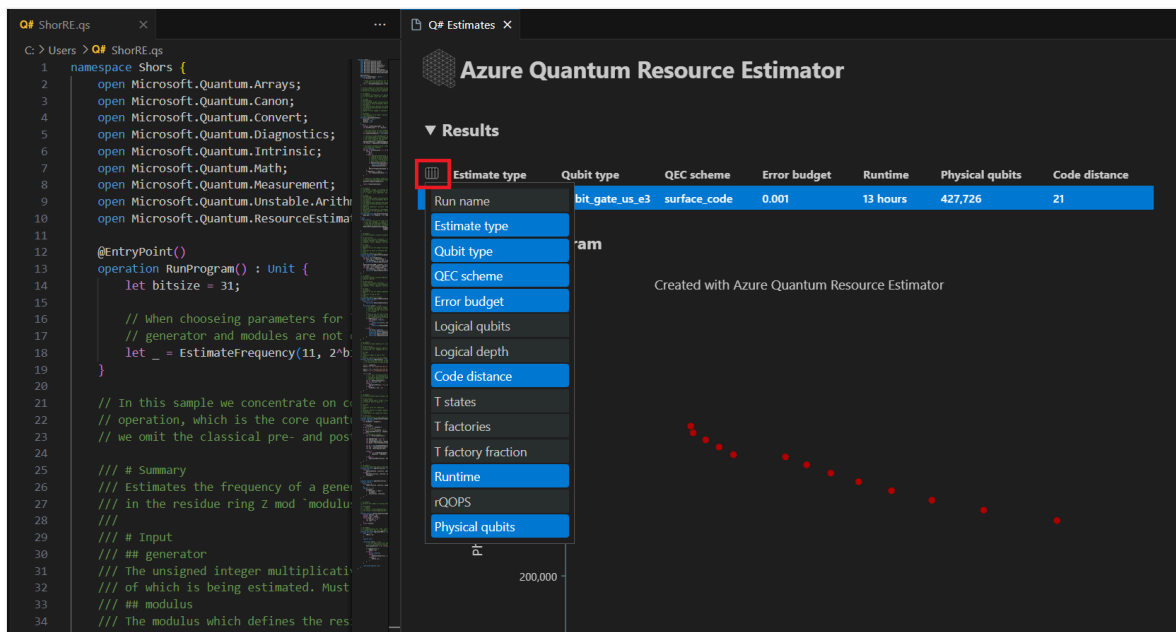
3. Especifique el **presupuesto de errores** o acepte el valor predeterminado 0,001. En este ejemplo, deje el valor predeterminado y presione **Entrar**.
4. Presione **Entrar** para aceptar el nombre de resultado predeterminado en función del nombre de archivo, en este caso, **ShorRE**.

Ver resultados

El estimador de recursos proporciona varias estimaciones para el mismo algoritmo, cada una de las cuales muestra inconvenientes entre el número de cúbits y el tiempo de ejecución. Comprender el equilibrio entre el tiempo de ejecución y la escala del sistema es uno de los aspectos más importantes de la estimación de recursos.

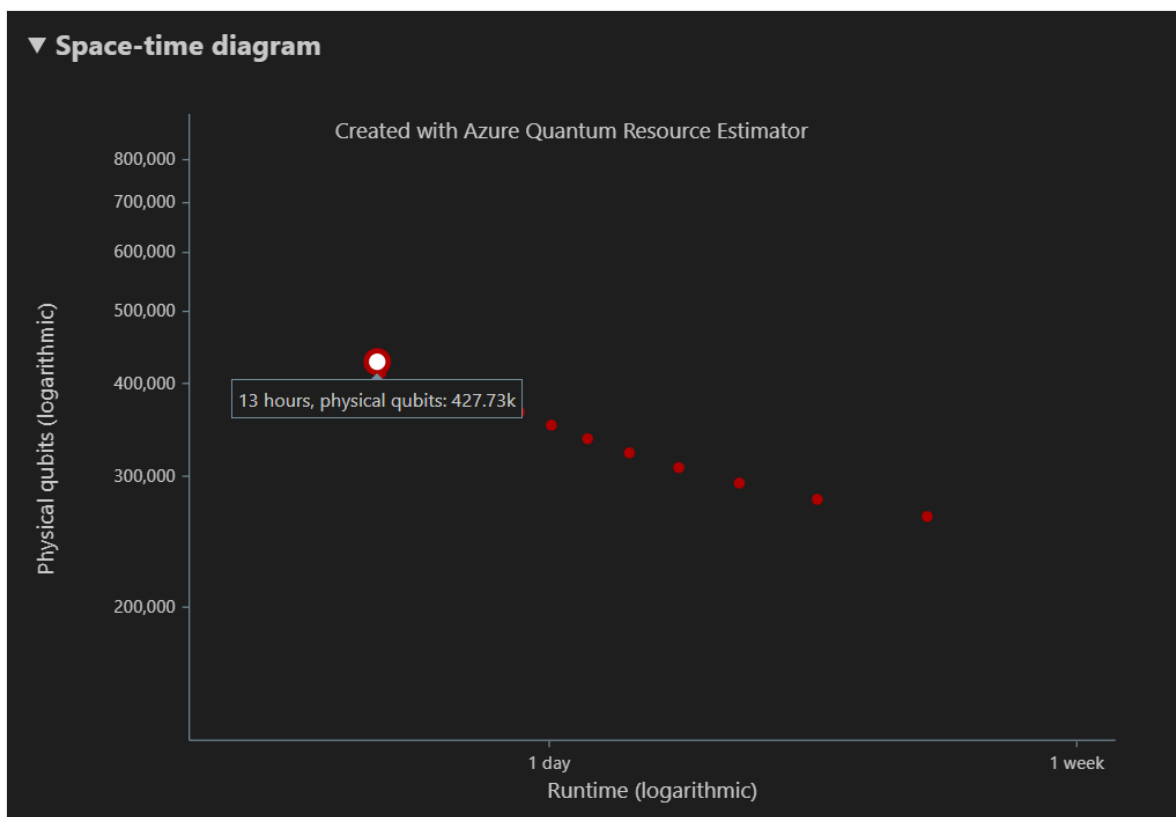
El resultado de la estimación de recursos se muestra en la ventana **Estimación de Q#**.

1. En la pestaña **Resultados**, se muestra un resumen de la estimación de recursos. Haga clic en el icono situado junto a la primera fila para seleccionar las columnas que desee mostrar. Puede seleccionar entre el nombre de ejecución, el tipo de estimación, el tipo de cúbit, el esquema qec, el presupuesto de errores, los cúbits lógicos, la profundidad lógica, la distancia del código, los estados de T, los generadores de T, la fracción de factoría de T, el tiempo de ejecución, rQOPS y los cúbits físicos.



En la **columna Tipo estimado** de la tabla de resultados, puede ver el número de combinaciones óptimas de $\{\text{número de cúbits}, \text{runtime}\}$ para el algoritmo. Estas combinaciones se pueden ver en el diagrama de tiempo espacial.

- En el **diagrama** space-time se muestran los inconvenientes entre el número de cúbits físicos y el tiempo de ejecución del algoritmo. En este caso, el estimador de recursos encuentra 13 combinaciones óptimas diferentes de muchas miles posibles. Puede mantener el puntero sobre cada $\{\text{número de cúbits}, \text{tiempo de ejecución}\}$ para ver los detalles de la estimación de recursos en ese momento.

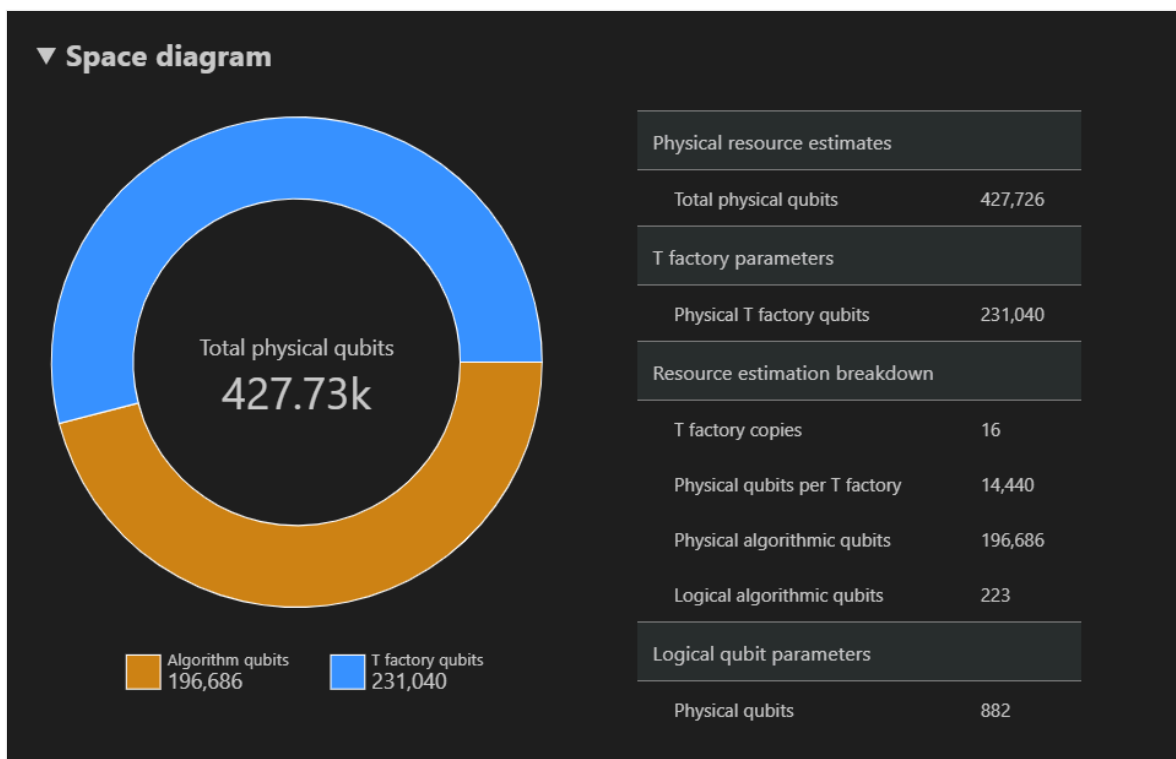


Para obtener más información, vea [Diagrama de tiempo espaciador](#).

⚠ Nota

Debe hacer clic en un punto del diagrama de tiempo espaciado, es decir, un {número de cúbits, un par runtime}, para ver el diagrama de espacio y los detalles de la estimación de recursos correspondiente a ese punto.

- El **diagrama space** muestra la distribución de cúbits físicos usados para el algoritmo y las **factorías de T, correspondientes** a un par {número de cúbits, runtime}. Por ejemplo, si selecciona el punto más a la izquierda en el diagrama de tiempo espacial, el número de cúbits físicos necesarios para ejecutar el algoritmo se 427726, 196686 de los cuales son cúbits de algoritmo y 231040 de los cuales son cúbits de fábrica de T.



- Por último, en la **pestaña Estimaciones** de recursos se muestra la lista completa de datos de salida del estimador de recursos correspondiente a un *par {número de cúbits, runtime}* . Se pueden inspeccionar los detalles de los costos contrayendo los grupos, que contienen más información. Por ejemplo, seleccione el punto más a la izquierda en el diagrama de tiempo espaciado y contraiga el **grupo Parámetros cuánticos lógicos** .

[Expandir tabla](#)

Parámetro de cúbit lógico	Value
Esquema de QEC	surface_code

Parámetro de cúbit lógico	Value
Distancia del código	21
Cúbits físicos	882
Tiempo de ciclo lógico	13 milisegundos
Tasa de errores de cúbit lógico	3,00E-13
Factor previo de cruce	0,03
Umbral de corrección de errores	0,01
Fórmula de tiempo de ciclo lógico	$(4 * \text{twoQubitGateTime} + 2 * \text{oneQubitMeasurementTime}) * \text{codeDistance}$
Fórmula de cúbits físicos	$2 * \text{codeDistance} * \text{codeDistance}$

Sugerencia

Haga clic en **Mostrar filas de detalles** para mostrar la descripción de cada salida de los datos del informe.

Para obtener más información, consulte [los datos de informe completos del estimador](#) de recursos.

Cambiar los target parámetros

Puede calcular el costo del mismo programa de Q# mediante otro tipo de cúbit, código de corrección de errores y presupuesto de errores. Abra la ventana Estimador de recursos seleccionando **Ver** -> **Paleta de comandos** y escriba `Q#: Calculate Resource Estimates`.

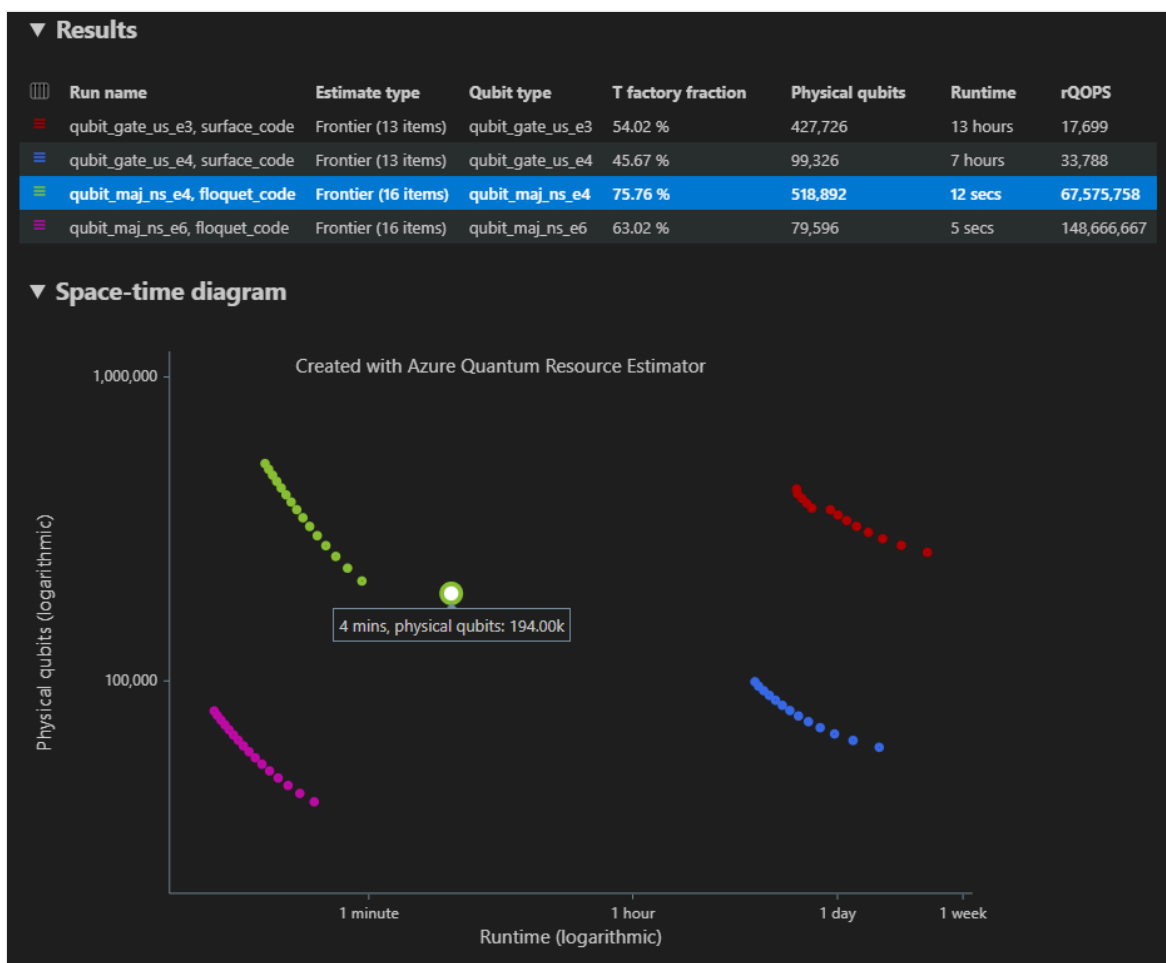
Seleccione cualquier otra configuración, por ejemplo, el parámetro qubit basado en Majorana, `qubit_maj_ns_e6`. Acepte el valor predeterminado del presupuesto de errores o escriba uno nuevo y presione **Entrar**. El estimador de recursos vuelve a ejecutar la estimación con los nuevos target parámetros.

Para obtener más información, consulte [Target parámetros](#) para el estimador de recursos.

Ejecución de varias configuraciones de parámetros

El estimador de recursos de Azure Quantum puede ejecutar varias configuraciones de parámetros y comparar los resultados de target la estimación de recursos.

1. Seleccione **Ver-> Paleta** de comandos o presione **Ctrl+Mayús+P** y escriba **Q#:**
Calculate Resource Estimates.
2. Seleccione **qubit_gate_us_e3, qubit_gate_us_e4, qubit_maj_ns_e4 + floquet_code** y **qubit_maj_ns_e6 + floquet_code** y haga clic en **Aceptar**.
3. Acepte el valor predeterminado del presupuesto de errores 0.001 y presione **Entrar**.
4. Presione **Entrar** para aceptar el archivo de entrada, en este caso, **ShorRE.qs**.
5. En el caso de varias configuraciones de parámetros, los resultados se muestran en filas diferentes en la **pestaña Resultados**.
6. En el **diagrama** space-time se muestran los resultados de todas las configuraciones de parámetros. La primera columna de la tabla de resultados muestra la leyenda de cada configuración de parámetros. Puede mantener el puntero sobre cada punto para ver los detalles de la estimación de recursos en ese momento.



7. Haga clic en un **punto {número de cúbits, runtime}** del diagrama de tiempo de espacio para mostrar el diagrama de espacio y los datos de informe correspondientes.

❗ Nota

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Pasos siguientes

- Descripción de los resultados del estimador de recursos
- Personalización de estimaciones de recursos en las características de la máquina
- Control de programas grandes con el estimador de recursos
- Tutorial: Estimación de los recursos de un problema de química cuántica

Comentarios

¿Le ha resultado útil esta página?



[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Personalización de los target parámetros del estimador de recursos

En este artículo se muestra cómo personalizar los target parámetros del estimador [de recursos de Azure Quantum](#) para que coincidan con las características de la máquina de destino. El estimador de recursos usa estos parámetros para calcular los recursos necesarios para ejecutar un algoritmo cuántico en un equipo cuántico.

⚠ Nota:

Si experimenta problemas al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#).

Requisitos previos

- Un entorno de Python con [Python y Pip](#) instalados.
- La versión más reciente de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- VS Code con las [extensiones kit de desarrollo](#) de Azure Quantum, [Python](#) y [Jupyter](#) instaladas.
- La biblioteca de Python más reciente `qdk` con las funcionalidades adicionales de `jupyter`.

Bash

```
python -m pip install --upgrade qdk[jupyter]
```

Parámetros de destino

El estimador de recursos calcula la estimación de recursos, como el número de cúbits y el tiempo de ejecución, que sería necesario para implementar un algoritmo cuántico determinado en un equipo cuántico con una tecnología de cúbit determinada y un conjunto fijo de opciones arquitectónicas.

El estimador de recursos toma un conjunto de entradas, con valores predefinidos para empezar fácilmente:

- Un [modelo](#) de cúbit físico, que define las propiedades de los cúbits físicos subyacentes.
- Un [esquema](#) de corrección de errores cuánticos (QEC), que es el esquema de corrección de errores cuánticos asumido.

- Un [presupuesto de errores](#), que es el error permitido general, definido como el número de veces que el programa puede producir un error.
- Restricciones en el nivel de componente, que son el número de ciclos [lógicos](#) y el número de copias de fábrica de T.
- [Unidades de destilación](#) para especificar algoritmos de destilación para la fábrica T.
- [Estimación de frontera de Pareto](#) para ejecutar varias estimaciones para el número de cúbits y tiempo de ejecución para el mismo algoritmo.

Sugerencia

Si ya conoce algunas estimaciones calculadas previamente para una operación, puede incorporarlas para optimizar la ejecución del estimador de recursos. Para obtener más información, consulte [Uso de estimaciones conocidas con el estimador](#) de recursos.

Parámetros de cúbit físicos

Cuando el Estimador de recursos modela las suposiciones de cúbit físico, usa dos conjuntos de instrucciones físicas diferentes para operar en los cúbits. El conjunto de instrucciones físicas puede ser basado en puertas o Majorana. Un conjunto de instrucciones basado en puertas proporciona la medición de un solo cúbit, puertas de un solo cúbit (por ejemplo, puertas T) y puertas de dos cúbits. Un conjunto de instrucciones Majorana proporciona una puerta T física, una medición de un solo cúbit y operaciones de medición conjunta de dos cúbits.

Puede elegir entre seis parámetros de cúbit predefinidos, cuatro de los cuales tienen conjuntos de instrucciones basados en puertas y dos tienen un conjunto de instrucciones de Majorana. Estos modelos de cúbits abarcan una gama de tiempos de operación y tasas de error, lo que permite una exploración suficiente de los costos de recursos necesarios para lograr aplicaciones cuánticas prácticas.

 Expandir tabla

Parámetro qubit	Clase de API de Python	Descripción
"qubit_gate_ns_e3" , "qubit_gate_ns_e4"	GATE_NS_E3 , GATE_NS_E4	Los tiempos de operación y las fidelidades pueden corresponder a futuras versiones de cúbits transmon superconductores , o cúbits de spin , que normalmente presentan tiempos de operación en el rango de nanosegundos. Para estos qubits, se supone que las operaciones de compuerta y medición toman 50 ns y 100 ns, respectivamente. Se supone que las tasas de error de puertas de un solo cúbit y de dos cúbits son 10^{-3} <i>tomando</i> como una target

Parámetro qubit	Clase de API de Python	Descripción
		realista, y 10^{-4} como una target optimista para un sistema ampliado.
<code>"qubit_gate_us_e3"</code> , <code>"qubit_gate_us_e4"</code>	<code>GATE_US_E3</code> , <code>GATE_US_E4</code>	Los tiempos de operación y las fidelidades pueden corresponder a versiones futuras de cúbits basados en iones , que normalmente tienen tiempos de operaciones en el régimen microsegundo. En función de las suposiciones habituales para cúbits de iones, se supone que las operaciones de puerta lógica y medición tardan 100 μ s. La tasa de errores para las puertas Clifford de un solo cúbit es de 10^{-3} como realista target y 10^{-4} como optimista target, mientras que la tasa de errores para las puertas no-Clifford de un solo cúbit (puerta T) es 10^{-6} . Para las puertas de dos qubits, la tasa de errores es de 10^{-3} como un valor realista target y 10^{-4} como un valor optimista target.
<code>"qubit_maj_ns_e4"</code> , <code>"qubit_maj_ns_e6"</code>	<code>MAJ_NS_E4</code> , <code>MAJ_NS_E6</code>	Los tiempos de operación y las fidelidades pueden asociarse con futuras versiones mejoradas de Majorana qubits . Para estos cúbits, se asume que las operaciones de puerta lógica y medición toman 100 ns. Para tener en cuenta la protección topológica en el hardware, se supone que las tasas de error de medición conjunta de un solo cúbit y de dos cúbits (tasas de error de Clifford) son 10^{-4} como opción realista target, y 10^{-6} como opción optimista target. Las operaciones que no son Clifford en esta arquitectura no tienen protección topológica, la tasa de errores de las puertas T físicas que no son Clifford es del 5 %.

⚠ Nota:

A menos que especifique un valor diferente, el valor predeterminado para el modelo de cúbits es `"qubit_gate_ns_e3"`.

Parámetros para parámetros de cúbit predefinidos

Como referencia, los parámetros de cúbit predefinidos completos son los siguientes:

Python

```
{
  "qubitParams": {
    "name": "qubit_gate_ns_e3",
    "instructionSet": "GateBased",
    "oneQubitMeasurementTime": "100 ns",
    "oneQubitGateTime": "50 ns",
```

```

        "twoQubitGateTime": "50 ns",
        "tGateTime": "50 ns",
        "oneQubitMeasurementErrorRate": 1e-3,
        "oneQubitGateErrorRate": 1e-3,
        "twoQubitGateErrorRate": 1e-3,
        "tGateErrorRate": 1e-3
    }
}

{
    "qubitParams": {
        "name": "qubit_gate_ns_e4",
        "instructionSet": "GateBased",
        "oneQubitMeasurementTime": "100 ns",
        "oneQubitGateTime": "50 ns",
        "twoQubitGateTime": "50 ns",
        "tGateTime": "50 ns",
        "oneQubitMeasurementErrorRate": 1e-4,
        "oneQubitGateErrorRate": 1e-4,
        "twoQubitGateErrorRate": 1e-4,
        "tGateErrorRate": 1e-4
    }
}

{
    "qubitParams": {
        "name": "qubit_gate_us_e3",
        "instructionSet": "GateBased",
        "oneQubitMeasurementTime": "100 µs",
        "oneQubitGateTime": "100 µs",
        "twoQubitGateTime": "100 µs",
        "tGateTime": "100 µs",
        "oneQubitMeasurementErrorRate": 1e-3,
        "oneQubitGateErrorRate": 1e-3,
        "twoQubitGateErrorRate": 1e-3,
        "tGateErrorRate": 1e-6
    }
}

{
    "qubitParams": {
        "name": "qubit_gate_us_e4",
        "instructionSet": "GateBased",
        "oneQubitMeasurementTime": "100 µs",
        "oneQubitGateTime": "100 µs",
        "twoQubitGateTime": "100 µs",
        "tGateTime": "100 µs",
        "oneQubitMeasurementErrorRate": 1e-4,
        "oneQubitGateErrorRate": 1e-4,
        "twoQubitGateErrorRate": 1e-4,
        "tGateErrorRate": 1e-6
    }
}

{

```

```

    "qubitParams": {
        "name": "qubit_maj_ns_e4",
        "instructionSet": "Majorana",
        "oneQubitMeasurementTime": "100 ns",
        "twoQubitJointMeasurementTime": "100 ns",
        "tGateTime": "100 ns",
        "oneQubitMeasurementErrorRate": 1e-4,
        "twoQubitJointMeasurementErrorRate": 1e-4,
        "tGateErrorRate": 0.05
    }
}

{
    "qubitParams": {
        "name": "qubit_maj_ns_e6",
        "instructionSet": "Majorana",
        "oneQubitMeasurementTime": "100 ns",
        "twoQubitJointMeasurementTime": "100 ns",
        "tGateTime": "100 ns",
        "oneQubitMeasurementErrorRate": 1e-6,
        "twoQubitJointMeasurementErrorRate": 1e-6,
        "tGateErrorRate": 0.01
    }
}

```

Pasar parámetros de cúbit predefinidos

Hay dos maneras de especificar mediante programación parámetros de cúbit predefinidos. Una manera es seleccionar el nombre del modelo de cúbit para la `qubitParams` clase al ejecutar `qsharp.estimate`. Por ejemplo, para seleccionar el `"qubit_maj_ns_e6"` parámetro qubit, escriba el código siguiente:

Python

```

from qdk import qsharp

qsharp.estimate("RunProgram()", params=
    {"qubitParams": {
        "name": "qubit_maj_ns_e6"
    }},
    })

```

Otra manera es pasar los parámetros de cúbit a la [EstimatorParams](#) clase mediante [QubitParams](#). Por ejemplo, para seleccionar el `MAJ_NS_E6` parámetro qubit, escriba el código siguiente:

Python

```

from qdk import qsharp
from qdk.estimator import EstimatorParams, QubitParams

params = EstimatorParams()
params.qubit_params.name = QubitParams.MAJ_NS_E6 # qubit_maj_ns_e6 qubit parameter

qsharp.estimate("RunProgram()", params=params)

```

Personalización de parámetros de cúbit predefinidos

Puede personalizar los parámetros de cúbit predefinidos especificando el nombre y actualizando cualquiera de los demás valores. Por ejemplo, para reducir la tasa de errores de las medidas conjuntas de dos cúbits en `qubit_maj_ns_e4`, escriba el código siguiente:

Python

```

qsharp.estimate("RunProgram()", params=
    {"qubitParams": {
        "name": "qubit_maj_ns_e4",
        "twoQubitJointMeasurementErrorRate": 1e-5,
    },
})

```

O bien, puede pasar las instrucciones en forma de una lista.

Python

```

from qsharp.estimator import EstimatorParams, QubitParams

params = EstimatorParams()
params.qubit_params.name = QubitParams.MAJ_NS_E4
params.qubit_params.two_qubit_joint_measurement_error_rate = 1e-5

```

Parámetros de cúbit para cúbits basados en puertas

 Expandir tabla

Identificador de Python	Tipo de datos	Descripción
<code>name</code>	cadena	Nombre del modelo de cúbits
<code>instruction_set</code>	"basado en puertas"	Tecnología de cúbit subyacente
<code>one_qubit_measurement_time</code>	cadena con formato de tiempo	Tiempo de operación para la medición de un solo cúbit (t_{meas}) en ns

Identificador de Python	Tipo de datos	Descripción
<code>one_qubit_gate_time</code>	cadena con formato de tiempo	Tiempo de operación para la puerta de un solo cúbit (t_{gate}) en ns
<code>two_qubit_gate_time</code>	cadena con formato de tiempo	Tiempo de operación para la puerta de dos cúbits en ns
<code>t_gate_time</code>	cadena con formato de tiempo	Tiempo de operación para una puerta no Clifford de un solo cúbit en ns
<code>one_qubit_measurement_error_rate</code>	flotante	Tasa de errores para la medición de un solo cúbit
<code>one_qubit_gate_error_rate</code>	flotante	Tasa de errores para una puerta Clifford de un solo cúbit (p)
<code>two_qubit_gate_error_rate</code>	flotante	Tasa de errores para una puerta Clifford de dos cúbits
<code>t_gate_error_rate</code>	flotante	Tasa de errores para preparar un estado no Clifford de un solo cúbit (p_T)
<code>idle_error_rate</code>	flotante	Tasa de errores correspondiente al estado idling (inactivo)

En el código siguiente se muestra cómo especificar parámetros de cúbit personalizados para un conjunto de instrucciones basado en puertas:

Python

```
from qdk.estimated import EstimatorParams, QubitParams

params = EstimatorParams()

params.qubit_params.name = "your_custom_name"
params.qubit_params.instruction_set = "gate_based"
params.qubit_params.t_gate_error_rate = 0.03
params.qubit_params.t_gate_time = "10 ns"
params.qubit_params.idle_error_rate = 0.02
```

⚠ Nota:

El valor predeterminado de `two_qubit_gate_time` y es `t_gate_time`, el valor predeterminado de `one_qubit_gate_time` y `two_qubit_gate_error_rate` `t_gate_error_rate` es `one_qubit_gate_error_rate` y el valor predeterminado de `idle_error_rate` es `one_qubit_measurement_error_rate`.

Parámetros de cúbit para cúbits de Majorana

 Expandir tabla

Identificador de Python	Tipo de datos	Descripción
<code>name</code>	cadena	Nombre del modelo de cúbits
<code>instruction_set</code>	"majorana"	Tecnología de cúbit subyacente
<code>one_qubit_measurement_time</code>	cadena con formato de tiempo	Tiempo de operación para la medición de un solo cúbit (t_{meas}) en ns
<code>two-qubit_joint_measurement_time</code>	cadena con formato de tiempo	Tiempo de operación para la medición de dos cúbits en ns
<code>t_gate_time</code>	cadena con formato de tiempo	Tiempo de operación para una puerta no Clifford de un solo cúbit en ns
<code>one_qubit_measurement_error_rate</code>	flotante	Tasa de errores para la medición de un solo cúbit
<code>two_qubit_joint_measurement_error_rate</code>	flotante	Tasa de errores para la medición de dos cúbits
<code>t_gate_error_rate</code>	flotante	Tasa de errores para preparar un estado no Clifford de un solo cúbit (p_T)
<code>idle_error_rate</code>	flotante	Tasa de errores correspondiente al estado idling (inactivo)

Una plantilla mínima para un conjunto de instrucciones basado en Majorana con todos los valores necesarios es:

Python

```
from qdk.estimator import EstimatorParams, QubitParams

params = EstimatorParams()

params.qubit_params.name = "your_custom_name"
params.qubit_params.instruction_set = "majorana"
params.qubit_params.one_qubit_measurement_time = "10 ns"
params.qubit_params.one_qubit_measurement_error_rate = 0.01
```

 **Nota:**

El valor predeterminado de `two_qubitJointMeasurementTime` y es `t_gate_time`, el valor predeterminado de `one_qubit_measurement_time` y

`two_qubit_joint_measurement_error_rate` `t_gate_error_rate` es `one_qubit_measurement_error_rate` y el valor predeterminado de `idle_error_rate` es `one_qubit_measurement_error_rate`.

Para `one_qubit_measurement_error_rate` y `two_qubit_joint_measurement_error_rate`, puede especificar las tasas de error que corresponden a lecturas de medida, `readout`, y procesamiento de medidas, `process`. Estos valores pueden ser `<double>` números o pares de números. Por ejemplo:

Python

```
params.qubit_params.two_qubit_joint_measurement_error_rate = \
    MeasurementErrorRate(process=0.00005, readout=0.00007)
```

ⓘ Nota:

Si especifica un valor numérico único para las tasas de error de un solo cúbit y de dos cúbits en la medida de cúbits de Majorana, las tasas de error de lectura y de proceso podrían ser iguales.

ⓘ Importante

Todos los valores que no se especifican tomarán un valor predeterminado. Por ejemplo, al especificar `"qubit": {"oneQubitGateTime": "200 ns"}` se modela un cúbit basado en puertas en el que tanto el tiempo de la puerta de dos cúbits como el tiempo de la puerta de un cúbit es de 200 ns. Para las unidades, debe especificar cadenas de tiempo, que son números de punto flotante de precisión doble, seguidos de un espacio y la unidad de tiempo para estos valores. Los sufijos de hora admitidos son `ns`, `µs` (o `us`), `ms` y `s`.

Esquemas de corrección de errores cuánticos

Para ejecutar aplicaciones cuánticas de escala práctica, las operaciones cuánticas deben tener tasas de error bajas. Estas tasas de error targets suelen ir más allá de las capacidades de los cúbits físicos crudos. Para superar esta limitación, la corrección cuántica de errores (QEC) y el cálculo tolerante a errores son dos técnicas cruciales que forman los bloques de creación de equipos cuánticos a gran escala. En primer lugar, QEC nos permite componer varios cúbits físicos propensos a errores y crear un cúbit lógico más confiable que conserva la información cuántica mejor que los cúbits físicos subyacentes.

El estimador de recursos usa la fórmula siguiente para modelar las tasas de error lógicas mediante un modelo exponencial,

$$P = ad^k \left(\frac{p}{p^*} \right)^{\frac{d+1}{2}}$$

donde a es un factor previo cruzado, d es la distancia del código, k es una potencia de coeficiente de distancia (normalmente establecida en 0), p es la tasa de error física y p^* es el umbral de corrección de errores cuánticos. El prefactor de cruce a se puede extraer numéricamente para simulaciones.

La distancia de código d es un parámetro que controla el número de errores que se pueden corregir. Por lo tanto, la distancia de código define la tasa de errores de los cúbits lógicos y el número de cúbits físicos necesarios para codificarlos. Tanto la precisión como el número de cúbits físicos aumentan con la distancia del código. El objetivo de un esquema QEC es encontrar la distancia mínima del código que puede lograr la tasa de errores necesaria establecida para una aplicación determinada.

La tasa de errores físicos p se extrae de los parámetros del cúbit como la tasa de error en el peor caso de cualquier operación física de Clifford realizada en el dispositivo. En concreto, $p = \max(\text{one\_qubit\_measurement\_error\_rate}, \text{one\_qubit\_gate\_error\_rate}, \text{two\_qubit\_gate\_error\_rate})$ para los parámetros de cúbit con un conjunto de instrucciones basado en puertas y $p = \max(\text{one\_qubit\_measurement\_error\_rate}, \text{two\_qubit\_joint\_measurement\_error\_rate})$ para los parámetros de cúbit con un conjunto de instrucciones Majorana. Los esquemas QEC suelen tener un umbral de tasa de error p^* por debajo del cual la corrección de errores suprime los errores.

El estimador de recursos de Azure Quantum admite dos esquemas QEC predefinidos: un código de superficie y un código de floquet.

[Expandir tabla](#)

Protocolo QEC	Clase de API de Python	Descripción
<code>surface_code</code>	<code>SURFACE_CODE</code>	El código de superficie basado en puertas se basa en arXiv:1208.0928 y arXiv:1009.3686 . El código de superficie Majorana se basa en arXiv:1909.03002 y arXiv:2007.00307 .
<code>floquet_code</code>	<code>FLOQUET_CODE</code>	Solo para cúbits de Majorana. El código de floquet se basa en arXiv:2202.11829 .

ⓘ Nota:

A menos que se especifique otro valor, el valor predeterminado para el esquema QEC es "surface_code".

Parámetros para esquemas QEC predefinidos

Los parámetros exactos de cada esquema QEC predefinido son los siguientes.

Python

```
{
  "qubitParams": {
    "instructionSet": "GateBased",
  }
  "qecScheme": {
    "name": "surface_code",
    "errorCorrectionThreshold": 0.01,
    "crossingPrefactor": 0.03,
    "distanceCoefficientPower": 0,
    "logicalCycleTime": "(4 * twoQubitGateTime + 2 * oneQubitMeasurementTime) *
codeDistance",
    "physicalQubitsPerLogicalQubit": "2 * codeDistance * codeDistance"
  }
}

{
  "qubitParams": {
    "instructionSet": "Majorana",
  }
  "qecScheme": {
    "name": "surface_code",
    "errorCorrectionThreshold": 0.0015,
    "crossingPrefactor": 0.08,
    "distanceCoefficientPower": 0,
    "logicalCycleTime": "20 * oneQubitMeasurementTime * codeDistance",
    "physicalQubitsPerLogicalQubit": "2 * codeDistance * codeDistance"
  }
}

{
  "qubitParams": {
    "instructionSet": "Majorana",
  }
  "qecScheme": {
    "name": "floquet_code",
    "errorCorrectionThreshold": 0.01,
    "crossingPrefactor": 0.07,
    "distanceCoefficientPower": 0,
    "logicalCycleTime": "3 * oneQubitMeasurementTime * codeDistance",
    "physicalQubitsPerLogicalQubit": "4 * codeDistance * codeDistance + 8 *
(codeDistance - 1)"
  }
}
```

```
}  
}
```

Traspaso de esquemas predefinidos de QEC

Hay dos maneras de especificar esquemas de QEC predefinidos. Puede seleccionar el nombre del modelo QEC para la `"qecScheme"` clase al ejecutar `qsharp.estimate`. Por ejemplo, para seleccionar el código de floquet, escriba:

Python

```
qsharp.estimate("RunProgram()", params=  
    {"qecScheme": {  
        "name": "floquet_code"  
    }}  
))
```

También puede pasar una lista de parámetros de estimación a la [EstimatorParams](#) clase usando la [QECScheme](#) clase. Por ejemplo, para seleccionar el código de floquet, escriba:

Python

```
from qdk import qsharp  
from qsharp.estimator import EstimatorParams, QubitParams, QECScheme  
  
params = EstimatorParams()  
params.items.qec_scheme.name = QECScheme.FLOQUET_CODE # floquet code QEC scheme  
  
qsharp.estimate("RunProgram()", params=params)
```

Personalización de esquemas de QEC predefinidos

Puede personalizar esquemas de QEC predefinidos especificando el nombre y actualizando cualquiera de los otros valores. Por ejemplo, para aumentar el factor previo del cruce en el código Floquet, escriba:

Python

```
qsharp.estimate("RunProgram()", params=  
    {"qecScheme": {  
        "name": "floquet_code",  
        "crossingPrefactor": 0.07,  
    }}  
))
```

⚠ Nota:

El valor predeterminado de `"logicalCycleTime"` y `"physicalQubitsPerLogicalQubit"` es, el valor predeterminado de `"oneQubitMeasurementTime"` es `"errorCorrectionThreshold"` 0.01 y el valor predeterminado de `"crossingPrefactor"` es 0.03.

Personaliza tus esquemas de QEC

El estimador de recursos puede abstraer un esquema QEC personalizado basado en la fórmula anterior proporcionando valores para el `"crossingPrefactor"` a , el `distanceCoefficientPower` k y el `"errorCorrectionThreshold"` p^* . También debe especificar el `"logicalCycleTime"`, que es el tiempo de ejecutar una sola operación lógica, y que depende de la distancia del código y de las suposiciones de tiempo de operación física de los cúbits físicos subyacentes. Por último, una segunda fórmula calcula, `"physicalQubitsPerLogicalQubit"` que es el número de cúbits físicos necesarios para codificar un cúbit lógico en función de la distancia del código.

Puede usar el código siguiente como plantilla para esquemas QEC:

Python

```
qsharp.estimate("RunProgram()", params=
    {"qecScheme": {
        "crossingPrefactor": <double>,
        "errorCorrectionThreshold": <double>,
        "distanceCoefficientPower": <integer>,
        "logicalCycleTime": <formula string>,
        "physicalQubitsPerLogicalQubit": <formula string>
    }}
))
```

Dentro de las fórmulas, puede usar las variables `one_qubit_gate_time`, `two_qubit_gate_time`, `one_qubit_measurement_time`, y `two_qubit_joint_measurement_time`, cuyos valores se obtienen del campo correspondiente de los parámetros del cúbit físico, así como la variable para la distancia de código calculada para el cúbit lógico, basada en las propiedades del cúbit físico, el umbral de corrección de errores y el prefactor de cruce. Las variables de hora y `eccDistance` se pueden usar para describir la `logicalCycleTime` fórmula. Para la fórmula `physicalQubitsPerLogicalQubit`, solo se puede usar el `eccDistance`.

Presupuesto de errores

El presupuesto total de errores ϵ establece el error tolerado general para el algoritmo, es decir, la probabilidad de error permitida del algoritmo. Su valor global debe estar comprendido entre 0 y 1 y el valor predeterminado es 0,001, que corresponde al 0,1 %. En otras palabras, el algoritmo puede producir un error máximo de una vez en 1000 ejecuciones. Este parámetro es muy específico de la aplicación.

Por ejemplo, si está ejecutando el algoritmo de Shor para factorizar enteros, se puede tolerar un valor grande para el presupuesto de errores, ya que uno puede comprobar que las salidas son realmente los factores primos de la entrada. Por otro lado, es posible que se necesite un presupuesto de errores más pequeño en el caso de un algoritmo que resuelva un problema con una solución, que no se puede comprobar de forma eficaz.

El presupuesto de errores corresponde a la suma de tres partes:

$$\epsilon = \epsilon_{\log} + \epsilon_{\text{dis}} + \epsilon_{\text{syn}}$$

Los errores lógicos $\epsilon_{\log}$ son el error de implementación de cúbits lógicos, el error de estado T ϵ_{dis} es el error de producir estados T a través de la destilación y el error de puerta de rotación ϵ_{syn} es el error de sintetizar puertas de rotación con ángulos arbitrarios.

ⓘ Nota:

A menos que se especifique otro valor, el presupuesto de errores ϵ se distribuye uniformemente entre el error lógico, el error de estado T y el error de puerta de rotación.

Tenga en cuenta que para la destilación y la síntesis de rotación, los respectivos presupuestos de errores ϵ_{dis} y ϵ_{syn} se distribuyen uniformemente entre todos los estados T necesarios y todas las puertas de rotación necesarias, respectivamente. Si no hay puertas de rotación en el algoritmo de entrada, el presupuesto de errores se distribuye uniformemente a errores lógicos y errores de estado T.

Superar el presupuesto de errores

Hay dos maneras de especificar el presupuesto de errores estableciendo un número entre 0 y 1. Puede exceder el presupuesto de errores al ejecutar `qsharp.estimate`. Por ejemplo, para seleccionar un presupuesto de errores de 1/3, escriba:

Python

```
qsharp.estimate("RunProgram()", params=
    {'errorBudget': 0.333
    })
```

También puede pasar los parámetros de presupuesto de error a la [EstimatorParams](#) clase .

Python

```
from qdk import qsharp
from qdk.estimator import EstimatorParams, QubitParams, QECScheme

params = EstimatorParams()
params.items.error_budget = 0.333 # error budget of 1/3

qsharp.estimate("RunProgram()", params=params)
```

Además, puede especificar individualmente cada componente del presupuesto de errores. La suma de todos los valores es el presupuesto total de errores y debe estar comprendido entre 0 y 1. Si un algoritmo cuántico no contiene estados T o rotaciones, los valores de `t_states` y `rotations` pueden ser 0 respectivamente.

En el código siguiente se muestra cómo especificar el parámetro de presupuesto de errores con estados T y rotaciones:

Python

```
from qdk.estimator import EstimatorParams, QubitParams

params = EstimatorParams()
params.error_budget.logical = 0.01
params.error_budget.t_states = 0.02
params.error_budget.rotations = 0.03
```

Restricciones

Puede usar la clase `"constraints"` para aplicar restricciones en el nivel de componente de fábrica T. Al ajustar las restricciones, puede optimizar las estimaciones para reducir el número de cúbits o para reducir el runtime.

[Expandir tabla](#)

Parámetro	Tipo de datos	Descripción
<code>logical_depth_factor</code>	flotante	Controlar el tiempo de ejecución. Si tiene un valor mayor que 1, el número inicial de ciclos lógicos, también denominado <i>profundidad</i> lógica, se multiplica por este número. Al reducir <code>logical_depth_factor</code> , se puede aumentar el número de invocaciones de la fábrica T en un cierto periodo de tiempo, lo que da lugar a que se necesiten menos copias de la fábrica T para

Parámetro	Tipo de datos	Descripción
		generar el mismo número de estados T. Al reducir el número de copias de fábrica de T, el tiempo de ejecución del algoritmo aumenta en consecuencia. El factor de escala del tiempo de ejecución total puede ser mayor, ya que la tasa de errores lógica necesaria aumenta debido al número adicional de ciclos.
<code>max_t_factories</code>	entero	Número máximo de copias de fábrica T. El estimador de recursos determina los recursos necesarios seleccionando el número óptimo de copias de fábrica de T que minimiza el número de cúbits físicos usados, sin tener en cuenta la sobrecarga de tiempo. El <code>max_t_factories</code> parámetro limita el número máximo de copias y, por tanto, ajusta el número de ciclos lógicos en consecuencia. Para obtener más información, consulte Estimación física de la fábrica T .
<code>max_duration</code>	cadena con formato de tiempo	Tiempo de ejecución máximo para el algoritmo. El estimador de recursos solo acepta una restricción de <code>max_duration</code> o <code>max_physical_qubits</code> a la vez, pero no dos. Si <code>max_duration</code> se especifica, el estimador de recursos intenta encontrar la mejor estimación de <code>max_physical_qubits</code> entre las soluciones restringidas por el número máximo especificado.
<code>max_physical_qubits</code>	entero	Número máximo de cúbits físicos para el algoritmo. El estimador de recursos solo acepta una restricción de <code>max_duration</code> o <code>max_physical_qubits</code> a la vez, pero no dos. Si <code>max_physical_qubits</code> se especifica, el estimador de recursos intenta encontrar la mejor estimación de <code>max_duration</code> entre las soluciones restringidas por el número máximo especificado.

En el código siguiente se muestra cómo especificar las restricciones de un algoritmo cuántico:

Python

```
from qdk.estimated import EstimatorParams

params = EstimatorParams()

params.constraints.max_duration = "1 s"
params.constraints.logical_depth_factor = 1.5
params.constraints.max_t_factories = 10
```

⚠ Nota:

Si el valor proporcionado a `max_duration` o `max_physical_qubits` es demasiado pequeño para encontrar una solución factible, el estimador de recursos devuelve un error. Si no se

especifican ni `max_duration` ni `max_physical_qubits` restricciones, el Estimador de recursos pretende encontrar una solución con el menor tiempo.

💡 Sugerencia

Puede usar `max_duration` y `max_physical_qubits` para influir en el espacio de la solución, posiblemente encontrar soluciones con un tiempo de ejecución más largo, pero un número menor de cúbits en comparación con las soluciones sin estas restricciones. Existe un equilibrio entre el tiempo de ejecución y el número de cúbits, y esta compensación se puede administrar eficazmente para algunos algoritmos, con efectos variables en distintos algoritmos. La tabla IV de [[arXiv:2211.07629](#)] ilustra el uso efectivo del equilibrio entre el número de cúbits y el tiempo de ejecución para los algoritmos de dinámica cuántica. Para más información, consulte el ejemplo de [Estimación de recursos cuánticos con restricciones de tiempo o número de cúbits](#).

Unidades de destilación

Se pueden proporcionar especificaciones para algoritmos de destilación para fábricas T con la clase `DistillationUnitSpecification`. La especificación puede ser predefinida o personalizada. Para especificar una especificación predefinida, seleccione el nombre de la unidad de destilación:

`15-1 RM` O `15-1 space-efficient`.

Python

```
from qdk.estimated import EstimatorParams, DistillationUnitSpecification

params = EstimatorParams()
unit = DistillationUnitSpecification()
unit.name = "15-1 RM" # predefined distillation unit

params.distillation_unit_specifications.append(unit)
```

En ambos casos, la notación `15-1` significa 15 estados de entrada T y 1 estado de salida T. La unidad de destilación `15-1 space-efficient` usa menos cúbits que `15-1 RM`, pero requiere más runtime. Para obtener más información, consulte [Tabla VI](#).

💡 Sugerencia

Obtiene un mejor rendimiento cuando se usan unidades de destilación predefinidas en lugar de unidades personalizadas.

Personalización de las unidades de destilación

Puede personalizar sus propias unidades de destilación. Los parámetros exactos de las unidades de destilación son los siguientes.

Python

```
qsharp.estimate("RunProgram()", params=
    {"distillationUnitSpecifications": {
        "displayName": <string>,
        "numInputTs": <int>,
        "numOutputTs": <int>,
        "failureProbabilityFormula": <string>,
        "outputErrorRateFormula": <string>,
        "physicalQubitSpecification": <protocol specific
parameters>,
        "logicalQubitSpecification": <protocol specific parameters>,
        "logicalQubitSpecificationFirstRoundOverride": <protocol
specific parameters>, # Only if "logicalQubitSpecification"
    }
})
```

Se espera que todos los parámetros numéricos sean positivos. `displayName` especifica cómo se mostrará la unidad de destilación en los resultados de salida.

El siguiente código muestra cómo especificar los parámetros de la unidad de destilación para un algoritmo cuántico mediante la clase `DistillationUnitSpecification` y la clase `ProtocolSpecificDistillationUnitSpecification`.

Python

```
from qdk.estimate import EstimatorParams, DistillationUnitSpecification,
ProtocolSpecificDistillationUnitSpecification

params = EstimatorParams()
unit = DistillationUnitSpecification()
unit.display_name = "T"
unit.failure_probability_formula = "c"
unit.output_error_rate_formula = "r"
unit.num_input_ts = 1
unit.num_output_ts = 2

physical_qubit_specification = ProtocolSpecificDistillationUnitSpecification()
physical_qubit_specification.num_unit_qubits = 1
physical_qubit_specification.duration_in_qubit_cycle_time = 2
unit.physical_qubit_specification = physical_qubit_specification
```

Las fórmulas de `failure_probability_formula` y `output_error_rate_formula` son fórmulas personalizadas con operaciones aritméticas básicas, constantes y solo tres parámetros:

- `clifford_error_rate`, también se indica como `c`.
- `readout_error_rate`, también se indica como `r`.
- `input_error_rate`, también se indica como `z`.

Consulte los ejemplos siguientes de fórmulas personalizadas con notación larga y corta. En estos ejemplos se muestran las fórmulas que se usan de forma predeterminada en la implementación estándar.

[Expandir tabla](#)

Parámetro	Fórmula larga	Fórmula corta
<code>failure_probability_formula</code>	"15,0 * tasa_de_error_de_entrada + 356,0 * tasa_de_error_de_clifford"	"15.0 * z + 356.0 * c"
<code>output_error_rate_formula</code>	35,0 * tasa_de_error_de_entrada ^ 3 + 7,1 * tasa_de_error_de_clifford	"35.0 * z ^ 3 + 7.1 * c"

Se debe proporcionar al menos uno de los parámetros `physical_qubit_specification` o `logical_qubit_specification`. Si solo se proporciona la anterior, la unidad de destilación se puede aplicar a cúbits físicos. Si solo se proporciona este último, la unidad de destilación se puede aplicar a cúbits lógicos. Si se proporcionan ambos, la unidad de destilación se puede aplicar a ambos tipos de cúbits.

El parámetro `logical_qubit_specification_first_round_override` solo se puede proporcionar si `logical_qubit_specification` se especifica. Si es así, invalida los valores de `logical_qubit_specification` cuando se aplican en la primera ronda de destilación. El valor `<protocol specific parameters>` necesario para `logical_qubit_specification_first_round_override` debe tener el esquema siguiente:

Python

```
{
    "numUnitQubits": <int>,
    "durationInQubitCycleTime": <double>
}
```

Estimación de la frontera de Pareto

Al calcular los recursos de un algoritmo, es importante tener en cuenta el equilibrio entre el número de cúbits físicos y el tiempo de ejecución del algoritmo. Podría considerar la asignación de tantos cúbits físicos como sea posible para reducir el tiempo de ejecución del algoritmo. Sin embargo, el número de cúbits físicos está limitado por el número de cúbits

físicos disponibles en el hardware cuántico. Comprender el equilibrio entre el tiempo de ejecución y la escala del sistema es uno de los aspectos más importantes de la estimación de recursos.

La estimación de la frontera de Pareto proporciona varias estimaciones para el mismo algoritmo, cada una de las cuales muestra inconvenientes entre el número de cúbits y el tiempo de ejecución.

ⓘ **Nota:**

Si ejecuta el estimador de recursos en Visual Studio Code con el **QDKcomando : Calcular estimaciones de recursos**, la estimación de frontera de Pareto está habilitada de forma predeterminada.

Si ejecuta el estimador de recursos en Python, debe especificar el `"estimateType"` parámetro como `"frontier"`.

Python

```
result = qsharp.estimate("RunProgram()", params=
    {"qubitParams": { "name": "qubit_maj_ns_e4" },
     "qecScheme": { "name": "surface_code" },
     "estimateType": "frontier", # Pareto frontier estimation
    }
)
```

Si desea visualizar los resultados de la estimación de la frontera de Pareto, puede usar la función `EstimatesOverview`. Esta función muestra los resultados de la estimación de frontera en la tabla y un diagrama de tiempo espaciador. Para obtener más información, vea [Diagrama espacio-tiempo](#).

Python

```
from qdk.widgets import EstimatesOverview

EstimatesOverview(result)
```

ⓘ **Nota:**

Si experimenta problemas al trabajar con el Estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Pasos siguientes

- [Descripción de los resultados del estimador de recursos](#)
- [Ejecución de la primera estimación de recursos](#)
- [Diferentes formas de ejecutar el estimador de recursos](#)
- [Control de programas grandes con el estimador de recursos](#)
- [Tutorial: Estimación de los recursos de un problema de química cuántica](#)

Last updated on 12/11/2025

Obtención del resultado del estimador de recursos

29/09/2025

Aprenda a interpretar y recuperar los parámetros y diagramas de salida del estimador de recursos. En este artículo se explica cómo obtener acceso mediante programación a los resultados del estimador de recursos en cuadernos de Jupyter Notebook en Visual Studio Code (VS Code).

Requisitos previos

- Un entorno de Python con [Python y Pip](#) instalados.
- La versión más reciente de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- VS Code con las [extensiones kit de desarrollo](#) de Azure Quantum, [Python](#) y [Jupyter](#) instaladas.
- Los paquetes y `qsharp-widgets` Azure Quantum `qsharp` más recientes.

Bash

```
python -m pip install --upgrade qsharp qsharp-widgets
```

⚠ Nota

Si ejecuta el estimador de recursos en VS Code desde la paleta de comandos, no se aplican los siguientes comandos. Para obtener más información, consulte [Diferentes formas de ejecutar el estimador de recursos](#).

Parámetros de salida

La salida del estimador de recursos es un informe que se imprime en la consola y al que puede acceder mediante programación. Por ejemplo, el siguiente fragmento de código muestra los valores de todos los parámetros del estimador de recursos:

Python

```
result['jobParams']
```

La tabla siguiente contiene el tipo de datos y una breve descripción de cada parámetro de salida, al que puede acceder mediante programación. ¿ESTOS PARÁMETROS SON EXHAUSTIVOS Y SIGUEN FORMANDO PARTE DEL CONJUNTO DE PARÁMETROS DE SALIDA? NO VEO EL ESTADO]

 Expandir tabla

Parámetro de salida de nivel superior	Tipo de datos	Descripción
status	string	El estado del trabajo, siempre Succeeded es .
jobParams	diccionario	Parámetros target del trabajo que se pasan como entrada.
physicalCounts	diccionario	Estimaciones de recursos físicos. Para obtener más información, consulte Recuentos físicos .
physicalCountsFormatted	diccionario	Las estimaciones de recursos físicos con formato para mostrarse en los datos del informe. Para obtener más información, consulte Recuentos físicos con formato .
logicalQubit	diccionario	Propiedades de cúbit lógico. Para obtener más información, consulte Qubit lógico .
tfactory	diccionario	Las propiedades del generador de T.
logicalCounts	diccionario	Estimaciones de recursos lógicos de diseño previo. Para obtener más información, consulte Recuentos lógicos .
reportData	diccionario	Datos de generación para el informe de estimación de recursos.

Recuentos físicos

El physicalCounts diccionario contiene las siguientes entradas:

 Expandir tabla

Parámetro de salida	Tipo de datos	Descripción
physicalQubits	number	Número total de cúbits físicos.
runtime	number	Tiempo de ejecución total para ejecutar el algoritmo en nanosegundos.
rqops	number	Número de operaciones cuánticas confiables por segundo (QOPS).

Parámetro de salida	Tipo de datos	Descripción
<code>breakdown</code>	diccionario	Desglose de estimaciones. Para obtener más información, consulte Desglose de recuentos físicos.

Desglose de recuentos físicos

El `breakdown` diccionario de `physicalCounts` contiene las siguientes entradas:

[Expandir tabla](#)

Parámetro de salida	Tipo de datos	Descripción
<code>algorithmicLogicalQubits</code>	number	Los cúbits lógicos necesarios para ejecutar el algoritmo y no incluyen recursos para factorías de T.
<code>algorithmicLogicalDepth</code>	number	Los ciclos lógicos necesarios para ejecutar el algoritmo y no incluyen recursos para factorías de T.
<code>logicalDepth</code>	number	El número posiblemente ajustado de ciclos que se calcula cada vez que el tiempo de ejecución de la factoría de T es más rápido, la ejecución del algoritmo.
<code>numTstates</code>	number	Número de estados T consumidos por el algoritmo.
<code>clockFrequency</code>	number	Número de ciclos lógicos por segundo.
<code>numTfactories</code>	number	Número de fábricas de T (suponiendo diseño uniforme de fábrica de T).
<code>numTfactoryRuns</code>	number	Número de frecuencia con la que se deben ejecutar todas las factorías T paralelas.
<code>physicalQubitsForTfactories</code>	number	Número de cúbits físicos para todas las factorías de T.
<code>physicalQubitsForAlgorithm</code>	number	Número de cúbits físicos para el diseño del algoritmo.
<code>requiredLogicalQubitErrorRate</code>	number	Tasa de errores lógica necesaria.
<code>requiredLogicalTstateErrorRate</code>	number	Tasa de errores de estado T lógico necesaria.
<code>numTsPerRotation</code>	number	Número de puertas T por rotación.
<code>cliffordErrorRate</code>	number	Tasa de errores de Clifford basada en los parámetros de cúbit.

Recuentos físicos con formato

El `physicalCounts` diccionario contiene las siguientes entradas:

 Expandir tabla

Parámetro de salida	Tipo de datos	Descripción
<code>runtime</code>	string	Tiempo de ejecución total como cadena de uso humano.
<code>rqops</code>	string	Número de operaciones cuánticas confiables por segundo (QOPS) con formato de sufijo de métrica.
<code>physicalQubits</code>	string	Número total de cúbits físicos con sufijo de métrica.
<code>algorithmicLogicalQubits</code>	string	Cúbits lógicos algorítmicos con sufijo de métrica.
<code>algorithmicLogicalDepth</code>	string	Profundidad lógica algorítmica con sufijo de métrica.
<code>logicalDepth</code>	string	Posiblemente se ajuste la profundidad lógica algorítmica con sufijo de métrica.
<code>numTstates</code>	string	Número de estados T con sufijo de métrica.
<code>numTfactories</code>	string	Número de copias de fábrica de T con sufijo de métrica.
<code>numTfactoryRuns</code>	string	Número de ejecuciones de generador de T con sufijo de métrica.
<code>physicalQubitsForAlgorithm</code>	string	Número de cúbits físicos para el algoritmo con sufijo de métrica.
<code>physicalQubitsForTfactories</code>	string	Número de cúbits físicos para factorías de T con sufijo de métrica.
<code>physicalQubitsForTfactoriesPercentage</code>	string	Número de cúbits físicos para todas las factorías de T en porcentaje al total.
<code>requiredLogicalQubitErrorRate</code>	string	Velocidad de error de cúbit lógico truncada.
<code>requiredLogicalTstateErrorRate</code>	string	Tasa de errores de estado T truncada.
<code>physicalQubitsPerLogicalQubit</code>	string	Número de cúbits físicos por cúbit lógico con sufijo de métrica.
<code>logicalCycleTime</code>	string	El tiempo del ciclo lógico de un cúbit lógico como una cadena que admite el usuario.

Parámetro de salida	Tipo de datos	Descripción
<code>clockFrequency</code>	string	Número de ciclos lógicos por segundo como una cadena que admite personas.
<code>logicalErrorRate</code>	string	Tasa de errores lógicos truncadas.
<code>tfactoryPhysicalQubits</code>	string	Número de cúbits físicos en la fábrica de T con sufijo de métrica (o mensaje que indica que no hay ningún generador de T).
<code>tfactoryRuntime</code>	string	El tiempo de ejecución de un único generador de T como cadena de compatibilidad humana (o mensaje que indica que no hay ningún generador de T).
<code>numInputTstates</code>	string	Número de estados de entrada T (o mensaje que indica que no hay ningún generador de T).
<code>numUnitsPerRound</code>	string	Número de unidades por destilación redonda, separadas por comas en una cadena (o mensaje que indica que no hay ningún generador de T).
<code>unitNamePerRound</code>	string	Los nombres de unidad de cada ronda de destilación, separadas por comas en una cadena (o mensaje que indica que no hay ningún generador de T).
<code>codeDistancePerRound</code>	string	Distancias de código por ronda de destilación, separadas por comas en una cadena (o mensaje que indica que no hay ningún generador de T).
<code>physicalQubitsPerRound</code>	string	Número de cúbits físicos por destilación, separados por comas en una cadena (o mensaje que no hay ningún generador de T).
<code>tfactoryRuntimePerRound</code>	string	El tiempo de ejecución de cada ronda de destilación, mostrado como cadenas humanas separadas por comas (o mensaje que indica que no hay ningún generador de T).
<code>tstateLogicalErrorRate</code>	string	Tasa de errores de estado T lógico truncado (o mensaje que indica que no hay ningún generador de T).
<code>logicalCountsNumQubits</code>	string	Número de cúbits (diseño previo) con sufijo de métrica.
<code>logicalCountsTCount</code>	string	Número de puertas T (diseño previo) con sufijo de métrica.

Parámetro de salida	Tipo de datos	Descripción
<code>logicalCountsRotationCount</code>	string	Número de puertas de rotación (diseño previo) con sufijo de métrica.
<code>logicalCountsRotationDepth</code>	string	Profundidad de rotación (diseño previo) con sufijo de métrica.
<code>logicalCountsCczCount</code>	string	Número de puertas CCZ (diseño previo) con sufijo de métrica.
<code>logicalCountsCcixCount</code>	string	Número de puertas de CCiX (diseño previo) con sufijo de métrica.
<code>logicalCountsMeasurementCount</code>	string	Número de medidas de un solo cúbit (diseño previo) con sufijo de métrica.
<code>errorBudget</code>	string	Presupuesto total de errores truncado.
<code>errorBudgetLogical</code>	string	Presupuesto de errores truncado para el error lógico.
<code>errorBudgetTstates</code>	string	Presupuesto de errores truncado para la destilación de estado T defectuoso.
<code>errorBudgetRotations</code>	string	Presupuesto de errores truncado para la síntesis de rotación errónea.
<code>numTsPerRotation</code>	string	Número con formato de Ts por rotación (puede ser Ninguno).

Cúbit lógico

El `physicalCounts` diccionario contiene las siguientes entradas:

 Expandir tabla

Parámetro de salida	Tipo de datos	Descripción
<code>codeDistance</code>	number	Distancia de código calculada para el cúbit lógico.
<code>physicalQubits</code>	number	Número de cúbits físicos para cada cúbit lógico.
<code>logicalCycleTime</code>	number	Tiempo para ejecutar una operación lógica.
<code>logicalErrorRate</code>	number	Velocidad de error lógica del cúbit lógico.

Recuentos lógicos

El `physicalCounts` diccionario contiene las siguientes entradas:

 Expandir tabla

Parámetro de salida	Tipo de datos	Descripción
<code>numQubits</code>	number	Número de cúbits de diseño previo.
<code>tCount</code>	number	Número de puertas T de diseño previo.
<code>rotationCount</code>	number	Número previo de diseño de puertas de rotación.
<code>rotationDepth</code>	number	Profundidad de rotación de diseño previo.
<code>cczCount</code>	number	Número de preproceso de puertas CCZ.
<code>ccixCount</code>	number	Número de preproceso de puertas CCiX.
<code>measurementCount</code>	number	Número de diseños previos de medidas de un solo cúbit.

Sugerencia

Si desea usar un conjunto calculado previamente de recuentos lógicos para un trabajo de estimación de recursos, puede usar la `LogicalCounts` operación de Python para pasar las estimaciones conocidas al Estimador de recursos. Para obtener más información, consulte [Uso de estimaciones conocidas con el estimador](#) de recursos.

Diagrama de espacio

La estimación general de recursos físicos consta del número total de cúbits físicos usados para las copias del algoritmo y del generador de T. Puede inspeccionar la distribución entre estos dos mediante el diagrama de espacio.

El diagrama de espacio muestra la proporción de los cúbits físicos usados para el algoritmo y las [factorías](#) de T. Tenga en cuenta que el número de copias de fábrica T contribuye al número de cúbits físicos para las fábricas T.

En Jupyter Notebook, puede acceder al diagrama de espacio mediante el `SpaceChart` widget desde el `qsharp-widgets` paquete.

Python

```
import qsharp

from qsharp_widgets import SpaceChart
SpaceChart(result)
```

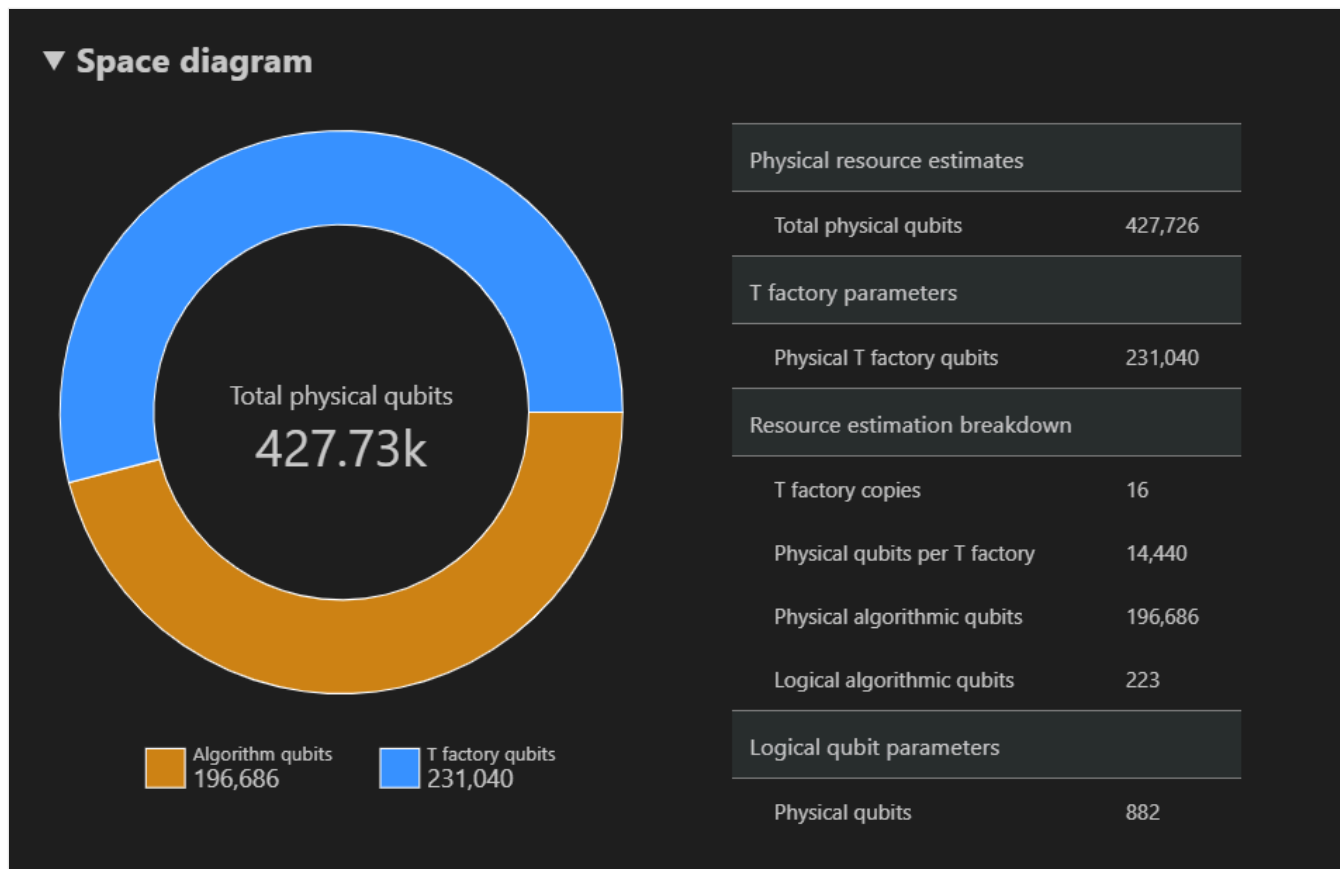


Diagrama circular que muestra la distribución de cúbits físicos totales entre cúbits de algoritmo y cúbits de fábrica de T. Hay una tabla con el desglose del número de copias de fábrica de T y el número de cúbits físicos por factoría de T.

Al ejecutar varias configuraciones de parámetros con la [estimación](#) de target frontera de Pareto, puede trazar el diagrama de espacio para una solución específica de . Por ejemplo, el código siguiente muestra cómo trazar el diagrama de espacio para la primera configuración de parámetros y el tercer tiempo de ejecución más corto.

Python

```
SpaceChart(result[0], 2) # First (estimate index=0) run and third (point index=2)
shortest runtime
```

Diagrama de tiempo espaciador

En la computación cuántica, hay un equilibrio entre el número de cúbits físicos y el tiempo de ejecución del algoritmo. Podría considerar la asignación de tantos cúbits físicos como sea posible para reducir el tiempo de ejecución del algoritmo. Sin embargo, el número de cúbits

físicos está limitado por el número de cúbits físicos disponibles en el hardware cuántico. Comprender el equilibrio entre el tiempo de ejecución y la escala del sistema es uno de los aspectos más importantes de la estimación de recursos.

Al calcular los recursos de un algoritmo, puede usar el diagrama de tiempo espaciado para visualizar los inconvenientes entre el número de cúbits físicos y el tiempo de ejecución del algoritmo.

ⓘ Nota

Para ver varias combinaciones óptimas en el diagrama de tiempo espacial, debe establecer el tipo de estimación en Estimación de frontera de Pareto. Si ejecuta el estimador de recursos en Visual Studio Code con el comando **QDK: Calcular estimaciones de recursos**, la estimación de frontera de Pareto está habilitada de forma predeterminada.

El diagrama de tiempo espaciado permite encontrar la combinación óptima de {número de cúbits, pares runtime} que satisfacen las restricciones del hardware cuántico. El diagrama muestra el número de cúbits físicos y el tiempo de ejecución del algoritmo para cada par {número de cúbits, runtime}.

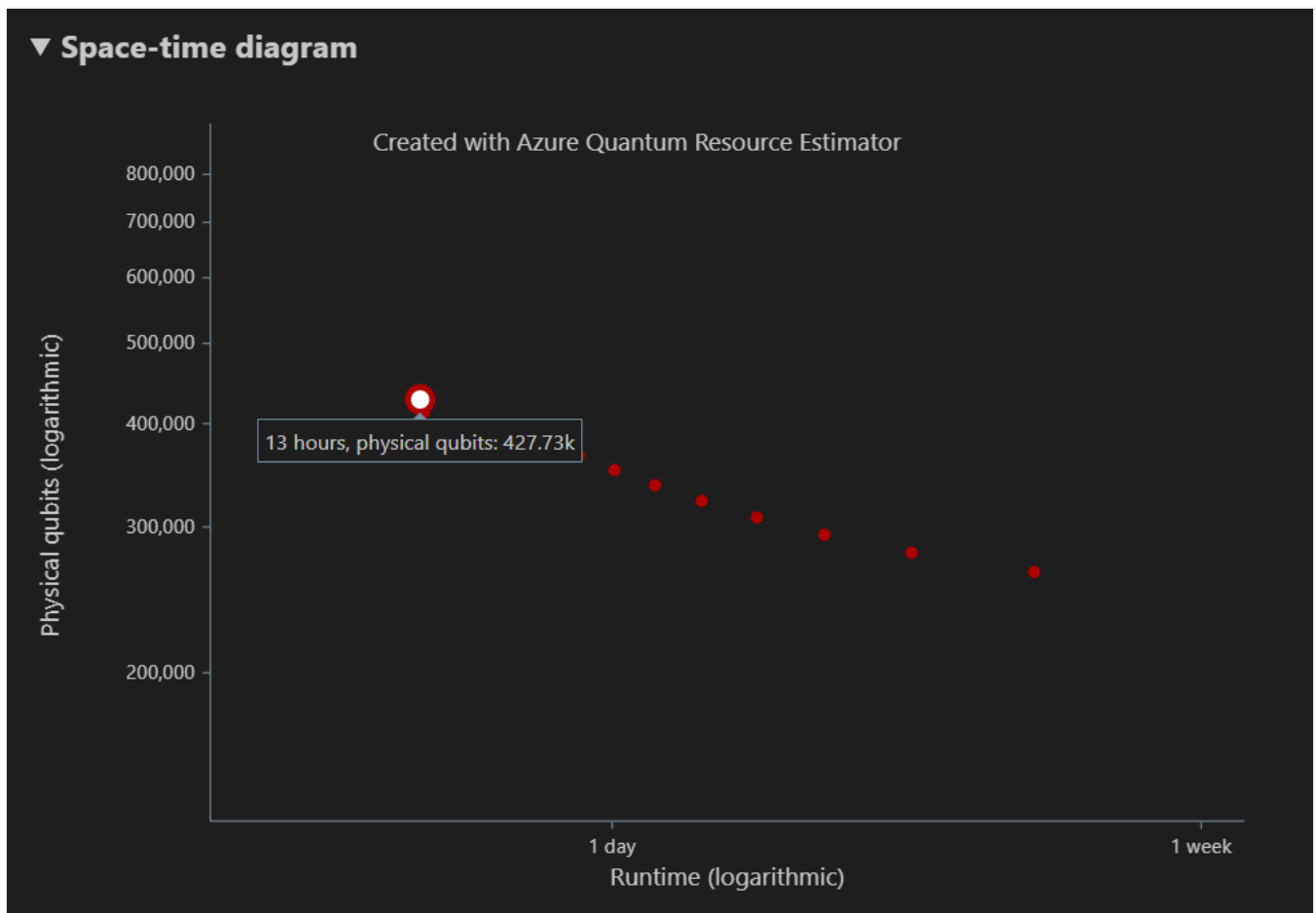
Para ejecutar el diagrama de tiempo espacial en Jupyter Notebook, puede usar el `EstimatesOverview` widget desde el `qsharp-widgets` paquete.

Python

```
import qsharp

from qsharp_widgets import EstimatesOverview

EstimatesOverview(result, colors=["#1f77b4", "#ff7f0e"], runNames=["e4 Surface Code", "e6 Floquet Code"])
```

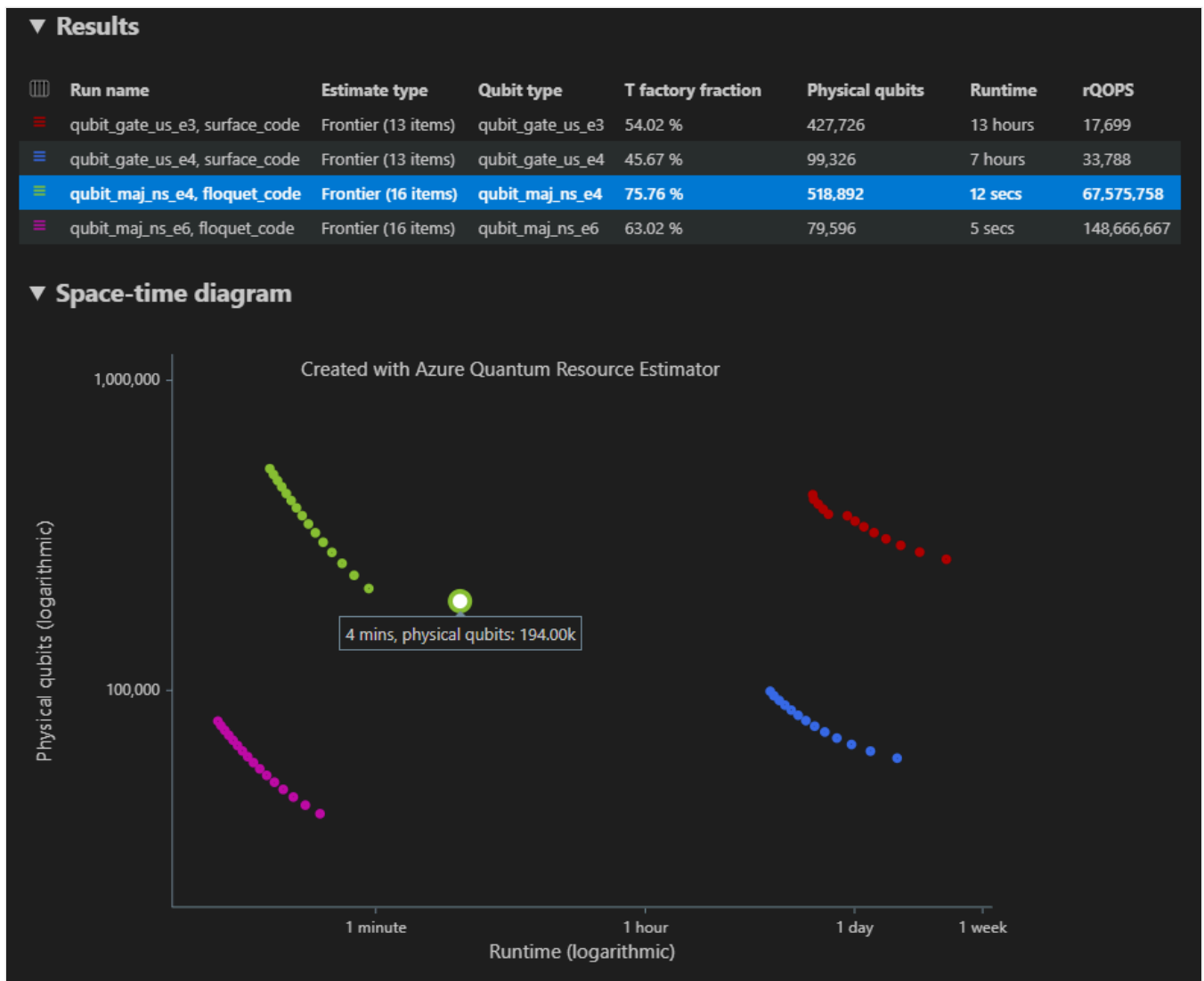


Captura de pantalla que muestra el diagrama de tiempo de cúbit del estimador de recursos.

💡 Sugerencia

Para ver los detalles de la estimación, puede mantener el puntero sobre cada punto del diagrama.

El diagrama de tiempo espaciado es especialmente útil al comparar varias configuraciones de target parámetros para el mismo algoritmo.



Captura de pantalla que muestra el diagrama de tiempo espaciador y la tabla de resultados al ejecutar varias configuraciones de parámetro en el Estimador de recursos.

ⓘ Nota

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Pasos siguientes

- Ejecución de la primera estimación de recursos
- Personalización de estimaciones de recursos en las características de la máquina
- Diferentes formas de ejecutar el estimador de recursos
- Control de programas grandes con el estimador de recursos
- Tutorial: Estimación de los recursos de un problema de química cuántica

Comparación de varias configuraciones de parámetros de target con el estimador de recursos

En este artículo, aprenderá a ejecutar varias configuraciones de parámetros de target al mismo tiempo y compararlas con el estimador de recursos de Azure Quantum .

El estimador de recursos de Azure Quantum permite ejecutar varias configuraciones de [target parámetros](#) como un único trabajo para que no necesite volver a ejecutar varios trabajos en el mismo programa cuántico.

Un trabajo puede estar formado por varios elementos o configuraciones de parámetros target. Algunos escenarios en los que puede que desee ejecutar varios elementos como un solo trabajo:

- Ejecute varios target parámetros con los mismos argumentos de operación en todos los elementos.
- Ejecute varios target parámetros con distintos argumentos de operación en todos los elementos.
- Compare fácilmente varios resultados en un formato tabular.
- Comparar fácilmente varios resultados en un gráfico.

Para obtener información sobre cómo ejecutar el estimador de recursos, consulte [Diferentes formas de usar el estimador](#) de recursos.

Requisitos previos

- La versión más reciente de [Visual Studio Code](#) [↗](#) o abrir [VS Code en la Web](#) [↗](#) .
- La versión más reciente de [Extensión de kit de desarrollo de Quantum](#) [↗](#) . Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).
- Instale la versión más reciente de las [extensiones de Python](#) [↗](#) y [Jupyter](#) [↗](#) para VS Code.
- La biblioteca de Python más reciente `qdk` .

Bash

```
python -m pip install --upgrade qdk
```


Ejecución de varias configuraciones con el estimador de recursos

Puede ejecutar configuraciones múltiples de parámetros de target como un único trabajo en Q# utilizando [Jupyter Notebook en VS Code](#). Puede pasar una lista de parámetros de target al parámetro `params` de la función `qsharp.estimate`.

En el ejemplo siguiente se muestra cómo ejecutar dos configuraciones de target parámetros como un único trabajo. La primera configuración usa los parámetros predeterminados target y la segunda configuración usa el `qubit_maj_ns_e6` parámetro qubit y el `floquet_code` esquema QEC.

En el mismo jupyter Notebook del Q# programa, agregue una nueva celda y ejecute el código siguiente:

Python

```
from qdk import qsharp

result_batch = qsharp.estimate("RunProgram()", params=[
    {}, # Default parameters
    {
        "qubitParams": {
            "name": "qubit_maj_ns_e6"
        },
        "qecScheme": {
            "name": "floquet_code"
        }
    }
])

result_batch.summary_data_frame(labels=["Gate-based ns, 10-3", "Majorana ns, 10-6"])
```

También puede construir una lista de parámetros de estimación target mediante la [EstimatorParams clase](#). En el código siguiente se muestra cómo procesar por lotes seis configuraciones de parámetros de target como un solo trabajo.

Python

```
from qdk import qsharp
from qdk.estimator import EstimatorParams, QubitParams, QECScheme

labels = ["Gate-based  $\mu$ s, 10-3", "Gate-based  $\mu$ s, 10-4", "Gate-based ns, 10-3",
          "Gate-based ns, 10-4", "Majorana ns, 10-4", "Majorana ns, 10-6"]

params = EstimatorParams(num_items=6)
params.error_budget = 0.333
params.items[0].qubit_params.name = QubitParams.GATE_US_E3
params.items[1].qubit_params.name = QubitParams.GATE_US_E4
params.items[2].qubit_params.name = QubitParams.GATE_NS_E3
```

```
params.items[3].qubit_params.name = QubitParams.GATE_NS_E4
params.items[4].qubit_params.name = QubitParams.MAJ_NS_E4
params.items[4].qec_scheme.name = QECScheme.FLOQUET_CODE
params.items[5].qubit_params.name = QubitParams.MAJ_NS_E6
params.items[5].qec_scheme.name = QECScheme.FLOQUET_CODE

qsharp.estimate("RunProgram()", params=params).summary_data_frame(labels=labels)
```

ⓘ Nota:

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Contenido relacionado

- [Descripción de los resultados del estimador de recursos](#)
- [Uso de diferentes SDK e IDE con el estimador de recursos](#)
- [Personalización de estimaciones de recursos en las características de la máquina](#)
- [Tutorial: Estimación de los recursos de un problema de química cuántica](#)

Last updated on 12/11/2025

Uso de estimaciones conocidas con el estimador de recursos

Artículo • 13/01/2025

En este artículo, aprenderá a usar estimaciones calculadas previamente y a optimizar la ejecución del estimador [de recursos de Azure Quantum](#). Si ya conoce algunas estimaciones de una operación, por ejemplo, de un documento publicado, una manera de reducir el tiempo de ejecución está tomando las estimaciones conocidas e incorporarlas al costo general del programa.

Para obtener información sobre cómo ejecutar el estimador de recursos, consulte [Diferentes formas de ejecutar el estimador de recursos](#).

Requisitos previos

- La última versión de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- La versión más reciente de [Extensión de kit de desarrollo de Quantum](#). Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

Si desea usar Python en VS Code, también necesita lo siguiente:

- Instale la versión más reciente de las [extensiones de Python](#) y [Jupyter](#) para VS Code.
- El paquete de Azure Quantum `qsharp` más reciente.

Bash

```
python -m pip install --upgrade qsharp
```

Uso de estimaciones conocidas para una operación

Algunos escenarios en los que es posible que desee realizar estimaciones calculadas previamente:

- Quiere probar un algoritmo nuevo descrito en un documento para comprobar si mejora el rendimiento del programa. Puede tomar estimaciones del documento e

incorporarlas al programa.

- Quiere desarrollar [el programa de arriba abajo](#), es decir, empezar a desarrollar a partir de la función principal y, a continuación, implementar niveles inferiores. Puede usar las estimaciones conocidas en el nivel superior con las estimaciones esperadas para todo el programa. A medida que avanza el proceso de desarrollo, los nuevos componentes comienzan a llamar a las estimaciones conocidas y las estimaciones esperadas se reemplazan por la implementación real. De esta manera, las estimaciones de todo el programa se conocen por adelantado y obtienen más precisión a medida que avanza el desarrollo.

Uso Q#

Puede usar la `AccountForEstimates` Q# operación para pasar estimaciones conocidas al Estimador de recursos.

❗ Nota

La operación especial es una operación `AccountForEstimates` intrínseca para el estimador de recursos. No es compatible con otra ejecución targets.

Por ejemplo, considere la siguiente Q# operación denominada `FactoringFromLogicalCounts` que toma una lista de estimaciones conocidas y una lista de cúbits.

Q#

```
import Microsoft.Quantum.ResourceEstimation.*;

operation FactoringFromLogicalCounts() : Unit {
    use qubits = Qubit[12581];

    AccountForEstimates(
        [TCount(12), RotationCount(12), RotationDepth(12),
         CczCount(3731607428), MeasurementCount(1078154040)],
        PSSPCLayout(), qubits);
}
```

La `AccountForEstimates` operación puede tomar los parámetros siguientes:

[Expandir tabla](#)

Funciones con <code>AccountForEstimates</code>	Descripción
<code>AuxQubitCount(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que el número de cúbits auxiliares es igual a <code>.amount</code>
<code>TCount(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que el número de puertas T es igual a <code>.amount</code>
<code>MeasurementCount(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que el número de medidas es igual a <code>.amount</code>
<code>RotationCount(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que el número de rotaciones es igual a <code>.amount</code>
<code>RotationDepth(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que la profundidad de rotación es igual a <code>.amount</code>
<code>CczCount(amount : Int)</code>	Devuelve una tupla que se puede pasar a la <code>AccountForEstimates</code> operación para especificar que el número de puertas CCZ es igual a <code>.amount</code>
<code>PSSPCLayout()</code>	Indica el diseño de Pauli Computation Secuencial de síntesis paralela (PSSPC). Para obtener más información, vea arXiv:2211.0769 .

ⓘ Nota

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Contenido relacionado

- [Descripción de los resultados del estimador de recursos](#)
- [Diferentes formas de ejecutar el estimador de recursos](#)
- [Personalización de estimaciones de recursos en las características de la máquina](#)
- [Tutorial: Estimación de los recursos de un problema de química cuántica](#)

Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

Optimización de la ejecución de programas grandes con el estimador de recursos

Artículo • 16/01/2025

En este artículo, aprenderá a optimizar el tiempo de ejecución al ejecutar programas grandes Q# con el [estimador](#) de recursos de Azure Quantum.

Para obtener información sobre cómo ejecutar el estimador de recursos, consulte [Diferentes formas de ejecutar el estimador](#) de recursos.

Requisitos previos

- La última versión de [Visual Studio Code](#) o abrir [VS Code en la Web](#).
- La versión más reciente de [Extensión de kit de desarrollo de Quantum](#). Para obtener más información sobre la instalación, consulte [Configuración de la extensión QDK](#).

Si desea usar Python en VS Code, también necesita lo siguiente:

- Instale la versión más reciente de las [extensiones de Python](#) y [Jupyter](#) para VS Code.
- El paquete de Azure Quantum `qsharp` más reciente.

Bash

```
python -m pip install --upgrade qsharp
```

Almacenamiento en caché manual con programas de Q#

Al enviar un trabajo de estimación de recursos al Estimador de recursos, el programa cuántico se evalúa completamente para extraer las estimaciones de recursos. Si desea calcular los recursos de una Q# operación que se invoca muchas veces, por ejemplo, en un bucle con muchas iteraciones, la ejecución del trabajo de estimación de recursos puede tardar mucho tiempo. Una manera de reducir los tiempos de ejecución largos es

ejecutar la operación una vez, calcular y almacenar en caché sus costos y usar los datos en llamadas posteriores. Esta técnica se denomina *almacenamiento en caché manual*.

El estimador target de recursos admite dos Q# funciones para realizar el almacenamiento en caché manual: `BeginEstimateCaching(name: String, variant: Int): Bool` y `EndEstimateCaching(): Unit`. `BeginEstimateCaching` function toma como entradas un `name` que es el nombre único del fragmento de código para el que desea almacenar en caché los costos y un entero `variant` que distingue diferentes variantes de costo para el mismo fragmento.

❗ Nota

Las dos operaciones `BeginEstimateCaching` especiales y `EndEstimateCaching` son operaciones intrínsecas para el estimador de recursos. No son compatibles con otra ejecución targets.

Por ejemplo, supongamos que tiene una Q# operación denominada `ExpensiveOperation` que se llama muchas veces en una iteración. Puede usar el almacenamiento en caché para reducir su tiempo de estimación:

Q#

```
operation ExpensiveOperation(c: Int, b : Bool): Unit {  
    if BeginEstimateCaching("MyNamespace.ExpensiveOperation",  
SingleVariant()) {  
        // Code block to be cached  
        EndEstimateCaching();  
    }  
}
```

Cuando `ExpensiveOperation` se usa repetidamente, `BeginEstimateCaching` se llama a cada vez. Cuando `BeginEstimateCaching` se llama por primera vez, devuelve `true` y comienza la acumulación de datos de costos. Esto hace que el código continúe con la ejecución del fragmento de código costoso. Cuando `EndEstimateCaching` se llama a , los datos de costo se almacenan para el uso futuro y se incorporan al costo total del programa.

Cuando `ExpensiveOperation` se llama a la segunda vez (y posteriormente), el Estimador de recursos busca los datos de costo almacenados (almacenados en caché), lo incorpora al costo total del programa y devuelve `false`. Esto hace que se omita un fragmento de código costoso, por lo que el estimador de recursos ejecuta el programa más rápido.

`EndEstimateCaching` debe colocarse al final de la condición y las regiones incluidas en `BeginEstimateCaching-EndEstimateCaching` se pueden anidar.

`SingleVariant()` indica que los datos de costo recopilados en la primera ejecución se pueden reutilizar en todas las ejecuciones posteriores del fragmento de código. Esto podría no ser siempre el caso. Por ejemplo, si el código tiene un costo diferente para los valores impares e pares de una variable "c", puede proporcionar un `variant` valor:

Q#

```
operation ExpensiveOperation(c: Int, b : Bool): Unit {  
    if BeginEstimateCaching("MyNamespace.ExpensiveOperation", c % 2) {  
        // Some code  
        EndEstimateCaching();  
    }  
}
```

En este caso, la memoria caché es diferente para los valores impares e pares de `c`. En otras palabras, los datos recopilados para los valores pares de `c` solo se reutilizan para los valores pares de `c` y lo mismo se aplica a los valores impares de `c`.

⚠ Nota

Si tiene algún problema al trabajar con el estimador de recursos, consulte la [página Solución de problemas](#) o póngase en contacto con AzureQuantumInfo@microsoft.com.

Contenido relacionado

- [Recuperar los resultados del estimador de recursos](#)
- [Diferentes formas de ejecutar el estimador de recursos](#)
- [Personalizar los parámetros de target del estimador de recursos](#)

Comentarios

¿Le ha resultado útil esta página?

👍 Sí

👎 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

Análisis de protocolos criptográficos con el estimador de recursos

El [estimador de recursos de Azure Quantum](#) está disponible en el [sitio web de Microsoft Quantum](#) para calcular los recursos necesarios para que un equipo cuántico escalado futuro interrumpa un algoritmo de cifrado determinado. Esta herramienta le ayuda a analizar el impacto de la computación cuántica en la seguridad de algunos métodos de cifrado clásicos y prepararse para un futuro seguro para la computación cuántica.

ⓘ Nota:

Para acceder al estimador de recursos en el [sitio web de Microsoft Quantum](#), solo necesita una cuenta microsoft.

El estimador de recursos para el planeamiento seguro para quantum toma un conjunto de cuatro target parámetros como entrada:

- **Algoritmo de cifrado clásico**, que son Rivest-Shamir-Adleman (RSA), Criptografía de curva elíptica (ECC) y Estándar de cifrado avanzado (AES).
- **Fuerza clave**, que son estándar, mejorada y más alta.
- **Tipo de cúbit**, que son topológicos y superconductores.
- **Tasa de errores de cúbits**, que son razonables y optimistas.

Uso del estimador de recursos para la planeación segura para quantum

Para usar el Estimador de recursos para la planeación segura para quantum, vaya al [sitio web de Microsoft Quantum](#).

1. Haga clic en las **flechas** para cambiar entre diferentes selecciones de parámetros de entrada preestablecidos. Vea que los campos se rellenan automáticamente.
2. Haga clic en **Preguntar a Copilot** si desea obtener más información sobre los datos de estimación de recursos.
3. Haga clic en **Descargar** para descargar el trazado del trabajo de estimación de recursos.
4. También **puede personalizar los parámetros** del algoritmo de cifrado. Debe seleccionar *al menos* una opción para cada parámetro de entrada. Puede seleccionar varias combinaciones de parámetros y comparar su seguridad con equipos cuánticos.

- Los resultados se muestran en un **trazado** que muestra el número de cúbits y el tiempo de ejecución que un equipo cuántico con la arquitectura seleccionada necesitaría interrumpir el algoritmo de cifrado. **Mantenga el puntero sobre los puntos del trazado** para ver más información sobre las estimaciones de recursos.
- En cualquier momento, puede **hacer a Copilot una pregunta sobre criptografía, estimación de recursos o computación cuántica**.

Quantum Resource Estimation and Cryptography

Public key cryptography needs to be upgraded

RSA and Elliptic Curve, two common public key algorithms, are selected.

A scaled quantum computer will need 5.8 million physical qubits and 20.8 hours to compromise Elliptic Curve, will require 25 million qubits and 1 day of runtime.

Insight: Current quantum computers do not threaten all cryptography. However, a scaled quantum computer with millions of qubits *will* compromise public key encryption.

Resource Estimation parameters

Encryption algorithm *	Key strength *	Qubit type *	Qubit error rate *	Clear estimates
☒ RSA ☒ Elliptic curve ☐ AES (symmetric)	☒ Standard ☐ Enhanced ☐ Highest	☐ Topological ☒ Superconducting	☒ Reasonable ☐ Optimistic	Ask Copilot

Physical qubits (millions)

Runtime

Encryption Algorithm: RSA encryption
Key strength: Standard
Qubit type: Superconducting
Qubit error rate: Reasonable

(25,172,932 million physical qubits, 1.07 days)

Copilot in Azure Quantum PREVIEW

Hi! Welcome to Copilot in Azure Quantum, here to assist you in your cryptography journey.

Feel free to ask questions about cryptography.

AI generated content may be inaccurate.

[Explain this resource estimation data.](#)

Based on the resource estimation data you provided, there are 2 different parameter combinations. Here's a summary of the results:

1. RSA with standard key strength, superconducting qubits, and a reasonable qubit error rate requires 25,172,932 physical qubits and takes approximately 1.07 days to run.
2. Elliptic Curve with standard key strength, superconducting qubits, and a reasonable qubit error rate requires 5,874,840 physical qubits and takes approximately 20.86 hours to run.

Comparing the two parameter combinations, the Elliptic Curve algorithm requires fewer physical qubits and has a shorter runtime than the RSA algorithm. However, both runtimes are short enough that neither algorithm can be considered secure against quantum computers.

AI generated content may be inaccurate.

[Ask me a question about cryptography or resource estimation](#)

0/500

Responses may be inaccurate. Review all responses for accuracy. Responses may be delayed during periods of high demand, please try again later if you experience issues. [Learn more about Copilot in Azure Quantum.](#) [Read the AI Terms of Use.](#)

Pasos siguientes

- Descripción de los resultados del estimador de recursos
- Diferentes formas de ejecutar el estimador de recursos
- Personalización de estimaciones de recursos en las características de la máquina
- Tutorial: Estimación de los recursos de un problema de química cuántica

Tutorial: Estimación de los recursos de un problema de química cuántica

En este tutorial, calculará los recursos físicos necesarios para calcular la energía de un hamiltoniano a la precisión química de 1 mHa mediante el [estimador](#) de recursos de Azure Quantum.

En este tutorial, aprenderá lo siguiente:

- ✓ Clone un repositorio de ejemplo de GitHub.
- ✓ Use archivos FCIDUMP como parámetros de argumento para aplicaciones de modelado y simulación química.
- ✓ Ejecute la estimación de recursos para un problema a gran escala, que es una muestra de química doble factorizada.

Requisitos previos

- Un entorno de Python con [Python y Pip](#) instalados.
- La versión más reciente de [Visual Studio Code](#) con [el Kit de desarrollo de Quantum](#) y las extensiones de [Python](#) instaladas.
- El paquete y los `qsharp` paquetes de `numpy` Azure Quantum `scipy` más recientes.

Bash

```
python -m pip install --upgrade qsharp numpy scipy
```

Sugerencia

No es necesario tener una cuenta de Azure para ejecutar el estimador de recursos.

Describir el problema

En este tutorial, evaluará las estimaciones de recursos físicos del algoritmo de cúbitización descrito en [Phys. Rev. Research 3, 033055 \(2021\)](#) para calcular la energía de un usuario proporcionado hamiltoniano a la precisión química de 1 mHa.

El algoritmo cuántico que calcula la energía del hamiltoniano se basa en la cúbitización factorizada doble. Los archivos de FCIDUMP (interacción de configuración completa)

proporcionados describen el Hamiltoniano en términos de integrales de uno y dos electrones y están disponibles a través de un URI HTTPS.

El *enfoque de cúbitización* se basa en la estimación de fase cuántica, pero en lugar de construir el estándar $U = \exp(-iH/\alpha)$ de la matriz hamiltoniana H , se toma

Extra close brace or missing open brace, que normalmente se puede implementar con menos recursos. Mediante la *factorización* doble, H se representa de forma compacta a través de una combinación de una elección sensata de orbitales y compresión.

Carga del ejemplo en Visual Studio Code

El código de este tutorial se puede encontrar en el repositorio de ejemplo [de QDK](#), en [estimación/df-química](#). Se recomienda clonar el repositorio en la máquina local para ejecutar el ejemplo.

Para clonar el repositorio, ejecute el siguiente comando desde el terminal:

Bash

```
git clone https://github.com/microsoft/qdk.git
```

Seleccionar y pasar un archivo FCIDUMP

En este ejemplo, el hamiltoniano se describe en términos de integrales de uno y dos electrones en el formato FCIDUMP. Puede elegir uno de los archivos FCIDUMP de la tabla siguiente o seleccionar su propio archivo FCIDUMP disponible en la máquina o en línea a través de un URI HTTPS accesible públicamente.

 Expandir tabla

URI	Nombre de instancia	Descripción
https://aka.ms/fcidump/XVIII-cas4-fb-64e-56o	XVIII-cas4-fb-64e56o	64 electrones, 56 espacio activo orbital de uno de los intermedios estables en el ciclo de fijación de carbono catalizado ruthenium .
https://aka.ms/fcidump/nitrogenase-54e-54o	nitrogenase_54orbital	54 electrones, 54 espacio activo orbital del núcleo activo de la nitrógenoasa .
https://aka.ms/fcidump/fe2s2-10e-40o	fe2s2-10e-40o	10 electrones, 40 espacio activo orbital de [2Fe, 2S] clúster .

URI	Nombre de instancia	Descripción
https://aka.ms/fcidump/polyyne-24e-24o	polyyne-24e-24o	24 electrones, 24 espacio activo orbital de la molécula poliyne.
https://aka.ms/fcidump/n2-10e-8o	n2-10e-8o	10 electrones, 8 espacio activo orbital de nitrógeno desasociado a 3 distancia de Angstrom.

Para pasar el archivo FCIDUMP, debe ejecutar el archivo `chemistry.py` y pasar el nombre de archivo o URI de FCIDUMP como argumento mediante `-f` o `--fcidumpfile`.

Bash

```
usage: chemistry.py [-h] [-f FCIDUMPPFILE]

options:
  -h, --help
  -f FCIDUMPPFILE, --fcidumpfile FCIDUMPPFILE
```

Ejecución del ejemplo de química

1. En Visual Studio Code, abra la carpeta donde ha clonado el repositorio de ejemplo de QDK.
2. Abra un nuevo terminal, **Terminal** -> **Nuevo terminal** y vaya al directorio donde se encuentra la muestra de química cuántica. Por ejemplo, si clona el repositorio de muestra de QDK en su equipo local, la ruta de acceso es `qdk/samples/estimation/df-chemistry`.
3. **Ejecute** el archivo `chemistry.py` y pase el archivo FCIDUMP. Por ejemplo, el siguiente comando descarga el archivo `n2-10e-8o` FCIDUMP en la carpeta de trabajo y ejecuta la estimación de recursos para él.

Bash

```
python chemistry.py -f https://aka.ms/fcidump/n2-10e-8o
```

Después, puede pasar la ruta de acceso al archivo descargado en su lugar.

Bash

```
python chemistry.py -f n2-10e-8o
```

4. El **resultado** de la estimación de recursos se muestra en el terminal. Por ejemplo, la siguiente salida muestra la estimación de recursos del archivo FCIDUMP `n2-10e-8o`.

Resultados

```
Algorithm runtime: 19 mins
Number of physical qubits required: 207.60k
For more detailed resource counts, see file resource_estimate.json
```

ⓘ Nota

Después de ejecutar el `chemistry.py` archivo, se crea un `resource_estimation.json` archivo en la carpeta de trabajo. El `resource_estimation.json` archivo contiene la salida detallada del estimador de recursos, como los parámetros del trabajo, los recuentos físicos, las propiedades de fábrica de T, los recuentos lógicos y las propiedades de cúbit lógicos.

Cambiar target parámetros

1. Abra el archivo `chemistry.py`.
2. Los target parámetros de la estimación de recursos se pueden encontrar en la llamada a `qsharp.estimate` del archivo `chemistry.py`. En el siguiente fragmento de código se muestran los parámetros usados en este tutorial.

Python

```
# Get resource estimates
res = qsharp.estimate(qsharp_string,
                      params={"errorBudget": 0.01,
                              "qubitParams": {"name": "qubit_maj_ns_e6"},
                              "qecScheme": {"name": "floquet_code"}})
```

3. Si desea cambiar los target parámetros, modifique el fragmento de código anterior. Por ejemplo, el siguiente fragmento de código muestra cómo cambiar el presupuesto de errores a 0.333. Para obtener más información, consulte [Personalización de los target parámetros del estimador de recursos](#).

Python

```
# Get resource estimates
res = qsharp.estimate(qsharp_string,
                      params={"errorBudget": 0.333,
```

```
"qubitParams": {"name": "qubit_maj_ns_e6"},  
"qecScheme": {"name": "floquet_code"}})
```

¿Por qué son importantes las aplicaciones de química de la computación cuántica?

Este tutorial representa un primer paso para integrar la estimación de recursos de soluciones cuánticas a problemas de estructura electrónica. Una de las aplicaciones más importantes de los equipos cuánticos escalados es resolver problemas de química cuántica. La simulación de sistemas mecánicos cuánticos complejos tiene el potencial de desbloquear avances en áreas como la captura de carbono, la inseguridad alimentaria y el diseño de mejores combustibles y materiales.

Por ejemplo, uno de los archivos FCIDUMP proporcionados en esta muestra, `nitrogenase_54orbital`, describe la enzima nitrogenasa. Si podría simular con precisión cómo funciona esta enzima en un nivel cuántico, podría ayudarnos a comprender cómo producirla a escala. Podrías reemplazar el proceso muy intensivo de energía que se utiliza para producir suficiente fertilizante para alimentar el planeta. Esto tiene el potencial de reducir la huella de carbono mundial y también para ayudar a abordar las preocupaciones relativas a la inseguridad alimentaria en una población creciente.

Si desea profundizar en sus conocimientos, estos son algunos experimentos que puede probar:

- Calcule algunos archivos FCIDUMP personalizados.
- Modifique las suposiciones en el target equipo cuántico proporcionando parámetros de cúbit personalizados.

Contenido relacionado

- El tutorial [Implementación del algoritmo](#) de búsqueda de Grover muestra cómo escribir un programa de Q# que usa el algoritmo de búsqueda de Grover para resolver un problema de coloración de grafos.
- En el tutorial [Escritura y simulación de programas de nivel de cúbit en Q#](#) se explora cómo escribir un programa de Q# que dirija directamente los cúbits específicos.
- En el tutorial [Explorar el entrelazamiento cuántico con Q#](#) se muestra cómo operar en cúbits con Q# para cambiar su estado y se muestran los efectos de la superposición y el entrelazamiento.
- [Quantum Katas](#) [↗](#) son tutoriales autodirigidos y ejercicios de programación basados en Jupyter Notebook destinados a enseñar los elementos de la computación cuántica y la programación de Q# al mismo tiempo.

Simulador cuántico disperso

El simulador disperso es el simulador local predeterminado para entornos de desarrollo de Azure Quantum y utiliza una representación dispersa de vectores de estado cuántico. Esta característica permite que el simulador disperso minimice la superficie de memoria utilizada para representar estados cuánticos, lo que permite simulaciones en un mayor número de cúbits. El simulador disperso es eficaz para representar estados cuánticos dispersos en la base computacional, es decir, aquellos estados cuánticos para los que la mayoría de los coeficientes de amplitud son cero en la base computacional. Por lo tanto, el simulador disperso permite a los usuarios explorar aplicaciones más grandes que lo que se puede representar mediante un simulador de estado completo que perderá memoria y tiempo en un número exponencialmente grande de amplitudes cero.

Para más información sobre el simulador disperso, consulte [Jaques y Häner \(arXiv:2105.01533\)](#).

Llamada al simulador disperso

El simulador disperso es el simulador local predeterminado en Visual Studio Code con la extensión de Azure Quantum Development Kit instalada.

 Expandir tabla

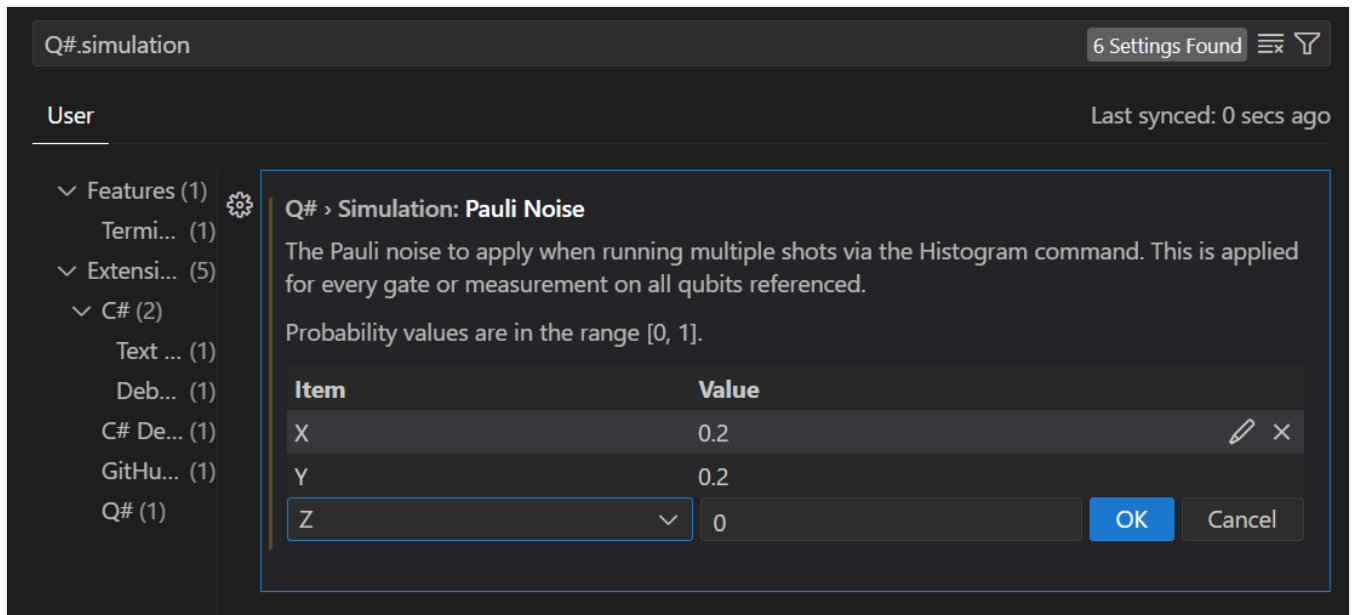
Escenario	Método
En un Q# programa de VS Code	Seleccione Ejecutar Q# archivo.
En una celda del cuaderno de Python	<pre>result=qsharp.eval("Program_Entry_Operation()") o result=qsharp.run("Program_Entry_Operation()", shots=##)</pre>
En una celda del <code>%%qsharp</code> cuaderno	<code>Program_Entry_Operation()</code>

Adición de ruido de Pauli al simulador de matrices dispersas

El simulador disperso admite la adición de ruido de Pauli a la simulación. Esta característica permite simular los efectos del ruido en las operaciones cuánticas y las medidas. El modelo de ruido se especifica mediante un diccionario de probabilidades de ruido de Pauli, donde las claves son los operadores `X` de Pauli, `Y`, y `Z` los valores son las probabilidades de aplicar el operador Pauli correspondiente. El modelo de ruido se puede usar en Q# programas, programas de Python o configurado en la configuración de VS Code.

Adición de ruido de Pauli mediante la configuración de VS Code

El ruido de Pauli se puede configurar globalmente en Visual Studio Code a través de la opción de configuración de usuario "Simulation:Pauli Noise".



La configuración se aplica a los resultados del histograma para todos los Q# programas que se ejecutan en Visual Studio Code, así como a todas las compuertas, medidas y cúbits a los que se hace referencia en el programa. La configuración es idéntica al uso de la `ConfigurePauliNoise()` función .

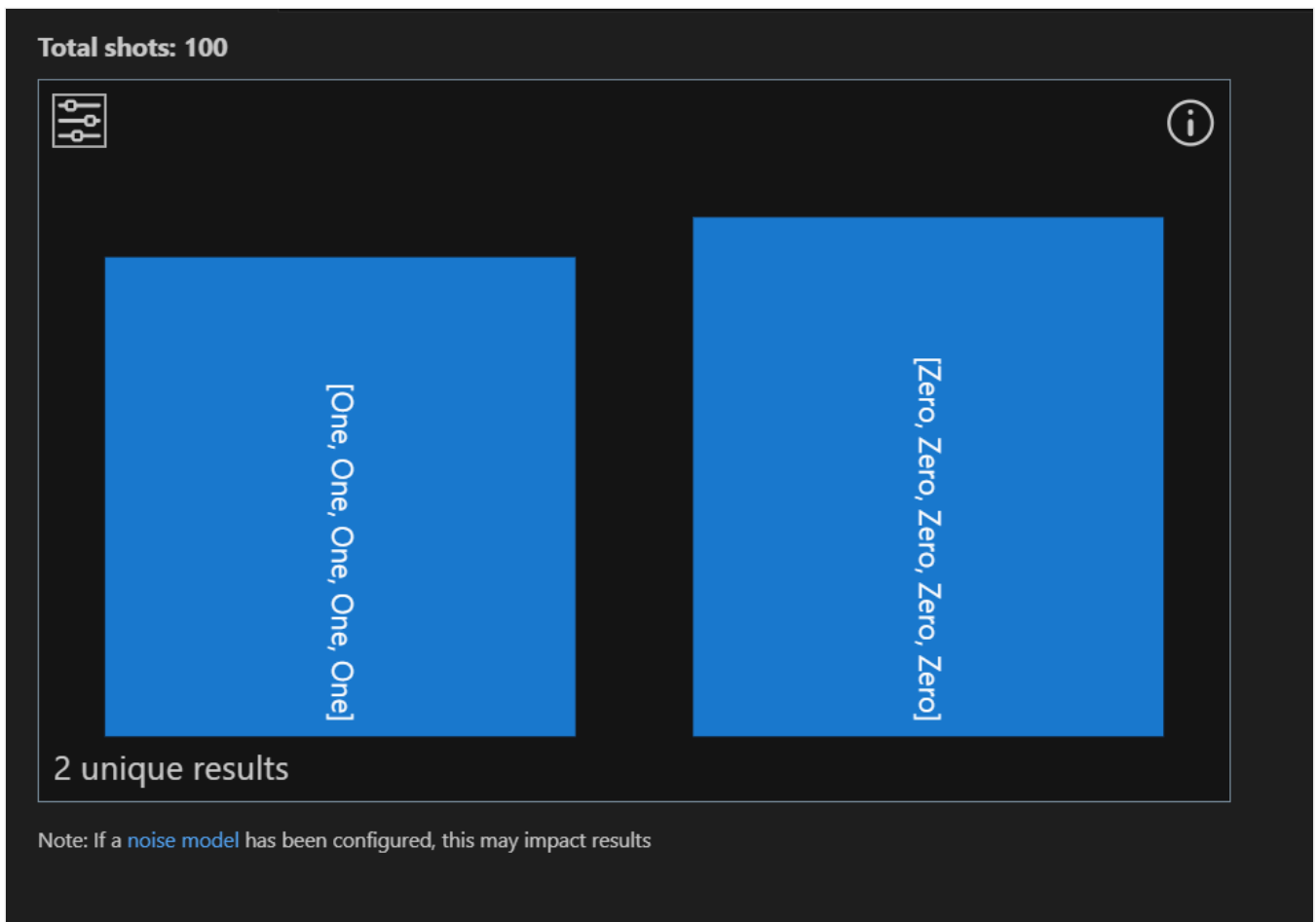
Ejecutar un histograma del siguiente programa de ejemplo de GHZ sin ruido configurado devolvería $|00000\rangle$ aproximadamente la mitad del tiempo y $|11111\rangle$ la otra mitad.

Q#

```
import Std.Diagnostics.*;
import Std.Measurement.*;

operation Main() : Result []{
    let num = 5;
    return GHZSample(num);
}

operation GHZSample(n: Int) : Result[] {
    use qs = Qubit[n];
    H(qs[0]);
    ApplyCNOTChain(qs);
    let results = MeasureEachZ(qs);
    ResetAll(qs);
    return results;
}
```



Sin embargo, agregar un 1% de ruido de volteo de bits muestra el estado empezando a difundirse, y con un 25% de ruido, el estado se vuelve indistinguible del ruido.



Adición de ruido de Pauli a Q# programas

Puede usar la `ConfigurePauliNoise()` función para establecer o modificar el modelo de ruido para programas individuales Q#. Con `ConfigurePauliNoise()`, puede controlar de forma granular dónde se aplica ruido en sus programas de Q#.

ⓘ Nota

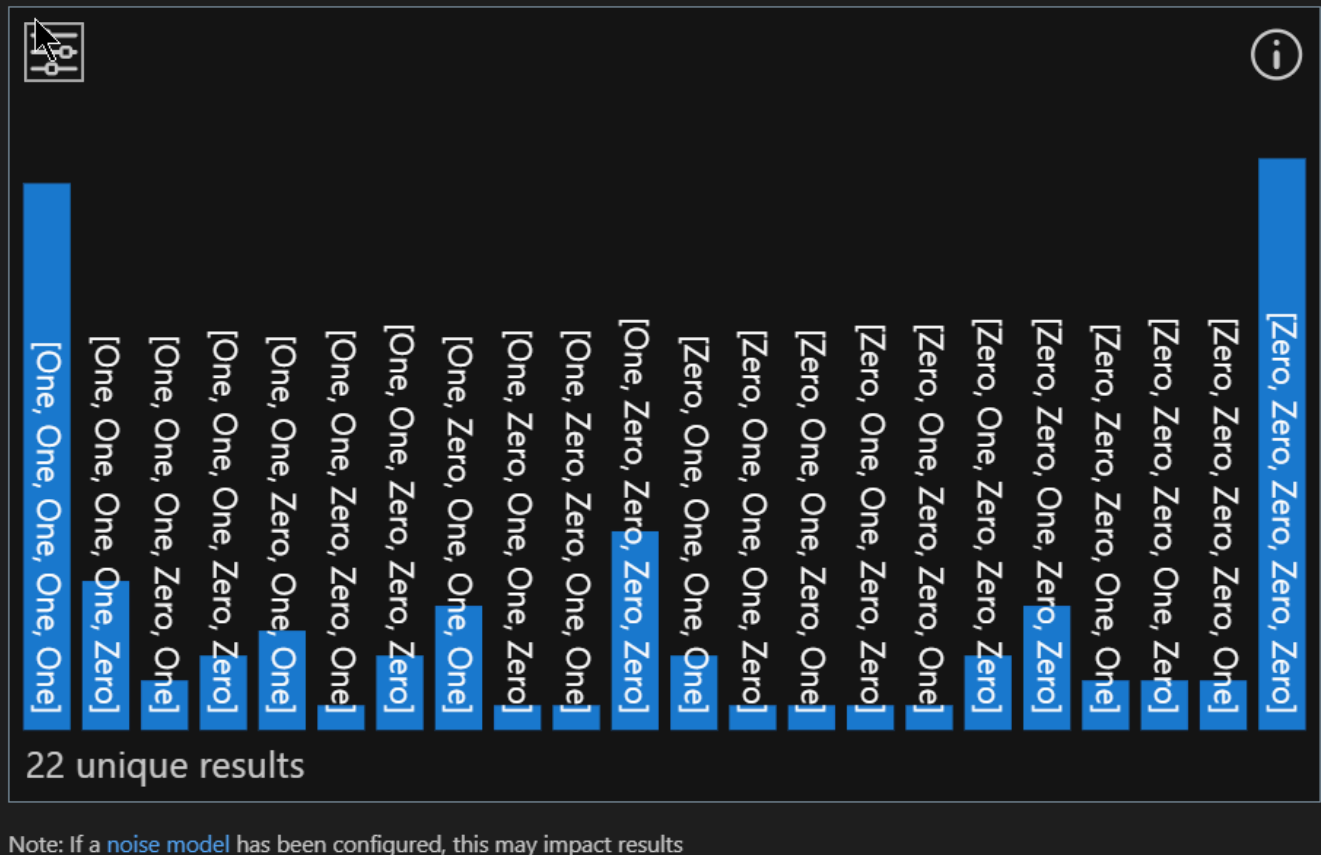
Si configura ruido en la configuración de VS Code, se aplicará a todos los Q# programas. Si configura ruido en un Q# programa mediante `ConfigurePauliNoise()`, omitirá cualquier configuración de VS Code durante la ejecución de ese programa.

Por ejemplo, en el programa anterior puede agregar ruido inmediatamente después de la asignación de cúbits:

Q#

```
...  
operation GHZSample(n: Int) : Result[] {  
    use qs = Qubit[n];  
    ConfigurePauliNoise(0.05, 0.0, 0.0); // 5% bit-flip noise applies to all  
operations  
    H(qs[0]);  
    ...  
}
```

Total shots: 100



o justo antes de la operación de medición.

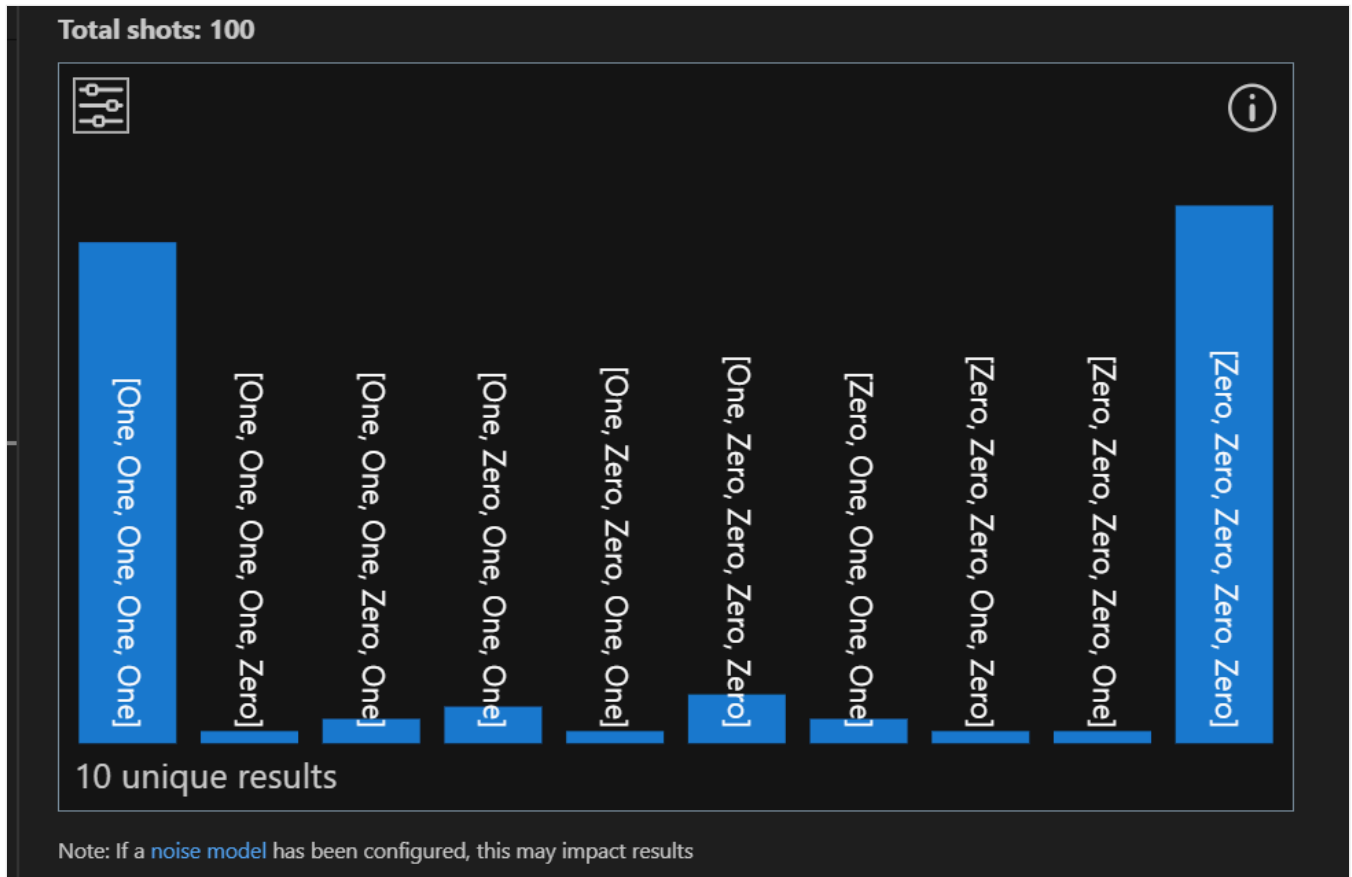
Q#

```
...  
use qs = Qubit[n];  
H(qs[0]);
```

```

ApplyCNOTChain(qs);
ConfigurePauliNoise(0.05, 0.0, 0.0); // noise applies only to measurement
operation
let results = MeasureEachZ(qs);
...

```



También puede usar `ConfigurePauliNoise()` para restablecer o borrar configuraciones de ruido. En este ejemplo, la configuración de ruido se establece después de la asignación de cúbits y, a continuación, se borra inmediatamente después de la operación Hadamard, de modo que el ruido solo se aplica a la `H()` operación.

```

Q#
...
operation GHZSample(n: Int) : Result[] {
    use qs = Qubit[n];
    ConfigurePauliNoise(0.05, 0.0, 0.0); // noise applies to H() operation
    H(qs[0]);
    ConfigurePauliNoise(0.0, 0.0, 0.0); // clear noise settings
...

```

Las siguientes funciones están disponibles en la `Qdk.Std.Diagnostics` biblioteca para configurar el ruido en Q# programas:

Función	Description	Example
<code>ConfigurePauliNoise()</code>	Configura el ruido de Pauli para una ejecución del simulador, con parámetros que representan probabilidades para las puertas X, Y y Z. La configuración de ruido se aplica a todas las puertas, mediciones y qubits posteriores en un programa Q#. Omite cualquier configuración de ruido en VS Code y se puede restablecer mediante llamadas posteriores <code>ConfigurePauliNoise()</code> .	<code>ConfigurePauliNoise(0.1, 0.0, 0.5)</code> o <code>ConfigurePauliNoise(BitFlipNoise(0.1))</code>
<code>BitFlipNoise()</code>	Configura el ruido para que sea la puerta X solo con una probabilidad específica. La configuración de ruido se aplica a todas las compuertas, medidas y cúbits posteriores de un Q# programa.	10% ruido de volteo de bits: <code>ConfigurePauliNoise(BitFlipNoise(0.1))</code> ≡ <code>ConfigurePauliNoise(0.1, 0.0, 0.0)</code>
<code>PhaseFlipNoise()</code>	Configura el ruido para que sea la puerta Z solo con la probabilidad especificada. La configuración de ruido se aplica a todas las compuertas, mediciones y cúbits posteriores en un programa Q#.	10% ruido de volteo por fase: <code>ConfigurePauliNoise(PhaseFlipNoise(0.1))</code> ≡ <code>ConfigurePauliNoise(0.0, 0.0, 0.1)</code>
<code>DepolarizingNoise()</code>	Configura el ruido para que sea una puerta X, Y o Z con probabilidades iguales.	6% ruido despolarizante <code>ConfigurePauliNoise(DepolarizingNoise(0.06))</code> ≡ <code>ConfigurePauliNoise(0.2, 0.2, 0.2)</code>
<code>NoNoise()</code>	Configura el modelo de ruido para no incluir ruido.	<code>ConfigurePauliNoise(NoNoise())</code> ≡ <code>ConfigurePauliNoise(0.0, 0.0, 0.0)</code>
<code>ApplyIdleNoise</code>	Aplica el ruido configurado a un solo cúbit durante la simulación.	... <code>use q = Qubit[2];</code> <code>ConfigurePauliNoise(0.1, 0.0, 0.0);</code>

Función	Description	Example
		<pre>ApplyIdleNoise(q[0]);</pre> <pre>...</pre>

Adición de ruido de Pauli a programas de Python o cuadernos Jupyter

La configuración de ruido de Pauli está disponible con el paquete `qsharp` de Python, y la capacidad de histograma con el paquete `qsharp_widgets`. El ruido se agrega como parámetro al `qsharp.run` método .

- `qsharp.BitFlipNoise`
- `qsharp.PhaseFlipNoise`
- `qsharp.DepolarizingNoise`

En el ejemplo siguiente se muestra el efecto del ruido de despolarización del 10% en una medida de estado de Bell.

Python

```
import qsharp
import qsharp_widgets
```

Python

```
%%qsharp

operation BellPair() : Result[] {
    use q = Qubit[2];
    H(q[0]);
    CNOT(q[0], q[1]);
    MResetEachZ(q)
}
```

Python

```
results = qsharp.run("BellPair()", 20, noise=qsharp.DepolarizingNoise(0.1))
results
```

El ruido arbitrario de Pauli se puede agregar al modelo de ruido especificando las probabilidades de cada operador de Pauli. Vamos a usar el programa de ejemplo de GHz anterior:

Python

```
%%qsharp
```

```
operation GHZSample() : Result[] {  
    use q = Qubit[5];  
    H(q[0]);  
    ApplyCNOTChain(q);  
    MResetEachZ(q)  
}
```

Esta ejecución del programa aplica ruido con una probabilidad del 20% (inversión de bits la mitad del tiempo e inversión de fase la mitad del tiempo),

Python

```
result = qsharp.run("GHZSample()", 20, noise=(0.1, 0.0, 0.1))  
display(qsharp_widgets.Histogram(result))
```

y esta ejecución aplica Pauli-Y ruido con 10% probabilidad.

Python

```
result = qsharp.run("GHZSample()", 20, noise=(0.0, 0.1, 0.0))  
display(qsharp_widgets.Histogram(result))
```

Last updated on 17/10/2025

Simuladores cuánticos de back-end de proveedores cuánticos

En este artículo se describen los simuladores de back-end disponibles en los proveedores cuánticos. Estos simuladores están disponibles para todos los usuarios de Azure Quantum y son una excelente manera de probar Q# los programas antes de ejecutarlos en un equipo cuántico real.

IonQ

IonQ proporciona un simulador idealizado acelerado por GPU que admite hasta 29 cúbits, usando el mismo conjunto de puertas que IonQ proporciona en su hardware cuántico. El simulador es un excelente lugar para preproducción de trabajos antes de ejecutarlos en un equipo cuántico real.

- Tipo de trabajo: `Simulation`
- Formato de datos: `ionq.circuit.v1`
- Id. de destino: `ionq.simulator`
- Perfil de ejecución de destino: base qir (representación intermedia cuántica)

Para obtener más información, consulte la página [del proveedor de IonQ](#).

PASQAL

El emulador de Emu-TN de PASQAL simula la evolución temporal de un estado cuántico mediante la ecuación de Schrödinger correspondiente a las acciones que realizan los láseres.

El emulador de Emu-TN se ejecuta en un clúster de nodos DE NVIDIA DGX, cada uno equipado con GPU nvidia A100, lo que permite la emulación de los procesadores cuánticos de PASQAL. Es una herramienta clave para crear prototipos y validar programas cuánticos antes de ejecutarlos en la QPU (unidad de procesamiento cuántico). Se pueden emular hasta 100 cúbits en matrices 2D para desarrollar aplicaciones industriales y para avanzar en el descubrimiento científico.

- Tipo de trabajo: `Simulation`
- Formato de datos: `application/json`
- Id. de destino: `pasqal.sim.emu-tn`
- Perfil de ejecución de destino: N/A

Para obtener más información, consulte la página [del proveedor PASQAL](#).

Quantinuum

Quantinuum proporciona dos herramientas del emulador:

Comprobadores de sintaxis : estas herramientas comprueban la sintaxis adecuada, la finalización de la compilación y la compatibilidad de la máquina, con el mismo compilador que el equipo cuántico que tienen como destino. Hay comprobadores de sintaxis para ambas máquinas del modelo de sistema H2.

- Tipo de trabajo: `Simulation`
- Formatos de datos: `honeywell.openqasm.v1`, `honeywell.qir.v1`
- Id. de destino:
 - Comprobador de sintaxis H2-1: `quantinuum.sim.h2-1sc`
 - Corrector de sintaxis H2-2: `quantinuum.sim.h2-2sc`
- Perfil de ejecución de destino: QIR Adaptive RI
- Precios: Gratis (\$0)

Emuladores : estas herramientas contienen un modelo físico detallado y un modelo de ruido realista del hardware real del modelo de sistema H2. Hay emuladores para ambas máquinas H2 junto con un emulador de Quantinuum basado en la nube.

- Tipo de trabajo: `Simulation`
- Formato de datos: `honeywell.openqasm.v1`, `honeywell.qir.v1`
- Id. de destino:
 - Emulador H2-1: `quantinuum.sim.h2-1e`
 - Emulador H2-2: `quantinuum.sim.h2-2e`
- Perfil de ejecución de destino: QIR Adaptive RI

El emulador de Quantinuum es un emulador basado en la serie H del modelo del sistema disponible de forma gratuita en la página [Código con Azure Quantum](#) [↗] . Para obtener más información, consulte la página [Quantinuum Emulator \(Emulador de Quantinuum\)](#) .

Para obtener más información sobre todos los emuladores de Quantinuum, consulte la página [del proveedor Quantinuum](#) .

Rigetti

Rigetti proporciona su máquina virtual cuántica (QVM), un simulador de código abierto para Quil. El destino de QVM acepta un programa Quil como texto y ejecuta ese programa en QVM hospedado en la nube y devuelve resultados simulados.

- Tipo de trabajo: `Simulation`

- Formatos de datos: `rigetti.quil.v1`, `rigetti.qir.v1`
- Id. de destino: `rigetti.sim.qvm`
- Perfil de ejecución de destino: base de QIR
- Precios: Gratis (\$0)

Consulte la [página del proveedor Rigetti](#) para más información.

Last updated on 16/10/2025

Proveedores de computación cuántica en Azure Quantum

Azure Quantum ofrece diversas soluciones cuánticas, como dispositivos de hardware cuánticos y simuladores cuánticos, que se pueden usar para ejecutar programas de computación cuántica. En este artículo se enumeran los proveedores a los que puede acceder con Azure Quantum y se proporciona una descripción de lo que ofrece cada proveedor.

 Expandir tabla

Provider	Description
 IONQ	Los equipos cuánticos basados en puertas de iones atrapados de IonQ son universales y dinámicamente reconfigurables en software, proporcionando hasta 25 qubits en las QPU de IonQ Aria y 36 qubits en las QPU IonQ Forte 1 y Forte Enterprise 1. Todos los cúbits están totalmente conectados, lo que significa que puede ejecutar una puerta de dos cúbits entre cualquier par. La implementación de operaciones de puerta cuántica se realiza manipulando iones Ytterbium con pulsos láser. IonQ proporciona un simulador cuántico acelerado por GPU que admite hasta 36 cúbits, mediante el mismo conjunto de puertas que IonQ proporciona en su hardware cuántico. Consulte la página del proveedor IonQ para más información.
 PASQAL	Los procesadores cuánticos neutros basados en átomos de PASQAL que funcionan a temperatura ambiente tienen tiempos de coherencia largos e impresionante conectividad de cúbits. Las operaciones se realizan con pinzas ópticas, usando luz láser para manipular registros cuánticos 1D y 2D con hasta cien cúbits. Para obtener más información, consulte la página del proveedor PASQAL .
 QUANTINUUM	Los equipos cuánticos de iones atrapados de Quantinuum tienen cúbits totalmente conectados de alta fidelidad y reutilización de cúbits. Las operaciones cuánticas son puertas basadas en láser con tasas de error bajas y tienen la capacidad de tomar medidas en el circuito medio. La generación de hardware System Model H2, impulsado por Honeywell, usa una arquitectura Quantum Charge-Coupled Device (QCCD). Quantinuum proporciona herramientas de emulación, dos emuladores del modelo de sistema H2, que contienen modelos físicos detallados y modelos de ruido del hardware cuántico real. Consulte la página del proveedor Quantinuum para más información.
 rigetti	Los sistemas de Rigetti están alimentados por procesadores cuánticos superconductores basados en cúbits. Ofrecen tiempos de puerta rápidos, lógica condicional de baja latencia y tiempos de ejecución rápidos del programa. En el nivel de chip, cada cúbit superconductor consta de una inductancia josephson no lineal en paralelo con un condensador de pérdida ultra baja para crear una estructura resonante en el rango de 3 a 6 GHz. Los cúbits se acoplan a un resonador de superconductor lineal para lectura. La combinación del cúbit, el resonador de lectura lineal y el cableado asociado proporciona un elemento de circuito cuántico de uso general capaz de codificar, manipular y leer información cuántica de forma confiable. Los procesadores de Rigetti usan matrices de cúbits acoplados entre sí con capacitaciones en chip. Las operaciones lógicas de un solo cúbit y multibit se implementan a

Provider	Description
	través de la aplicación de pulsos de microondas o DC. Consulte la página del proveedor Rigetti para más información.

ⓘ Importante

Los dispositivos de hardware cuántico siguen siendo una tecnología emergente. Estos dispositivos tienen algunas limitaciones y requisitos en cuanto a los programas cuánticos que se ejecutan en ellos. Para más información, consulte los [target tipos de perfil en Azure Quantum](#).

Para obtener información sobre qué proveedores de computación cuántica están disponibles en su región, consulte [Disponibilidad global de proveedores de Azure Quantum](#).

Disponibilidad de cúbits para proveedores de computación cuántica

Los asociados de proveedor de Microsoft ofrecen una amplia gama de disponibilidad de cúbits para sus procesadores de hardware y simuladores.

 Expandir tabla

Nombre de destino	Número de cúbits
Simulador cuántico de IonQ	29 cúbits
IonQ Aria 1	25 cúbits
IonQ Forte 1	36 qubits
IonQ Forte Enterprise 1	36 qubits
PASQAL Emu-TN	100 qubits
PASQAL Fresnel1	100 qubits
Comprobador de sintaxis quantinuum H2-1	32 qubits
Comprobador de sintaxis quantinuum H2-2	32 qubits
Emulador Quantinuum H2-1	20 qubits
Emulador Quantinuum H2-2	20 qubits
Quantinuum H2-1	20 qubits

Nombre de destino	Número de cúbits
Quantinuum H2-2	32 qubits
Máquina virtual cuántica rigetti (QVM)	30 qubits
Rigetti Ankaa-3	84 qubits
Rigetti Cepheus-1-36Q	36 qubits

Próximamente en Azure Quantum

Azure Quantum es una plataforma para la innovación. Dado que los asociados de hardware cuántico del ecosistema de Azure Quantum no dejan de crecer, puede explorar estas futuras soluciones de hardware cuántico.

 Expandir tabla

Provider	Description
	Los circuitos superconductores de pila completa de Quantum Circuits, Inc incluyen comentarios en tiempo real que permiten la corrección de errores y puertas de entrelazamiento independientes de la codificación. Puede llevar a cabo el registro previo hoy en la versión preliminar privada de QCI de Azure Quantum.

Last updated on 15/10/2025

Diferentes tipos de target perfiles en Azure Quantum

Los dispositivos cuánticos siguen siendo una tecnología emergente y, desafortunadamente, no todos ellos pueden ejecutar todos los programas de Q#. Por lo tanto, debe tener en cuenta algunas restricciones mientras desarrolla programas cuánticos. Los target tipos de perfil definen las capacidades de los dispositivos cuánticos que utilizas target con tus programas de Q#. El Kit de desarrollo de Quantum (QDK) tiene un conjunto de diferentes target tipos de perfil, que admiten todas las funcionalidades de los dispositivos cuánticos actuales que están disponibles en Azure Quantum.

En este artículo se describen los distintos tipos de perfiles de target Azure Quantum, sus limitaciones y cómo configurarlos en el QDK.

Perfiles de destino y sus limitaciones

Actualmente, Azure Quantum y QDK administran diferentes target perfiles, en función de su capacidad de ejecutar [programas de representación intermedia cuántica \(QIR\)](#).

- **Unrestricted:** Este perfil puede ejecutar cualquier QIR programa y, por tanto, cualquier programa de Q#, dentro de los límites de memoria para simuladores o el número de cúbits para equipos cuánticos físicos.
- **Base:** Este perfil puede ejecutar cualquier programa de Q# que no requiera el uso de los resultados de las mediciones de cúbits para controlar el flujo del programa. Dentro de un programa de Q# destinado a este tipo de QPU, los valores de tipo `Result` no admiten la comparación de igualdad.
- **Adaptive RI:** Este perfil tiene una capacidad limitada de usar los resultados de las mediciones de cúbits para controlar el flujo del programa. Dentro de un programa de Q# destinado a este tipo de QPU, se pueden comparar valores de tipo `Result` como parte de condiciones dentro `if` de las sentencias en las operaciones, lo que permite la medición a mitad del circuito.
- **Adaptive RIF:** Este perfil tiene las mismas funcionalidades que el Adaptive RI perfil, pero también admite operaciones de punto flotante.

Creación y ejecución de programas para perfil Unrestrictedtarget

Unrestricted target Los perfiles pueden ejecutar todo el programa de Q#, por lo que puede escribir código de Q# sin necesidad de tener en cuenta las restricciones de funcionalidad.

Azure Quantum no proporciona ningún dispositivo targets real que admita este perfil. Sin embargo, puede ejecutar Unrestricted programas de Q# que tengan este perfil en los simuladores que se proporcionan con el QDK.

Configuración del Unrestrictedtarget perfil

Si no establece manualmente su perfil QIRtarget, entonces el compilador establece automáticamente el perfil target. El compilador elige el perfil más restrictivo que todavía permite que el programa se ejecute en el dispositivo target de Azure Quantum que elija.

Para establecer manualmente el QIRtarget perfil en **Sin restricciones**, elija una de las siguientes opciones:

- Si configura un proyecto de Q#, agregue el siguiente comando al archivo del `qsharp.json` proyecto:

JSON

```
{
  "targetProfile": "unrestricted"
}
```

- Si trabaja en un `.qs` archivo que no forma parte de un proyecto de Q#, establezca el perfil directamente en el target código de Q#. Para ello, incluya `@EntryPoint(Unrestricted)` justo antes de la operación de punto de entrada en el programa, incluso cuando esa operación sea la predeterminada `Main`.
- En Python, llame al método `qsharp.init` para establecer el perfil target.

Python

```
qsharp.init(target_profile=qsharp.TargetProfile.Unrestricted)
```

Creación y ejecución de programas para perfil Basetarget

Base target Los perfiles pueden ejecutar una amplia variedad de aplicaciones de Q#, con la restricción que no pueden usar los resultados de las mediciones de cúbits para controlar el flujo del programa. En concreto, los valores de tipo `Result` no admiten la comparación de igualdad.

Por ejemplo, no puede ejecutar la siguiente `FlipQubitOnZero` operación en `:Basetarget`

Q#

```
@EntryPoint(Base)
operation FlipQubitOnZero() : Unit {
    use q = Qubit();
    if M(q) == Zero {
        X(q);
    }
}
```

La operación `FlipQubitOnZero` falla cuando se ejecuta este código en un `Basetarget` porque el dispositivo target no puede usar el resultado de una medición de cúbit para llevar a cabo la lógica condicional mientras se ejecuta el algoritmo cuántico. Si planea ejecutar algoritmos en un `Basetarget` dispositivo, asegúrese de que el código no contiene ningún `if` bloque para la [bifurcación condicional](#) que dependa de cúbits medidos para evaluar una condición lógica.

Configuración del Basetarget perfil

Para establecer manualmente el `QIRtarget` perfil en **Base**, elija una de las siguientes opciones:

- Si configura un proyecto de Q#, agregue el siguiente comando al archivo del `qsharp.json` proyecto:

JSON

```
{
  "targetProfile": "base"
}
```

- Si trabaja en un `.qs` archivo que no forma parte de un proyecto de Q#, establezca el perfil directamente en el target código de Q#. Para ello, incluya `@EntryPoint(Base)` justo antes de la operación de punto de entrada en el programa, incluso cuando esa operación sea la predeterminada `Main`.
- En Python, llame el método `qsharp.init` para establecer el perfil target.

Python

```
qsharp.init(target_profile=qsharp.TargetProfile.Base)
```

Soportado targets para Basetarget el perfil

Por ahora, Azure Quantum proporciona acceso a lo siguiente `Basetargets`:

Provider	Simulador	QPU
IonQ	<code>ionq.simulator</code>	<code>ionq.qpu.*</code>
Rigetti	<code>rigetti.sim.*</code>	<code>rigetti.qpu.*</code>

Para más información sobre estos proveedores en Azure Quantum, consulte [Proveedor de IonQ](#) y [Proveedor de Rigetti](#).

Creación y ejecución de programas para perfil Adaptive Rltarget

Adaptive RI target perfiles pueden ejecutar una variedad más ancha de aplicaciones de Q# que Base perfiles, pero todavía tienen algunas limitaciones. A diferencia de los perfiles Basetarget, Adaptive Rltargets admiten medidas de circuito medio.

Con las medidas del circuito medio, puede medir de forma selectiva los cúbits en cualquier punto del programa cuántico, no solo el final. A continuación, puede usar los resultados de medición para otras operaciones del programa, como la bifurcación condicional con `if` bloques. Los qubits que mides a mitad del circuito colapsan a un estado clásico (cero o uno), pero los qubits no medidos permanecen en su estado cuántico.

Cuando se mide un cúbit en Q#, se devuelve un valor de tipo `Result`. Si desea usar este resultado en una instrucción condicional, debe comparar directamente en la instrucción condicional. Los bloques condicionales correspondientes no pueden contener instrucciones `return` ni `set`.

Por ejemplo, se permite el siguiente código de Q# en :Adaptive Rltarget

Q#

```
@EntryPoint(Adaptive_RI)
operation MeasureQubit(q : Qubit) : Result {
    return M(q);
}

operation SetToZero(q : Qubit) : Unit {
    if MeasureQubit(q) == One { X(q); }
}
```

Configuración del Adaptive Rltarget perfil

Para establecer manualmente el QIRtarget perfil en **Adaptive RI**, elija una de las siguientes opciones:

- Si configura un proyecto de Q#, agregue el siguiente comando al archivo del `qsharp.json` proyecto:

JSON

```
{  
  "targetProfile": "adaptive_ri"  
}
```

- Si trabaja en un `.qs` archivo que no forma parte de un proyecto de Q#, establezca el perfil directamente en el target código de Q#. Para ello, incluya `@EntryPoint(Adaptive_RI)` justo antes de la operación de punto de entrada en el programa, incluso cuando esa operación sea la predeterminada `Main`.
- En Python, llame el método `qsharp.init` para establecer el perfil target.

Python

```
qsharp.init(target_profile=qsharp.TargetProfile.Adaptive_RI)
```

Soportado targets para Adaptive RI target el perfil

Por ahora, Quantinuum es el único proveedor de Azure Quantum que tiene Adaptive RI targets.

- **Emuladores:** `quantinuum.sim.h2-1e` y `quantinuum.sim.h2-2e`
- **QPU:** `quantinuum.qpu.h2-1` y `quantinuum.qpu.h2-2`

Para más información sobre las ofertas de Quantinuum en Azure Quantum, consulte [Quantinuum Emulators \(Emuladores de Quantinuum\)](#).

Creación y ejecución de programas para perfil Adaptive RIFtarget

Adaptive RIF target perfiles tienen todas las capacidades de Adaptive RI, pero también admiten programas de Q# que contienen cálculos de punto flotante.

Por ejemplo, se permite el siguiente código de Q# en :Adaptive RIFtarget

Q#

```
@EntryPoint(Adaptive_RIF)
operation DynamicFloat() : Double {
    use q = Qubit();
    H(q);
    mutable f = 0.0;
    if M(q) == One {
        f = 0.5;
    }
    Reset(q);
    return f;
}
```

Configuración del Adaptive RIFtarget perfil

Para establecer manualmente el QIRtarget perfil en **Adaptive RIF**, elija una de las siguientes opciones:

- Si configura un proyecto de Q#, agregue el siguiente comando al archivo del `qsharp.json` proyecto:

JSON

```
{
  "targetProfile": "adaptive_rif"
}
```

- Si trabaja en un `.qs` archivo que no forma parte de un proyecto de Q#, establezca el perfil directamente en el target código de Q#. Para ello, incluya `@EntryPoint(Adaptive_RIF)` justo antes de la operación de punto de entrada en el programa, incluso cuando esa operación sea la predeterminada `Main`.
- En Python, llame el método `qsharp.init` para establecer el perfil target.

Python

```
qsharp.init(target_profile=qsharp.TargetProfile.Adaptive_RIF)
```

Soportado targets para Adaptive RItarget el perfil

Por ahora, Azure Quantum no tiene Adaptive RIFtargets. Sin embargo, puede ejecutar programas para Adaptive RIFtargets en el simulador local en el QDK.

Disponibilidad global de proveedores de Azure Quantum

La disponibilidad de los proveedores de computación cuántica IonQ, Pasqal, Quantinuum y Rigetti varía en función del país o región donde se registre la cuenta de facturación. Seleccione la pestaña de cada proveedor para ver su disponibilidad por país o región.

IonQ

IonQ ofrece un plan de pago por uso a través de Azure Quantum. Para más información sobre los recursos de IonQ en Azure Quantum, consulte [Proveedor de IonQ y targets](#).

IonQ está disponible en los siguientes países o regiones:

- Armenia
- Australia
- Austria
- Belarus
- Belgium
- Bulgaria
- Canada
- Chile
- Colombia
- Croatia
- Cyprus
- Czechia
- Denmark
- Estonia
- Finland
- France
- Germany
- Greece
- Hungary
- Iceland
- India
- Indonesia
- Ireland
- Israel
- Italy
- Kenya

- Korea
- Latvia
- Liechtenstein
- Lithuania
- Luxembourg
- Malaysia
- Malta
- Monaco
- Netherlands
- Nueva Zelanda
- Nigeria
- Norway
- Poland
- Portugal
- Puerto Rico
- Romania
- Arabia Saudí
- Serbia
- Singapore
- Slovakia
- Slovenia
- Sudáfrica
- Spain
- Sweden
- Switzerland
- Taiwan
- Thailand
- Türkiye
- Emiratos Árabes Unidos
- Reino Unido
- United States
- Vietnam

IonQ no está disponible en los siguientes países o regiones:

- Afghanistan
- Albania
- Algeria
- Andorra

- Angola
- Argentina
- Azerbaijan
- Bahrain
- Bangladesh
- Barbados
- Belize
- Bermuda
- Bolivia
- Bosnia-Herzegovina
- Botswana
- Brazil
- Brunei
- Cabo Verde
- Cameroon
- Islas Caimán
- Costa Rica
- Costa de Marfil
- Curaçao
- República Dominicana
- Ecuador
- Egypt
- El Salvador
- Ethiopia
- Islas Feroe
- Fiji
- Georgia
- Ghana
- Guatemala
- Honduras
- RAE de Hong Kong
- Iraq
- Jamaica
- Japan
- Jordan
- Kazakhstan
- Kuwait
- Kyrgyzstan
- Lebanon
- Libya
- RAE de Macao

- Mauritius
- Mexico
- Moldova
- Mongolia
- Montenegro
- Morocco
- Namibia
- Nepal
- Nicaragua
- Macedonia del Norte
- Oman
- Pakistan
- Autoridad palestina
- Panama
- Paraguay
- Peru
- Philippines
- Qatar
- Russia
- Rwanda
- San Kitts y Nevis
- Senegal
- Sri Lanka
- Tajikistan
- Tanzania
- Trinidad y Tobago
- Tunisia
- Turkmenistan
- Islas Vírgenes estadounidenses
- Uganda
- Ukraine
- Uruguay
- Uzbekistan
- Ciudad del Vaticano
- Venezuela
- Yemen
- Zambia
- Zimbabwe

Proveedor IonQ

04/10/2025

Los equipos cuánticos de IonQ realizan cálculos mediante la manipulación de los estados de energía hiperfinos de iones de iterbio con láseres. Los átomos son los cúbits de la naturaleza: cada cúbit es idéntico dentro de un programa y en los distintos programas. Las operaciones lógicas también se pueden realizar en cualquier par arbitrario de cúbits, lo que permite utilizar programas cuánticos complejos sin las restricciones de la conectividad física. ¿Quiere saber más? Lea la [introducción a la tecnología de los equipos cuánticos de iones atrapados](#) de IonQ.

- Publicador: [IonQ](#)
- Identificador de proveedor: `ionq`

Los siguientes targets están disponibles en este proveedor:

[Expandir tabla](#)

Nombre de destino	Id. de destino	Número de cúbits	Descripción
Simulador cuántico	Simulador de IonQ	29 cúbits	Simulador idealizado basado en la nube de IonQ. Sin costo.
IonQ Aria 1	ionq.qpu.aria-1	25 cúbits	Computadora cuántica de iones atrapados IonQ Aria 1.
IonQ Forte 1	ionq.qpu.forte-1	36 qubits	El ordenador cuántico de iones atrapados IonQ Forte 1.
IonQ Forte Enterprise 1	ionq.qpu.forte-enterprise-1	36 qubits	Computadora cuántica de iones atrapados IonQ Forte Enterprise 1.

Las targets de IonQ corresponden a un perfil de **QIR Base**. Para más información sobre este target perfil y sus limitaciones, consulte [Descripción target de los tipos de perfil en Azure Quantum](#).

Simulador cuántico

Simulador idealizado acelerado por GPU que admite hasta 29 cúbits, con el mismo conjunto de puertas que IonQ proporciona en su hardware cuántico. Esto hace que sea excelente lugar para realizar trabajos previos antes de ejecutarlos en un equipo cuántico real.

- Tipo de trabajo: `Simulation`

- Formato de datos: `ionq.circuit.v1`
- Id. de destino: `ionq.simulator`
- Perfil de ejecución objetivo: [QIR Base](#)

Ordenador cuántico IonQ Aria

IonQ Aria es el buque insignia de los equipos cuánticos de iones atrapados de IonQ, con un sistema reconfigurable dinámicamente de 25 cúbits. Para obtener más información, consulte [IonQ Aria \(ionq.com\)](https://ionq.com) [↗].

Importante

Debiasing está habilitado en los sistemas Aria de forma predeterminada, y los trabajos enviados están sujetos a tarifas basadas en debiasing. Para obtener más información sobre el desbiaseo y cómo deshabilitar o habilitar el servicio, consulte [Mitigación de sesgos](#).

- Tipo de trabajo: `Quantum Program`
- Formato de datos: `ionq.circuit.v1`
- Id. de destino: `ionq.qpu.aria-1`
- Perfil de ejecución objetivo: [QIR Base](#)

 Expandir tabla

Nombre de parámetro	Tipo	Obligatorio	Descripción
<code>shots</code>	Int	No	Número de disparos experimentales.

Tiempos del sistema

 Expandir tabla

Medida	Duración media del tiempo
T1	10-100 segundos
T2	1 segundo
Puerta de un solo cúbit	135 μ s
Puerta de dos cúbits	600 μ s

Fidelidad del sistema

 Expandir tabla

Operación	Fidelidad promedio
Puerta de un solo cúbit	99,95 % (correo no deseado corregido)
Puerta de dos cúbits	99,6 % (sin corrección de SPAM)
SPAM*	99.61%

* Preparación y medición del estado (SPAM): esta medida determina la precisión con la que un equipo cuántico puede establecer un cúbit en su estado inicial y, a continuación, medir el resultado al final.

IonQ Aria está disponible a través del plan pago por uso. Para obtener más información, consulte los [precios de Azure Quantum](#).

Ordenador cuántico IonQ Forte

IonQ Forte es el equipo cuántico de iones atrapados de rendimiento más alto de IonQ, disponible comercialmente. Con un sistema configurable por software de 36 cúbits. Para obtener más información, consulte [IonQ Forte \(ionq.com\)](#) .

Importante

Debiasing está habilitado en el sistema Forte de forma predeterminada, y los trabajos enviados están sujetos a tarifas basadas en la desbiasización. Para obtener más información sobre el desbiaseo y cómo deshabilitar o habilitar el servicio, consulte [Mitigación de sesgos](#).

- Tipo de trabajo: `Quantum Program`
- Formato de datos: `ionq.circuit.v1`
- Id. de destino: `ionq.qpu.forte-1`
- Perfil de ejecución objetivo: [QIR Base](#)

 Expandir tabla

Nombre de parámetro	Tipo	Obligatorio	Descripción
<code>shots</code>	Int	No	Número de disparos experimentales.

Ordenador cuántico IonQ Forte Enterprise

IonQ Forte Enterprise es el equipo cuántico de iones atrapados de rendimiento más alto de IonQ, disponible comercialmente. Con un sistema configurable por software de 36 cúbits. Para obtener más información, consulte [IonQ Forte Enterprise \(ionq.com\)](https://ionq.com).

El Forte Enterprise es una versión de los sistemas de clase Forte que se han adaptado y endurecido para su implementación en un entorno estándar de centro de datos, lo que lo hace más adecuado para tareas de nivel empresarial orientadas a la producción. El hardware y el rendimiento de la computación cuántica base son los mismos. La principal diferencia entre IonQ Forte y IonQ Forte Enterprise se encuentra en sus casos de implementación y uso previstos, no en sus especificaciones principales de rendimiento. Aunque ambos sistemas cuentan con las mismas métricas de alto rendimiento, Forte Enterprise está diseñado específicamente para la integración en un entorno de centro de datos.

ⓘ Importante

Debiasing está habilitado en el sistema Forte Enterprise de forma predeterminada y los trabajos enviados están sujetos a precios basados en desbiasing. Para obtener más información sobre el desbiaseo y cómo deshabilitar o habilitar el servicio, consulte [Mitigación de sesgos](#).

- Tipo de trabajo: `Quantum Program`
- Formato de datos: `ionq.circuit.v1`
- Id. de destino: `ionq.qpu.forte-enterprise-1`
- Perfil de ejecución objetivo: [QIR Base](#)

[Expandir tabla](#)

Nombre de parámetro	Tipo	Obligatorio	Descripción
<code>shots</code>	Int	No	Número de disparos experimentales.

Formato de entrada

En Q#, la salida de una medida cuántica es un valor de tipo `Result`, que solo puede tomar los valores `Zero` y `One`. Cuando se define una operación de Q#, solo se puede enviar al hardware ionQ si el tipo de valor devuelto es una colección de `Result`s, es decir, si la salida de la operación es el resultado de una medida cuántica. La razón de esto es que IonQ crea un

histograma a partir de los valores devueltos, por lo que restringe el tipo de valor devuelto para `Result` simplificar la creación de este histograma.

Las targets de IonQ corresponden a la [QIR Base profile](#). Este perfil no puede ejecutar operaciones cuánticas que requieran el uso de los resultados de las mediciones de cúbits para controlar el flujo del programa.

Formato de salida

Al enviar un programa cuántico al simulador de IonQ, devuelve el histograma creado por las medidas. El simulador de IonQ no toma muestras de la distribución de probabilidad creada por un programa cuántico, sino que devuelve la distribución escalada al número de disparos. Esto es más evidente cuando envía un circuito de disparo único. Verá múltiples resultados de medición en el histograma para una toma. Este comportamiento es inherente al simulador de IonQ, mientras que IonQ QPU ejecuta realmente el programa y agrega los resultados.

Funcionalidades adicionales

Aquí se enumeran las funcionalidades adicionales compatibles con el hardware de IonQ.

[Expandir tabla](#)

Capacidad	Descripción
Mitigación de errores	Uso de la reducción del sesgo para minimizar el ruido y maximizar el rendimiento algorítmico en el hardware IonQ
Compatibilidad con puertas nativas	Definir y ejecutar circuitos directamente en puertas nativas de hardware IonQ
Simulación del modelo de ruido	Simulación del perfil de ruido que encontrarán los circuitos al ejecutarlos en hardware de IonQ diferente.

Los usuarios pueden aprovechar estas funcionalidades adicionales a través de parámetros de transferencia en los proveedores de Q# y Qiskit de Azure Quantum.

Mitigación de errores

IonQ proporciona la opción de habilitar *la mitigación* de errores cuánticos al enviar trabajos al hardware de IonQ. La mitigación de errores es un proceso de nivel de compilador que ejecuta y ejecuta varias variaciones simétricas de un circuito y, a continuación, agrega los resultados a la vez que mitiga el impacto de los errores de hardware y la decoherencia de cúbits. A

diferencia de las técnicas de *corrección de errores* cuánticos, la mitigación de errores no requiere una gran sobrecarga de puertas y qubits.

Debiasing es el proceso de crear ligeras variaciones de un circuito determinado que *deben* ser idénticas en una máquina sin ruido ideal, utilizando técnicas como diferentes asignaciones de cúbits, descomposición de puertas y soluciones de pulso, y luego ejecutando esas variaciones.

El afilado y *el promedio* son opciones para agregar los resultados de las variaciones. Promediar se basa igualmente en todos los resultados de variación, mientras que Afilado filtra los resultados erróneos, y puede ser más confiable para ciertos tipos de algoritmos.

Para obtener más información, consulte [Desviar y afilar](#). Para obtener los precios de mitigación de errores, consulte [Precios](#) de IonQ.

Habilitación de la mitigación de errores

ⓘ Nota

Debiasing está habilitado de forma predeterminada en los sistemas Aria y Forte.

En Azure Quantum, la mitigación de errores se puede habilitar o deshabilitar para los trabajos enviados con Q# o con Qiskit.

Para habilitar la mitigación de errores, agregue un parámetro opcional para la target máquina:

Python

```
option_params = {  
    "error-mitigation": {  
        "debias": True  
    }  
}
```

Para deshabilitar la mitigación de errores, establezca el parámetro `False` en :

Python

```
option_params = {  
    "error-mitigation": {  
        "debias": False  
    }  
}
```


ⓘ Nota

Si también usa la simulación del modelo de ruido de IonQ, estos parámetros se pueden incluir aquí, por ejemplo:

Python

```
option_params = {  
    "error-mitigation": {  
        "debias": False  
    },  
    "noise": {  
        "model": "aria-1",  
        "seed": 100  
    }  
}
```

Para obtener más información, consulte [Simulación](#) de modelos de ruido.

Ejecución de un trabajo en Azure Quantum con mitigación de errores

En este ejemplo se usa un generador de números aleatorios simple.

En primer lugar, importe los paquetes necesarios e inicie el perfil base:

Python

```
import qsharp  
import azure.quantum  
qsharp.init(target_profile=qsharp.TargetProfile.Base)
```

A continuación, defina la función .

Q#

```
%%qsharp  
import Std.Measurement.*;  
import Std.Arrays.*;  
import Std.Convert.*;  
  
operation GenerateRandomBit() : Result {  
    use target = Qubit();  
  
    // Apply an H-gate and measure.  
    H(target);  
    return M(target);  
}
```

and compile the `operation`:

```
```python
MyProgram = qsharp.compile("GenerateRandomBit()")
```

Conéctese a Azure Quantum, seleccione la target máquina y configure los parámetros de ruido para el emulador:

Python

```
MyWorkspace = azure.quantum.Workspace(
 resource_id = "",
 location = ""
)

MyTarget = MyWorkspace.get_targets("ionq.qpu.aria-1")
```

Especificar la `error-mitigation` configuración

Python

```
option_params = {
 "error-mitigation": {
 "debias": True
 }
}
```

Pase la configuración de mitigación de errores al enviar el trabajo:

Python

```
job = MyTarget.submit(MyProgram, "Experiment with error mitigation", shots = 10,
input_params = option_params)
job.get_results()
```

En Qiskit, se pasan parámetros opcionales a la configuración de la máquina target antes de enviar la tarea.

Python

```
circuit.name = "Single qubit random - Debias: True"
backend.options.update_options(**option_params)
```

```
job = backend.run(circuit, shots=500)
```

### ⚠ Nota

Si no proporciona el `error-mitigation` parámetro, la target máquina usará su configuración predeterminada, que se encuentra *habilitada* para los sistemas Aria y Forte.

## Compatibilidad y uso de puertas nativas

De manera predeterminada, IonQ permite especificar un circuito cuántico mediante un conjunto abstracto de puertas cuánticas, llamado `qis`, que permite flexibilidad y portabilidad al escribir un algoritmo sin preocuparse por la optimización del hardware.

Sin embargo, en algunos casos de uso avanzados, puedes definir un circuito directamente en puertas nativas para acercarte más al hardware y omitir la optimización. El conjunto de puertas nativas es el conjunto de puertas cuánticas que se ejecutan físicamente en el procesador cuántico y se asignan al circuito como parte de la ejecución.

Para obtener más información, consulte [Guía de inicio con puertas nativas \(ionq.com\)](https://ionq.com/guides/native-gates).

Para usar el conjunto de puertas nativas al enviar trabajos de Qiskit a Azure Quantum, especifique el parámetro `gateset` al inicializar el back-end como en el ejemplo siguiente:

Python

```
Here 'provider' is an instance of AzureQuantumProvider
backend = provider.get_backend("ionq.qpu.aria-1", gateset="native")
```

[Expandir tabla](#)

| Nombre de parámetro  | Tipo   | Obligatorio | Descripción                                                                                                                                                                                                                                                    |
|----------------------|--------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gateset</code> | cadena | No          | Especifica el conjunto de puertas que se va a usar para definir un circuito. El valor <code>qis</code> corresponde a las puertas abstractas (comportamiento predeterminado) y <code>native</code> a las <a href="#">puertas nativas del hardware de IonQ</a> . |

Para obtener más información sobre los trabajos de Qiskit, consulte [Enviar un circuito con Qiskit](#).

# Simulación del modelo de ruido

Incluso lo mejor del hardware cuántico actual tiene ruido inherente y conocer las características de ruido del target sistema puede ayudarle a refinar los algoritmos y obtener una predicción más realista de los resultados al ejecutar el circuito en hardware. IonQ proporciona una *simulación* de modelo de ruido que introduce ruido en el circuito mediante una "huella digital de ruido" específica del target hardware. Para obtener más información, consulte [Introducción a la simulación](#) de modelo de ruido de hardware.

## Parámetros del modelo de ruido

 Expandir tabla

| Nombre de parámetro | Valores                                   | Descripción                                                                                                                                                                                                                                                                                                                    |
|---------------------|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>noise</code>  | <code>model</code> , <code>seed</code>    | Habilita la simulación del modelo de ruido                                                                                                                                                                                                                                                                                     |
| <code>model</code>  | <code>ideal</code> , <code>aria-1</code>  | Especifica el modelo de ruido para el target hardware. <ul style="list-style-type: none"><li><code>ideal</code> - No se introduce ningún ruido en el circuito. Esto es lo mismo que no habilitar la simulación de ruido.</li><li><code>aria-1</code> – usa el modelo de ruido para el computador cuántico IonQ Aria.</li></ul> |
| <code>seed</code>   | Entero entre 1 y $2^{31}$ (2,147,483,648) | Permite especificar un valor semilla para el ruido pseudoaleatorio y el muestreo de disparos, creando resultados de ruido reproducibles. Si no se especifica el parámetro , se crea un valor aleatorio <code>seed</code> .                                                                                                     |

## Conciencia de disparo

La simulación del modelo de ruido es *sensible a disparos*; es decir, muestrea mediciones del estado de salida en función del número de disparos proporcionados. En Azure Quantum, el `shots` parámetro se envía con el trabajo y es necesario para `aria-1` los modelos de ruido. Si no se especifica ningún `shot` valor, se usa un valor predeterminado de `1000` . Si se usa el `ideal` modelo de ruido, se omite el `shots` parámetro .

## Capacidad de cúbito

Aunque el `ideal` modelo de ruido permite simular hasta 29 cúbits con el simulador cuántico de IonQ, los modelos de ruido específicos del hardware se limitan a la capacidad de cúbit real del target hardware, que es de 25 cúbits para el `aria-1` modelo de ruido.

## Habilitación de la simulación del modelo de ruido

En Azure Quantum, la simulación de modelos de ruido se puede habilitar o deshabilitar para los trabajos enviados con Q# o con Qiskit.

Para habilitar la simulación del modelo de ruido, agregue un parámetro opcional para la target máquina, por ejemplo:

Python

```
option_params = {
 "noise": {
 "model": "aria-1", # targets the Aria quantum computer
 "seed" : 1000 # If seed isn't specified, a random value is used
 }
}
```

### ⚠ Nota

Si también usa la mitigación de errores de IonQ, estos parámetros se pueden incluir aquí, por ejemplo:

Python

```
option_params = {
 "error-mitigation": {
 "debias": False
 },
 "noise": {
 "model": "aria-1",
 "seed": 1000
 }
}
```

Para obtener más información, consulte [Mitigación](#) de errores.

## Ejecución de un trabajo con simulación de modelo de ruido

Puede usar el mismo programa de ejemplo mostrado anteriormente en [la mitigación](#) de errores y agregar o reemplazar la configuración del modelo de ruido en `option_params`;

Python

```
option_params = {
 "error-mitigation": {
 "debias": True
 },
 "noise": {
 "model": "aria",
 "seed": 1000
 }
}
```

A continuación, pase los parámetros opcionales al enviar el trabajo:

Python

```
job = MyTarget.submit(MyProgram, "Experiment with noise model simulation", shots =
10, input_params = option_params)
job.get_results()
```

En Qiskit, se pasan parámetros opcionales a la configuración de la máquina target antes de enviar la tarea.

Python

```
circuit.name = "Single qubit random - Debias: True"
backend.options.update_options(**option_params)
job = backend.run(circuit, shots=500)
```

## Precios

Para ver el plan de facturación de IonQ, consulte [Precios de Azure Quantum](#).

## Límites y cuotas

Se realiza un seguimiento de las cuotas de IonQ en función de la unidad de uso de QPU, que es la *captura de puerta de cúbit (QGS)*. El uso de los recursos se acredita en su cuenta.

Cada programa cuántico consta de  $N$  puertas lógicas cuánticas de uno o más cúbits, y se ejecuta un número determinado de repeticiones. El número de capturas de puerta se calcula mediante la fórmula siguiente:

$$QGS = N \cdot C$$

donde:

- $N$  es el número de puertas de uno o dos cúbits enviadas
- $C$  es el número de disparos de ejecución solicitados

## Estado de IonQ

Para obtener información sobre los retrasos en el procesamiento de trabajos de QPU de IonQ, consulte [la página de estado de IonQ](#) .

## Procedimientos recomendados de IonQ y grafo de conectividad

Para ver los procedimientos recomendados para la QPU de IonQ, consulte [Procedimientos recomendados de IonQ \(ionq.com\)](#) .

# Directiva de soporte técnico para IonQ en Azure Quantum

18/09/2025

En este artículo se describe la directiva de soporte técnico de Microsoft que se aplica cuando se usa el proveedor IonQ en Azure Quantum. El artículo se aplica a cualquiera de los targets de este proveedor.

Si usa el proveedor IonQ y encuentra problemas inesperados que no puede solucionar sin asistencia, puede [crear un caso de Soporte técnico de Azure](#) para ponerse en contacto con el equipo de Soporte técnico de Azure y obtener ayuda.

No obstante, en algunas situaciones, el equipo de Soporte técnico de Azure tendrá que redirigirle al equipo de soporte técnico de IonQ. O bien, puede recibir una respuesta más rápida poniéndose en contacto con IonQ directamente en el [sitio web](#) de soporte técnico de IonQ.

## Preguntas más frecuentes

### **P: ¿Cuál es la directiva de soporte técnico para usar las ofertas de IonQ a través de Azure Quantum?**

Microsoft proporcionará compatibilidad solo con la plataforma Azure y el servicio Azure Quantum. IonQ proporciona compatibilidad con hardware, simuladores y otros productos y targets se proporcionará directamente mediante IonQ. Para más información sobre el Soporte técnico de Azure, consulte los [planes de soporte técnico de Azure](#). Para obtener información sobre el soporte técnico de IonQ, consulte el [sitio web de soporte técnico de IonQ](#).

### **P: ¿Qué ocurre si creo un caso para el equipo de soporte técnico de Azure y resulta que un proveedor de terceros (como IonQ) necesita intervenir para solucionar el problema?**

El ingeniero de soporte técnico creará un paquete de redireccionamiento. Se trata de un documento PDF que contiene información sobre su caso y que puede proporcionar al equipo de soporte técnico de IonQ. El ingeniero de soporte técnico también le dará consejos e instrucciones sobre cómo comunicarse con IonQ para continuar con la solución del problema. Para obtener más información, consulte [la página](#) estado de IonQ.



## **P: ¿Qué ocurre si creo un caso para el equipo de soporte técnico de IonQ y resulta que hay un problema con el servicio Azure Quantum?**

El equipo de soporte técnico de IonQ le ayudará a ponerse en contacto con Microsoft y le proporcionará un paquete de redireccionamiento. Este paquete es un documento PDF que puede usar para continuar con su consulta con el equipo de Soporte técnico de Azure.

## **Aviso de declinación de responsabilidades sobre la información de terceros**

Los productos de otros fabricantes que se mencionan en este artículo han sido creados por compañías independientes de Microsoft. Microsoft no ofrece ninguna garantía, ya sea implícita o de otro tipo, sobre la confiabilidad o el rendimiento de dichos productos.

## **Declinación de responsabilidad sobre la información de contacto de terceros**

Microsoft proporciona información de contacto de otros proveedores para ayudarle a encontrar información adicional sobre este tema. Esta información de contacto puede cambiar sin previo aviso. Microsoft no garantiza la precisión de la información de contacto de terceros.

# Proveedor de PASQAL

02/06/2025

Los equipos cuánticos de PASQAL controlan átomos neutros con pinzas ópticas, usando luz láser para manipular registros cuánticos con hasta cien cúbits.

- Publicador: [PASQAL](#)
- Identificador de proveedor: `pasqal`

Los siguientes destinos están disponibles por este proveedor:

[Expandir tabla](#)

| Nombre de destino       | Id. de destino                  | Número de cúbits            | Descripción                                                                                                                                     |
|-------------------------|---------------------------------|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">Emu-TN</a>  | <code>pasqal.sim.emu-tn</code>  | 100 cúbits en redes 1D y 2D | Simula la evolución temporal de un estado cuántico mediante la ecuación de Schrödinger correspondiente a las acciones que realizan los láseres. |
| <a href="#">Fresnel</a> | <code>pasqal.qpu.fresnel</code> | 100 qubits                  | Equipo cuántico de átomos neutros de PASQAL.                                                                                                    |

## Emulador

El emulador de Emu-TN de PASQAL simula la evolución temporal de un estado cuántico mediante la ecuación de Schrödinger correspondiente a las acciones que realizan los láseres.

El emulador de Emu-TN se ejecuta en un clúster de nodos DGX, cada uno equipado con GPU nvidia A100, lo que permite la emulación de los procesadores cuánticos de PASQAL. Es una herramienta clave para crear prototipos y validar programas cuánticos antes de ejecutarlos en la QPU . Se pueden emular hasta 100 cúbits en matrices 2D para desarrollar aplicaciones industriales y para avanzar en el descubrimiento científico.

- Tipo de trabajo: `Simulation`
- Formato de datos: `application/json`
- Id. de destino: `pasqal.sim.emu-tn`
- Perfil de ejecución objetivo: N/A

## Fresnel

Fresnel es el equipo cuántico de PASQAL basado en átomos neutros. Los átomos neutros, controlados por pinzas ópticas, componen una matriz de 100 cúbits.

Los dispositivos cuánticos de átomos neutros usan láseres altamente centrados, denominados pinzas ópticas, para interceptar y manipular átomos neutros individualmente para crear matrices de cúbits 1D o 2D en configuraciones arbitrarias. La generación PASQAL actual de dispositivos usa alrededor de 100 átomos de rubidio para cálculos. Cada cúbit se representa mediante un estado de energía de dos niveles en un átomo de Rubidium, normalmente un estado de tierra y un estado de Rydberg que es un estado de alta energía.

- Tipo de trabajo: `Quantum program`
- Formato de datos: `application/json`
- Id. de destino: `pasqal.qpu.fresnel`
- Perfil de ejecución objetivo: N/A

## Pulser SDK

En la QPU PASQAL, los átomos individuales se capturan en posiciones bien definidas en lasttices 1D o 2D. [Pulser](#) es un marco para componer, simular y ejecutar secuencias de pulso en dispositivos cuánticos de átomos neutros. Para obtener más información, consulte [la documentación](#) de Pulser.

Para instalar paquetes del SDK de Pulser, ejecute el código siguiente:

Python

```
!pip -q install pulser-simulation #Only for using the local Qutip emulator
included in Pulser
!pip -q install pulser-core
```

## Formato de datos de entrada

Los destinos PASQAL aceptan archivos JSON como formato de datos de entrada. Para enviar las secuencias de pulso, debe convertir los objetos Pulser en una cadena JSON que se puede usar como datos de entrada.

Python

```
Convert the sequence to a JSON string
def prepare_input_data(seq):
 input_data = {}
 input_data["sequence_builder"] = json.loads(seq.to_abstract_repr())
 to_send = json.dumps(input_data)
```

```
#print(json.dumps(input_data, indent=4, sort_keys=True))
return to_send
```

Antes de enviar el trabajo cuántico a PASQAL, debe establecer los parámetros de formato de datos de entrada y salida adecuados. Por ejemplo, el código siguiente establece el formato `pasqal.pulser.v1` de datos de entrada en y el formato de datos de salida en `pasqal.pulser-results.v1`.

Python

```
Submit the job with proper input and output data formats
def submit_job(target, seq):
 job = target.submit(
 input_data=prepare_input_data(seq), # Take the JSON string previously
 defined as input data
 input_data_format="pasqal.pulser.v1",
 output_data_format="pasqal.pulser-results.v1",
 name="PASQAL sequence",
 shots=100 # Number of shots
)
```

Para obtener más información sobre cómo enviar trabajos al proveedor PASQAL, consulte [Envío de un circuito a PASQAL mediante pulser SDK](#).

## Precios

Para ver el plan de facturación de PASQAL, visite [Precios](#) de Azure Quantum.

## Límites y cuotas

Las cuotas de PASQAL se aplican al uso del emulador y la QPU y se pueden aumentar con una incidencia de soporte técnico. Para ver los límites y cuotas actuales, vaya a la sección **Operaciones** y seleccione la hoja **Cuotas** del área de trabajo en [Azure Portal](#). Para obtener más información, consulte [Cuotas de Azure Quantum](#).

# Directiva de soporte técnico para PASQAL en Azure Quantum

Artículo • 18/03/2025

En este artículo se describe la directiva de soporte técnico de Microsoft que se aplica al usar el proveedor PASQAL en Azure Quantum. El artículo se aplica a cualquiera de los [destinos](#) ofrecidos por PASQAL.

## Soporte técnico de Azure

En general, los problemas con PASQAL en Microsoft Azure Quantum se abordan mejor con el equipo de soporte técnico de Azure Quantum directamente mediante [la creación de un caso](#) de Soporte técnico de Azure. Este equipo tiene todos los detalles necesarios para aislar el problema que está experimentando y dirigir el informe al punto de contacto adecuado. El equipo de Soporte técnico de Azure también tiene los medios para llegar al soporte de PASQAL fuera del horario comercial para los problemas más urgentes.

## Compatibilidad con Pasqal

En algunas situaciones, el equipo de soporte técnico de Azure deberá redirigirle al equipo de soporte técnico de PASQAL. Puede recibir una respuesta más rápida al ponerse en contacto [directamente](#) con PASQAL.

## Escala de tiempo de soporte técnico

Pasqal siempre intentará resolver problemas con la integración de Microsoft Azure tan rápido como sea práctico. Algunos problemas tardarán más tiempo en solucionarse que otros, pero las tablas siguientes indican el tiempo que puede esperar para una respuesta inicial de Pasqal después de que se notifique un problema.

[Expandir tabla](#)

| Severidad | Definición                                                     | Respuesta SLA    |
|-----------|----------------------------------------------------------------|------------------|
| Alto      | Error en la conectividad, degradación completa del rendimiento | Un día laborable |
| Media     | Degradación del rendimiento                                    | Dos días         |

| Severidad | Definición                                           | Respuesta SLA        |
|-----------|------------------------------------------------------|----------------------|
|           |                                                      | laborables           |
| Bajo      | Preguntas y problemas relacionados con la aplicación | Tres días laborables |

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Proveedor de Quantinuum

02/10/2025

Quantinuum proporciona acceso a sistemas de iones atrapados con cúbits totalmente conectados de alta fidelidad y la capacidad de tomar medidas en el circuito medio.

- Publicador: [Quantinuum](#)
- Identificador de proveedor: `quantinuum`

## Targets

### ⚠ Advertencia

Quantinuum retirará el hardware H1-1 el 15 de octubre de 2025. Si está en un plan que usa el sistema H1, cambie a un plan que admita el hardware H2.

Los siguientes targets están disponibles en este proveedor:

[Expandir tabla](#)

| Nombre del destino                           | Id. de destino        | Número de cúbits | Description                                                                                                                                            |
|----------------------------------------------|-----------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">Comprobador de sintaxis H2-1</a> | quantinuum.sim.h2-1sc | 56 qubits        | Úselo para validar los programas cuánticos en el compilador H2-1 antes de enviarlos a hardware o emuladores en la plataforma de Quantinuum. Sin costo. |
| <a href="#">comprobador de sintaxis H2-2</a> | quantinuum.sim.h2-2sc | 56 qubits        | Úselo para validar programas cuánticos en el compilador H2-2 antes de enviarlos a hardware o emuladores en la plataforma de Quantinuum. Sin costo.     |
| <a href="#">Emulador H2-1</a>                | quantinuum.sim.h2-1e  | 56/32 qubits     | Usa un modelo físico realista y un modelo de ruido de H2-1. La simulación de 56 cúbits solo está disponible como simulación de estabilizador           |
| <a href="#">Emulador H2-2</a>                | quantinuum.sim.h2-2e  | 56/32 qubits     | Usa un modelo físico realista y un modelo de ruido de H2-2. La simulación de 56 cúbits solo está disponible como simulación de estabilizador           |
| <a href="#">H2-1</a>                         | quantinuum.qpu.h2-    | 56 qubits        | Dispositivo de iones atrapados H2-1 de                                                                                                                 |

| Nombre del destino | Id. de destino      | Número de cúbits | Description                                        |
|--------------------|---------------------|------------------|----------------------------------------------------|
|                    | 1                   |                  | Quantinuum.                                        |
| H2-2               | quantinuum.qpu.h2-1 | 56 qubits        | Dispositivo de iones atrapados H2-2 de Quantinuum. |

targets de Quantinuum se corresponde a un perfil de **QIR Adaptive RI**. Para más información sobre este target perfil y sus limitaciones, consulte [Descripción target de los tipos de perfil en Azure Quantum](#).

Todos los circuitos híbridos integrados ahora son compatibles con Quantinuum targets . Para obtener más información sobre el envío de trabajos híbridos integrados, consulte [Informática híbrida](#) integrada.

Para empezar a usar el proveedor Quantinuum en Azure Quantum, consulte [Introducción a Q# y un cuaderno](#) de Azure Quantum.

#### Sugerencia

Los trabajos cuánticos enviados en una sesión tienen **acceso exclusivo** al hardware de Quantinuum, siempre que pongas en cola los trabajos con una diferencia de un minuto entre ellos. Después, cualquier trabajo se acepta y se gestiona con la lógica estándar de puesta en cola y priorización. Para más información, consulte [sesiones en Azure Quantum](#).

## Comprobadores de sintaxis

Se recomienda que los usuarios validen primero su código con un Corrector sintáctico. Se trata de una herramienta para comprobar que la sintaxis sea correcta, que la compilación finaliza y que la máquina sea compatible. Los correctores sintácticos usan el mismo compilador que el equipo cuántico que target. Por ejemplo, el comprobador de sintaxis H2-1 usa el mismo compilador que H2-1. La pila de compilación completa se ejecuta con la excepción de las operaciones cuánticas reales. Si el código se compila, el corrector sintáctico devuelve el estado `success` y un resultado con todo ceros. Si el código no se compila, el comprobador de sintaxis devuelve un estado erróneo y proporciona el error devuelto para ayudar a los usuarios a depurar su sintaxis de circuito. Los correctores sintácticos permiten a los desarrolladores validar su código en cualquier momento, incluso cuando las máquinas están desconectadas.

- Tipo de trabajo: `Simulation`
- Formatos de datos: `honeywell.openqasm.v1`, `honeywell.qir.v1`



- Id. de destino:
  - Comprobador de sintaxis H2-1: `quantinuum.sim.h2-1sc`
  - Corrector de sintaxis H2-2: `quantinuum.sim.h2-2sc`
- Perfil de ejecución de destino: [QIR Adaptive RI](#)

El uso de los comprobadores de sintaxis se ofrece de forma gratuita.

## Emulador de Quantinuum (basado en la nube)

El emulador de Quantinuum está disponible de forma gratuita en la página de [Codificación con Azure Quantum](#), en el sitio web de Azure Quantum, donde puede escribir código de Q# y enviar sus trabajos al emulador de Quantinuum sin una cuenta de Azure. El emulador de Quantinuum es un emulador cuántico basado en statevector que usa un modelo de ruido físico realista y parámetros de error generalizados basados en el rendimiento típico de un [equipo cuántico del modelo de sistema H2](#). La simulación cuántica realizada es la misma que el [emulador del modelo de sistema H2](#), pero la rutina de optimización del circuito clásico se reduce para aumentar el rendimiento. La compatibilidad con [la informática](#) híbrida integrada está planeada para una fecha futura.

## Emulador de System Model H2

Después de validar la sintaxis de su código con el Comprobador de sintaxis H2-1, los usuarios pueden aprovechar el emulador de System Model H2 de Quantinuum, una herramienta de emulación que contiene un modelo físico detallado y un modelo de ruido realista del hardware del modelo del sistema H2 real. Puede encontrar más información sobre el modelo de ruido en la *hoja* de datos del producto del emulador de System Model H2 que se encuentra en la [página System Model H2](#). El System Model H2 Emulator utiliza una API para el envío de trabajos idéntica a la del hardware del System Model H2, lo que permite una transición sin problemas de la emulación al hardware. Para ayudar a maximizar la productividad y acortar el tiempo de desarrollo, el emulador H2 está disponible incluso mientras el hardware está sin conexión.

- Tipo de trabajo: `Simulation`
- Formato de datos: `quantinuum.openqasm.v1`
- Id. de destino:
  - Emulador H2-1: `quantinuum.sim.h2-1e`
  - Emulador H2-2: `quantinuum.sim.h2-2e`
- Perfil de ejecución de destino: [QIR Adaptive RI](#)

El uso del emulador de System Model H2 se ofrece de forma gratuita con una suscripción de hardware. Para obtener más información, consulte [Precios de Azure Storage](#).

## Modelo de sistema H2

La generación Quantinuum System Model H2 de equipos cuánticos, con tecnología de Honeywell, está formada por un dispositivo de carga cuántica acoplado (QCCD) con dos secciones lineales conectadas y actualmente tiene 1 máquina, el H2-1. Puede encontrar más información en la *hoja* de datos del producto System Model H2 que se encuentra en la [página System Model H2](#) [↗](#). Se recomienda a los usuarios probar la compatibilidad de su código mediante el envío de trabajos a un [comprobador de sintaxis](#) y el [emulador del modelo de sistema H2](#) antes de enviarlos a las target máquinas.

Si un usuario envía un trabajo a la máquina H2-1 y la máquina H2-1 no está disponible, el trabajo permanece en la cola de esa máquina hasta que la máquina esté disponible.

El hardware del modelo del sistema H2 se actualiza continuamente a lo largo del ciclo de vida del producto. A los usuarios se les proporciona acceso al hardware más actualizado, avanzado y capaz que haya disponible.

- Tipo de trabajo: `Quantum Program`
- Formato de datos: `quantinuum.openqasm.v1`
- Id. de destino:
  - H2-1: `quantinuum.qpu.h2-1`
  - H2-2: `quantinuum.qpu.h2-2`
- Perfil de ejecución de destino: [QIR Adaptive RI](#)

## Especificaciones técnicas del modelo del sistema H2

Los detalles técnicos de System Model H2 se pueden encontrar en las hojas de datos del producto de Quantinuum en la página [System Model H2](#) [↗](#) junto con vínculos a la especificación de Quantinuum y repositorios de datos de volumen cuántico y cómo citar el uso de sistemas Quantinuum.

## Funcionalidades adicionales

Las funcionalidades adicionales disponibles a través de la API Quantinuum se enumeran aquí.

| Capability                                                             | Description                                                                             |
|------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <a href="#">Medición y restablecimiento a mitad de circuito (MCMR)</a> | Medir cúbits en medio de un circuito y reutilizarlos                                    |
| <a href="#">Puertas ZZ angulares arbitrarias</a>                       | Realice directamente rotaciones de puerta de ángulo arbitrario de 2 cúbits              |
| <a href="#">Puerta de entrelazamiento general SU(4)</a>                | Realice directamente rotaciones de puerta de ángulo arbitrario de 2 cúbits              |
| <a href="#">Parámetros de ruido del emulador</a>                       | Experimentación con los parámetros de ruido usados en los emuladores de Quantinuum      |
| <a href="#">Optimizaciones de TKET en Quantinuum Stack</a>             | Experimente activando diferentes niveles de optimización de TKET en la pila Quantinuum. |

Los usuarios pueden aprovechar estas funcionalidades adicionales a través de funciones de circuito o parámetros de transferencia en los proveedores de Azure Quantum Q# y Qiskit.

## Medición y restablecimiento a mitad de circuito

Mid-circuit Measurement and Reset (MCMR) permite a los usuarios medir cúbits en medio de un circuito y restablecerlos. Esto permite la funcionalidad para la corrección de errores cuánticos, así como la capacidad de reutilizar cúbits dentro del circuito.

Debido a la estructura de nivel interno de cúbits de iones interceptados, una medida de circuito medio puede dejar el cúbit en un estado no computacional. Todas las medidas del circuito medio deben ir seguidas de un restablecimiento si el cúbit se va a usar de nuevo en ese circuito. Los ejemplos de código siguientes muestran esto.

Cuando se mide un subconjunto de cúbits en medio del circuito, la información clásica de estas mediciones se puede usar para acondicionar elementos futuros del circuito. Los ejemplos también resaltan este uso.

Para obtener información sobre MCMR en sistemas Quantinuum, consulte la página [MCMR](#) en [Quantinuum Docs](#).

### MCMR con proveedor Q#

En Q#, la `MResetZ` función se puede usar tanto para medir un cúbit como para restablecerla. Para obtener más información sobre esta función, consulte [MResetZ](#) la documentación de Q#.

Q#

```
%%qsharp
import Std.Measurement.*;

operation ContinueComputationAfterReset() : Result[] {
 // Set up circuit with 2 qubits
 use qubits = Qubit[2];

 // Perform Bell Test
 H(qubits[0]);
 CNOT(qubits[0], qubits[1]);

 // Measure Qubit 1 and reset it
 let res1 = MResetZ(qubits[1]);

 // Continue additional computation, conditioned on qubits[1] measurement
 outcome
 if res1 == One {
 X(qubits[0]);
 }
 CNOT(qubits[0], qubits[1]);

 // Measure qubits and return results
 let res2 = Measure([PauliZ, PauliZ], qubits);
 return [res1, res2];
}
```

## Puertas ZZ de ángulo arbitrario

El conjunto de puertas nativas de Quantinuum incluye puertas de ángulo arbitrario ZZ. Sirve para reducir el número de puertas de 2 cúbits para muchos algoritmos cuánticos y secuencias de puertas. Para obtener información sobre las puertas ZZ de ángulo arbitrario en los sistemas de Quantinuum, visite la sección [Puertas de Ángulo Parametrizado ZZ](#) en [Quantinuum Docs](#).

Puertas ZZ de ángulo arbitrario con proveedor de Q#

En Q#, la puerta ZZ de ángulo arbitrario se implementa con la operación `Rzz`.

Q#

```
%%qsharp
import Std.Intrinsic.*;
import Std.Measurement.*;
import Std.Arrays.*;

operation ArbitraryAngleZZExample(theta : Double) : Result[] {
```

```
// Set up circuit with 2 qubits
use qubits = Qubit[2];

// Create array for measurement results
mutable resultArray = [Zero, size = 2];

H(qubits[0]);
Rz(theta, qubits[0]);
Rz(theta, qubits[1]);
X(qubits[1]);

// Add Arbitrary Angle ZZ gate
Rzz(theta, qubits[0], qubits[1]);

// Measure qubits and return results
for i in IndexRange(qubits) {
 resultArray w/= i <- M(qubits[i]);
}

return resultArray;
}
```

## Puerta de entrelazamiento general SU(4)

El conjunto de puertas nativas de Quantinuum incluye una puerta de entrelazamiento general SU(4). Tenga en cuenta que los circuitos cuánticos enviados al hardware cambian de base a la puerta ZZ de entrelazamiento completo y la puerta RZZ de ángulo arbitrario. Los circuitos solo cambian de base a la puerta de entrelazamiento general SU(4) si los usuarios lo deciden. Para obtener información sobre el Entangler General SU(4) en los sistemas Quantinuum, consulte la página [Puertas de Ángulo Parametrizado SU\(4\)](#) en [Quantinuum Docs](#).

Puerta de entrelazamiento general SU(4) con proveedor de Q#

En Q#, la puerta de entrelazamiento general SU(4) se implementa a través del perfil QIR de Quantinuum. Para usarlo, defina una función con una función intrínseca personalizada que coincida con la firma de perfil QIR y use esta función dentro de la `SU4Example` operación.

Para asegurarse de que el circuito se ejecuta con la puerta de entrelazamiento general SU(4), pase las siguientes opciones en la pila Quantinuum:

- `nativetq: Rxxyzz` para evitar los cambios de base a otras puertas nativas.
- `noreduce: True` para evitar optimizaciones adicionales del compilador (opcional).

Q#

```

%%qsharp
import Std.Math.*;

operation __quantum__qis__rxyzz__body(a1 : Double, a2 : Double, a3 : Double,
q1 : Qubit, q2 : Qubit) : Unit {
 body intrinsic;
}

operation SU4Example() : Result[] {
 use qs = Qubit[2];

 // Add SU(4) gate
 __quantum__qis__rxyzz__body(PI(), PI(), PI(), qs[0], qs[1]);

 MResetEachZ(qs)
}

```

Ahora compile la operación:

Python

```
MyProgram = qsharp.compile("GenerateRandomBit()")
```

Conéctese a Azure Quantum, seleccione la target máquina y configure los parámetros de ruido para el emulador:

Python

```

MyWorkspace = azure.quantum.Workspace(
 resource_id = "",
 location = ""
)

MyTarget = MyWorkspace.get_targets("quantinuum.sim.h2-1e")

Update TKET optimization level desired
option_params = {
 "nativetq": `Rxyzz`,
 "noreduce": True
}

```

Incluya la opción `noreduce` cuando envíe el trabajo.

Python

```

job = MyTarget.submit(MyProgram, "Submit a program with SU(4) gate", shots =
10, input_params = option_params)

```

```
job.get_results()
```

## Parámetros de ruido del emulador

Los usuarios tienen la opción de experimentar con los parámetros de ruido de los emuladores de Quantinuum. **Aquí solo se resaltan** algunos de los parámetros de ruido disponibles que muestran cómo pasar por los parámetros de los proveedores de Azure Quantum.

Para obtener más información sobre el conjunto completo de parámetros de ruido disponibles, consulte la página [Quantinuum Emulators \(Emuladores de Quantinuum\)](#) en la [documentación de Quantinuum](#).

Parámetros de ruido del emulador con el proveedor de Q#

En primer lugar, importe los paquetes necesarios e inicie el perfil base:

Python

```
import qsharp
import azure.quantum
qsharp.init(target_profile=qsharp.TargetProfile.Base)
```

A continuación, defina la función .

Q#

```
%%qsharp
import Std.Measurement.*;
import Std.Arrays.*;
import Std.Convert.*;

operation GenerateRandomBit() : Result {
 use target = Qubit();

 // Apply an H-gate and measure.
 H(target);
 return M(target);
}
```

y compile la operación:

Python

```
MyProgram = qsharp.compile("GenerateRandomBit()")
```

Conéctese a Azure Quantum, seleccione la target máquina y configure los parámetros de ruido para el emulador:

Python

```
MyWorkspace = azure.quantum.Workspace(
 resource_id = "",
 location = ""
)

MyTarget = MyWorkspace.get_targets("quantinuum.sim.h2-1e")

Update the parameter names desired
Note: This is not the full set of options available.
For the full set, see the Quantinuum Emulators page
option_params = {
 "error-params": {
 "p1": 4e-5,
 "p2": 3e-3,
 "p_meas": [3e-3, 3e-3],
 "p_init": 4e-5,
 "p_crosstalk_meas": 1e-5,
 "p_crosstalk_init": 3e-5,
 "p1_emission_ratio": 6e-6,
 "p2_emission_ratio": 2e-4
 }
}
```

Pase las opciones de ruido del emulador al enviar el trabajo:

Python

```
job = MyTarget.submit(MyProgram, "Experiment with Emulator Noise Parameters",
 shots = 10,
 input_params = option_params)

job.get_results()
```

Para desactivar el modelo de ruido del emulador, establezca la `error-model` opción en `False`. De manera predeterminada, se establece en `True`.

Python

```
option_params = {
 "error-model": False
```



```
}
```

Para usar el emulador de estabilizador, establezca la `simulator` opción en `stabilizer`. De manera predeterminada, se establece en `state-vector`.

Python

```
option_params = {
 "simulator": "stabilizer"
}
```

## Compilación de TKET en la pila Quantinuum

Los circuitos enviados a los sistemas Quantinuum, **excepto los envíos híbridos integrados**, se ejecutan automáticamente a través de pasos de compilación TKET para hardware Quantinuum. Esto permite optimizar automáticamente los circuitos para los sistemas Quantinuum y ejecutarlos de forma más eficaz.

Puede encontrar más información sobre los pasos de compilación específicos aplicados en la documentación, específicamente en la sección [pytket-quantinuum](#) [pytket-quantinuum Pasos de compilación](#).

En la pila de software Quantinuum, el nivel de optimización aplicado se establece con el parámetro `tket-opt-level`. *La configuración de compilación predeterminada para todos los circuitos enviados a los sistemas Quantinuum es el nivel de optimización 2.*

Los usuarios que quieran experimentar con los pasos de compilación TKET y ver las optimizaciones que se aplicarían a sus circuitos *antes* de la envío de trabajos pueden consultar el cuaderno `Quantinuum_compile_without_api.ipynb` en la [pytket-quantinuum carpeta Ejemplos](#).

Para desactivar la compilación de TKET en la pila, se puede definir una opción diferente, `no-opt`, en `True` dentro de `option_params`. Por ejemplo, `"no-opt": True`.

Para obtener más información sobre `pytket`, consulte los vínculos siguientes:

- [pytket](#)
- [pytket Manual de usuario](#)

Compilación de TKET con el proveedor de Q#

En primer lugar, importe los paquetes necesarios e inicie el perfil base:

Python

```
import qsharp
import azure.quantum
qsharp.init(target_profile=qsharp.TargetProfile.Base)
```

A continuación, defina la función .

Q#

```
%%qsharp
import Std.Measurement.*;
import Std.Arrays.*;
import Std.Convert.*;

operation GenerateRandomBit() : Result {
 use target = Qubit();

 // Apply an H-gate and measure.
 H(target);
 return M(target);
}
```

y compile la operación:

Python

```
MyProgram = qsharp.compile("GenerateRandomBit()")
```

Conéctese a Azure Quantum, seleccione la target máquina y configure los parámetros de ruido para el emulador:

Python

```
MyWorkspace = azure.quantum.Workspace(
 resource_id = "",
 location = ""
)

MyTarget = MyWorkspace.get_targets("quantinuum.sim.h2-1e")

Update TKET optimization level desired
option_params = {
 "tket-opt-level": 1
}
```

Pase la opción de optimización al enviar el trabajo:

Python

```
job = MyTarget.submit(MyProgram, "Experiment with TKET Compilation", shots =
10, input_params = option_params)
job.get_results()
```

## Especificaciones técnicas

Los detalles técnicos del System Model H2 y de los emuladores System Model H2 se pueden encontrar en la página [Guía del usuario de hardware](#) junto con las especificaciones de Quantinuum y los repositorios de datos de volumen cuántico, y [Cómo citar Quantinuum Systems](#).

## Disponibilidad de destino

Los equipos cuánticos de Quantinuum están diseñados para actualizarse continuamente, lo que permite a los clientes tener acceso a las funcionalidades de hardware más recientes, ya que Quantinuum mejora continuamente las fidelidades de las puertas, los errores de memoria y la velocidad del sistema.

El hardware de Quantinuum recorre cíclicamente los períodos comerciales y los períodos de desarrollo. En los períodos comerciales, el hardware está disponible para procesar los trabajos mediante un sistema de colas. En los períodos de desarrollo, el hardware está sin conexión mientras se aplican las actualizaciones.

Todos los meses se envía un calendario a los usuarios de Quantinuum con información sobre los períodos comerciales y de desarrollo. Si no ha recibido este calendario, envíe un correo electrónico a [QCsupport@quantinuum.com](mailto:QCsupport@quantinuum.com).

El estado de un target indica su capacidad actual de procesar trabajos. Entre los posibles estados de un target elemento se incluyen:

- **Disponible:** target está en línea, procesa los trabajos enviados y acepta los nuevos.
- **Degradado:** target está aceptando trabajos, pero no los está procesando en este momento.
- **No disponible:** target está sin conexión y no acepta nuevos envíos de trabajo.

Para el equipo targets cuántico Quantinuum, **Disponible** y **Degradado** corresponden a períodos comerciales, mientras que **No disponible** corresponde a los períodos de desarrollo en los que la máquina está sin conexión para las actualizaciones.

La información de estado actual se puede recuperar de la pestaña *Proveedores* de un área de trabajo en [Azure Portal](#).

## Pricing

Para ver los planes de facturación de Quantinuum, consulte [Precios de Azure Quantum](#).

## Límites y cuotas

Se realiza un seguimiento de las cuotas de Quantinuum en función de la unidad de crédito de uso de QPU, *Crédito cuántico de hardware (HQC)*, para trabajos enviados a equipos cuánticos de Quantinuum, y créditos cuánticos de hardware de emulador (eHQC) para trabajos enviados a emuladores.

Los HQC y los eHQC se usan para calcular el costo de ejecutar un trabajo y se calculan en función de la fórmula siguiente:

$$HQC = 5 + C(N_{1q} + 10N_{2q} + 5N_m)/5000$$

where:

- $N_{1q}$  es el número de operaciones de un cúbit en un circuito.
- $N_{2q}$  es el número de operaciones nativas de dos cúbits en un circuito. La puerta nativa equivale a CNOT de hasta varias puertas de un cúbit.
- $N_m$  es el número de operaciones de preparación y medida del estado (SPAM) en un circuito, incluida la preparación del estado implícito inicial, las medidas intermedias y finales, y los restablecimientos de estado.
- $C$  es el número de disparos.

### ⓘ Nota

El coste total de los HQC incluye todas las puertas y medidas a través de cualquier rama condicional o flujo de control. Esto puede tener un mayor impacto en los trabajos híbridos integrados.

Las cuotas se basan en la selección del plan y se pueden aumentar con una incidencia de soporte técnico. Para ver los límites y cuotas actuales, vaya a la sección **Operaciones** y seleccione la hoja **Cuotas** del área de trabajo en [Azure Portal](#). Para más información, consulte [Cuotas de Azure Quantum](#).

# Directiva de soporte técnico para Quantinuum en Azure Quantum

Artículo • 16/10/2024

En este artículo se describe la directiva de soporte técnico de Microsoft que se aplica cuando se usa el proveedor Quantinuum en Azure Quantum. El artículo se aplica a cualquiera de los targets de este proveedor.

Si usa el proveedor Quantinuum y encuentra problemas inesperados, puede [crear una incidencia de soporte técnico de Azure](#).

En algunas situaciones, el equipo de soporte de Azure tendrá que redirigirle al equipo de soporte de Quantinuum. Puede recibir una respuesta más rápida poniéndose en contacto con Quantinuum directamente en [QCSupport@quantinuum.com](mailto:QCSupport@quantinuum.com).

## Directiva de soporte técnico de Quantinuum: versión preliminar pública

El objetivo del soporte técnico es identificar y corregir defectos o mal funcionamientos que provocan que el hardware target Quantinuum no se realice de acuerdo con las especificaciones y documentación acordadas ("Problemas"). El soporte técnico solo cubre la versión publicada actual disponible con carácter general para los clientes. Aunque nuestros expertos cuánticos pueden ayudarle a solucionar problemas de algoritmos, no somos responsables de ningún problema, incidencia o defecto del algoritmo. Para minimizar los problemas de programación, los usuarios pueden ejecutar sus algoritmos con el corrector de sintaxis antes de ejecutarlos en el hardware.

### 1.1 Ponerse en contacto con el soporte técnico en Quantinuum

Para crear una solicitud de soporte técnico para Quantinuum, envíe un informe del incidente por correo electrónico a [QCSupport@quantinuum.com](mailto:QCSupport@quantinuum.com). El ingeniero de guardia le responderá en el plazo establecido en el Acuerdo de Nivel de Servicio correspondiente (tabla 2). Tenga en cuenta que solo se notificarán al equipo de Quantinuum y tendrán el plazo de resolución establecido en el Acuerdo de Nivel de Servicio adecuado los informes de incidentes que contengan toda la información siguiente. Los informes sin la información necesaria no desencadenan esta respuesta. Todas las solicitudes deben proporcionarse en inglés y se responden en inglés.

Tabla 1: Información de contacto de soporte técnico y horas de funcionamiento estándar

 Expandir tabla

| Correo electrónico                                                   | Horas de funcionamiento estándar                   |
|----------------------------------------------------------------------|----------------------------------------------------|
| <a href="mailto:QCsupport@quantinum.com">QCsupport@quantinum.com</a> | 8 a. m. – 5 p. mst en días laborables <sup>1</sup> |

<sup>1</sup> "Días laborables" significa de lunes a viernes, pero excluyendo los días festivos nacionales, legales o bancarios.

El informe de incidentes requiere la siguiente información:

- **Descripción:** una descripción detallada del problema que tiene, lo que está observando y los pasos que ya ha dado para clasificar y comprender el problema.
- **Contacto principal:** un nombre de contacto principal y un número de teléfono en caso de que se requiera más información.
- **Gravedad del incidente:** la gravedad del incidente, según nuestras definiciones de gravedad.

Una vez que se presenta un informe, se notifica al equipo de Quantinum y responde dentro de la ventana del Acuerdo de Nivel de Servicio requerido. Recibirá una confirmación del problema una vez que se haya recibido y se haya notificado al ingeniero de guardia. En función de la gravedad, el ingeniero de guardia puede solicitar más información o puede llamarle al número de contacto que aparece en el informe.

Durante las sesiones reservadas (dedicadas), Quantinum proporciona al menos un especialista cuántico disponible para el soporte al cliente para la totalidad de la sesión dedicada. Quantinum se reserva el derecho a cambiar al especialista durante la sesión, pero hará todos los esfuerzos comercialmente razonables para asegurarse de que el nuevo especialista está preparado para seguir proporcionando soporte técnico.

Durante las ejecuciones en cola, si el trabajo se ejecuta en la cola general y encuentra un problema, puede enviar la solicitud de problema en línea en:

[QCsupport@quantinum.com](mailto:QCsupport@quantinum.com). El plazo de respuesta previsto es de tres (3) días laborables.

## 1.2 Acuerdo de Nivel de Servicio de respuesta e índice de gravedad de Quantinum

Si el ingeniero de Quantinum determina que el problema no está dentro de los límites de nuestros compromisos de respuesta a incidentes o no tiene la clasificación de

gravedad adecuada, puede optar a su entera discreción por ajustar la gravedad o dar por terminado el soporte técnico si la nueva gravedad no lo requiere.

Algunos problemas pueden ser más fáciles de solucionar que otros, y es posible que no podamos resolver completamente el problema con nuestra respuesta inicial. Si no podemos resolver razonablemente el problema, haremos todo lo posible para realizar una evaluación del problema y el tiempo de resolución estimado.

*Tabla 2: Definición del índice de gravedad*

[Expandir tabla](#)

| Gravedad   | Significado            | Descripción                                                                |
|------------|------------------------|----------------------------------------------------------------------------|
| Gravedad 1 | Impacto significativo  | Rendimiento del sistema degradado por debajo del nivel mínimo aceptado     |
| Gravedad 2 | Impacto urgente o alto | El estado del trabajo no se ha actualizado; problemas para recuperar datos |
| Gravedad 3 | No urgente             | Varios                                                                     |

En la tabla siguiente se muestra el Acuerdo de Nivel de Servicio de respuesta que cumple Quantinuum para los informes de incidentes de la gravedad correspondiente. Tenga en cuenta que estos definen la rapidez con la que responderemos a los problemas; no garantizan los tiempos de resolución.

*Tabla 3: Acuerdo de Nivel de Servicio de respuesta para varias gravedades*

[Expandir tabla](#)

| Gravedad   | Acuerdo de Nivel de Servicio de respuesta           |
|------------|-----------------------------------------------------|
| Gravedad 1 | Respuesta en 1 hora en horario comercial            |
| Gravedad 2 | Respuesta en 1 día laborable en horario comercial   |
| Gravedad 3 | Respuesta en 3 días laborables en horario comercial |

## Preguntas más frecuentes

**P: ¿Qué ocurre si creo un caso para el equipo de Soporte técnico de Azure y resulta que un proveedor de terceros**

## **(como Quantinuum) necesita intervenir para solucionar el problema?**

El ingeniero de soporte técnico crea un paquete de redirección automáticamente. Se trata de un documento PDF que contiene información sobre su caso y que puede proporcionar al equipo de soporte técnico de Quantinuum. El ingeniero de soporte técnico también le ofrece consejos e instrucciones sobre cómo ponerse en contacto con Quantinuum para continuar con la solución de problemas.

### **P: ¿Qué ocurre si creo un caso para el equipo de soporte técnico de Quantinuum y resulta que hay un problema con el servicio Azure Quantum?**

El equipo de soporte técnico de Quantinuum le ayuda a ponerse en contacto con Microsoft y proporcionarle un paquete de redirección. Se trata de un documento PDF que puede usar para continuar con su consulta con el equipo de Soporte técnico de Azure.

## **Aviso de declinación de responsabilidades sobre la información de terceros**

Los productos de otros fabricantes que se mencionan en este artículo han sido creados por compañías independientes de Microsoft. Microsoft no ofrece ninguna garantía, ya sea implícita o de otro tipo, sobre la confiabilidad o el rendimiento de dichos productos.

## **Declinación de responsabilidad sobre la información de contacto de terceros**

Microsoft proporciona información de contacto de otros proveedores para ayudarle a encontrar información adicional sobre este tema. Esta información de contacto puede cambiar sin previo aviso. Microsoft no garantiza la precisión de esta información de contacto de terceros.

---

## **Comentarios**

¿Le ha resultado útil esta página?

 **Sí**

 **No**





# Proveedor Rigetti

12/08/2025

Los [procesadores cuánticos Rigetti](#) son máquinas universales de modelo de puerta basadas en cúbits superconductores ajustables. Las características del sistema y las características del dispositivo incluyen funcionalidades de lectura mejoradas, una aceleración en los tiempos de procesamiento cuánticos, tiempos de puerta rápidos para varias familias de puertas entrelazadas, muestreo rápido a través del restablecimiento de registro activo y control paramétrico.

- Publicador: [Rigetti](#)
- Identificador de proveedor: `rigetti`

El proveedor Rigetti pone a disposición lo siguiente targets :

[Expandir tabla](#)

| Nombre de destino                              | Id. de destino            | Número de cúbits | Descripción                                                             |
|------------------------------------------------|---------------------------|------------------|-------------------------------------------------------------------------|
| <a href="#">Máquina virtual cuántica (QVM)</a> | rigetti.sim.qvm           | -                | Simulador de código abierto para programas Quil, Q#y Qiskit. Sin costo. |
| <a href="#">Ankaa-3</a>                        | rigetti.qpu.ankaa-3       | 84 cúbits        |                                                                         |
| <a href="#">Cepheus-1-36Q</a>                  | rigetti.qpu.cepheus-1-36q | 36 qubits        |                                                                         |

 **Nota**

Los simuladores y hardware targets de Rigetti no admiten programas Cirq.

El targets de Rigetti corresponde a un perfil **QIR Base**. Para más información sobre este target perfil y sus limitaciones, consulte [Descripción target de los tipos de perfil en Azure Quantum](#).

## Equipos de Quantum

Todas las [QPU](#) disponibles públicamente de Rigetti están disponibles a través de Azure Quantum. Esta lista está sujeta a cambios sin previo aviso.

### Ankaa-3

Un procesador cuántico de 84 cúbits.

- Tipo de trabajo: Quantum Program
- Formato de datos: `rigetti.quil.v1`, `rigetti.qir.v1`
- Id. de destino: `rigetti.qpu.ankaa-3`
- Perfil de ejecución de destino: [QIR Base](#)

## Cepheus-1-36Q

Un procesador cuántico de 36 qubits construido con una matriz de chips de 9 qubits dispuestos en mosaico.

- Tipo de trabajo: Quantum Program
- Formato de datos: `rigetti.quil.v1`, `rigetti.qir.v1`
- Id. de destino: `rigetti.qpu.cepheus-1-36q`
- Perfil de ejecución de destino: [QIR Base](#)

## Simuladores

La [máquina virtual cuántica \(QVM\)](#) es un simulador de código abierto para [Quil](#).

`rigetti.sim.qvm` target acepta un [programa](#) Quil como texto y ejecuta ese programa en QVM hospedado en la nube y devuelve resultados simulados.

- Tipo de trabajo: Simulation
- Formatos de datos: `rigetti.quil.v1`, `rigetti.qir.v1`
- Id. de destino: `rigetti.sim.qvm`
- Perfil de ejecución de destino: [QIR Base](#)
- Precios: gratis (0 USD)

## Precios

Para ver el plan de facturación de Rigetti, consulte [Precios de Azure Quantum](#).

## Formato de entrada

Todos Rigetti targets actualmente aceptan dos formatos:

- `rigetti.quil.v1`, que es el texto de un programa de [Quil](#).
- `rigetti.qir.v1`, que es el bitcode de QIR.

Todos los targets también toman el parámetro entero opcional `count` para definir el número de capturas que se van a ejecutar. Si se omite, el programa solo se ejecuta una vez.

## Quil

All Rigetti targets acepta el `rigetti.quil.v1` formato de entrada, que es el texto de un [programa Quil](#), que representa el lenguaje de instrucción cuántica. De forma predeterminada, los programas se compilan mediante [quilk](#) antes de ejecutarse. Sin embargo, quilc no admite las características de control de nivel de pulso ([Quil-T](#)), por lo que, si desea usar esas características, debe proporcionar un programa de [native Quil](#) (que también contiene Quil-T) como entrada y especificar el parámetro de entrada `skipQuilc: true`.

Para facilitar la construcción de un programa de Quil, puede usar [pyQuil](#) junto con el paquete [pyquil-for-azure-quantum](#). Sin este paquete, `pyQuil` se puede usar para *construir* programas de Quil, pero no para enviarlos a Azure Quantum.

## QIR

Todo el hardware Rigetti admite la ejecución de trabajos compatibles con Quantum Intermediate Representation (QIR) con el perfil [QIR Base, v1](#) como `rigetti.qir.v1`. QIR proporciona una interfaz común que admite muchos lenguajes cuánticos y target plataformas para el cálculo cuántico y permite la comunicación entre lenguajes y máquinas de alto nivel. Por ejemplo, puede enviar trabajos de Q#, Quil o Qiskit al hardware Rigetti, y Azure Quantum controla la entrada automáticamente. Para obtener más información, vea [Quantum Intermediate Representation](#).

## Selección del formato de entrada correcto

¿Debe usar Quil u otro lenguaje compatible con QIR? Todo depende del caso de uso final. QIR es más accesible para muchos usuarios, mientras que Quil es más eficaz hoy en día.

Si usa Qiskit, Q#, u otro kit de herramientas que admita la generación de QIR, y la aplicación funciona en Rigetti targets a través de Azure Quantum, QIR es adecuado para usted. QIR tiene una especificación en constante evolución, y Rigetti sigue aumentando el soporte para programas QIR más avanzados a medida que pasa el tiempo, lo que no se puede compilar hoy puede bien compilar mañana.

Por otro lado, los programas de Quil expresan el conjunto completo de funcionalidades disponibles para los usuarios de sistemas Rigetti desde cualquier plataforma, incluido Azure Quantum. Si desea adaptar la descomposición de las puertas cuánticas o escribir programas en

el nivel de pulso, querrá trabajar en Quil, ya que esas funcionalidades aún no están disponibles a través de QIR.

## Ejemplos

La manera más fácil de enviar trabajos de Quil es usar el paquete [pyquil-for-azure-quantum](#), ya que permite usar las herramientas y la documentación de la biblioteca [pyQuil](#).

También puede construir programas de Quil manualmente y enviarlos directamente mediante el paquete `azure-quantum`.

### Uso de pyquil-for-azure-quantum

Python

```
from pyquil.gates import CNOT, MEASURE, H
from pyquil.quil import Program
from pyquil.quilbase import Declare
from pyquil_for_azure_quantum import get_qpu, get_qvm

Note that some environment variables must be set to authenticate with Azure
Quantum
qc = get_qvm() # For simulation

program = Program(
 Declare("ro", "BIT", 2),
 H(0),
 CNOT(0, 1),
 MEASURE(0, ("ro", 0)),
 MEASURE(1, ("ro", 1)),
).wrap_in_numshots_loop(5)

Optionally pass to_native_gates=False to .compile() to skip the compilation
stage
result = qc.run(qc.compile(program))
data_per_shot = result.readout_data["ro"]
Here, data_per_shot is a numpy array, so you can use numpy methods
assert data_per_shot.shape == (5, 2)
ro_data_first_shot = data_per_shot[0]
assert ro_data_first_shot[0] == 1 or ro_data_first_shot[0] == 0

Let's print out all the data
print("Data from 'ro' register:")
for i, shot in enumerate(data_per_shot):
 print(f"Shot {i}: {shot}")
```

# Directiva de soporte técnico para Rigetti en Azure Quantum

Artículo • 16/10/2024

En este artículo se describe la directiva de soporte técnico de Microsoft que se aplica cuando se usa el proveedor Rigetti en Azure Quantum. El artículo se aplica a cualquiera de las [targets](#) ofertas del proveedor rigetti.

## Puntos de contacto de soporte técnico

### Soporte técnico de Azure

En general, los problemas con el proveedor Rigetti en Microsoft Azure Quantum se abordan mejor con el equipo de soporte técnico de Azure Quantum directamente mediante la [creación de un caso de soporte técnico de Azure](#). Este equipo tiene todos los detalles necesarios para aislar el problema que está experimentando y dirigir el informe al punto de contacto adecuado. El equipo de soporte técnico de Azure también tiene los medios para comunicarse con el soporte técnico de Rigetti fuera del horario comercial para los problemas más urgentes.

### Compatibilidad con Rigetti

En algunas situaciones, el equipo de soporte de Azure tendrá que redirigirle al equipo de soporte técnico de Rigetti. Puede recibir una respuesta más rápida al ponerse en contacto con Rigetti directamente en <https://rigetti.zendesk.com/hc/>.

## Escala de tiempo del soporte técnico

Rigetti siempre intentará resolver los problemas con la integración de Microsoft Azure tan rápido como sea posible. Algunos problemas tardarán más tiempo en solucionarse que otros, pero en las tablas siguientes se indica el tiempo que puede esperar para una respuesta inicial de Rigetti después de señalar un problema.

[Expandir tabla](#)

| Gravedad | Definición                                                     | Acuerdo de Nivel de Servicio de respuesta |
|----------|----------------------------------------------------------------|-------------------------------------------|
| Alto     | Error en la conectividad; degradación completa del rendimiento | 1 día laborable                           |
| Media    | Degradación del rendimiento                                    | 2 días laborables                         |
| Bajo     | Preguntas y problemas relacionados con la aplicación           | 5 días laborables                         |

Los *días laborables* se definen como de lunes a viernes, sin contar los días festivos de EE. UU. Por ejemplo, si se notifica un incidente de gravedad media el martes, se devolverá una respuesta al final de la jornada laboral del jueves, si los días entre medio también son laborables.

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Vectores y matrices en la computación cuántica

Artículo • 01/04/2025

Cierta familiaridad con el álgebra lineal es esencial para comprender la computación cuántica. En este artículo se presentan los conceptos básicos del álgebra lineal y cómo trabajar con vectores y matrices en la computación cuántica.

## Vectores

Un vector de columna, o abreviadamente vector,  $v$  de dimensión (o tamaño)  $n$  es una colección de  $n$  números complejos  $(v_1, v_2, \dots, v_n)$  organizados como una columna:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

La norma de un vector  $v$  se define como  $\sqrt{v^\dagger v}$ . Un vector se denomina *vector de unidad* si su norma es 1.

El *adjunto* de un vector  $v$  es un vector de fila que se indica como  $v^\dagger$  y se define como la transposición conjugada de  $v$ . Para un vector de columna  $v$  de dimensión  $n$ , el adjunto es un vector de fila de dimensión  $1 \times n$ :

$$\begin{bmatrix} v_1 & \vdots & v_n \end{bmatrix}^\dagger = [v_1^* \quad \cdots \quad v_n^*]$$

donde  $v_i^*$  denota el conjugado complejo de  $v_i$ .

Con el álgebra lineal, el estado de un cúbit  $\psi = a|0\rangle + b|1\rangle$  se describe como un vector de estado cuántico  $\begin{bmatrix} a \\ b \end{bmatrix}$ , donde  $|a|^2 + |b|^2 = 1$ . Para obtener más información, consulte [El cúbit](#).

## Producto escalar

Dos vectores se pueden multiplicar con el *producto escalar*, también conocido como *producto interior*. Como indica el nombre, el resultado del producto escalar de dos vectores es un escalar. El producto escalar proporciona la proyección de un vector a otro y se usa para expresar un vector como una suma de otros vectores más sencillos. El



producto escalar entre dos vectores columna  $u$  y  $v$  se denota como  $\langle u, v \rangle = u^\dagger v$  y se define como

$$\langle u, v \rangle = u^\dagger v = [u_1^* \quad \cdots \quad u_n^*] \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = u_1^* v_1 + \cdots + u_n^* v_n.$$

Con el producto escalar, la norma de un vector  $v$  se puede escribir como  $\sqrt{\langle v, v \rangle}$ .

Puede multiplicar un vector por un número  $a$  para formar un nuevo vector cuyas entradas se multiplican por  $a$ . También se pueden sumar dos vectores  $u$  y  $v$  para formar un nuevo vector cuyas entradas son la suma de las entradas  $u$  y  $v$ . Estas son las operaciones siguientes:

$$au + bv = \begin{bmatrix} au_1 + bv_1 \\ au_2 + bv_2 \\ \vdots \\ au_n + bv_n \end{bmatrix}$$

## Matrices

Una matriz de tamaño  $m \times n$  es una colección de  $m \cdot n$  números complejos organizados en  $m$  filas y  $n$  columnas, como se muestra a continuación:

$$M = \begin{bmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ M_{21} & M_{22} & \cdots & M_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{bmatrix}$$

### ⓘ Nota

Tenga en cuenta que un vector de dimensión  $n$  es simplemente una matriz de tamaño  $n \times 1$ .

Las operaciones cuánticas se representan mediante *matrices cuadradas*, es decir, el número de filas y columnas es igual. Por ejemplo, las operaciones de un solo cúbit se representan mediante matrices  $2 \times 2$ , como las matrices de Pauli  $X$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

### 💡 Sugerencia

En Q#, la operación Pauli  $X$  se representa mediante la  $\text{X}$  operación.

Al igual que con los vectores, puede multiplicar una matriz con un número  $c$  para obtener una nueva matriz donde cada entrada se multiplica con  $c$  y se pueden agregar dos matrices del mismo tamaño para generar una nueva matriz cuyas entradas son la suma de las entradas respectivas de las dos matrices.

## Multiplicación de matrices

También puede multiplicar una matriz  $M$  de  $m \times n$  y una matriz  $N$  de dimensión  $n \times p$  para obtener una nueva matriz  $P$  de dimensión  $m \times p$  de la siguiente manera:

$$\begin{aligned} & \begin{bmatrix} M_{11} & M_{12} & \cdots & M_{1n} \\ M_{21} & M_{22} & \cdots & M_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{m1} & M_{m2} & \cdots & M_{mn} \end{bmatrix} \begin{bmatrix} N_{11} & N_{12} & \cdots & N_{1p} \\ N_{21} & N_{22} & \cdots & N_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ N_{n1} & N_{n2} & \cdots & N_{np} \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & \cdots & P_{1p} \\ P_{21} & P_{22} & \cdots & P_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ P_{m1} & P_{m2} & \cdots & P_{mp} \end{bmatrix} \end{aligned}$$

donde las entradas de  $P$  son  $P_{ik} = \sum_j M_{ij}N_{jk}$ . Por ejemplo, la entrada  $P_{11}$  es el producto escalar de la primera fila de  $M$  con la primera columna de  $N$ . Tenga en cuenta que, dado que un vector es simplemente un caso especial de una matriz, esta definición se extiende a la multiplicación matriz-vector.

## Tipos especiales de matrices

Una matriz cuadrada especial es la *matriz de identidad*, denotada  $\mathbb{I}$ , que tiene todos sus elementos diagonales iguales a 1 y los elementos restantes iguales a 0:

$$\mathbb{I} =$$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

.

Para una matriz cuadrada  $A$ , una matriz  $B$  es su *inversa* si  $AB = BA = \mathbb{I}$ . Si una matriz  $A$

tiene un inverso, la matriz inversa es única y se escribe como  $A^{-1}$ .

Para cualquier matriz  $M$ , la transposición contigua o conjugada de  $M$  es una matriz  $N$ , tal como  $N_{ij} = M_{ji}^*$ . El elemento contiguo de  $M$  se indica como  $M^\dagger$ .

Una matriz  $U$  es *unitaria* si  $UU^\dagger = U^\dagger U = \mathbb{I}$  o de forma equivalente,  $U^{-1} = U^\dagger$ . Una propiedad importante de las matrices unitarias es que conservan la norma de un vector. Esto ocurre porque

$$\langle v, v \rangle = v^\dagger v = v^\dagger U^{-1} U v = v^\dagger U^\dagger U v = \langle U v, U v \rangle.$$

#### ⚠ Nota

Las operaciones cuánticas se representan mediante matrices unitarias, que son matrices cuadradas cuyo adjunto es igual a su inversa.

Una matriz  $M$  se denomina *Hermitiana* si  $M = M^\dagger$ .

En la computación cuántica, básicamente solo se encuentran dos matrices: hermitica y unitaria.

## Producto tensorial

Otra operación importante es el producto tensor, también llamado producto directo de matrices o producto de Kronecker.

Considere los dos vectores  $v = \begin{bmatrix} a \\ b \end{bmatrix}$  y  $u = \begin{bmatrix} c \\ d \end{bmatrix}$ . Su producto tensorial se indica como  $v \otimes u$ , y da como resultado una matriz de bloques.

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a \begin{bmatrix} c \\ d \end{bmatrix} & b \begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix}$$

#### ⚠ Nota

Tenga en cuenta que el producto tensor se distingue de la multiplicación de matrices, que es una operación completamente diferente.

El producto tensor se usa para representar el estado combinado de varios cúbits. La eficacia real de la computación cuántica proviene del aprovechamiento de varios cúbits para realizar cálculos. Para más detalles, consulte Operaciones en varios cúbits.

## Contenido relacionado

- El cúbit
- [Varios cúbits](#)
- [Notación de Dirac](#)
- Mediciones de Pauli

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# El cúbit en la computación cuántica

Artículo • 14/02/2025

Así como los bits son el objeto fundamental de la información en la computación clásica, los [cúbits](#) (bits cuánticos) son el objeto fundamental de la información en la computación cuántica. Para entender esta correspondencia, este artículo examina el ejemplo más sencillo: un solo cúbit.

## Representación de un cúbit

Mientras que un bit, o dígito binario, puede tener el valor 0 o 1, un cúbit puede tener un valor que sea de 0, 1 o una superposición cuántica de 0 y 1.

El estado de un solo cúbit puede describirse mediante un vector de columna bidimensional de norma unitaria, es decir, la magnitud al cuadrado de sus entradas debe sumar 1. Este vector, llamado vector de estado cuántico, contiene toda la información necesaria para describir el sistema cuántico de un cúbit, al igual que un solo bit contiene toda la información necesaria para describir el estado de una variable binaria. Para conocer los conceptos básicos de vectores y matrices en la computación cuántica, consulte [Vector y matrices](#).

Cualquier vector de columna bidimensional de números reales o complejos con norma 1 representa un posible estado cuántico mantenido por un cúbit. Así,  $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$  representa un estado de cúbit si  $\alpha$  y  $\beta$  son números complejos que satisfacen  $|\alpha|^2 + |\beta|^2 = 1$ .

Algunos ejemplos de vectores de estado cuántico válidos que representan cúbits son  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  y  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Estos dos vectores forman una base para el espacio vectorial que describe el estado del cúbit. Esto significa que cualquier vector de estado cuántico puede escribirse como una suma de estos vectores de base. En concreto, el vector  $\begin{bmatrix} x \\ y \end{bmatrix}$  puede escribirse como  $x \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Aunque cualquier rotación de estos vectores serviría como base perfectamente válida para el cúbit, elegimos dar prioridad a esta, a la que llamamos *base computacional*.

Tomamos estos dos estados cuánticos como correspondientes a los dos estados de un bit clásico, a saber, 0 y 1. La convención estándar es elegir:

$$0 \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$1 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

aunque también se podría optar por la opción contraria. Por lo tanto, fuera del número infinito de posibles vectores de estado cuántico de un solo cúbit, solo dos corresponden a estados de bits clásicos; todos los demás estados cuánticos no.

## Medida de un cúbit

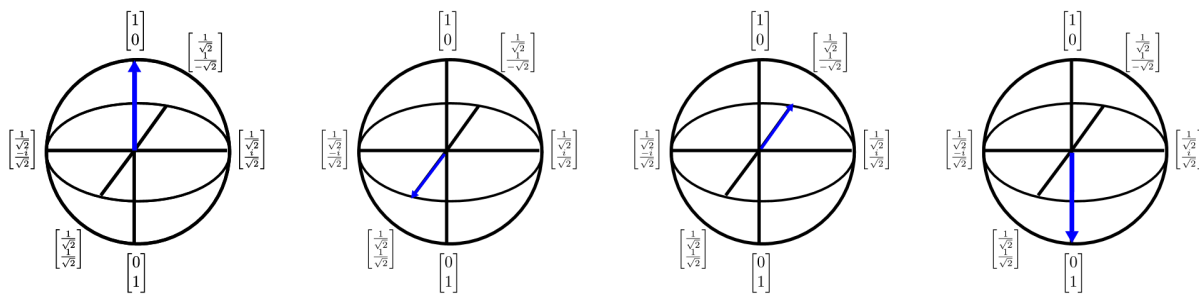
Ahora que sabemos cómo representar un cúbit, veremos el concepto de [medida](#) para hacernos una idea de lo que representan estos estados. Una medida corresponde a la idea informal de "mirar" un cúbit, que inmediatamente contrae el estado cuántico a uno de los dos estados clásicos  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  o  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Cuando se mide un cúbit dado por el vector de estado cuántico  $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ , se obtiene el resultado 0 con probabilidad  $|\alpha|^2$  y el resultado 1 con probabilidad  $|\beta|^2$ . En el resultado 0, el nuevo estado del cúbit es  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ ; en el resultado 1, su estado es  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ . Observe que estas probabilidades suman 1 debido a la condición de normalización  $|\alpha|^2 + |\beta|^2 = 1$ .

Las propiedades de la medida también significan que el signo global del vector de estado cuántico es irrelevante. Negar un vector equivale a  $\alpha \rightarrow -\alpha$  y  $\beta \rightarrow -\beta$ . Como la probabilidad de medir 0 y 1 depende de la magnitud al cuadrado de los términos, la inserción de estos signos no cambia en absoluto las probabilidades. Estas fases se denominan comúnmente "[fases globales](#)" y, más generalmente, pueden tener la forma  $e^{i\phi}$  en lugar de solo  $\pm 1$ .

Una propiedad importante final de la medida es que no daña necesariamente todos los vectores de estado cuántico. Si uno comienza con un cúbit en el estado  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ , que corresponde al estado *clásico* 0, medir este estado siempre produce el resultado 0 y deja el estado cuántico sin cambios. En este sentido, si solo hay bits clásicos (por ejemplo, cúbits que son  $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$  o  $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ ), la medición no daña el sistema. Esto significa que podemos replicar los datos clásicos y manipularlos en un equipo cuántico igual que se podría hacer en un equipo clásico. Sin embargo, la capacidad de almacenar información en ambos estados a la vez es lo que eleva la computación cuántica más allá de lo que es posible de forma clásica, y priva a los equipos cuánticos de la capacidad de copiar datos cuánticos indiscriminadamente. Consulte también el [teorema de no clonación](#).

# Visualización de cúbits y transformaciones mediante la esfera bloch

Los cúbits también pueden representarse en 3D mediante la esfera de [Bloch](#). La esfera de Bloch permite describir un estado cuántico de un solo cúbit (que es un vector complejo bidimensional) como un vector tridimensional de valor real. Esto es importante porque nos permite visualizar los estados de un solo cúbit y, por tanto, desarrollar un razonamiento que puede resultar muy valioso para entender los estados de varios cúbits (donde, lamentablemente, la representación de la esfera de Bloch no funciona). La esfera de Bloch puede visualizarse de la siguiente manera:



Las flechas de este diagrama muestran la dirección hacia la que apunta el vector de estado cuántico y cada transformación de la flecha puede considerarse como una rotación alrededor de uno de los ejes cardinales. Aunque es muy tentador pensar en un cálculo cuántico como en una secuencia de rotaciones, es un reto utilizar esa noción para diseñar y describir algoritmos. Para mitigar este problema, Q# proporciona un lenguaje para describir estas rotaciones.

## Operaciones de un solo cúbit

Los equipos cuánticos procesan los datos aplicando un conjunto universal de puertas cuánticas que pueden emular cualquier rotación del vector de estado cuántico. Esta noción de universalidad es similar a la noción de universalidad de la computación tradicional (por ejemplo, la clásica), en la que se considera que un conjunto de puertas es universal si toda transformación de los bits de entrada puede realizarse mediante un circuito de longitud finita. En la computación cuántica, las transformaciones válidas que se nos permite realizar en un cúbit son las transformaciones unitarias y la medida. La *operación adjunta* o la transpuesta conjugada compleja es de crucial importancia para la computación cuántica porque se necesita para invertir las transformaciones cuánticas.

Las operaciones de un solo cúbit o las puertas cuánticas de un solo cúbit se pueden clasificar en dos categorías: puertas de Clifford y puertas no Clifford. Las Puertas No Clifford constan solo de puerta  $T$  (también conocida como Puerta  $\pi/8$ ).

$$T = \begin{bmatrix} 1 & 0 & 0 & e^{i\pi/4} \end{bmatrix}.$$

El conjunto estándar de puertas Clifford de un solo cúbit, que se incluye de forma predeterminada en Q#, incluye

$$H = \frac{1}{\sqrt{2}}$$

$$\begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}$$

$$, \text{ \texttt{S} =}$$

$$\begin{bmatrix} 1 & 0 & 0 & i \end{bmatrix}$$

$$= T^2, \text{ \texttt{X} =}$$

$$\begin{bmatrix} 0 & 1 & 1 & 0 \end{bmatrix}$$

$$= HT^4H, \text{ \texttt{Z} =}$$

$$Y = \begin{bmatrix} 0 & -i & i & 0 \end{bmatrix} = T^2HT^4HT^6, \quad Z = \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix} = T^4.$$

Aquí, las operaciones  $X$ ,  $Y$  y  $Z$  se utilizan con especial frecuencia y se denominan operadores de [Pauli](#) en honor a su creador, Wolfgang Pauli. Junto con la puerta no Clifford (la puerta  $T$ ), estas operaciones pueden formularse para aproximar cualquier transformación unitaria sobre un solo cúbit.

Mientras que las anteriores son las puertas primitivas más populares para describir operaciones en el nivel lógico de la pila (piense en el nivel lógico como el nivel del algoritmo cuántico), a menudo resulta práctico considerar las operaciones menos básicas en el nivel algorítmico, por ejemplo, operaciones más cercanas al nivel de descripción de una función. Afortunadamente, Q# también tiene métodos para implementar unitarios de nivel superior que, a su vez, permiten implementar algoritmos generales sin descomponer explícitamente todo hasta las puertas Clifford y  $T$ .

La más sencilla de estas primitivas es la rotación de un solo cúbit. Normalmente se consideran tres rotaciones de un solo cúbit:  $R_x$ ,  $R_y$  y  $R_z$ . Para visualizar la acción de la rotación  $R_x(\theta)$ , por ejemplo, imagine que apunta con el pulgar derecho a lo largo de la dirección del eje  $x$  de la esfera de Bloch y gire el vector con su mano a través de un ángulo de  $\theta/2$  radianes. Este factor de confusión de 2 surge del hecho de que los vectores ortogonales están separados  $180^\circ$  cuando se trazan en la esfera de Bloch, pero en realidad están separados  $90^\circ$  grados geoméricamente. Las matrices unitarias correspondientes son:



$$R_z(\theta) = e^{-i\theta Z/2} = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix},$$

$$R_x(\theta) = e^{-i\theta X/2} = HR_z(\theta)H = \begin{bmatrix} \cos(\theta/2) & -i \sin(\theta/2) \\ -i \sin(\theta/2) & \cos(\theta/2) \end{bmatrix},$$

$$R_y(\theta) = e^{-i\theta Y/2} = SHR_z(\theta)HS^\dagger = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}.$$

Al igual que se pueden combinar tres rotaciones cualesquiera para realizar una rotación arbitraria en tres dimensiones, desde la representación de la esfera de Bloch se puede ver que cualquier matriz unitaria también se puede escribir como una secuencia de tres rotaciones. En concreto, para toda matriz unitaria  $U$  existe  $\alpha, \beta, \gamma, \delta$  de manera que  $U = e^{i\alpha}R_x(\beta)R_z(\gamma)R_x(\delta)$ . Por lo tanto,  $R_z(\theta)$  y  $H$  también forman un conjunto de puertas universales, aunque no es un conjunto discreto porque  $\theta$  puede tomar cualquier valor. Por esta razón, y debido a las aplicaciones en la simulación cuántica, estas puertas continuas son cruciales para la computación cuántica, especialmente a nivel de diseño de algoritmos cuánticos. Para lograr una implementación de hardware tolerante a los errores, en última instancia se compilarán en secuencias de compuertas discretas que se aproximen a estas rotaciones.

## Contenido relacionado

- [Varios cúbits](#)
- [Notación de Dirac](#)
- [Medidas de Pauli](#)
- [Puertas T y fábricas T](#)
- [Circuitos cuánticos](#)

## Comentarios

¿Le ha resultado útil esta página?

☐ Sí

☐ No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

# Operaciones en varios cúbits

Artículo • 16/09/2024

En este artículo se revisan las reglas que se usan para compilar estados de varios cúbits a partir de estados de un solo cúbit y se analizan las operaciones de puerta necesarias para incluir en un conjunto de puertas para formar un equipo cuántico universal de varios cúbits. Estas herramientas son necesarias para comprender los conjuntos de puertas que se usan habitualmente en Q# el código. También son importantes para obtener intuición sobre por qué los efectos cuánticos, como el entrelazamiento o la interferencia, representan la computación cuántica más eficaz que la computación clásica.

## Puertas de un solo cúbit y de varios cúbits

La verdadera potencia de la computación cuántica solo se vuelve evidente a medida que aumenta el número de cúbits. Los cúbits únicos poseen algunas características contra intuitivas, como la capacidad de estar en más de un estado en un momento dado. Sin embargo, si todo lo que tuviera en un equipo cuántico fuera puertas de un solo cúbit, una calculadora y ciertamente un superequipo clásico enanoría su potencia computacional.

La capacidad de computación cuántica surge, en parte, porque la dimensión del espacio vectorial de los vectores de estado cuántico crece exponencialmente con el número de cúbits. Esto significa que, aunque un solo cúbit se puede modelar de forma trivial, simular un cálculo cuántico de 50 cúbits podría insertar los límites de los superequipos existentes. Al aumentar el tamaño del cálculo solo un cúbit *adicional, se duplica* la memoria necesaria para almacenar el estado y se duplica *aproximadamente* el tiempo de cálculo. Esta duplicación rápida de la potencia computacional es la razón por la que un equipo cuántico con un número relativamente pequeño de cúbits puede superar con creces los superequipos más eficaces de la actualidad y del futuro para algunas tareas de cálculo.

## Estados de dos cúbits

Si tiene dos cúbits independientes, uno en el estado  $\psi = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$  y el otro en el estado  $\phi = \begin{bmatrix} \gamma \\ \delta \end{bmatrix}$ , el estado de dos cúbits correspondiente se da mediante el producto tensor (o [producto](#) Kronecker) de vectores, que se define como se indica a continuación.

$$\psi \otimes \phi = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \otimes \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \\ \beta \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta \end{bmatrix}.$$

Por lo tanto, dados dos estados  $\psi$  de un solo cúbit y  $\phi$ , cada una de las dimensiones 2, el estado  $\psi \otimes \phi$  de dos cúbits correspondiente es de 4 dimensiones. El vector

$$\begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix}$$

representa un estado cuántico en dos cúbits si

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1.$$

Por lo general, puede ver que el estado cuántico de  $n$  cúbits se representa mediante un vector de unidad  $v_1 \otimes v_2 \otimes \dots \otimes v_n$  de dimensión  $2 \cdot 2 \cdot 2 \dots = 2^n$  mediante esta construcción. Al igual que con los cúbits

únicos, el vector de estado cuántico de varios cúbits contiene toda la información necesaria para describir el comportamiento del sistema. Para obtener más información sobre los vectores y los productos tensoriales, vea [Vectores y matrices en computación cuántica](#).

La base computacional para los estados de dos cúbits está formada por los productos tensor de estados de base de un cúbit. Por ejemplo, tiene

$$00 \equiv$$

Missing \end{bmatrix}

$$\otimes \begin{bmatrix} 1 & 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

\,\qqquad 01

$$\begin{bmatrix} \equiv 1 \\ 00 \end{bmatrix}$$

$\otimes$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ 000 \end{bmatrix}$$

$\backslash, \backslash 10$

$$\begin{bmatrix} \equiv 0 \\ 11 \end{bmatrix}$$

$\otimes$

$$\begin{bmatrix} 0 \end{bmatrix}$$

amp;=

Missing \end{bmatrix}

$$\otimes \begin{bmatrix} 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 00 \\ 0 \\ 1 \end{bmatrix}$$

$$\quad \quad \quad \end{bmatrix} \quad \quad \quad \end{align} \quad \quad \quad$$

Tenga en cuenta que, aunque siempre puede tomar el producto tensor de dos estados de un solo cúbit para formar un estado de dos cúbits, no todos los estados cuánticos de dos cúbits se pueden escribir como el producto tensor de dos estados de un solo cúbit. Por ejemplo, no hay ningún estado  $\psi = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$  y  $\phi = \begin{bmatrix} \gamma \\ \delta \end{bmatrix}$  cuyo producto tensorial sea el estado

$$|\psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Este estado de dos cúbits, que no se puede escribir como el producto tensorial de los estados de un solo cúbit, se denomina "estado entrelazado". Se dice que los dos cúbits están [entrelazados](#). En términos generales, dado que el estado cuántico no se puede pensar como un producto tensorial de estados de cúbits únicos, la información que contiene el estado no se limita a ninguno de los cúbits individualmente. En su lugar, la información se almacena de forma no local en las correlaciones entre los dos estados. Esta no localidad de la información es una de las principales características distintivas de la computación cuántica sobre la computación clásica y es esencial para una serie de protocolos cuánticos, incluida la corrección de errores cuánticos.

## Medición de estados de dos cúbits

La medición de estados de dos cúbits es muy similar a las mediciones de un solo cúbit. La medición del estado

$$\begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix}$$

produce 00 con probabilidad  $|\alpha_{00}|^2$ , 01 con probabilidad  $|\alpha_{01}|^2$ , 10 con probabilidad  $|\alpha_{10}|^2$  y 11 con probabilidad  $|\alpha_{11}|^2$ . Los nombres de las variables  $\alpha_{00}$ ,  $\alpha_{01}$ ,  $\alpha_{10}$  y  $\alpha_{11}$

se eligieron expresamente para que esta conexión estuviera clara. Después de la medición, si el resultado es 00, el estado cuántico del sistema de dos cúbits se ha contraído y ahora está

$$|00\rangle \equiv \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

También es posible medir solo un cúbit de un estado cuántico de dos cúbits. Cuando se mide solo un cúbit de un estado de dos cúbits, el impacto de la medición es sutilmente diferente de medir dos cúbits. Esto se debe a que todo el estado no se contrae a un estado de base computacional, sino que solo se contrae a un subsistema. Es decir, medir un cúbit de un estado de dos cúbits solo contrae el subsistema relacionado con un estado de base computacional.

Para ver esto, considere la posibilidad de medir el primer cúbit del siguiente estado, que se forma aplicando la transformación *Hadamard*  $H$  en dos cúbits establecidos inicialmente en la  $|0\rangle$  estado:

$$H^{\otimes 2} \left( \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Missing  $\end{bmatrix}$

$$\otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix} \frac{1}{2}$$

$$\begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

$$0 \text{ } 1 \text{ } 1 \text{ } 1 \text{ } 1 = \begin{cases} \text{mapsto} \text{matriz} \text{ resultado } 0 \text{ } \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \text{ } \text{matriz} \end{cases}$$

$$0 \text{ } \text{matriz} \text{ } \text{resultado} = 1 \text{ } \& \{2\} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \sqrt{\frac{1}{2}} \text{ } \text{matriz} \end{cases}.$$

\$\$ Ambos resultados tienen una probabilidad del 50 % de producirse. Esto se puede intuir del hecho de que el estado cuántico antes de la medición no cambia si 0 se intercambia con 1 en el primer cúbit.

$$P(\text{outcome} = 1) = \sum_{e_k \text{ en el conjunto } S} \psi^\dagger e_k e_k^\dagger \psi.$$

En este artículo se usa el *formato little-endian* para etiquetar la base computacional. En el formato little-endian, primero se incluyen los bits menos significativos. Por ejemplo, el número cuatro en formato little-endian se representa mediante la cadena de bits 001.

La acción que tiene una medida de este tipo en el estado se puede expresar matemáticamente como

El lector cauteloso puede preocuparse de lo que sucede si el denominador es cero. Aunque este estado no está definido, no es necesario preocuparse por tales eventualidades porque la probabilidad es cero.

$$P(\text{medida del primer cúbit} = 1) = (\psi^\dagger e_1)(e_1^\dagger \psi) + (\psi^\dagger e_3)(e_3^\dagger \psi) = |e_1^\dagger \psi|^2 + |e_3^\dagger \psi|^2.$$

$$\frac{1}{4} \left| \right|$$

Missing \left or extra \right

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

$$|\text{right}|^2 = \frac{1}{2}.$$

que coincide perfectamente con nuestra intuición. Del mismo modo, el estado después de que el primer cúbit se mida como 1 se puede escribir como

$$\frac{\frac{e_1}{2} + \frac{e_3}{2}}{\sqrt{\frac{1}{2}}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

de nuevo de acuerdo con nuestra intuición.

## Operaciones de dos bits cúbits

Como en el caso de un solo cúbit, cualquier transformación unitaria es una operación válida en los cúbits. En general, una transformación unitaria en  $n$  cúbits es una matriz  $U$  de tamaño  $2^n \times 2^n$  (de modo que actúa sobre vectores de tamaño  $2^n$ ), tal como  $U^{-1} = U^\dagger$ . Por ejemplo, la puerta CNOT (controlled-NOT) es una puerta de dos cúbits de uso frecuente y se representa mediante la siguiente matriz unitaria:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

También podemos formar puertas de dos cúbits aplicando puertas de un solo cúbit en ambos cúbits. Por ejemplo, si aplica las puertas

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

y

$$\begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

a los cúbits primero y segundo, respectivamente, esto equivale a aplicar la unidad de dos cúbits dada por su producto tensor:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} e & f & g & h \end{bmatrix} = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{bmatrix}$$

Por lo tanto, puede formar puertas de dos cúbits tomando el producto tensor de algunas puertas de un solo cúbit conocidas. Algunos ejemplos de puertas de dos cúbits incluyen  $H \otimes I$ ,  $X \otimes I$  y  $X \otimes Z$ .

Tenga en cuenta que, aunque dos puertas de un solo cúbit definen una puerta de dos cúbits tomando su producto tensorial, lo contrario no es verdad. No todas las puertas de dos cúbits se pueden escribir como el producto tensorial de las puertas de un solo cúbit. Este tipo de puerta se denomina puerta de *entrelazamiento*. Un ejemplo de una puerta de entrelazamiento es la puerta CNOT.

La intuición detrás de una puerta no controlada se puede generalizar a puertas arbitrarias. Una puerta controlada en general es una puerta que actúa como identidad a menos que un cúbit específico sea 1. Se indica un unitario controlado, controlado en este caso en el cúbit etiquetado  $x$ , con un  $\Lambda_x(U)$ . Por ejemplo:  $\Lambda_0(U) |e_1\rangle \otimes |\psi\rangle = |e_1\rangle \otimes U|\psi\rangle$  y  $\Lambda_0(U) |e_0\rangle \otimes |\psi\rangle = |e_0\rangle \otimes |\psi\rangle$

, donde  $e_0, e_1$  son los vectores de base computacional para un solo qubit correspondiente a los valores 0 y 1. Por ejemplo, considere la siguiente puerta controlada —  $Z$  entonces puede expresar esto como

$$U_{\Lambda_0(Z)} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

$$= (I \otimes H) \text{CNOT} (I \otimes H).$$

La compilación de operaciones unitarias controladas de forma eficaz es un desafío importante. La manera más sencilla de implementar esto requiere formar una base de datos de versiones controladas de puertas fundamentales y reemplazar todas las puertas fundamentales de la operación unitaria original por su equivalente controlado. Esto suele ser bastante ineficiente y, a menudo, las conclusiones inteligentes se pueden usar para reemplazar solo algunas puertas por versiones controladas para lograr el mismo impacto. Por este motivo, el marco proporciona la capacidad de realizar el método naïve de control o permitir al usuario definir una versión controlada del unitario si se conoce una versión optimizada.

Las puertas también se pueden controlar mediante información clásica. Una puerta no controlada de forma clásica, por ejemplo, es simplemente una puerta NOT normal, pero solo se aplica si un bit clásico es 1, al contrario que un bit cuántico. En este sentido, se puede pensar en una puerta controlada de forma clásica como una instrucción if en el código cuántico, donde la puerta se aplica solo en una rama del código.

Al igual que en el caso de un solo qubit, un conjunto de puertas de dos qubits es universal si cualquier matriz unitaria de  $4 \times 4$  se puede aproximar mediante un producto de puertas de este conjunto a una precisión arbitraria. Un ejemplo de un conjunto de puertas universales es la puerta Hadamard, la puerta T y la puerta CNOT. Al tomar productos de estas puertas, puede aproximarse a cualquier matriz unitaria en dos qubits.

## Sistemas de varios qubits

Seguimos exactamente los mismos patrones explorados en el caso de dos qubits para crear estados cuánticos de varios qubits a partir de sistemas más pequeños. Estos estados se construyen mediante la formación de productos tensoriales de estados más pequeños. Por ejemplo, considere la posibilidad de codificar la cadena de bits 1011001 en un equipo cuántico. Puede codificar esto como

$$1011001 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Las puertas cuánticas funcionan exactamente de la misma manera. Por ejemplo, si desea aplicar la *puerta X* al primer qubit y, a continuación, realizar un CNOT entre los qubits segundo y tercer qubits, puede expresar esta transformación como

$$(X \otimes I \otimes I) \text{CNOT}_{23} (I \otimes I \otimes I)$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\otimes$$

Missing `\end{bmatrix}`

$\otimes \mathbb{1}$

$\mathbb{1}$

$\otimes \mathbb{1}$

$[\otimes]$

$\otimes \text{end}\{\text{bmatrix}\}$

Missing  $\text{end}\{\text{bmatrix}\}$

$\otimes \mathbb{1} \text{end}\{\text{bmatrix}\} \equiv 0011001$ .  $\text{end}\{\text{align}\}$

En muchos sistemas de cúbits, a menudo es necesario asignar y desasignar cúbits que sirven como memoria temporal para el equipo cuántico. Se dice que este tipo de cúbits es *auxiliar*. De forma predeterminada, puede suponer que el estado del cúbit se inicializa en  $e_0$  *tras la asignación*. También puede suponer que se devuelve de nuevo a  $e_0$  antes de la desasignación. Esta suposición es importante, porque si un cúbit auxiliar se entrelaza con otro registro de cúbits cuando se desasigna, el proceso de desasignación daña el cúbit auxiliar. Por este motivo, siempre se supone que estos cúbits se revierten a su estado inicial antes de liberarse.

Por último, aunque es necesario agregar nuevas puertas al conjunto de puertas para lograr la computación cuántica universal para dos equipos cuánticos de cúbits, no es necesario introducir nuevas puertas en el caso de varios cúbits. Las puertas  $H$ ,  $T$  y CNOT forman una puerta universal establecida en muchos cúbits porque cualquier transformación unitaria general se puede dividir en una serie de dos rotaciones de dos cúbits. A continuación, puede aprovechar la teoría desarrollada para el caso de dos cúbits y usarla de nuevo aquí cuando tenga muchos cúbits.

#### ⓘ Nota

Aunque la notación algebraica lineal que se ha usado hasta ahora puede usarse para describir estados de varios cúbits, se vuelve cada vez más complicado a medida que aumenta el tamaño de los estados. El vector de columna resultante para una cadena de 7 bits de longitud, por ejemplo, es de 128 dimensiones, lo que hace que expresarlo con la notación descrita anteriormente sea muy complicado. En su lugar, *se usa la notación Dirac*, una abreviatura simbólica que simplifica la representación de los estados cuánticos. Para obtener más información, vea [Notación Dirac](#).

## Contenido relacionado

- [Notación de Dirac](#)
- [Medidas de Pauli](#)
- [Puertas T y fábricas de T](#)
- [Circuitos cuánticos](#)
- [Oráculos cuánticos](#)

## Comentarios

¿Le ha resultado útil esta página?

👍 Sí

👎 No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)



# Notación dirac en la computación cuántica

Artículo • 17/01/2025

[notación dirac](#) es una manera concisa y eficaz de describir los estados y las operaciones cuánticos. Se llama después del físico Paul Dirac, quien desarrolló la notación en los años 30. La notación dirac se usa en la computación cuántica para describir los estados cuánticos, las operaciones cuánticas y las medidas cuánticas.

En este artículo se presenta la notación dirac y se muestra cómo usarla para describir los estados y las operaciones cuánticos.

## Vectores en notación Dirac

Hay dos tipos de vectores en notación de Dirac: el vector bra , correspondiente a un vector de fila, y el vector ket , correspondiente a un vector de columna. Por esa razón, la notación de Dirac también se denomina notación *bra-ket*.

Si  $\psi$  es un vector de columna, puede escribirlo en notación Dirac como  $|\psi\rangle$ , donde el  $|\cdot\rangle$  denota que es un vector de *ket*.

Del mismo modo, el vector de fila  $\psi^\dagger$  se expresa como  $\langle\psi|$ , que es un vector *bra*. En otras palabras,  $\psi^\dagger$  se obtiene al aplicar la conjugación compleja de entrada a los elementos de la transposición de  $\psi$ . La notación bra-ket implica directamente que  $\langle\psi|\psi\rangle$  es el producto interior del vector  $\psi$  por sí mismo, que es por definición 1.

De forma más general, si  $\psi$  y  $\phi$  son vectores de estado cuántico, su producto interior es  $\langle\phi|\psi\rangle$ . Este producto interior implica que la probabilidad de que la medida del estado  $|\psi\rangle$  sea  $|\phi\rangle$  es  $|\langle\phi|\psi\rangle|^2$ .

Los estados de base computacional 0 y 1 se representan como \$

$$\begin{bmatrix} 1 & 0 \end{bmatrix}$$

$=|\text{ket}\{0\}\rangle$

$$\begin{bmatrix} 0 & 1 \end{bmatrix}$$

$=|\text{ket}\{1\}\rangle$ , respectivamente.

## Ejemplo: Representación de la operación de Hadamard con la notación de Dirac

Vamos a aplicar la puerta Hadamard  $H$  a los estados cuánticos  $|0\rangle$  y  $|1\rangle$  mediante la notación Dirac:

$$\frac{1}{\sqrt{2}}$$

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$=H|0\rangle=|+\rangle$$

$$\frac{1}{\sqrt{2}}$$

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

$$=H|1\rangle=|-\rangle$$

Los estados resultantes corresponden a los vectores unitarios en las direcciones  $+x$  y  $-x$  en la esfera de Bloch. Estos estados también pueden expandirse con la notación de Dirac como sumas de  $|0\rangle$  y  $|1\rangle$ :

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

## Vectores de base computacional

Cada estado cuántico siempre se puede expresar como sumas de vectores de base computacional y estas sumas se expresan fácilmente mediante la notación Dirac. Lo contrario también es cierto en el sentido de que los estados  $|+\rangle$  y  $|-\rangle$  también forman una base para los estados cuánticos. Esto se puede ver en el hecho de que

$$|0\rangle = \frac{1}{\sqrt{2}}(|+\rangle + |-\rangle)$$

$$|1\rangle = \frac{1}{\sqrt{2}}(|+\rangle - |-\rangle)$$

Como ejemplo de notación de Dirac, consideremos el bra-ket  $\langle 0|1\rangle$ , que es el producto interior de 0 y 1. Se puede escribir como

$$\langle 0|1\rangle = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0.$$

Este ejemplo indica que  $|0\rangle$  y  $|1\rangle$

son vectores ortogonales, lo que significa que  $\langle 0|1\rangle = \langle 1|0\rangle = 0$

. También por definición,  $\langle 0 | 0 \rangle = \langle 1 | 1 \rangle = 1$ , lo que significa que los dos vectores de base computacional también se pueden llamar *ortonormales*.

Estas propiedades ortonormales se usan en el ejemplo siguiente. Si se tiene el estado  $|\psi\rangle = \frac{3}{5}|\ket{1}\rangle + \frac{4}{5}|\ket{0}\rangle$ , y puesto que  $\langle 1 | 0 \rangle = 0$ , la probabilidad de medir 1 es

$$\|\langle 1 | \psi \rangle\|^2 = \left| \frac{3}{5} \langle 1 | 1 \rangle + \frac{4}{5} \langle 1 | 0 \rangle \right|^2 = \frac{9}{25}.$$

## Notación del producto tensorial

La notación Dirac es muy útil para expresar el **producto tensor**. El producto Tensor es importante en la computación cuántica porque el vector de estado descrito por dos registros cuánticos no relacionados es los productos tensor de los dos vectores de estado.

El  $\psi \otimes \phi$  del producto tensor para dos vectores de estado cuántico  $\phi$  y  $\psi$  se escribe en notación Dirac como  $|\psi\rangle \otimes |\phi\rangle$ . Por convención, también puede escribir el producto tensor como  $|\psi\rangle |\phi\rangle = |\psi\phi\rangle$ .

Por ejemplo, el estado con dos cúbits inicializados en el estado cero es  $|\ket{0}\rangle \otimes |\ket{0}\rangle = |\ket{00}\rangle$ .

## Ejemplo: Descripción de la superposición con la notación de Dirac

Como otro ejemplo de cómo puede usar la notación Dirac para describir un estado cuántico, tenga en cuenta las siguientes formas equivalentes de escribir un estado cuántico que sea una superposición **igual** en cada cadena de bits de longitud *n* posible.

$$H^{\otimes n} |0\rangle = \frac{1}{2^{n/2}} \sum_{j=0}^{2^n-1} |j\rangle = |+\rangle^{\otimes n}.$$

Aquí tal vez se pregunte por qué la suma va de 0 a  $2^n - 1$  para  $n$  bits. En primer lugar, observe que hay  $2^n$  configuraciones diferentes que pueden tomar  $n$  bits. Para verlo, recuerde que un bit puede tomar 2 valores, pero dos bits pueden tomar 4 valores, y así sucesivamente. En general, esto significa que hay  $2^n$  cadenas de bits posibles diferentes, pero el mayor valor codificado en cualquiera de ellas es  $1 \cdots 1 = 2^n - 1$  y, por lo tanto, es el límite superior de la suma. Como nota al margen, en este ejemplo no se ha utilizado  $|+\rangle^{\otimes n} = |+\rangle$  como analogía de  $|\ket{0}^{\otimes n}\rangle = |\ket{0}\rangle$ . Esta convención notaria está reservada para el estado de base computacional con cada cúbit

inicializado en cero. Aunque esta convención es razonable en este caso, no se emplea en la literatura de computación cuántica.

## Expresión de la linealidad con la notación de Dirac

Otra característica de la notación de Dirac es que es lineal. Por ejemplo, para dos números complejos  $\alpha$  y  $\beta$ , puede escribir:

$$|\psi\rangle \otimes (\alpha |\phi\rangle + \beta |\chi\rangle) = \alpha |\psi\rangle |\phi\rangle + \beta |\psi\rangle |\chi\rangle.$$

Es decir, la notación del producto tensorial se puede distribuir en notación de Dirac para que los productos tensoriales de vectores de estado terminen pareciendo una multiplicación normal.

Los vectores bra siguen una convención similar a la de los vectores ket. Por ejemplo, el vector  $\langle\psi| \langle\phi|$  es equivalente al vector de estado  $\psi^\dagger \otimes \phi^\dagger = (\psi \otimes \phi)^\dagger$ . Si el vector ket  $|\psi\rangle$  es  $\alpha |0\rangle + \beta |1\rangle$ , la versión del vector bra del vector es  $\langle\psi| = \langle\psi|^\dagger = (\alpha^* \langle 0| + \beta^* \langle 1|)$ .

Como ejemplo, imaginemos que desea calcular la probabilidad de medir el estado  $|\psi\rangle = \frac{3}{5} |1\rangle + \frac{4}{5} |0\rangle$  con un programa cuántico para medir que los estados sean  $|+\rangle$  o  $|-\rangle$ . La probabilidad de que el dispositivo diga que el estado es  $|-\rangle$  es:

$$\langle\psi| \langle - | \psi \rangle^2 = \left| \frac{1}{\sqrt{2}} (\langle 0| - \langle 1|) \left( \frac{3}{5} |1\rangle + \frac{4}{5} |0\rangle \right) \right|^2 = \left| -\frac{3}{5\sqrt{2}} + \frac{4}{5\sqrt{2}} \right|^2 = \frac{1}{50}.$$

El hecho de que el signo negativo aparezca en el cálculo de la probabilidad es una manifestación de la interferencia cuántica, que es uno de los mecanismos que hace que la computación cuántica tenga ventajas sobre la clásica.

## ketbra o producto exterior

El último elemento que vale la pena explicar en la notación de Dirac es *ketbra* o producto exterior. El producto exterior se representa en la notación de Dirac como  $|\psi\rangle \langle\phi|$ . El producto exterior se define mediante la multiplicación de matrices como  $|\psi\rangle \langle\phi| = \psi \phi^\dagger$  para los vectores de estado cuántico  $\psi$  y  $\phi$ . El ejemplo más sencillo, y posiblemente el más común de esta notación, es

$$|0\rangle \langle 0| =$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \end{bmatrix}$$

=

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\quad \ket{1}\bra{1} =$$

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix}$$

=

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

. \$\$

Los ketbras suelen llamarse proyectores porque proyectan un estado cuántico sobre un valor fijo. Como estas operaciones no son unitarias (y ni siquiera conservan la norma de un vector), un equipo cuántico no puede aplicar un proyector de forma determinista. Sin embargo, los proyectores hacen un buen trabajo para describir efecto que la medida tiene sobre un estado cuántico. Por ejemplo, si mide un estado  $|\psi\rangle$  para que sea 0, la transformación resultante que experimenta el estado como resultado de la medida es:

$$\ket{\psi} \rightarrow \frac{(\ket{0}\bra{0})\ket{\psi}}{\langle 0 | \psi \rangle} = \ket{0},$$

como cabría esperar si mide el estado y lo encontró como  $|0\rangle$ . Repetimos que estos proyectores no pueden aplicarse a un estado en un equipo cuántico de forma determinista. En cambio, pueden aplicarse aleatoriamente con el resultado  $|0\rangle$  que aparece con alguna probabilidad fija. La probabilidad de que dicha medida tenga éxito puede escribirse como el valor de la expectativa del proyector cuántico en el estado:

$$\langle \psi | (|0\rangle \langle 0|) | \psi \rangle = |\langle \psi | 0 \rangle|^2,$$

lo que ilustra que los proyectores ofrecen una nueva forma de expresar el proceso de medida.

Si, en su lugar, considera que la medida del primer cúbit de un estado de varios cúbits es 1, también podemos describir este proceso de forma cómoda mediante proyectores y la notación de Dirac:

$$P(\text{first qubit} = 1) = \langle \psi | \left( |1\rangle\langle 1| \otimes \mathbb{I}^{\otimes n-1} \right) | \psi \rangle$$

Aquí, la matriz de identidades puede escribirse en notación de Dirac como:

$$\mathbb{I} = |0\rangle\langle 0| + |1\rangle\langle 1| =$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

.

Si hay dos cúbits, el proyector se puede expandir como:

$$|1\rangle\langle 1| \otimes \mathbb{I} = |1\rangle\langle 1| \otimes (|0\rangle\langle 0| + |1\rangle\langle 1|) = |10\rangle\langle 10| + |11\rangle\langle 11|$$

Puede ver que esto es coherente con la discusión sobre las probabilidades de medida de los estados de varios cúbits mediante la notación de vectores de columna:

$$P(\text{primer cúbit} = 1) = \langle \psi | (|10\rangle\langle 10| + |11\rangle\langle 11|) | \psi \rangle = |e_{10}|^2 + |e_{11}|^2$$

que coincide con la discusión sobre la medida de varios cúbits. Sin embargo, la generalización de este resultado al caso de varios cúbits es algo más sencilla de expresar utilizando la notación de Dirac que la notación de vectores de columna, y es totalmente equivalente al tratamiento anterior.

## Operadores de densidad

Otro operador útil para expresar con la notación de Dirac es el *operador de densidad*, también conocido como *operador de estado*. Como el vector de estado cuántico, el operador de densidad describe el estado cuántico de un sistema. Aunque los vectores de estado cuántico solo pueden representar *estados puros*, los operadores de densidad también pueden representar *estados mixtos*.

En general, una matriz dada  $\rho$  es un operador de densidad válido si se cumplen las siguientes condiciones:

- $\rho$  es una matriz de números complejos
- $\rho = \rho^\dagger$  (es decir,  $\rho$  es hermitiano)
- Cada valor propio de  $\rho$  no es negativo
- Todos los valores propios de  $\rho$  suman 1

En conjunto, estas condiciones garantizan que  $\rho$  puede considerarse como un conjunto. Un operador de densidad para un vector de estado cuántico  $|\psi\rangle$  tiene la forma  $\rho = \sum_i p_i |\psi_i\rangle \langle \psi_i|$  es una descomposición de valores propios de  $\rho$ , entonces  $\rho$  describe el conjunto  $\rho = |\psi_i\rangle$  con probabilidad  $p_i$ .

Los estados cuánticos puros son aquellos que se caracterizan por un único vector ket o función de onda, y no pueden escribirse como una mezcla estadística (o *combinación convexa*) de otros estados cuánticos. Un estado cuántico mixto es un conjunto estadístico de estados puros.

En una esfera Bloch, los estados puros se representan mediante un punto en la superficie de la esfera, mientras que los estados mixtos se representan mediante un punto interior. El estado mixto de un solo cúbit se representa mediante el centro de la esfera, por simetría. La pureza de un estado se puede visualizar como el grado en el que está cerca de la superficie de la esfera.

Este concepto de representar el estado como una matriz, en lugar de un vector, suele ser conveniente porque proporciona una manera cómoda de representar cálculos de probabilidad, y también permite describir tanto la incertidumbre estadística como la incertidumbre cuántica dentro del mismo formalismo.

Un operador de densidad  $\rho$  representa un estado puro si y solo si:

- $\rho$  puede escribirse como un producto externo de un vector de estado,  $\rho = |\psi\rangle \langle \psi|$
- $\rho = \rho^2$
- $\text{tr}(\rho^2) = 1$

Para saber lo cerca que está un determinado operador de densidad  $\rho$  es puro, se puede ver el seguimiento (es decir, la suma de los elementos diagonales) de  $\rho^2$ . Un operador de densidad representa un estado puro si y sólo si  $\text{tr}(\rho^2) = 1$ .

## Secuencias de puertas de Q# equivalentes a estados cuánticos

Un último punto que vale la pena plantear sobre la notación cuántica y el lenguaje de programación Q#: al principio de este documento, mencionamos que el estado cuántico es el objeto básico de información en la computación cuántica. Puede sorprender entonces que en Q# no exista la noción de estado cuántico. En cambio, todos los estados se describen únicamente por las operaciones utilizadas para prepararlos. El ejemplo anterior lo ilustra perfectamente. En lugar de expresar una superposición uniforme sobre cada cadena de bits cuánticos en un registro, puede representar el resultado como  $H^{\otimes n} |0\rangle$ . Esta descripción exponencialmente más corta del estado no

solo tiene la ventaja de que puede razonar de forma clásica sobre él, sino que también define de forma concisa las operaciones necesarias para propagarse por la pila de software con el fin de implementar el algoritmo. Por esta razón, Q# está diseñado para emitir secuencias de puertas en lugar de estados cuánticos; sin embargo, a nivel teórico, las dos perspectivas son equivalentes.

## Temas relacionados

- [Medidas de Pauli](#)
- [Puertas T y fábricas de T](#)
- [Circuitos cuánticos](#)
- [Oráculos cuánticos](#)

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)



# Medidas de Pauli de un solo cúbit y de varios cúbits

Artículo • 16/09/2024

A medida que trabaja con Q#, observa que *las* medidas de Pauli son un tipo común de medida. Las medidas de Pauli generalizan las medidas de base computacional para incluir medidas en otras bases y de paridad entre diferentes cúbits. En tales casos, es habitual analizar la medición de un operador Pauli, que es un operador como  $X$ ,  $Y$ ,  $Z$  o  $Z \otimes Z$ ,  $XX \otimes$ ,  $X \otimes Y$ , etc. Para conocer los conceptos básicos de la medición cuántica, consulte [El cúbit](#) y [varios cúbits](#).

La discusión de la medición en términos de operadores pauli es común en el subcampo de corrección de errores cuánticos.

La guía de Q# sigue una convención similar; en este artículo, se explica esta vista alternativa de las medidas.

## 💡 Sugerencia

En Q#, los operadores pauli de varios cúbits suelen representarse mediante matrices de tipo `Pauli[]`. Por ejemplo, para representar  $X \otimes Z \otimes Y$ , puede usar la matriz `[PauliX, PauliZ, PauliY]`.

Antes de profundizar en los detalles de cómo pensar una medida de Pauli, es útil pensar en el efecto que tiene en el estado cuántico la medición de un único cúbit dentro de un equipo cuántico. Imagine un estado cuántico de  $n$  cúbits; a continuación, la medición de un cúbit descarta de inmediato la mitad de las  $2^n$  posibilidades en las que podría estar ese estado. En otras palabras, la medida proyecta el estado cuántico en uno de los dos semiespacios. Puede generalizar la forma en que piensa en la medición para reflejar esta intuición.

Para identificar estos subespacios de forma concisa, necesitamos un lenguaje para describirlos. Una manera de describir los dos subespacios es especificarlos a través de una matriz que solo tenga dos valores propios únicos, que por convención se toman como  $\pm 1$ . Para ver un ejemplo sencillo de cómo describir subespacios de esta manera, piense en  $Z$ :

$$Z =$$

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

. \end{align} \$\$

Al leer los elementos diagonales de la matriz  $Z$  de Pauli, podemos ver que  $Z$  tiene dos vectores propios,  $|0\rangle$  y  $|1\rangle$ , con los valores propios correspondientes  $\pm 1$ . Por lo tanto, si una medida del cúbit da `Zero` como resultado (correspondiente al estado  $|0\rangle$ ), se sabe que el estado del cúbit es un *estadopropio*  $+ 1$  del *operador*  $Z$ . Del mismo modo, si el resultado es `One`, se sabe que el estado del cúbit es un *estadopropio*  $- 1$  de  $Z$ . En el lenguaje de las medidas de Pauli, este proceso se denomina "medida de Pauli  $Z$ ," y es totalmente equivalente a realizar una medición de base computacional.

Cualquier matriz de  $2 \times 2$  que sea una transformación unitaria de  $Z$  también cumple estos criterios. Es decir, también podríamos usar una matriz  $A = U^\dagger Z U$ , donde  $U$  es cualquier otra matriz unitaria, para proporcionar una matriz que defina los dos resultados de una medida en sus vectores propios  $\pm 1$ . La notación de las medidas de Pauli hace referencia a esta equivalencia unitaria mediante la identificación de las medidas  $X, Y, Z$  como medidas equivalentes que se podrían hacer para obtener información de un cúbit. Estas medidas se proporcionan aquí para mayor comodidad.

[Expandir tabla](#)

| Medidas de Pauli | Transformación unitaria |
|------------------|-------------------------|
| $Z$              | $I$                     |
| $X$              | $H$                     |
| $Y$              | $HS^\dagger$            |

Es decir, usando este lenguaje, `&entre comillas; measure Y` es equivalente a aplicar  $HS^\dagger$  y, a continuación, medir en la base computacional, donde `S` es una operación cuántica intrínseca a veces llamada `phase gate`; y se pueden simular mediante la matriz unitaria

$$S = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

También es equivalente a aplicar  $HS^\dagger$  al vector de estado cuántico y, a continuación, medir  $Z$ , de modo que la siguiente operación sea equivalente a `Measure([PauliY], [q])`:

Q#

```
operation MeasureY(qubit : Qubit) : Result {
 mutable result = Zero;
 within {
 Adjoint S(q);
```

```

 H(q);
 } apply {
 set result = M(q);
 }
 return result;
}

```

A continuación, se encuentra el estado correcto al transformarse de nuevo en la base computacional, lo que equivale a aplicar  $SH$  al vector de estado cuántico; en el fragmento de código, la transformación de vuelta a la base computacional se controla automáticamente con el uso del `within ... apply` bloque.

En Q#, el resultado--- que es, la información clásica extraída de la interacción con el estado--- se proporciona mediante un `Result` valor  $j \in \{\text{Zero}, \text{One}\}$  que indica si el resultado está en el *espaciopropio* $(-1)^j$  propio del operador Pauli medido.

## Medidas de varios cúbits

Las medidas de los operadores de Pauli de varios cúbits se definen de forma similar, como se ve a continuación:

$$ZZ \otimes I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \text{amp; } 1 & 0 & 0 & 0 \\ \text{amp; } 0 & 1 & 0 & 0 \\ \text{amp; } 0 & 0 & 1 & 0 \end{bmatrix}.$$

Por lo tanto, los productos tensoriales de dos operadores  $Z$  de Pauli forman una matriz formada por dos espacios que constan de valores propios  $+1$  y  $-1$ . Al igual que con el caso de un único cúbit, ambos constituyen un semiespacio, lo que significa que la mitad del espacio vectorial accesible pertenece al espacio propio  $+1$ ; y la mitad restante, al espacio propio  $-1$ . En general, es fácil ver a partir de la definición del producto tensorial que cualquier producto tensorial de los operadores  $Z$  de Pauli y la identidad también obedecen a esto. Por ejemplo,

$$Z \otimes I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Como antes, cualquier transformación unitaria de estas matrices también describe dos espacios medio etiquetados con  $\pm 1$  valores propios. Por ejemplo,

$X \otimes X = H \otimes H (Z \otimes Z) H \otimes H$  de la identidad que  $Z = HXH$ . De forma similar al caso de un cúbit, todas las medidas pauli de dos cúbits se pueden escribir como  $U^\dagger (Z \otimes I) U$  para  $4 \times 4$  matrices *unitarias*  $U$ . Las transformaciones se enumeran en la tabla siguiente.

ⓘ Nota

En esta tabla, **SWAP** se usa para indicar la matriz

Misplaced &

\end{align}utilizado para simular la operación **SWAP** intrínseca .&}\begin{

Expandir tabla

| Medidas de Pauli       | Transformación unitaria                           |
|------------------------|---------------------------------------------------|
| $Z \otimes \mathbf{1}$ | $\mathbf{1} \otimes \mathbf{1}$                   |
| $X \otimes \mathbf{1}$ | $H \otimes \mathbf{1}$                            |
| $Y \otimes \mathbf{1}$ | $HS^\dagger \otimes \mathbf{1}$                   |
| $\mathbf{1} \otimes Z$ | SWAP                                              |
| $\mathbf{1} \otimes X$ | $(H \otimes \mathbf{1})$ SWAP                     |
| $\mathbf{1} \otimes Y$ | $(HS^\dagger \otimes \mathbf{1})$ SWAP            |
| $Z \otimes Z$          | $\text{CNOT}_{10}$                                |
| $X \otimes Z$          | $\text{CNOT}_{10}(H \otimes \mathbf{1})$          |
| $Y \otimes Z$          | $\text{CNOT}_{10}(HS^\dagger \otimes \mathbf{1})$ |
| $Z \otimes X$          | $\text{CNOT}_{10}(\mathbf{1} \otimes H)$          |
| $X \otimes X$          | $\text{CNOT}_{10}(H \otimes H)$                   |
| $Y \otimes X$          | $\text{CNOT}_{10}(HS^\dagger \otimes H)$          |
| $Z \otimes Y$          | $\text{CNOT}_{10}(\mathbf{1} \otimes HS^\dagger)$ |
| $X \otimes Y$          | $\text{CNOT}_{10}(H \otimes HS^\dagger)$          |
| $Y \otimes Y$          | $\text{CNOT}_{10}(HS^\dagger \otimes HS^\dagger)$ |

En este caso, la operación **CNOT** aparece por el siguiente motivo. Cada medida de Pauli que no incluye la **1** matriz es equivalente a una unidad a  $Z \otimes Z$  por el razonamiento anterior. Los valores propios de  $Z \otimes Z$  solo dependen de la paridad de los cúbits que componen cada vector de base computacional, y las operaciones controlled-NOT sirven para calcular esta paridad y almacenarla en el primer bit. Después, una vez que se mide

el primer bit, podemos recuperar la identidad del semiespacio resultante, lo que equivale a medir el operador de Pauli.

Además, aunque puede ser tentador suponer que la medición  $de Z \otimes Z$  es la misma que la medición secuencial de  $Z \otimes y$ , a continuación  $\mathbb{1} \otimes Z$ , *estas suposición sería falsa. La razón es que la medición de  $Z \otimes Z$  proyecta el estado cuántico en el estado puro de  $+1$  o  $-1$  de estos operadores. Medir  $Z \otimes \mathbb{1}$  y luego  $\mathbb{1} \otimes Z$  proyecta el vector de estado cuántico primero en un semiespacio de  $Z \otimes \mathbb{1}$  y, luego, en un semiespacio de  $\mathbb{1} \otimes Z$* . Como hay cuatro vectores de base computacional, procesar ambas medidas reduce el estado a un cuarto de espacio y, por tanto, lo reduce a un único vector de base computacional.

## Correlaciones entre bits cuánticos

Otra manera de ver la medición de los productos tensoriales de las matrices de Pauli, como  $X \otimes X$  o  $Z \otimes Z$ , es que estas medidas permiten ver la información almacenada en las correlaciones entre los dos cúbits. Medir  $X \otimes \mathbb{1}$  permite examinar la información almacenada localmente en el primer cúbit. Aunque ambos tipos de medidas son igualmente valiosos en la computación cuántica, el primero ilustra la capacidad de la computación cuántica. Revela que, en la computación cuántica, a menudo la información que quiere aprender no se almacena en un único cúbit, sino que se almacena de forma no local en todos los cúbits a la vez y, por tanto, solo al mirarla a través de una medida conjunta (por ejemplo,  $Z \otimes Z$ ) hace que se manifieste esta información.

Los operadores arbitrarios de Pauli, como  $X \otimes Y \otimes Z \otimes \mathbb{1}$ , también se pueden medir. Todos estos productos tensoriales de los operadores de Pauli solo tienen dos valores propios  $\pm 1$ , y ambos espacios propios constituyen semiespacios del espacio vectorial entero. Por lo tanto, coinciden con los requisitos indicados anteriormente.

En  $Q^\#$ , estas medidas devuelven  $j$  si la medida produce un resultado en el espacio puro de signo  $(-1)^j$ . Tener medidas de Pauli como una característica integrada en  $Q^\#$  es útil porque medir estos operadores requiere cadenas largas de puertas NO controladas y transformaciones de base para describir la puerta U diagonalizante necesaria para expresar la operación como un producto tensor de  $Z$  y  $\mathbb{1}$ . Al poder especificar que quiere realizar una de estas medidas predefinidas, no es necesario preocuparse por cómo transformar la base de forma que una medida de base computacional proporcione la información necesaria.  $Q^\#$  controla automáticamente todas las transformaciones de base necesarias.

# El teorema de la no clonación

La información cuántica es poderosa. Permite hacer cosas increíbles, como números de factor exponencialmente más rápidos que los algoritmos clásicos más conocidos, o simular eficazmente sistemas de electrones correlacionados que requieren clásicamente un costo exponencial para simular con precisión. Sin embargo, la eficacia de computación cuántica tiene limitaciones. Una de estas limitaciones está dada por el *teorema de la no clonación*.

El teorema de la no clonación tiene un nombre adecuado. No permite la clonación de estados cuánticos genéricos por parte de un equipo cuántico. La demostración del teorema es sumamente sencilla. Aunque una prueba completa del teorema de no clonación es demasiado técnica para este artículo, la prueba en el caso de que no haya cúbits auxiliares adicionales esté dentro del ámbito.

Para este tipo de equipo cuántico, la operación de clonación debe describirse con una matriz unitaria. La medida cuántica no está permitida, porque dañaría el estado cuántico que se va a clonar. Para simular la operación de clonación, queremos que la matriz unitaria usada tenga la propiedad que  $U \ket{\psi} \ket{0} = \ket{\psi} \ket{\psi}$  para cualquier estado  $|\psi\rangle$ . La propiedad de linealidad de la multiplicación de matrices implica entonces que para cualquier segundo estado cuántico  $|\phi\rangle$ ,

$$\begin{aligned} U \left[ \frac{1}{\sqrt{2}} (\ket{\phi} + \ket{\psi}) \right] \ket{0} &= \frac{1}{\sqrt{2}} U (\ket{\phi} \ket{0} + \ket{\psi} \ket{0}) \\ &= \frac{1}{\sqrt{2}} (\ket{\phi} \ket{\phi} + \ket{\psi} \ket{\psi}) \end{aligned}$$

Esto proporciona la intuición fundamental que subyace al teorema de no clonación: cualquier dispositivo que copie un estado cuántico desconocido debe inducir errores en al menos algunos de los estados que copia. Aunque la suposición clave de que el clonador actúa linealmente en el estado de entrada se puede infringir mediante la adición y medición de cúbits auxiliares, estas interacciones también filtran información sobre el sistema a través de las estadísticas de las medidas y evitan también la clonación exacta en estos casos.

El teorema de la no clonación es importante para la comprensión cualitativa de la computación cuántica, ya que si pudiera clonar estados cuánticos de forma económica, se le concedería una capacidad casi mágica de aprender a partir de los estados cuánticos. De hecho, podría infringir el alabado principio de incertidumbre de Heisenberg. Como alternativa, podría usar un clonador óptimo para tomar una única muestra de una distribución cuántica compleja y aprender todo lo posible sobre esa

distribución a partir de tan solo una muestra. Esto sería como si volteas una moneda y observaste cabezas y luego diciéndoles a un amigo sobre el resultado que les hace responder " ¡Ah la distribución de esa moneda debe ser Bernoulli con  $p = 0.512643 \dots$  !" Tal declaración sería no útil porque un bit de información (el resultado de los encabezados) simplemente no puede proporcionar los muchos bits de información necesarios para codificar la distribución sin información previa sustancial. Del mismo modo, sin información previa no se puede clonar perfectamente un estado cuántico, del mismo modo que no se puede preparar un conjunto de dichas monedas sin conocer  $p$ .

La información no es gratuita en la computación cuántica. Cada cúbit medido proporciona un único bit de información, y el teorema de la no clonación muestra que no hay ninguna puerta trasera que se pueda aprovechar para evitar el equilibrio fundamental entre la información obtenida sobre el sistema y la perturbación ejercida sobre él.

## Contenido relacionado

- [Puertas T y fábricas de T](#)
- [Circuitos cuánticos](#)
- [Oráculos cuánticos](#)
- [Algoritmo de Grover](#)

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Teoría del algoritmo de búsqueda de Grover

Artículo • 20/01/2025

En este artículo encontrará una explicación teórica detallada de los principios matemáticos que hacen que el algoritmo de Grover funcione.

Para obtener una implementación práctica del algoritmo de Grover para resolver problemas matemáticos, consulte [Implementación del algoritmo](#) de búsqueda de Grover.

## Instrucción del problema

El algoritmo de Grover acelera la solución a búsquedas de datos no estructuradas (o *problema de búsqueda*), ejecutando la búsqueda en menos pasos que cualquier algoritmo clásico. Cualquier tarea de búsqueda se puede expresar con una función  $f(x)$  abstracta que acepta elementos de búsqueda  $x$ . Si el elemento  $x$  es una solución para la tarea de búsqueda,  $f(x) = 1$ . Si el elemento  $x$  no es una solución,  $f(x) = 0$ . El *problema de búsqueda* consiste en buscar cualquier elemento  $x_0$ , de forma que  $f(x_0) = 1$ .

Cualquier problema que le permita comprobar si un valor determinado  $x$  es una solución válida (una "Sí o ningún problema") se puede formular en términos del problema de búsqueda. Estos son algunos ejemplos:

- Problema de satisfacibilidad booleana: dado un conjunto de valores booleanos, ¿el conjunto satisface una fórmula booleana determinada?
- Problema de vendedores de viajes: ¿Cuál es el bucle más corto posible que conecta todas las ciudades?
- Problema de búsqueda de base de datos: ¿contiene una tabla de base de datos el registro  $x$ ?
- Problema de factorización de enteros: ¿Es el número  $N$  divisible por el número  $x$ ?

La tarea que pretende resolver el algoritmo de Grover puede expresarse de la siguiente manera: dada una función clásica  $f(x) : \{0, 1\}^n \rightarrow \{0, 1\}$ , donde  $n$  es el tamaño de los bits del espacio de búsqueda, encontrar una entrada  $x_0$  para la que  $f(x_0) = 1$ . La complejidad del algoritmo se mide por el número de usos de la función  $f(x)$ . De forma clásica, en el peor de los casos, tenemos que evaluar  $f(x)$  un total de  $N - 1$  veces, donde  $N = 2^n$ , probando todas las posibilidades. Después de  $N - 1$  elementos, debe ser el último elemento. El algoritmo cuántico de Grover puede resolver este problema



mucho más rápido, con una velocidad cuadrática. La velocidad cuadrática implica que solo se requieren  $\sqrt{N}$  evaluaciones, en comparación con  $N$ .

## Esquema del algoritmo

Supongamos que hay  $N = 2^n$  elementos aptos para la tarea de búsqueda y los indexamos asignando a cada elemento un entero de 0 a  $N - 1$ . Además, supongamos que hay  $M$  entradas válidas diferentes, lo que significa que hay  $M$  entradas para las que  $f(x) = 1$ . Los pasos del algoritmo son los siguientes:

1. Comience con un registro de  $n$  cúbits iniciado en el estado  $|0\rangle$ .
2. Prepare el registro en una superposición uniforme aplicando  $H$  a cada cúbit del registro:

$$|\text{register}\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

3. Aplica las siguientes operaciones al registro  $N_{\text{optimal}}$  veces:
  - a. El oráculo de fase  $O_f$  que aplica un desplazamiento de fase condicional de  $-1$  para los elementos de la solución.
  - b. Aplique  $H$  a cada cúbit del registro.
  - c. Un desplazamiento de fase condicional de  $-1$  a cada estado de base computacional excepto  $|0\rangle$ . Esto se puede representar mediante la operación unitaria  $-O_0$ , del mismo modo que  $O_0$  representa el desplazamiento de la fase condicional a  $|0\rangle$  solamente.
  - d. Aplique  $H$  a cada cúbit del registro.
4. Mida el registro para obtener el índice de un elemento que es una solución con una probabilidad muy alta.
5. Comprueba si se trata de una solución válida. Si no es así, vuelve a empezar.

$N_{\text{optimal}} = \left\lceil \left\lfloor \frac{\pi}{4} \sqrt{\frac{N}{M}} - \frac{1}{2} \right\rfloor \right\rceil$  es el número óptimo de iteraciones que maximiza la probabilidad de obtener el elemento correcto midiendo el registro.

### ⓘ Nota

La aplicación conjunta de los pasos 3.b, 3.c y 3.d se conoce normalmente como *operador de difusión de Grover*.

La operación unitaria general aplicada al registro es:

$$(-H^{\otimes n} O_0 H^{\otimes n} O_f)^{N_{\text{optimal}}} H^{\otimes n}$$

# Seguimiento del estado del registro paso a paso

Para ilustrar el proceso, vamos a seguir las transformaciones matemáticas del estado del registro para un caso simple en el que solo tenemos dos cúbits y el elemento válido es  $|01\rangle$ .

1. El registro se inicia en el estado.

$$|\text{register}\rangle = |00\rangle$$

2. Después de aplicar  $H$  a cada cúbit, el estado del registro se transforma en:

$$|\text{register}\rangle = \frac{1}{\sqrt{4}} \sum_{i \in \{0,1\}^2} |\text{ket}\{i\}\rangle = \frac{1}{2} (|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

3. A continuación, se aplica el oráculo de fase para obtener:

$$|\text{register}\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle + |11\rangle)$$

4. A continuación,  $H$  actúa de nuevo en cada cúbit para dar:

$$|\text{register}\rangle = \frac{1}{2} (|00\rangle + |01\rangle - |10\rangle + |11\rangle)$$

5. Ahora, el cambio de fase condicional se aplica en cada estado, excepto  $|00\rangle$ :

$$|\text{register}\rangle = \frac{1}{2} (|00\rangle - |01\rangle + |10\rangle - |11\rangle)$$

6. Por último, la primera iteración de Grover finaliza aplicando  $H$  de nuevo para obtener:

$$|\text{register}\rangle = |01\rangle$$

Siguiendo los pasos anteriores, el elemento válido se encuentra en una sola iteración. Como verá más adelante, esto se debe a que para  $N=4$  y un solo elemento válido, el número óptimo de veces es  $N_{\text{óptimo}} = 1$ .

## Explicación geométrica

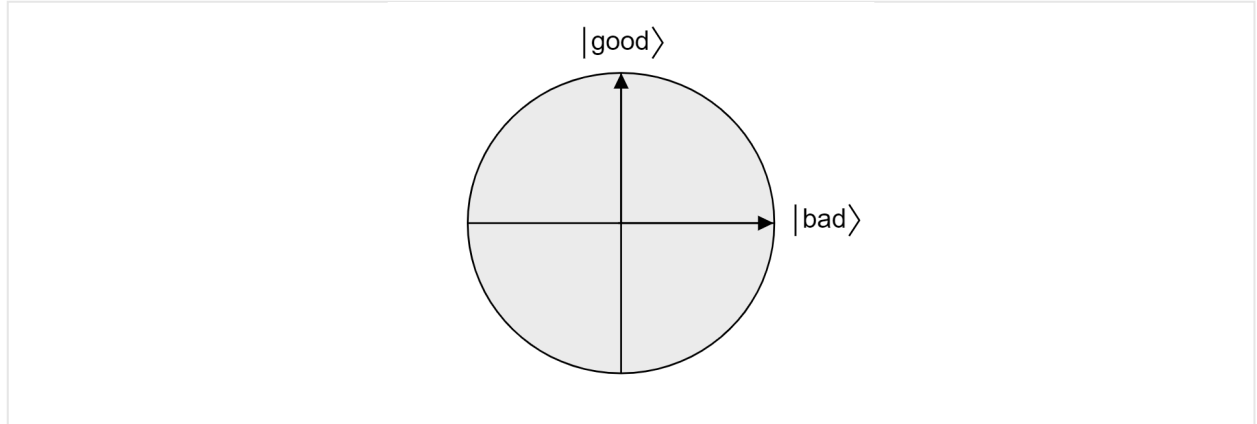
Para ver por qué funciona el algoritmo de Grover, vamos a estudiar el algoritmo desde una perspectiva geométrica. Suponiendo que hay  $M$  soluciones válidas, la superposición de todos los estados cuánticos que *no son* una solución al problema de búsqueda:

$$|\text{bad}\rangle = \frac{1}{\sqrt{N-M}} \sum_{x:f(x)=0} |x\rangle$$

La superposición de todos los estados que *son* una solución del problema de búsqueda:

$$|\text{good}\rangle = \frac{1}{\sqrt{M}} \sum_{x:f(x)=1} |x\rangle$$

Puesto que *good* y *bad* son conjuntos mutuamente excluyentes, porque un elemento no puede ser válido y no válido, los estados son ortogonales. Ambos estados forman la base ortogonal de un plano en el espacio vectorial. Podemos usar este plano para visualizar el algoritmo.



Ahora, supongamos que  $|\psi\rangle$  es un estado arbitrario que reside en el plano abarcado por  $|\text{good}\rangle$  y  $|\text{bad}\rangle$ . Cualquier estado que se encuentra en ese plano se puede expresar como:

$$|\psi\rangle = \alpha |\text{good}\rangle + \beta |\text{bad}\rangle$$

donde  $\alpha$  y  $\beta$  son números reales. Ahora, vamos a presentar el operador de reflexión  $R_{|\psi\rangle}$ , donde  $|\psi\rangle$  es cualquier estado de cúbit que se encuentra en el plano. El operador se define como:

$$R_{|\psi\rangle} = 2 |\psi\rangle \langle \psi| - \mathcal{I}$$

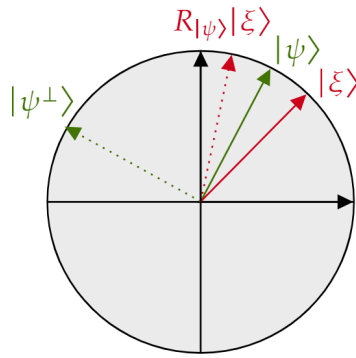
Se denomina operador de reflexión sobre  $|\psi\rangle$  porque se puede interpretar geométricamente como reflexión sobre la dirección de  $|\psi\rangle$ . Para verlo, tome la base ortogonal del plano formado por  $|\psi\rangle$  y su complemento ortogonal  $|\psi^\perp\rangle$ . Cualquier estado  $|\xi\rangle$  del plano se puede descomponer de esta forma:

$$|\xi\rangle = \mu |\psi\rangle + \nu |\psi^\perp\rangle$$

Si se aplica el operador  $R_{|\psi\rangle}$  a  $|\xi\rangle$ :

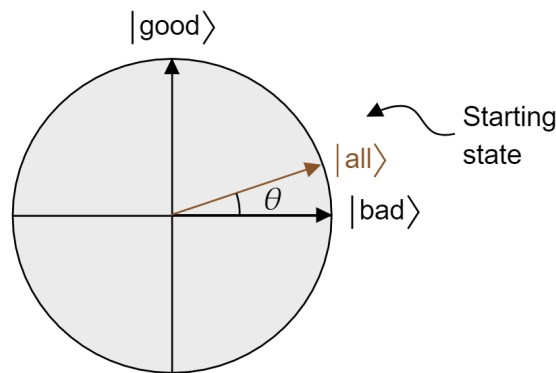
$$R_{|\psi\rangle} |\xi\rangle = \mu |\psi\rangle - \nu |\psi^\perp\rangle$$

El operador  $R_{|\psi\rangle}$  invierte el componente ortogonal en  $|\psi\rangle$ , pero deja el componente  $|\psi\rangle$  sin cambios. Por lo tanto,  $R_{|\psi\rangle}$  es una reflexión sobre  $|\psi\rangle$ .



El algoritmo de Grover, después de la primera aplicación de  $H$  a cada cúbit, empieza con una superposición uniforme de todos los estados. Esto se puede escribir como:

$$|all\rangle = \sqrt{\frac{M}{N}} |good\rangle + \sqrt{\frac{N-M}{N}} |bad\rangle$$



Y, por tanto, el estado reside en el plano. Tenga en cuenta que la probabilidad de obtener un resultado correcto cuando la medición de la superposición igual es simplemente  $|\langle good | all \rangle|^2 = M/N$ , que es lo que se esperaría de una estimación aleatoria.

El oráculo  $O_f$  agrega una fase negativa a cualquier solución al problema de búsqueda. Por lo tanto, se puede escribir como una reflexión sobre el eje  $|bad\rangle$ :

$$O_f = R_{|bad\rangle} = 2|bad\rangle\langle bad| - I$$

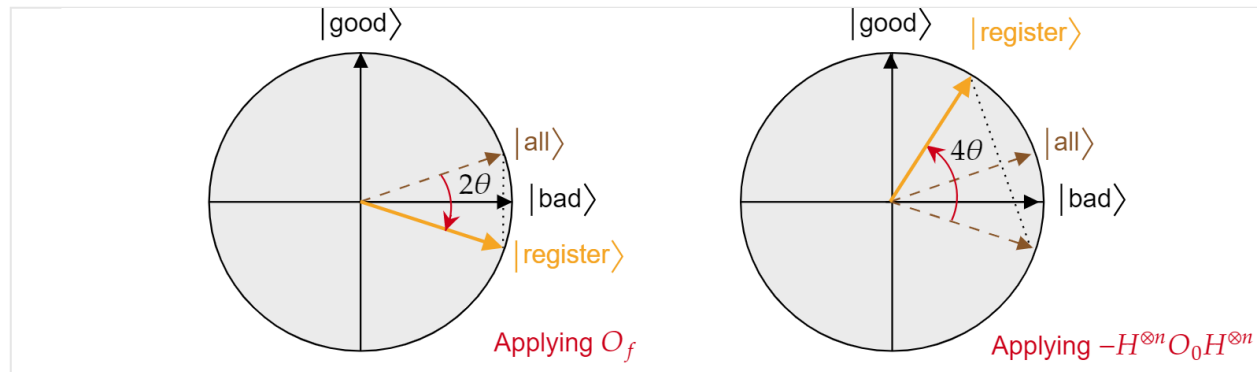
Análogamente, el desplazamiento de fase condicional  $O_0$  es simplemente una reflexión invertida sobre el estado  $|0\rangle$ :

$$O_0 = R_{|0\rangle} = -2|0\rangle\langle 0| + I$$

Sabiendo este hecho, es fácil comprobar que la operación de difusión de Grover  $-H^{\otimes n}O_0H^{\otimes n}$  también es una reflexión sobre el estado  $|all\rangle$ . Simplemente haga lo siguiente:

$$H^{\otimes n} O_{\{0\}} H^{\otimes n} = 2H^{\otimes n} |0\rangle\langle 0| H^{\otimes n} - H^{\otimes n} \mathbb{I} H^{\otimes n} = 2|\text{all}\rangle\langle \text{all}| - \mathbb{I} = R_{|\text{all}\rangle}$$

Se ha demostrado que cada iteración del algoritmo de Grover es una composición de dos reflejos  $R_{|\text{bad}\rangle}$  y  $R_{|\text{all}\rangle}$ .



El efecto combinado de cada iteración de Grover es una rotación en sentido contrario a las agujas del reloj de un ángulo  $2\theta$ . Afortunadamente, el ángulo  $\theta$  es fácil de encontrar. Dado que  $\theta$  es simplemente el ángulo entre  $|\text{all}\rangle$  y  $|\text{bad}\rangle$ , podemos usar el producto escalar para encontrar el ángulo. Se sabe que  $\cos \theta = \langle \text{all} | \text{bad} \rangle$ , por lo que necesitamos calcular  $\langle \text{all} | \text{bad} \rangle$ . A partir de la descomposición de  $|\text{all}\rangle$  en términos de  $|\text{bad}\rangle$  y  $|\text{good}\rangle$ , se obtiene lo siguiente:

$$\theta = \arccos(\langle \text{all} | \text{bad} \rangle) = \arccos\left(\sqrt{\frac{N-M}{N}}\right)$$

El ángulo entre el estado del registro y el  $|\text{buen}\rangle$  estado disminuye con cada iteración, lo que da lugar a una mayor probabilidad de medir un resultado válido. Para calcular esta probabilidad, solo tiene que calcular  $|\langle \text{el buen} | \text{registro} \rangle|^2$ . El ángulo entre  $|\text{good}\rangle$  y  $|\text{register}\rangle$  se indica como  $\gamma(k)$ , donde  $k$  es el recuento de iteraciones:

$$\gamma = \frac{\pi}{2} - \theta - 2k\theta = \frac{\pi}{2} - (2k+1)\theta$$

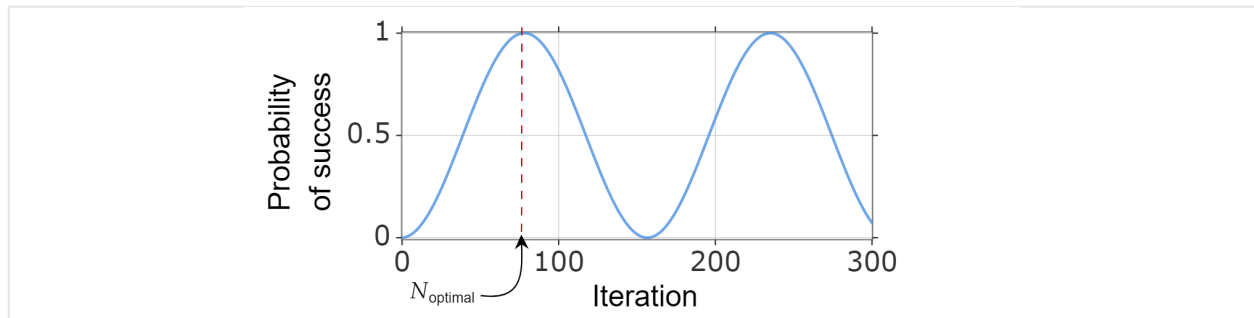
Por lo tanto, la probabilidad de éxito es:

$$P(\text{success}) = \cos^2(\gamma(k)) = \sin^2\left[(2k+1) \arccos\left(\sqrt{\frac{N-M}{N}}\right)\right]$$

## Número óptimo de iteraciones

Como la probabilidad de éxito se puede escribir como una función del número de iteraciones, podemos encontrar el número óptimo de iteraciones  $N_{\text{optimal}}$  al calcular el

entero positivo más pequeño que (aproximadamente) maximiza la función de probabilidad de éxito.



Se sabe que  $\sin^2 x$  alcanza su primer máximo para  $x = \frac{\pi}{2}$ , por lo que:

$$\frac{\pi}{2} = (2k_{\text{optimal}} + 1) \arccos \left( \sqrt{\frac{N-M}{N}} \right)$$

Este es el resultado:

$$k_{\text{optimal}} = \frac{\pi}{4 \arccos \left( \sqrt{1 - M/N} \right)} - 1/2 = \frac{\pi}{4 \arccos \left( \sqrt{\frac{N-M}{N}} \right)} - \frac{1}{2} + O \left( \sqrt{\frac{M}{N}} \right)$$

Donde en el último paso,  $\arccos \sqrt{1-x} = \sqrt{x} + O(x^{3/2})$ .

Por lo tanto, puede elegir  $N_{\text{optimal}} = \left\lfloor \frac{\pi}{4 \arccos \left( \sqrt{\frac{N-M}{N}} \right)} - \frac{1}{2} \right\rfloor$ .

## Análisis de complejidad

En el análisis anterior, necesitamos  $O \left( \sqrt{\frac{N}{M}} \right)$  consultas del oráculo  $O_f$  para encontrar

un elemento válido. Sin embargo, ¿se puede implementar el algoritmo de forma eficaz en términos de complejidad de tiempo?  $O_0$  se basa en la computación de operaciones booleanas en  $n$  bits y se sabe que se puede implementar mediante  $O(n)$  puertas.

También hay dos capas de  $n$  puertas Hadamard. Por lo tanto, ambos componentes solo requieren  $O(n)$  puertas por iteración. Como  $N = 2^n$ , se deduce que

$O(n) = O(\log(N))$ . Por lo tanto, si necesitamos  $O \left( \sqrt{\frac{N}{M}} \right)$  iteraciones y  $O(\log(N))$

puertas por iteración, la complejidad de tiempo total (sin tener en cuenta la implementación del oráculo) es  $O \left( \sqrt{\frac{N}{M}} \log(N) \right)$ .

La complejidad general del algoritmo depende en última instancia de la complejidad de la implementación del  $O_{\text{fdeoracle}}$ . Si una evaluación de funciones es mucho más complicada en un equipo cuántico que en una clásica, el tiempo de ejecución del

algoritmo general será más largo en el caso cuántico, aunque técnicamente use menos consultas.

## Referencias

Si desea seguir aprendiendo sobre el algoritmo de Grover, puede comprobar cualquiera de los siguientes orígenes:

- [Documento original de Lov K. Grover](#) ↗
- Sección de algoritmos de búsqueda cuántica de Nielsen, M. A. & Chuang, I. L. (2010). Quantum Computation and Quantum Information (Computación cuántica e información cuántica).
- [Algoritmo de Grover en Arxiv.org](#) ↗

## Contenido relacionado

- [Entrelazamiento en la computación cuántica](#)
- [Vectores y matrices](#)
- [Puertas T y fábricas de T](#)
- [Varios cúbits](#)

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) ↗ | [Obtener ayuda en Microsoft Q&A](#)

# Convenciones de los diagrama de circuitos cuánticos

Artículo • 16/01/2025

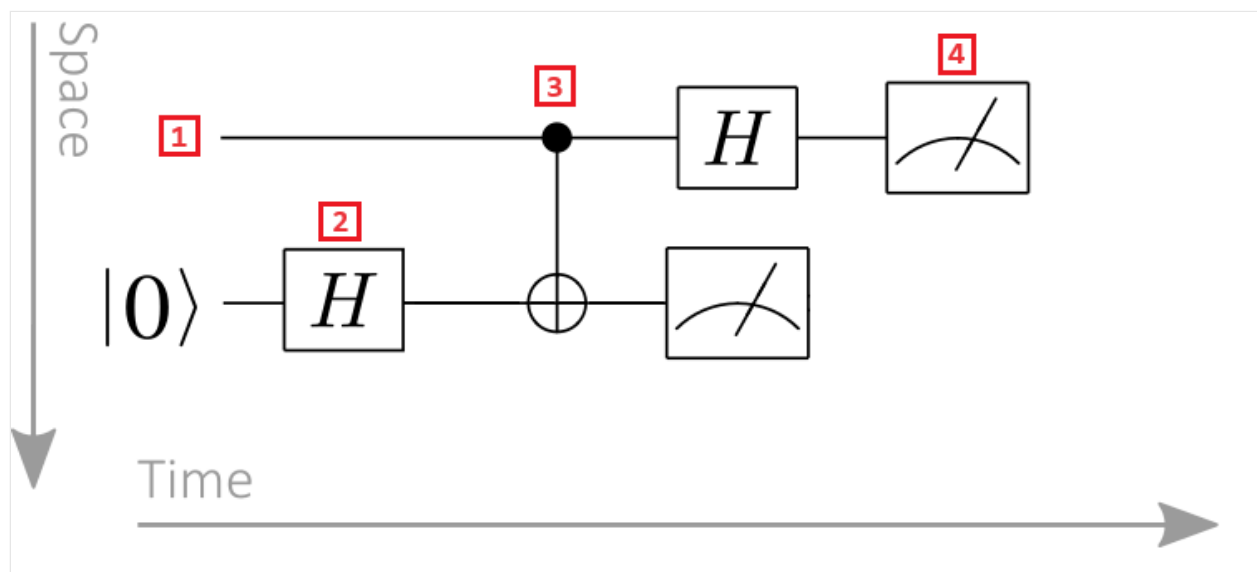
A veces, los algoritmos cuánticos son más fáciles de entender en un diagrama de circuitos que en la representación de matriz escrita equivalente. En este artículo se explica cómo leer diagramas de circuitos cuánticos y sus convenciones.

Para obtener más información, consulte [Visualización de diagramas de circuitos cuánticos](#).

## Lectura de diagramas de circuitos cuánticos

En un circuito cuántico, el tiempo fluye de izquierda a derecha. Las puertas cuánticas se ordenan cronológicamente; la puerta más a la izquierda es la que se aplica primero a los cúbits.

Tome el siguiente diagrama de circuitos cuánticos como ejemplo:



1. **Registro de cúbits:** los registros de cúbits se muestran como líneas horizontales, con cada línea que representa un cúbit. La línea superior es el registro de cúbits con la etiqueta 0, la segunda línea es el registro de cúbits con la etiqueta 1, etc.
2. **Puerta cuántica:** las operaciones cuánticas se representan mediante *puertas cuánticas*. El término puerta cuántica es análogo al de las puertas lógicas clásicas. Las puertas que actúan en uno o varios registros de cúbit se indican como un cuadro. En este ejemplo, el símbolo representa una operación Hadamard.
3. **Puerta controlada:** las puertas controladas actúan en dos o más cúbits. En este ejemplo, el símbolo representa una puerta CNOT. El círculo negro representa el

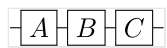


cúbit de control y la cruz dentro de un círculo representa el target cúbit.

4. **Operación de medida:** el símbolo de medidor representa una operación de medición. La operación de medición toma un registro de cúbits como entrada y genera información clásica.

## Aplicación de puertas cuánticas

Dado que el tiempo fluye de izquierda a derecha, la puerta más izquierda se aplica primero por ejemplo, la acción del siguiente circuito cuántico es la matriz  $unitaria CBA$ .



### ⚠ Nota

La multiplicación de matrices sigue la convención opuesta: primero se aplica la matriz más a la derecha. Sin embargo, en los diagramas de circuitos cuánticos, la puerta de la izquierda se aplica primero. En ocasiones, esta diferencia puede provocar confusión, por lo que es importante tener en cuenta esta diferencia importante entre la notación algebraica lineal y los diagramas de circuitos cuánticos.

## Entradas y salidas de circuitos cuánticos

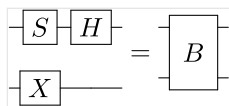
En un diagrama de circuito cuántico, los cables que entran en una puerta cuántica representan los cúbits que se introducen en la puerta cuántica y los cables que salen de la puerta cuántica representan los cúbits que se generan desde la puerta cuántica.

El número de entradas de una puerta cuántica es igual al número de salidas de una puerta cuántica. Esto se debe a que las operaciones cuánticas son unitarias y, por tanto, reversibles. Si una puerta cuántica tuviera más salidas que las entradas, no sería reversible y, por tanto, no unitaria, que es una contradicción.

Por este motivo, cualquier cuadro dibujado en un diagrama de circuito debe tener exactamente el mismo número de cables que entran y que salen.

## Operaciones de varios cúbits

Los diagramas de circuitos de varios cúbits siguen convenciones similares a los de un solo cúbit. Por ejemplo, se puede definir una operación  $unitariadedoscúbitsB$  para que sea  $(HS \otimes X)$ , por lo que el circuito cuántico equivalente es el siguiente:



También podemos ver que  $B$  tiene una acción en un solo registro de dos cúbits en lugar de dos registros de un cúbit, según el contexto en el que se use el circuito.

Quizás la propiedad más útil de estos diagramas de circuitos abstractos es que permiten describir algoritmos cuánticos complicados de forma general sin tener que compilarlos en puertas fundamentales. Esto significa que puede comprobar si su intuición sobre el flujo de datos de un algoritmo cuántico grande es cierta sin necesidad de comprender todos los detalles de cómo funciona cada una de las subrutinas dentro del algoritmo.

## Puertas controladas

Las puertas controladas por quantum son puertas de dos cúbits que aplican una puerta de un solo cúbit a un target cúbit si un cúbit de control está en un estado específico.

Por ejemplo, considere una puerta controlada cuántica, indicada  $\Lambda(G)$ , donde el valor de un solo cúbit controla la aplicación de la operación de  $G$ . La puerta

$\Lambda(G)$  se puede entender examinando el ejemplo siguiente de una entrada de estado del producto:

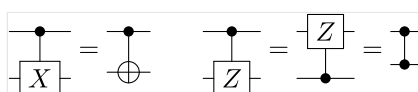
$$\Lambda(G) (\alpha|0\rangle + \beta|1\rangle) |\psi\rangle = \alpha|0\rangle |\psi\rangle + \beta|1\rangle G|\psi\rangle$$

Es decir, la puerta controlada aplica  $G$  al registro que contiene  $\psi$  si y solo si el cúbit de control toma el valor 1. En general, estas operaciones controladas se describen en diagramas de circuitos mediante el símbolo siguiente:



Aquí, el círculo negro denota el cúbit en el que se controla la puerta, y un cable vertical denota el unitario que se aplica cuando el cúbit de control toma el valor 1.

Para los casos especiales en *losque*  $G = X$  y  $G = Z$ , se usa la notación siguiente para describir la versión controlada de las puertas (tenga en cuenta que la puerta controlada- $X$  es la puerta CNOT):



Q# proporciona métodos para generar automáticamente la versión controlada de una operación, para que el programador no tenga que codificar manualmente estas

operaciones. A continuación se muestra un ejemplo:

Q#

```
operation PrepareSuperposition(qubit : Qubit) : Unit
is Ctl { // Auto-generate the controlled specialization of the operation
 H(qubit);
}
```

## Puertas controladas clásicamente

Las puertas cuánticas también se pueden aplicar después de una medida, donde el resultado de la medida actúa como un bit de control clásico.

El siguiente símbolo representa una puerta controlada clásicamente, donde  $G$  se aplica condicionada al valor 1 del bit de control clásico:



## Operador de medida

Las operaciones de medida toman un registro de cúbits, lo mide y genera el resultado como información clásica.

Una operación de medida se indica mediante un símbolo de medidor, siempre toma como entrada un registro de cúbit (que se indica mediante una línea continua) y genera información clásica (que se indica mediante una línea doble). En concreto, el símbolo de la operación de medición tiene el siguiente aspecto:



En Q#, el `Measure` operador implementa la operación de medición.

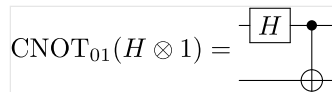
## Ejemplo: Transformación unitaria

Considere la transformación unitaria  $\text{CNOT}_{01}(H \otimes 1)$ . Esta secuencia de puertas tiene una importancia fundamental para la computación cuántica porque crea un estado de dos cúbits máximamente entrelazados:

$$\text{CNOT}_{01}(H \otimes 1) \ket{00} = \frac{1}{\sqrt{2}} (\ket{00} + \ket{11})$$

Las operaciones con esta complejidad o mayor son omnipresentes en los algoritmos cuánticos y en la corrección de errores cuánticos.

El diagrama de circuitos para preparar este estado cuántico máximamente entrelazado es:



El símbolo detrás de la puerta Hadamard representa una puerta CNOT, donde el círculo negro indica el cúbit de control y la cruz dentro de un círculo indica el target cúbit. En la representación anterior, el circuito cuántico actúa sobre dos cúbits (o dos registros equivalentes que constan de un cúbit).

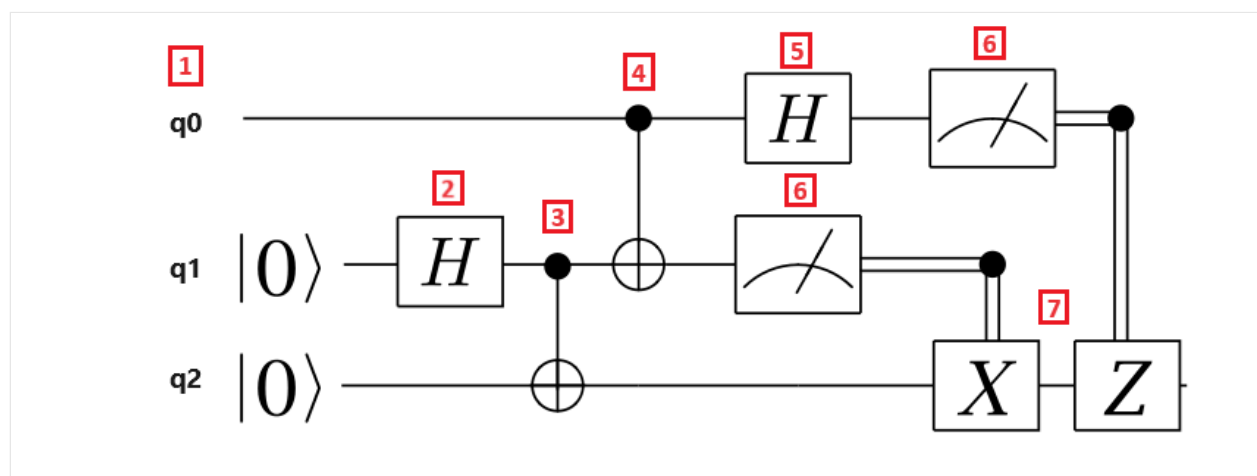
## Ejemplo: Diagrama del circuito de teletransportación

La teletransportación cuántica es uno de los mejores algoritmos cuánticos para ilustrar los componentes del circuito.

La teletransportación cuántica es un protocolo que permite transmitir un estado cuántico de un cúbit a otro, con la ayuda de un estado entrelazado compartido entre el remitente y el receptor, y la comunicación clásica entre ellos.

Con fines de aprendizaje, el remitente se denomina Alice, el receptor se llama Bob y el cúbit que se va a teletransportar se denomina cúbit del mensaje. Alice y Bob mantienen un cúbit cada uno, y Alice tiene un cúbit adicional que es el cúbit del mensaje.

En el diagrama de circuito siguiente se muestra el protocolo de teletransportación:



Vamos a desglosar los pasos del protocolo de teletransportación:

1. El registro **de cúbits q0** es el cúbit del mensaje, el registro **de cúbits q1** es el cúbit de Alice y el registro **de cúbits q2** es el cúbit de Bob. El cúbit del mensaje está en un estado desconocido y los cúbits de Alice y Bob están en estado  $|0\rangle$ .
2. Se aplica una **puerta** Hadamard al **cúbit** de Alice. Dado que el cúbit está en estado  $|0\rangle$ , el estado resultante es  $\frac{1}{\sqrt{2}}(\ket{0} + \ket{1})$ .
3. Una **puerta** CNOT se aplica a **los cúbits** de Alice y Bob. El cúbit de Alice es el cúbit de control y el cúbit de Bob es el target cúbit. El estado resultante es  $\frac{1}{\sqrt{2}}(\ket{00} + \ket{11})$ . Alice y Bob ahora comparten un estado entrelazado.
4. Se aplica una **puerta** CNOT al **cúbit del mensaje y al cúbit** de Alice. Dado que el cúbit de Alice también está entrelazado con el cúbit de Bob, el estado resultante es un estado entrelazado de tres cúbits.
5. Se aplica una **puerta** Hadamard al cúbit **del** mensaje.
6. Alice **mide** sus dos cúbits e indica los resultados de la medición a Bob, que no se refleja en el circuito. Los resultados de la medición son dos bits clásicos, que pueden tomar los valores 00, 01, 10 o 11.
7. Dos puertas **de Pauli controladas** clásicamente X y Z se aplican al **cúbit** de Bob, en función del valor 1 del bit de resultado. El estado resultante es el estado de cúbit del mensaje original.

## Contenido relacionado

- [Entrelazamiento cuántico](#)
- [Algoritmo de Grover](#)
- [Representación intermedia cuántica](#)
- [Vectores y matrices](#)

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Entrelazamiento y correlaciones

Artículo • 16/09/2024

El entrelazamiento es un concepto fundamental en la mecánica cuántica que describe una correlación cuántica entre sistemas cuánticos. Cuando dos o más cúbits están entrelazados, el estado de un cúbit depende del estado del otro cúbit, incluso si están lejos. Esta correlación cuántica es una característica única de los sistemas cuánticos que no tiene un homólogo clásico.

En este artículo se proporciona información general sobre el entrelazamiento, las correlaciones y se explica cómo crear un entrelazamiento mediante puertas cuánticas.

## ¿Qué es el entrelazamiento?

Imagine que tiene dos cúbits,  $A$  y  $B$ . Los cúbits son independientes entre sí, lo que significa que la información sobre el estado del cúbit  $A$ , sea cual sea, pertenece solo al cúbit  $A$ . Del mismo modo, la información sobre el estado del cúbit  $B$  pertenece al cúbit  $B$ . En este caso, los cúbits no están entrelazados, porque no comparten ninguna información sobre sus estados.

Ahora imagine que se entrelazan los cúbits. Si los cúbits  $A$  y  $B$  están entrelazados, la información sobre el estado del cúbit  $A$  no es independiente del estado del cúbit  $B$ . Cuando está entrelazado, la información se comparte entre ambos cúbits y no hay ninguna manera de conocer el estado de cúbit  $A$  o cúbit  $B$ . Solo puede describir el estado del sistema global, no el estado de los cúbits individuales.

El entrelazamiento es una **correlación cuántica** entre dos o más partículas. Si dos partículas están entrelazadas, no se pueden describir de forma independiente, sino solo como un sistema entero.

Se pueden entrelazar dos o más partículas aunque estén separadas por grandes distancias. Esta correlación es más fuerte que cualquier correlación clásica y es un recurso clave para las tareas de procesamiento de información cuántica, como la teletransportación cuántica, la criptografía cuántica y la computación cuántica. Si quiere aprender a teletransportar un cúbit mediante el entrelazamiento, consulte [este módulo en la ruta](#) de entrenamiento de Azure Quantum.

📌 Nota

El entrelazamiento es una propiedad de sistemas de varios cúbits, no de cúbits únicos. Es decir, no se puede entrelazar un solo cúbit.

## Definición del entrelazamiento en sistemas cuánticos

Imagine dos cúbits  $A$  y  $B$  de forma que el estado de  $|\phi\rangle$  del sistema global sea:

$$|\phi\rangle = \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle)$$

### ⓘ Nota

En notación Dirac,  $|0_A 0_B\rangle = |0\rangle_A |0\rangle_B$ . La primera posición corresponde al primer cúbit y la segunda posición corresponde al segundo cúbit.

$|\phi\rangle$  del sistema global está en una superposición de los estados  $|00\rangle$  y  $|11\rangle$ . ¿Pero cuál es el estado individual del cúbit  $A$ ? ¿Y del cúbit  $B$ ? Si intenta describir el estado del cúbit  $A$  sin tener en cuenta el estado de cúbit  $B$ , se produce un error. Los subsistemas  $A$  y  $B$  *están entrelazados* y no se pueden describir de forma independiente.

Si mide ambos cúbits, solo se pueden realizar dos resultados:  $|\ket{00}\rangle$  y  $|\ket{11}\rangle$ , cada uno con la misma probabilidad de  $\frac{1}{2}$ . La probabilidad de obtener los estados  $|01\rangle$  y  $|10\rangle$  es cero.

¿Pero qué ocurre si solo mide un cúbit? Cuando dos partículas están entrelazadas, los resultados de la medición también se correlacionan. Es decir, cualquier operación que ocurra con el estado de un cúbit en un par entrelazado, también afecta al estado del otro cúbit.

Si solo mide el cúbit  $A$  y obtiene el estado  $|0\rangle$ , significa que el sistema global se contrae al estado  $|00\rangle$ . Este es el único resultado posible, ya que la probabilidad de medir  $|01\rangle$  es cero. Por lo tanto, sin medir el bit cuántico  $B$ , puede estar seguro de que el segundo cúbit también está en *estado 0*. Los resultados de la medición se correlacionan porque los cúbits están entrelazados.

El estado  $|\phi\rangle$  cuántico se denomina **estado Bell**. Hay cuatro estados de Campana:

$$|\ket{\phi^+}\rangle = \frac{1}{\sqrt{2}}|\ket{00}\rangle + \frac{1}{\sqrt{2}}|\ket{11}\rangle$$

$$|\ket{\phi^-}\rangle = \frac{1}{\sqrt{2}}|\ket{00}\rangle - \frac{1}{\sqrt{2}}|\ket{11}\rangle$$

$$|\psi^+\rangle = \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle$$

$$|\psi^-\rangle = \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$$

#### ⓘ Nota

En este ejemplo se usan dos cúbits, pero el entrelazamiento cuántico no se limita a dos cúbits. En general, es posible que los sistemas de varios cúbits compartan entrelazamiento.

## Creación de un entrelazamiento con operaciones cuánticas

Puede usar operaciones cuánticas para crear un entrelazamiento cuántico. Una de las formas más comunes de crear entrelazamiento a dos cúbits en el estado  $|00\rangle$  es aplicando la operación *Hadamard*  $H$  y la operación *controladaNOT* para transformarlos en el estado  $|\text{phibell}^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ .

La *operación CNOT* toma dos cúbits como entrada, una actúa como cúbit de control y la otra es el cúbit de destino. La **CNOT** operación invierte el estado del cúbit de destino si, y solo si, el estado del cúbit de control es  $|1\rangle$ .

[Expandir tabla](#)

| Entrada      | Salida       |
|--------------|--------------|
| $ 00\rangle$ | $ 00\rangle$ |
| $ 01\rangle$ | $ 01\rangle$ |
| $ 10\rangle$ | $ 11\rangle$ |
| $ 11\rangle$ | $ 10\rangle$ |

Así es como funciona:

1. Tome dos cúbits en el estado  $|00\rangle$ . El primer cúbit es el cúbit de control y el segundo cúbit es el cúbit de destino.
2. Cree un estado de superposición solo en el cúbit de control aplicando  $H$ .



$$\frac{1}{\sqrt{2}}(|0_c\rangle + |1_c\rangle)$$

### ⚠ Nota

Los subíndices *cy* *tespecifican los cúbits* de control y destino.

3. Aplique el *operador CNOT* al cúbit de control, que se encuentra en un estado de superposición y el cúbit de destino, que se encuentra en el estado  $|0_t\rangle$ .

$$CNOT \frac{1}{\sqrt{2}}(|\ket{0_c} + \ket{1_c}\ket{0_t}\rangle) = \frac{1}{\sqrt{2}}(|\ket{0_c 0_t} + \ket{1_c 0_t}\rangle)$$

$$= \frac{1}{\sqrt{2}}(CNOT |0_c 0_t\rangle + CNOT |1_c 0_t\rangle)$$

$$\frac{1}{\sqrt{2}}$$

Extra close brace or missing open brace

### 💡 Sugerencia

Para obtener información sobre cómo entrelazar dos cúbits con Q#, consulte [Inicio rápido: Creación del primer Q# programa](#).

## Separación y entrelazamiento cuántico

El entrelazamiento se puede ver como la falta de separabilidad: un estado está entrelazado cuando no es separable.

Un estado cuántico es separable si se puede escribir como un estado de producto de los subsistemas. Es decir, un estado  $|\ket{\phi}\rangle_{AB}$  es separable si se puede escribir como una combinación de estados de producto de los subsistemas, es decir,  $|\ket{\phi}\rangle_{AB} = |\ket{a}\rangle_A |\ket{b}\rangle_B$ .

## Entrelazamiento en estados puros

Un estado cuántico puro es un único vector de ket, como el estado  $\frac{1}{\sqrt{2}}(|\ket{0}\rangle + |\ket{1}\rangle)$ .

Los estados puros no se pueden escribir como una mezcla estadística (o *combinación convexa*) de otros estados cuánticos.

En la [esfera](#) Bloch, los estados puros se representan mediante un punto en la superficie de la esfera, mientras que los estados mixtos se representan mediante un punto interior.

Un estado **Extra close brace or missing open brace** está entrelazado si no se puede escribir como una combinación de estados de producto de los subsistemas, es decir,  $AB = |a\rangle_A |\otimes b\rangle_B$ .

Por ejemplo, considere el estado  $\frac{1}{\sqrt{2}} (\ket{00} + \ket{10} + \ket{01} + \ket{11})$

Al principio, el estado  $|\psi\rangle_{AB}$  no se parece a un estado de producto, pero si se vuelve a escribir el estado como

$$\frac{1}{\sqrt{2}} (\ket{0}_A + \ket{1}_A) \otimes \frac{1}{\sqrt{2}} (\ket{0}_B + \ket{1}_B) = \ket{+}_A \ket{+}_B$$

el estado  $|\psi\rangle_{AB}$  es un estado de producto, por lo que no está entrelazado.

## Entrelazamiento en estados mixtos

Los estados cuánticos mixtos son un conjunto estadístico de estados puros. Para describir los estados mixtos es más fácil usar su matriz *densidad*  $\rho$  en lugar de la notación de ket.

Un estado *mixto* es separable si se puede escribir como una *combinación* convexa de estados de producto de los subsistemas, como

$$\rho = \sum_j p_j \rho_j^A \otimes \rho_j^B$$

donde  $p_j \geq 0$ ,  $\sum p_j = 1$  y  $\rho_j^A \geq 0$ ,  $\rho_j^B \geq 0$ .

Para más información, consulte [Matrices](#) de densidad.

Un estado *mixto* está entrelazado si no es separable, es decir, no se puede escribir como una combinación convexa de estados de producto.

### ⓘ Nota

- Si un estado *entrelazado* es puro, solo contiene correlaciones cuánticas.
- Si se mezcla un estado *entrelazado*, contiene correlaciones clásicas y cuánticas.

# Descripción de las correlaciones clásicas

Las correlaciones clásicas se deben a la falta de conocimiento del estado del sistema. Es decir, hay cierta aleatoriedad asociada a la correlación clásica, pero se puede eliminar mediante la obtención de conocimientos.

Por ejemplo, considere dos cuadros, cada uno que contenga una bola. Sabes que ambas bolas son del mismo color, azul o rojo. Si abre una caja y descubre que la bola dentro es azul, entonces sabemos que la otra bola también es azul. Por lo tanto, están correlacionados. Sin embargo, la incertidumbre que tenemos al abrir la caja se debe a nuestra falta de conocimiento, no es fundamental. La bola era azul antes de abrir la caja. Por lo tanto, se trata de una correlación clásica, no una correlación cuántica.

El estado cuántico mixto del sistema formado por los dos cuadros  $\rho_{boxes}$  se puede escribir como

$$\rho_{boxes} = \frac{1}{2} (\ket{rojo\ rojo}\bra{rojo} \otimes \ket{\phantom{rojo}}\bra{B}) + \frac{1}{2} (\ket{azul}\bra{azul_A} \otimes \ket{\phantom{azul}}\bra{B})$$

Tenga en cuenta que el estado  $\rho_{boxes}$  es separable, donde  $p_1 p_2 = \frac{1}{2}$  después solo contiene correlaciones clásicas. Otro ejemplo de un estado separador mixto es

$$\rho = \frac{1}{2} (\ket{0}\bra{0_A} \otimes \ket{0}\bra{0_B}) + \frac{1}{2} (\ket{1}\bra{1_A} \otimes \ket{1}\bra{1_B})$$

Ahora, tenga en cuenta el siguiente estado:

$$\rho = \frac{1}{4} (\ket{00}\bra{00} + \ket{00}\bra{11} + \ket{11}\bra{00} + \ket{11}\bra{11}) = \ket{\phi^+}\bra{\phi^+}$$

En este caso, nuestro conocimiento del estado es perfecto, sabemos con máxima certeza que el sistema  $AB$  está en el estado  $\ket{\text{phibell}^+}$  y  $\rho$  es un estado puro. Por lo tanto, no hay correlaciones clásicas. Pero si medimos un observable en el subsistema  $A$ , obtenemos un resultado aleatorio que nos proporciona información sobre el estado del subsistema  $B$ . Esta aleatoriedad es fundamental, es decir, estas son correlaciones cuánticas.

Un ejemplo de un estado cuántico que contiene correlaciones clásicas y cuánticas es

$$\rho = \frac{1}{2} (\ket{\phi^+}\bra{\phi^+} + \ket{\phi^-}\bra{\phi^-})$$

## ❗ Nota

Un estado separable solo contiene correlaciones clásicas.

# Contenido relacionado

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Introducción a la corrección de errores cuánticos

Artículo • 16/09/2024

En este artículo se explican los conceptos básicos de la corrección de errores cuánticos, los tipos de errores cuánticos y algunos códigos comunes de corrección de errores cuánticos. También proporciona un ejemplo de cómo corregir errores mediante el código de tres cúbits.

## ¿Qué es la corrección de errores cuánticos?

La corrección cuántica de errores (QEC) es una técnica que nos permite proteger la información cuántica frente a errores. La corrección de errores es especialmente importante en los equipos cuánticos, ya que los algoritmos cuánticos eficaces usan equipos cuánticos a gran escala, que son sensibles al ruido.

El principio básico detrás de la corrección de errores cuánticos es que se aumenta el número de bits que se usan para codificar una cantidad determinada de información. Esta redundancia permite que el código detecte y corrija errores.

Las tasas de error de los equipos cuánticos suelen ser mayores que los errores del equipo clásico debido a los desafíos asociados a la creación y el funcionamiento de sistemas cuánticos. El ruido, la decoherencia y las imperfecciones en puertas cuánticas pueden provocar errores en cálculos cuánticos. Los equipos cuánticos actuales tienen tasas de error en el intervalo del 1 % al 0,1 %. En otras palabras, esto significa que, en promedio, una de cada 100 a 1000 operaciones de puerta cuántica produce un error.

## Tipos de errores cuánticos

Hay dos tipos fundamentales de errores cuánticos: volteos de bits y volteos de fase.

Los errores de volteo de bits se producen cuando un cúbit cambia de  $|0\rangle$  a  $|1\rangle$  o viceversa. Los errores de volteo de bits también se conocen como  $\sigma_x$ -error, ya que asignan los estados de cúbit  $\sigma_x|1\rangle = |0\rangle$  y  $\sigma_x|0\rangle = |1\rangle$ . Este error es análogo a un error de volteo de bits clásico.

Los errores de volteo de fase se producen cuando un cúbit cambia su fase. También se conocen como  $\sigma_z$ -error, ya que asignan los estados de cúbit  $\sigma_z|1\rangle = -|1\rangle$  y  $\sigma_z|0\rangle = |0\rangle$ . Este tipo de error no tiene ninguna analogía clásica.

En la computación cuántica, los errores cuánticos pueden manifestarse como volteos de bits, volteos de fase o una combinación de ambos.

## ¿Cómo funciona la corrección cuántica de errores?

Los códigos de corrección de errores cuánticos funcionan codificando la información cuántica en un conjunto mayor de cúbits, denominados *cúbits físicos*. El estado conjunto de los cúbits físicos representa un *cúbit* lógico.

Los cúbits físicos están sujetos a errores debido a la decoherencia y las imperfecciones en las puertas cuánticas. El código está diseñado para que se puedan detectar y corregir errores mediante la medición de algunos de los cúbits del código.

Por ejemplo, imagine que desea enviar el mensaje  $|0\rangle$  de un solo cúbit. Puede usar tres cúbits físicos para codificar el mensaje, enviando  $|000\rangle$ , que se conoce como codeword. Este código de corrección de errores es un *código* de repetición, porque el mensaje se repite tres veces.

Ahora, imagine que se produce un único error de volteo de bits durante la transmisión para que lo que recibe el destinatario sea el estado  $|010\rangle$ . En este escenario, el destinatario puede deducir que el mensaje previsto es  $|000\rangle$ . Sin embargo, si el mensaje está sujeto a dos errores de volteo de bits, el destinatario puede deducir un mensaje incorrecto. Por último, si se voltean los tres bits para que el mensaje  $|000\rangle$  original se convierta en  $|111\rangle$ , el destinatario no tiene forma de saber si se produjo un error.

La distancia de código de un código QEC es el número mínimo de errores que cambian una palabra de código a otra, es decir, el número de errores que no se pueden detectar. La distancia *decódigod* se puede definir como

$$d = 2t + 1$$

donde  $t$  es el número de errores que el código puede corregir. Por ejemplo, el código de tres bits puede detectar y corregir un error de volteo de bits, por lo que  $t = 1$  y, por tanto, la distancia del código es  $d = 3$ .

Tenga en cuenta que los códigos de repetición, como el código de tres bits usado en este ejemplo, solo pueden corregir errores de volteo de bits y no los errores de volteo de fase. Para corregir ambos tipos de errores, se necesitan códigos de corrección de errores cuánticos más sofisticados.

## Tipos de códigos QEC

Hay muchos tipos diferentes de códigos QEC, cada uno con sus propias propiedades y ventajas. Algunos códigos de QEC comunes son:

- **Código de repetición:** el código de corrección de errores cuánticos más sencillo, donde un solo cúbit se codifica en varios cúbits repitiéndolo varias veces. El código de repetición puede corregir errores de volteo de bits, pero no errores de volteo por fases.
- **Código Shor:** el primer código de corrección de errores cuánticos, desarrollado por Peter Shor. Codifica un cúbit lógico en nueve cúbits físicos. El código Shor puede corregir un error de volteo de un solo bit o un error de volteo de fase, pero no puede corregir ambos tipos de errores al mismo tiempo.
- **Código steane:** se trata de un código de siete cúbits que puede corregir errores de volteo de bits y de volteo de fase. Tiene la ventaja de ser tolerante a errores, lo que significa que el propio proceso de corrección de errores no introduce errores adicionales.
- **Código de superficie:** se trata de un código de corrección de errores topológico que usa una red multidimensional de cúbits bidimensionales para codificar cúbits lógicos. Tiene un umbral elevado de corrección de errores y se considera una de las técnicas más prometedoras para la computación cuántica a gran escala y tolerante a errores. El estimador [de recursos de Azure Quantum usa el](#) código de superficie.
- **Código Hastings-Haah:** este código de corrección de errores cuánticos ofrece mejores costos de tiempo espaciado que los códigos de superficie en cúbits majorana en muchos regímenes. En el caso de los conjuntos de instrucciones basados en puertas, la sobrecarga es mayor, lo que hace que este enfoque sea menos eficaz que el código de superficie.

## Ejemplo: el código de tres cúbits

El código de corrección de errores de tres cúbits es un código de repetición simple que puede detectar y corregir un error de volteo de un bit. Codifica un único cúbit lógico en tres cúbits físicos repitiendo el cúbit tres veces.

### Sugerencia

Consulte el [Q# ejemplo de código para el código](#)  de tres cúbits.

Imagine que desea enviar un cúbit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ . Para evitar errores, codifique los estados  $|0\rangle$  y  $|1\rangle$  en un estado conjunto de tres cúbits. Los dos estados de base lógicos son  $|0_L\rangle = |000\rangle$  y  $|1_L\rangle = |111\rangle$ .

Por lo tanto, el único cúbit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  se codifica como:

$$|\psi\rangle_{000} = \alpha|0_L\rangle + \beta|1_L\rangle = \alpha|000\rangle + \beta|111\rangle$$

Vamos a desglosar los pasos del código de tres cúbits.

## Preparación de los cúbits

En primer lugar, codifica el único cúbit  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  en un estado conjunto de tres cúbits.

A continuación, preparará dos cúbits adicionales en el estado  $|0\rangle$ . Por lo tanto, el estado global de los tres cúbits es  $(\alpha|0\rangle + \beta|1\rangle)|00\rangle = \alpha|000\rangle + \beta|100\rangle$ .

Por último, puede codificar el único cúbit en un estado conjunto de tres cúbits mediante la aplicación de dos operaciones CNOT. El primer CNOT usa el primer cúbit como control y actúa en el segundo cúbit, produciendo  $\alpha|000\rangle + \beta|110\rangle$ . El segundo CNOT usa el primer cúbit como control y actúa en el tercer cúbit. El estado de los tres cúbits es ahora  $\alpha|000\rangle + \beta|111\rangle$ .

## Envío de cúbits

Envía los tres cúbits. Suponiendo que solo se pueden producir errores de volteo de un bit, los cúbits recibidos se encuentran en uno de los estados siguientes:

[Expandir tabla](#)

| Estado                                 | Error       |
|----------------------------------------|-------------|
| $\alpha 000\rangle + \beta 111\rangle$ | Sin errores |
| $\alpha 100\rangle + \beta 011\rangle$ | Cúbit 1     |
| $\alpha 010\rangle + \beta 101\rangle$ | Cúbit 2     |
| $\alpha 001\rangle + \beta 110\rangle$ | Cúbit 3     |

## Adición de cúbits auxiliares



En primer lugar, se presentan dos cúbits más, preparados en el estado  $|00\rangle$ . Este par auxiliar de cúbits se usa para extraer información del error sin medir ni obtener información directamente sobre el estado lógico.

A continuación, se llevan a cabo cuatro operaciones CNOT: las dos primeras usan los cúbits primero y segundo recibidos como control y actúan sobre el primer cúbit auxiliar, y las dos últimas operaciones usan los primeros y los tres cúbits recibidos como control y actúan en el segundo bit auxiliar. El estado total de los cinco cúbits es ahora:

[Expandir tabla](#)

| Estado                                             | Error       |
|----------------------------------------------------|-------------|
| $(\alpha 000\rangle + \beta 111\rangle) 00\rangle$ | Sin errores |
| $(\alpha 100\rangle + \beta 011\rangle) 11\rangle$ | Cúbit 1     |
| $(\alpha 010\rangle + \beta 101\rangle) 10\rangle$ | Cúbit 2     |
| $(\alpha 001\rangle + \beta 110\rangle) 01\rangle$ | Cúbit 3     |

## Recuperación del síndrome de error

Para recuperar la información de error, mida los dos cúbits auxiliares en los estados  $|0\rangle$  de base computacional y  $|1\rangle$ . Al hacerlo, se recupera el estado conjunto, que se denomina *síndrome de error* porque ayuda a diagnosticar los errores en los cúbits recibidos.

Ahora sabe en cuál de los cuatro estados posibles están los tres cúbits recibidos. Puede corregir el error aplicando la operación de corrección. En este caso, está tratando con errores de volteo de bits, por lo que la corrección es una *operación*  $\sigma_x$  aplicada a uno (o ninguno) de los cúbits.

Por ejemplo, si el síndrome de error es  $|00\rangle$ , los cúbits recibidos están en el estado  $\alpha|000\rangle + \beta|111\rangle$ , que es el estado que envió originalmente. Si el síndrome de errores es  $|11\rangle$ , los cúbits recibidos se encuentran en el estado  $\alpha|100\rangle + \beta|011\rangle$ . Hay un error de volteo en el primer cúbit, que puede corregirse aplicando una operación  $\sigma_x$  al primer cúbit.

[Expandir tabla](#)

| Síndrome de error | Contraer estado                          | Corrección                   |
|-------------------|------------------------------------------|------------------------------|
| $ 00\rangle$      | $\alpha  000\rangle + \beta  111\rangle$ | No hacer nada                |
| $ 01\rangle$      | $\alpha  100\rangle + \beta  011\rangle$ | Aplicar $\sigma_x$ a cúbit 3 |
| $ 10\rangle$      | $\alpha  010\rangle + \beta  101\rangle$ | Aplicar $\sigma_x$ a cúbit 2 |
| $ 11\rangle$      | $\alpha  001\rangle + \beta  110\rangle$ | Aplicar $\sigma_x$ a cúbit 1 |

## Extracción del cúbit original

Por último, para extraer el único cúbit que quería transmitir originalmente, se aplican dos operaciones CNOT: una usa el primer cúbit como control y actúa en el segundo cúbit, y otro usa el primer cúbit como control y actúa en el tercero.

El estado del primer cúbit es ahora  $\alpha |0\rangle + \beta |1\rangle$ , que es el cúbit original que quería transmitir.

### Importante

El código QEC no obtiene información sobre los coeficientes  $\alpha$  y  $\beta$ , por lo tanto, las superposiciones del estado computacional permanecen intactas durante la corrección.

## Comentarios

¿Le ha resultado útil esta página?  Sí  No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Representación intermedia cuántica

Artículo • 14/02/2025

Quantum Intermediate Representation (QIR) es una representación intermedia que actúa como una interfaz común entre los lenguajes de programación cuántica o los marcos de trabajo y las plataformas de cálculo cuánticos de destino. QIR especifica un conjunto de reglas para representar programas cuánticos mediante un lenguaje y un formato independiente del hardware en el IR de LLVM. El QIR es un proyecto que ha desarrollado la Alianza QIR de la que Microsoft es miembro.

## ¿Qué es una representación intermedia?

Un patrón común en los compiladores clásicos es empezar por compilar el idioma de origen en una representación intermedia. Una representación intermedia es, como su propio nombre indica, un paso intermedio en la conversión de instrucciones del código fuente al lenguaje de la máquina.

Una representación intermedia actúa como una representación abstracta de un programa. Todos los programas, independientemente del lenguaje en el que estén escritos, se traducen en esta representación intermedia mediante lo que se conoce como un compilador front-end, mientras que un componente back-end es el responsable de traducir esa representación intermedia en una representación de máquina. La representación intermedia permite separar los idiomas de origen de las plataformas de hardware y permite crear un compilador de manera modular, donde cada lenguaje nuevo solo requiere que se admita un front-end nuevo en todas las plataformas para las que está disponible un back-end.

Normalmente, la representación intermedia está diseñada para permitir que se representen muchos idiomas de origen diferentes. Además, en este nivel intermedio también es posible realizar alguna optimización y reorganización del circuito que hace que la implementación final sea más eficaz. Una vez conocida la plataforma de ejecución final target, la representación intermedia se puede compilar en código ejecutable real.

Este enfoque permite que muchos idiomas de origen compartan un conjunto común de optimizadores y generadores ejecutables. También facilita la compilación de un único lenguaje de origen para muchos distintos targets. La representación intermedia proporciona una plataforma común que se puede compartir entre muchos orígenes y targets permite una gran cantidad de reutilización en la maquinaria del compilador.

# ¿Qué es Quantum Intermediate Representation?

QIR es una representación intermedia para programas cuánticos que desarrolla la [Alianza QIR](#), a la que pertenece Microsoft. Proporciona una interfaz común que admite muchos lenguajes y target plataformas para el cálculo cuántico. Puede pensar que QIR es un lenguaje universal de nivel medio que permite la comunicación entre lenguajes de alto nivel y máquinas. Aunque Q# compila en QIR, QIR no es específico de Q#: cualquier marco de programación cuántica puede aprovechar QIR para representar un programa cuántico. Es independiente del hardware, lo que significa que no especifica una instrucción cuántica o un conjunto de puertas, lo que se deja al entorno informático target.

QIR se basa en el popular compilador clásico [LLVM](#) de código abierto. LLVM es una colección de tecnologías de compiladores y cadenas de herramientas modulares y reutilizables que ha sido adaptada por un amplio conjunto de lenguajes. QIR especifica un conjunto de reglas para representar construcciones cuánticas en LLVM, pero no requiere extensiones ni modificaciones en LLVM.

El hecho de que LLVM sea la cadena de herramientas subyacente significa que QIR es por naturaleza capaz de procesar la lógica clásica y cuántica. Esta característica resulta esencial para los algoritmos híbridos clásicos-cuánticos, que se han vuelto cada vez más importantes para las aplicaciones de la computación cuántica. Además, permite sacar provecho de las herramientas de compilación y optimización del sector informático clásico y, por tanto, reducir el costo de la escritura de traducciones.

Muchos de los sectores principales de computación cuántica ya han adoptado QIR. Por ejemplo, NVIDIA, Oak Ridge National Laboratory, Quantinuum, Quantum Circuits Inc. y Rigetti Computing están creando cadenas de herramientas que sacan provecho de QIR.

Para obtener más información, consulte la [especificación](#) QIR. Si está interesado en las herramientas y proyectos del compilador que usan QIR, observe estos [repositorios de QIR](#).

## ¿Qué es la Alianza QIR?

QIR Alliance es un esfuerzo conjunto para desarrollar una representación intermedia cuántica con el objetivo de permitir la interoperabilidad completa dentro del ecosistema cuántico, reducir el esfuerzo de desarrollo de todas las partes y proporcionar una representación adecuada para los procesadores cuánticos actuales y futuros heterogéneos.

Los SDK y lenguajes cuánticos aparecen y evolucionan a un ritmo rápido, junto con nuevos procesadores cuánticos con funcionalidades únicas y distintas entre sí. Para proporcionar interoperabilidad entre los nuevos lenguajes y las nuevas funcionalidades de hardware, es imperativo que el ecosistema desarrolle y comparta una representación intermedia que funcione con hardware cuántico presente y futuro.

Con su trabajo colectivo y su asociación, la Alianza QIR tiene como objetivo:

- Reduzca el trabajo de desarrollo necesario para todas las partes mediante la promoción de la interoperabilidad entre diferentes marcos y lenguajes.
- Habilite el desarrollo de bibliotecas compartidas tanto para el desarrollo de aplicaciones cuánticas como para el desarrollo de compiladores cuánticos.
- Se basa en la tecnología de compilador de última generación y saca provecho de las herramientas, bibliotecas y aprendizajes existentes de la informática de alto rendimiento.
- Permita la evolución incremental y progresiva en la manera en que los cálculos clásicos y cuánticos pueden interactuar a nivel de hardware.
- Proporcione flexibilidad para conectar con facilidad las tecnologías emergentes, de manera que permita la experimentación con funcionalidades de hardware distintas y diferenciadas.

La Alianza QIR forma parte de la [Joint Development Foundation de la Fundación Linux](#) y trabaja en estándares abiertos. Los miembros fundadores incluyen Microsoft, así como Quantinuum, Oak Ridge National Laboratory, Quantum Circuits Inc. y Rigetti Computing.

## ¿Cómo es Quantum Intermediate Representation?

Dado que QIR se basa en LLVM, es similar a LLVM.

Por ejemplo, tenga en cuenta el código siguiente Q# para generar un par Bell:

```
Q#

operation CreateBellPair(q1 : Qubit, q2 : Qubit) : Unit {
 H(q1);
 CNOT(q1, q2);
}
```

Cuando se compila en QIR, se convierte en:

```
define void @CreateBellPair__body(%Qubit* %q1, %Qubit* %q2) {
entry:
 call void @__quantum__qis__h(%Qubit* %q1)
 call void @__quantum__qis__cnot(%Qubit* %q1, %Qubit* %q2)
 ret void
}
```

En este fragmento de código, puede ver algunas características de QIR:

- Las operaciones de Q# (o cualquier otro lenguaje de programación cuántica) se representan mediante funciones LLVM.
- Los cúbits se representan como punteros a un tipo de estructura opaca denominado `%Qubit`.

Aunque el QIR para la operación `CreateBellPair` es muy sencillo, QIR hereda todas las funcionalidades de LLVM para expresar bucles, condicionales y otro flujo de control complejo. QIR también hereda la capacidad de LLVM de expresar cálculos clásicos arbitrarios.

Para obtener más información, vea la sesión de los desarrolladores de Microsoft del [evento Q2B de 2021](#).

## ¿Por qué es Quantum Intermediate Representation importante?

QIR es una herramienta esencial para ejecutar algoritmos cuánticos en hardware real. Pero las representaciones intermedias pueden desempeñar un papel importante, incluso si solo quiere desarrollar algoritmos en un nivel más teórico.

Por ejemplo, una aplicación habilitada por QIR es usar [el compilador](#) Clang, un front-end del lenguaje C para LLVM, para compilar QIR en código de máquina ejecutable para un clásico target. Esto proporciona una ruta de acceso sencilla para compilar un simulador en C o C++ mediante la implementación de las instrucciones cuánticas, lo que puede simplificar la creación de simuladores cuánticos.

Además, puede usar la representación intermedia para generar código que se proporcione posteriormente como entrada en un simulador cuántico, en lugar de un dispositivo real, que puede usar un lenguaje diferente al código fuente. De esta manera, puede comparar y usar como punto de referencia diferentes lenguajes o simuladores con facilidad mediante un marco común.

En términos de optimización de código, hay pasos de optimización que se pueden realizar en el nivel intermedio que pueden hacer que la implementación general del algoritmo sea más eficaz. La investigación de esta optimización del código de entrada puede ayudarle a comprender mejor dónde hacer que los algoritmos sean más eficaces y cómo mejorar los lenguajes de programación cuánticos.

Otra aplicación consiste en usar la infraestructura "pass" estándar de LLVM para crear optimizadores de código cuántico que operan en QIR. El enfoque independiente del lenguaje y del hardware de QIR permite reutilizar esos optimizadores para diferentes lenguajes de cálculo y plataformas de computación sin apenas esfuerzo.

## Contenido relacionado

- [Introducción a Q#](#)
- [Introducción a la estimación de recursos](#)
- [Proveedor Rigetti](#)

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

## Artículo • 16/10/2024

Lo importante es que definir un oráculo de esta manera para cada estado de base computacional  $|x\rangle |y\rangle$  también define cómo actúa  $O$  para cualquier otro estado. Este comportamiento sigue inmediatamente del hecho de que  $O$ , *aligualquetodas* las



operaciones cuánticas, es lineal en el estado en el que actúa. Considere la operación Hadamard, por ejemplo, que se define  $H|0\rangle = |+\rangle$  y  $H|1\rangle = |-\rangle$ . Si desea saber cómo actúa  $H$  en  $|+\rangle$ , puede usar que  $H$  es lineal,

$$\begin{aligned} H|\text{ket}{+}\rangle &= \frac{1}{\sqrt{2}} (H|\text{ket}{0}\rangle + |\text{ket}{1}\rangle) = \frac{1}{\sqrt{2}} (H|\text{ket}{1}\rangle + |\text{ket}{0}\rangle) \\ &= \frac{1}{\sqrt{2}} (|\text{ket}{+}\rangle + |\text{ket}{-}\rangle) = \frac{1}{2} (|\text{ket}{+}\rangle + |\text{ket}{-}\rangle + |\text{ket}{+}\rangle - |\text{ket}{-}\rangle) = |\text{ket}{0}\rangle \end{aligned}$$

Al definir el  $O$  de oracle, puede usar de formas similares que cualquier estado  $|\psi\rangle$  en  $n + m$  cúbits se puede describir como

$$|\psi\rangle = \sum_{x \in \{0, 1\}^n, y \in \{0, 1\}^m} \alpha(x, y) |\text{ket}{x}\rangle |\text{ket}{y}\rangle.$$

Aquí,  $\alpha : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \mathbb{C}$  representa los coeficientes del estado  $|\psi\rangle$ . Así,

$$\begin{aligned} O|\psi\rangle &= O \sum_{x \in \{0, 1\}^n, y \in \{0, 1\}^m} \alpha(x, y) |\text{ket}{x}\rangle |\text{ket}{y}\rangle \\ &= \sum_{x \in \{0, 1\}^n, y \in \{0, 1\}^m} \alpha(x, y) O|\text{ket}{x}\rangle |\text{ket}{y}\rangle \\ &= \sum_{x \in \{0, 1\}^n, y \in \{0, 1\}^m} \alpha(x, y) |\text{ket}{x}\rangle |\text{ket}{y} \oplus f(x)\rangle. \end{aligned}$$

## Oráculos de fase

Como alternativa, puede codificar  $f$  en un *Oracle* aplicando una *fase* basada en la entrada a  $O$ . Por ejemplo, puede definir *Oracle* que

$$O|\text{ket}{x}\rangle = (-1)^{f(x)} |\text{ket}{x}\rangle.$$

Si un oráculo de fase actúa en un registro inicialmente en un estado de base computacional  $|x\rangle$ , esta fase es una fase global y, por lo tanto, no se puede observar. Pero este tipo de oráculo puede ser un recurso eficaz si se aplica a una superposición o como una operación controlada. Por ejemplo, considere un oráculo de fase  $O_f$  para una función  $f$  de un único cúbit. A continuación,

$$\begin{aligned} O_f|\text{ket}{+}\rangle &= O_f \frac{1}{\sqrt{2}} (|\text{ket}{0}\rangle + |\text{ket}{1}\rangle) \\ &= \frac{1}{\sqrt{2}} ((-1)^{f(0)} |\text{ket}{0}\rangle + (-1)^{f(1)} |\text{ket}{1}\rangle) \\ &= \frac{1}{2} ((-1)^{f(0)} (|\text{ket}{0}\rangle + |\text{ket}{1}\rangle) + (-1)^{f(1) - f(0)} (|\text{ket}{0}\rangle - |\text{ket}{1}\rangle)) \\ &= (-1)^{f(0)} \frac{1}{\sqrt{2}} (|\text{ket}{0}\rangle + |\text{ket}{1}\rangle) = (-1)^{f(0)} |\text{ket}{+}\rangle \end{aligned}$$

### ⚠ Nota

Tenga en cuenta que  $Z^{-1} = Z^\dagger = Z$  y, por lo tanto,  $Z^{f(0)-f(1)} = Z^{f(1)-f(0)}$ .

Por lo general, ambas vistas de oráculos se pueden ampliar para representar funciones clásicas, que devuelven números reales en lugar de solo un solo bit.

Elegir la mejor manera de implementar un oráculo depende en gran medida de cómo se va a usar este oráculo dentro de un algoritmo determinado. Por ejemplo, el [algoritmo de Deutsch-Jozsa](#) se basa en el oráculo implementado de la primera manera, mientras que el [algoritmo de Grover](#) se basa en el oráculo implementado de la segunda manera.

Para obtener más información, consulte la discusión en [Gilyén 1711.00465](#).

## Contenido relacionado

- [Algoritmo de Grover](#)
- [Representación intermedia cuántica](#)
- [Vectores y matrices](#)
- [Puertas T y fábricas de T](#)

---

## Comentarios

¿Le ha resultado útil esta página?

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

# El rol de las puertas T y las factorías de T en la computación cuántica

Artículo • 16/09/2024

En este artículo se describe el rol de las puertas T y las factorías de T en la computación cuántica tolerante a errores. Dado un algoritmo cuántico, la estimación de los recursos necesarios para ejecutar las puertas T y las factorías de T es fundamental para determinar la viabilidad del algoritmo. El estimador de recursos de Azure Quantum calcula el número de estados T necesarios para ejecutar el algoritmo, el número de cúbits físicos para una única factoría de T y el tiempo de ejecución de la factoría de T.

## Conjunto universal de puertas cuánticas

Según los [criterios](#) de DiVincenzo, un equipo cuántico escalable debe ser capaz de implementar un **conjunto universal de puertas cuánticas**. Un *conjunto* universal contiene todas las puertas necesarias para realizar cualquier cálculo cuántico, es decir, cualquier cálculo debe volver a descomponerse en una secuencia finita de puertas universales. Como mínimo, un equipo cuántico debe poder mover cúbits únicos a cualquier posición de la esfera bloch (mediante puertas de un solo cúbit), así como introducir entrelazamiento en el sistema, lo que requiere una puerta de varios cúbits.

Solo hay cuatro funciones que asignan un bit a un bit en un equipo clásico. En cambio, hay un número infinito de transformaciones unitarias en un solo cúbit en un equipo cuántico. Por lo tanto, ningún conjunto finito de operaciones cuánticas primitivas o puertas, puede replicar exactamente el conjunto infinito de transformaciones unitarias permitidas en la computación cuántica. Esto significa que, a diferencia de la computación clásica, es imposible que un equipo cuántico implemente exactamente todos los programas cuánticos posibles utilizando un número finito de puertas. Por tanto, los equipos cuánticos no pueden ser universales en el mismo sentido que los equipos clásicos. Como resultado, cuando decimos que un conjunto de puertas es *universal* para la computación cuántica, nos referimos en realidad a algo ligeramente más débil de lo que significa en la computación clásica.

Para la universalidad, es necesario que un equipo cuántico solo *aproxima* cada matriz unitaria dentro de un error finito mediante una secuencia de puerta de longitud finita.

En otras palabras, un conjunto de puertas es un conjunto de puertas universal si cualquier transformación unitaria puede escribirse aproximadamente como un producto de puertas de este conjunto. Es necesario que para cualquier límite de error prescrito, existen puertas  $G, G_2, \dots, G_N$  del conjunto de {1} puertas de modo que

$$G_N G_{N-1} \cdots G_2 G_1 \approx U.$$

Dado que la convención para la multiplicación de matriz es multiplicar de derecha a izquierda la primera operación de puerta en esta secuencia,  $G_N$ , es realmente la última aplicada al vector de estado cuántico. Más formalmente, decimos que un conjunto de puertas es universal si para cada tolerancia de error  $\epsilon > 0$  existe  $G_1, \dots, G_N$  de forma que la distancia entre  $G_N \dots G_1$  y  $U$  sea como máximo  $\epsilon$ . Idealmente, el valor de  $N$  necesario para alcanzar esta distancia de  $\epsilon$  debería escalar polilogarítmicamente con  $1/\epsilon$ .

Por ejemplo, el conjunto formado por puertas Hadamard, CNOT y T es un conjunto universal, desde el que se puede generar cualquier cálculo cuántico (en cualquier número de cúbits). El hadamard y el conjunto de puertas T generan cualquier puerta de un solo cúbit:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad T =$$

$$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

En un equipo cuántico, las puertas cuánticas se pueden clasificar en dos categorías: puertas clifford y puertas que no son clifford, en este caso, la puerta T. Los programas cuánticos creados solo a partir de puertas de Clifford se pueden simular de forma eficaz mediante un equipo clásico y, por lo tanto, se requieren puertas que no sean Clifford para obtener la ventaja cuántica. En muchos esquemas de corrección de errores cuánticos (QEC), las llamadas puertas Clifford son fáciles de implementar, es decir, requieren muy pocos recursos en términos de operaciones y cúbits para implementar tolerancia a errores, mientras que las puertas que no son Clifford son bastante costosas al requerir tolerancia a errores. En un conjunto de puertas cuánticas universales, la puerta T se usa normalmente como puerta que no es Clifford.

El conjunto estándar de puertas clifford de un solo cúbit, incluido de forma predeterminada en Q#, incluye

$$H = \frac{1}{\sqrt{2}}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$, \quad S =$$

$$\begin{bmatrix} 1 & 0 \\ i & 1 \end{bmatrix}$$

$$= T^2, \text{ modulo } X =$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= HT^4H,$$

Misplaced &

Junto con la puerta que no es Clifford (la puerta T), estas operaciones se pueden componer para aproximar cualquier transformación unitaria en un solo cúbit.

## Generadores de T en el estimador de recursos de Azure Quantum

La preparación de la puerta T que no es Clifford es fundamental porque las otras puertas cuánticas no son suficientes para el cálculo cuántico universal. Para implementar operaciones que no sean Clifford para algoritmos a escala práctica, se requieren puertas T de baja tasa de errores (o estados T). Sin embargo, pueden ser difíciles de implementar directamente en cúbits lógicos y también pueden ser difíciles para algunos cúbits físicos.

En un equipo cuántico tolerante a errores, los estados T de baja tasa de errores necesarios se producen mediante una fábrica de destilación de estado T o una factoría de T para abreviar. Estas fábricas de T suelen implicar una secuencia de rondas de destilación, donde cada ronda toma muchos estados T ruidosos codificados en un código de distancia menor, los procesa mediante una unidad de destilación y genera menos estados T ruidosos codificados en un código de distancia mayor, con el número de rondas, unidades de destilación y distancias que todos los parámetros pueden variar. Este procedimiento se itera, donde los estados T de salida de una ronda se introducen en la siguiente ronda como entradas.

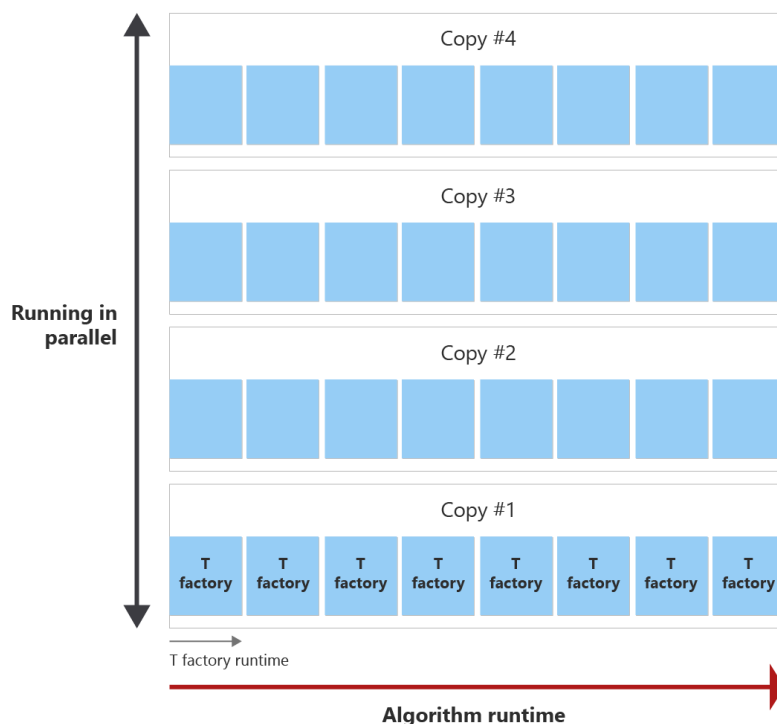
En función de la duración de la factoría de T, el [estimador](#) de recursos de Azure Quantum determina la frecuencia con la que se puede invocar una factoría de T antes de que supere el tiempo de ejecución total del algoritmo y, por tanto, cuántos estados T se pueden generar durante el tiempo de ejecución del algoritmo. Normalmente, se requieren más estados T que lo que se puede producir dentro de las invocaciones de un único generador de T durante el tiempo de ejecución del algoritmo. Para generar más estados de T, el estimador de recursos usa copias de las factorías de T.

### Estimación física de fábrica de T

El estimador de recursos calcula el número total de estados de T necesarios para ejecutar el algoritmo y el número de cúbits físicos para un único generador de T y su tiempo de ejecución.

El objetivo es generar todos los estados de T dentro del entorno de ejecución del algoritmo con lo menos copias de fábrica de T posibles. En el diagrama siguiente se muestra un ejemplo del tiempo de ejecución del algoritmo y el tiempo de ejecución de una factoría de T. Puede ver que el tiempo de ejecución de la factoría de T es más corto que el tiempo de ejecución del algoritmo. En este ejemplo, una fábrica de T puede destilar un estado T. Surgen dos preguntas:

- ¿Con qué frecuencia se puede invocar el generador de T antes del final del algoritmo?
- ¿Cuántas copias de la ronda de destilación de fábrica de T son necesarias para crear el número de estados T necesarios durante el tiempo de ejecución del algoritmo?



Antes del final del algoritmo, se puede invocar la factoría T ocho veces, que se denomina ronda de destilación. Por ejemplo, si necesita 30 estados T, se invoca una única factoría de T ocho veces durante el tiempo de ejecución del algoritmo y, por tanto, crea ocho estados T. A continuación, necesita cuatro copias de la ronda de destilación de fábrica T en paralelo para destilar los 30 estados T necesarios.

❗ Nota

Tenga en cuenta que las copias de fábrica de T y las invocaciones de fábrica de T no son las mismas.

Las fábricas de destilación de estado T se implementan en una secuencia de rondas, donde cada ronda consta de un conjunto de copias de unidades de destilación que se ejecutan en paralelo. El estimador de recursos calcula cuántos cúbits físicos se necesitan para ejecutar un generador de T y durante cuánto tiempo se ejecuta la factoría de T, entre otros parámetros necesarios.

Solo puede realizar invocaciones completas de un generador de T. Por lo tanto, puede haber situaciones en las que el tiempo de ejecución acumulado de todas las invocaciones de generador de T sea menor que el tiempo de ejecución del algoritmo. Dado que los cúbits se reutilizan por diferentes rondas, el número de cúbits físicos para un generador de T es el número máximo de cúbits físicos usados para una ronda. El tiempo de ejecución de la factoría de T es la suma del tiempo de ejecución en todas las rondas.

#### ⓘ Nota

Si la tasa de errores de puerta de T física es inferior a la tasa de errores de estado T lógico necesaria, el estimador de recursos no puede realizar una buena estimación de recursos. Al enviar un trabajo de estimación de recursos, es posible que encuentre que no se encuentra el generador de T porque la tasa de errores de estado T lógica necesaria es demasiado baja o demasiado alta.

Para obtener más información, vea apéndice C de evaluación de [los requisitos para escalar a ventaja cuántica](#) ↗ práctica.

## Contenido relacionado

- [Circuitos cuánticos](#)
- [Oráculos cuánticos](#)
- [Algoritmo de Grover](#)
- [Representación intermedia cuántica](#)

## Comentarios

¿Le ha resultado útil esta página?

👍 Sí

👎 No





# Solución de problemas en Azure Quantum

Al trabajar con el servicio Azure Quantum, es posible que experimente problemas relacionados con la conexión o el trabajo. Vea cómo puede resolver estos problemas.

## Conexión al área de trabajo

### Problema: No se puede autenticar en Azure Quantum mediante pytket-azure CI

Al intentar autenticarse en Azure Quantum a través del `pytket-azure` paquete en un entorno de CI mediante las variables de entorno "AZURE\_TENANT\_ID", "AZURE\_CLIENT\_ID" y "AZURE\_CLIENT\_SECRET", podría producirse el error:

Símbolo del sistema de Windows

Code: `InsufficientPermissions`

Message: `There are not enough permissions to perform this operation.`

Para resolver este problema, intente autenticarse usando una cadena de conexión y la variable de entorno "AZURE\_QUANTUM\_CONNECTION\_STRING" en su lugar. Para obtener más información, consulte [Conexión con una cadena de conexión](#).

Python

```
connection_string = "[Copy connection string]"
```

```
import os
```

```
os.environ["AZURE_QUANTUM_CONNECTION_STRING"] = connection_string
```

## Envío de trabajos

### Problema: Falta targets

Si el target en el que quieres ejecutar tu trabajo no aparece en la lista disponible de target, probablemente necesites actualizar a la versión más reciente del [Quantum Development Kit \(QDK\) para Visual Studio Code](#). Para obtener más información, consulte [Actualización del QDK](#).

## Problema: la operación ha devuelto un código de estado "No autorizado" no válido

Pasos para resolver este problema:

1. Abra Azure Portal (<https://portal.azure.com>) y autentique su cuenta.
2. En **Navegar**, seleccione **Suscripciones** y seleccione su suscripción.
3. Seleccione **Control de acceso (IAM)**.
4. En **Comprobar acceso**, busque su dirección de correo electrónico y seleccione la cuenta.
5. En la lista no deberían aparecer los roles **Propietario** o **Colaborador**.
6. Seleccione la pestaña **Asignaciones de roles**.

### ⚠ Nota

Si no ve la pestaña Asignaciones de roles, es posible que tenga que expandir el portal a pantalla completa o cerrar el panel de asignaciones de tu nombre.

7. Seleccione la **lista desplegable Rol**, seleccione **Propietario** o **Colaborador** y escriba la dirección de correo electrónico y seleccione su cuenta.
8. Seleccione **Guardar**.
9. Ahora debería aparecer su cuenta configurada con el rol **Propietario** o **Colaborador**.
10. Vuelva a crear el área de trabajo de Azure Quantum y, a continuación, envíe un trabajo para esta nueva área de trabajo.

## Problema: "AuthorizationFailure: esta solicitud no está autorizada para realizar esta operación"

Si se produce un error en el envío de un trabajo con este mensaje aunque tenga una conexión válida al servicio Azure Quantum, es posible que la cuenta de almacenamiento esté configurada para bloquear el acceso a la red pública. El servicio Azure Quantum solo admite cuentas de almacenamiento a través del acceso público a Internet.

Para comprobar la cuenta de almacenamiento:

- En la página área de trabajo cuántica de Azure Portal, seleccione **Información general** y seleccione la cuenta de almacenamiento.

- En la página de la cuenta de almacenamiento, en **Seguridad y redes**, seleccione **Redes**.
- En la **pestaña Firewalls y redes virtuales del acceso** a la red pública, asegúrese de que la opción **Habilitar todas las redes** está seleccionada.

## Problema: mensaje de error "No se ha podido compilar el programa" al intentar enviar un programa de Q# a través de la CLI

Al intentar enviar un trabajo desde el símbolo del sistema mediante el comando `az quantum submit`, es posible que encuentre el siguiente mensaje de error:

### Azure CLI

```
az quantum job submit ...
Failed to compile program.
Command ran in 21.181 seconds (init: 0.457, invoke: 20.724)
```

Este error se produce cuando hay un problema con el programa Q# que hace que se produzca un error en la compilación.

## Problema: Error del compilador "Número incorrecto de parámetros de puerta"

Al enviar un trabajo a Quantinuum desde un Jupyter Notebook local o un entorno de línea de comandos, y utilizando el traductor heredado de QASM (OPENQASM 2.0), es posible que encuentre este error:

### Símbolo del sistema de Windows

```
Job ID <jobId> failed or was cancelled with the message: 1000: Compile error:
[<file, line>] Wrong number of gate parameters
```

Este error se produce cuando se usa una coma "," u otro carácter que no sea de punto como separador decimal, como es habitual en muchos idiomas. Reemplace los separadores decimales que no son de período por puntos ".".

### Q#

```
// replace this line:
rx(1,5707963267948966) q[0];

// with this:
rx(1.5707963267948966) q[0];
```

## Problema: Error del compilador "no disponible para la configuración de compilación actual"

Al ejecutar una celda de código de Q# en un cuaderno de Jupyter Notebook en VS Code, es posible que encuentre el error:

Símbolo del sistema de Windows

```
<function name> not found. Found a matching item '<function name>' that is not available for the current compilation configuration
```

Este error indica que se establece el perfil qIR (representación intermedia cuántica) target en **Básico** y la función en cuestión requiere el perfil **Sin restricciones** target. Si no especifica un target tipo de perfil, el compilador establecerá automáticamente el target como **Sin restricciones** para usted.

## Problema: la operación ha devuelto un código de estado "Prohibido" no válido

Cuando envíes tu primer trabajo, es posible que recibas un código de error "denegado".

Este problema puede originarse durante la creación del área de trabajo: Azure Quantum no puede completar la asignación de roles que vincula el nuevo área de trabajo a la cuenta de almacenamiento especificada. Un escenario típico para esta situación se produce si la pestaña o la ventana del explorador web se cierra antes de que se complete la creación del área de trabajo.

Puede verificar que está encontrando este problema de asignación de roles siguiendo estos pasos:

- Vaya a la nueva área de trabajo cuántica en Azure Portal.
- En **Información general** > **Elementos esenciales** > **Cuenta de almacenamiento**, seleccione el vínculo de la cuenta de almacenamiento.
- En el panel de navegación izquierdo, seleccione **Control de acceso (IAM)**.
- Seleccione **Asignaciones de roles**.
- Compruebe que el área de trabajo aparezca como **Colaborador**.
- Si el área de trabajo no aparece como **Colaborador**, tiene una de las opciones siguientes:
  - Cree un área de trabajo y asegúrese de que se haya completado el proceso de creación antes de cerrar la pestaña o ventana del explorador web.
  - Añada la asignación de roles adecuada en la cuenta de almacenamiento
    - Control de acceso (IAM) > Agregar asignación de roles
    - Rol > Colaborador

- Asignar acceso a > usuario, grupo o entidad de servicio
- Seleccionar > [nombre del área de trabajo]
- Guardar

## Problema: El trabajo falla con el código de error: QIRPreProcessingFailed

Al enviar un trabajo a un proveedor de Rigetti, se produce un error en el trabajo y se notifica en la consola de administración de trabajos en Azure Portal:

### Resultados

```
Error code: QIRPreProcessingFailed
Error message: No match found for output recording set converter from
outputrecordingset.v2.labeled to outputrecordingset.v1.nonlabeled
```

Este error puede deberse a un conflicto de dependencia con una versión anterior de `pyqir` o `qiskit-qir`. Desinstale todas las versiones de `pyqir`, `pyqir-*` y `qiskit-qir` en el equipo local y, a continuación, instale o actualice el `azure-quantum` paquete de Python mediante el parámetro `[qiskit]` :

### Shell

```
pip install --upgrade azure-quantum[qiskit]
```

## Problema: Recuperación de información básica sobre trabajos con errores

Después de enviar un trabajo a un hardware target, el trabajo podría estar en la cola durante varias horas, o incluso uno o dos días, antes de que se produzca un error.

Para recuperar más información sobre el error:

- Para ver la salida o el mensaje de error devuelto, use el método `get_results()` con el objeto `job`:

### Python

```
job.get_results()
```

- En el área de trabajo de Azure Portal, seleccione > operaciones) y, a continuación, seleccione el nombre del trabajo para abrir un panel de detalles.

- En el área de trabajo de Azure Portal, seleccione **Operaciones > Proveedores**. Verificar la disponibilidad de la máquina target. Los trabajos enviados a targets con un estado **degradado** pueden permanecer en la cola durante más tiempo de lo habitual. A veces, los trabajos se procesan, pero a veces agotan el tiempo de espera y devuelven un error de *target no disponible*.

## Problema: Se me pide autenticarme al conectarme de forma programática a mi área de trabajo.

Si usa el SDK de Python de Azure Quantum, por ejemplo, en Jupyter Notebook y se conecta al área de trabajo mediante la clase `AzureQuantumProvider`, puede experimentar una ventana emergente para autenticarse en Azure cada vez que ejecute su script.

Este ventana emergente se produce porque el token de seguridad se restablece cada vez que usted ejecuta el script.

Para resolver este problema, ejecute `az login` mediante la CLI de Azure. Para obtener más información, consulte [az login](#).

## Problema: Después de actualizar el paquete azure-quantum, obtengo el error "ModuleNotFoundError: no hay ningún módulo denominado qiskit.tools" al supervisar un trabajo.

A partir de Qiskit 1.0, el módulo `qiskit.tools`, que es necesario para la función `job_monitor()`, está en desuso. Para supervisar los trabajos, use las funciones `wait_for_final_state()` o `result`.

### Python

```
job = MyTarget.run(circuit, shots=100)

to wait until the job is complete
job.wait_for_final_state()

to return the results of the job
result = job.result()
```

## Estimador de recursos de Azure Quantum

Los siguientes problemas podrían impedir que los trabajos de estimación de recursos se completen. Vea cómo resolver estos problemas.

## Problema: El algoritmo cuántico debe contener al menos un estado T o una medida

Para tener en cuenta la asignación de un programa cuántico arbitrario a una matriz 2D de cúbits lógicos, el estimador de recursos asume que *Cálculo Secuencial de Síntesis Paralela Pauli (PSSPC)* se realiza en el programa de entrada. En ese enfoque, todas las operaciones de Clifford se conmutan a través de todas las puertas T, las puertas de rotación y las operaciones de medición, dejando una sola operación Clifford que se puede evaluar clásicamente de forma eficaz. Por lo tanto, un programa cuántico que no contiene estados T, por ejemplo de puertas T o puertas de rotación, o operaciones de medición no requiere ningún recurso físico de computación cuántica. Para obtener más información sobre el cálculo secuencial de síntesis paralela de Pauli, vea [arXiv:2211.07629](https://arxiv.org/abs/2211.07629), [Apéndice D](#) [↗](#).

### Resultados

Error message: Algorithm requires at least one T state or measurement to estimate resources

## Problema: La tasa de errores de puerta T física es demasiado alta

La *tasa de errores de estado T lógico* depende del presupuesto de errores y del número de estados T del programa cuántico. [Los generadores de T](#) se usan para crear estados T con la tasa de errores de estado T lógico necesaria a partir de puertas T físicas, que tienen una tasa de errores de puerta T física. Normalmente, la tasa de errores de puerta T física es mayor que la tasa de errores de puerta T lógica necesaria. En algunos escenarios, la tasa de errores de puerta T física es significativamente mayor en comparación con la tasa de errores de estado T lógico necesaria, de modo que no se puede encontrar ningún generador de T que pueda producir estados T lógicos de calidad suficiente.

### Resultados

Error message: No T factory can be found, because the required logical T state error rate is too low

Esto es lo que podría hacer en este escenario:

- Aumente el presupuesto de errores, ya sea total o en la parte correspondiente a los estados T.
- Reducir la tasa de errores de puerta T física en los parámetros de los cúbits.
- Reduzca el número de estados T en el programa cuántico reduciendo las puertas T, las puertas de rotación y las puertas toffoli.

## Problema: La tasa de errores de puerta T física es demasiado baja

También hay el escenario opuesto, en el que la tasa de errores de puerta T física es inferior a la tasa de errores de estado T lógico necesaria. En tales casos, no se requiere ningún generador de T, ya que la tasa de errores de puerta T física ya es de calidad suficiente. Sin embargo, este escenario requiere una consideración cuidadosa del impacto de las unidades de transferencia que transfieren los estados T físicos desde la distancia del código 1 hasta la distancia de código del algoritmo (vea [arXiv:2211.07629](#), [Apéndice C](#)). En general, en presencia de fábricas tipo T, el costo de las unidades de transferencia es insignificante.

### Resultados

Error message: No T factory can be found, because the required logical T state error rate is too high; transfer units are necessary to perform a resource estimation accurately. One possibility to circumvent this problem is to increase the physical T gate error rate of the qubit parameters.

Esto es lo que podría hacer en este escenario:

- Aumente la tasa de errores de la puerta física T en los parámetros del cúbit a la tasa de errores del estado lógico T necesaria.
- Reduzca el margen de error o solo la parte de los estados T.

## Problema: La tasa de errores debe ser un número entre 0 y 1

Las tasas de error siempre deben ser valores entre 0 y 1. Además, para que la corrección de errores sea efectiva, la tasa de errores física para las puertas y las medidas debe estar por debajo de un valor que dependa de las propiedades del código de corrección de errores y de la tasa de errores lógica necesaria.

Esto es lo que podría hacer en este escenario:

- Aumente el presupuesto de errores, ya sea total o la parte para los errores lógicos.
- Reduzca las tasas de errores físicos en los parámetros de cúbit.

## Problema: Las restricciones del tiempo de ejecución máximo y el número máximo de cúbits físicos son mutuamente excluyentes

El estimador de recursos solo acepta una de las restricciones `maxDuration` o `maxPhysicalQubits` a la vez, pero no dos. Si proporciona restricciones `maxDuration` y `maxPhysicalQubits` para un



único trabajo, devuelve el `BothDurationAndPhysicalQubitsProvided` error.

## Problema: Ejecución del contenedor de recuentos de estimación de QIR: símbolo no definido `__quantum__rt__result_record_output`

Este error se produce al generar QIR para los circuitos de Qiskit a través del paquete de Python `qiskit_qir` sin establecer el parámetro `record_output` en `False`.

Para evitar este error, realice una de las siguientes acciones:

- Use el `azure_quantum` paquete de Python para enviar circuitos Qiskit a Azure Quantum (recomendado).
- Al usar el paquete de Python `qiskit_qir`, asegúrese de establecer el parámetro `record_output` en `False` antes de enviar su circuito.

## Creación de un área de trabajo de Azure Quantum

Es posible que se produzcan los siguientes problemas al usar Azure Portal para crear un área de trabajo.

### Problema: no se puede acceder al formulario de creación del área de trabajo en Azure Portal; Se le pide que se registre en una suscripción en su lugar.

Este problema se produce porque no tiene una suscripción activa.

Por ejemplo, es posible que se haya registrado en la [suscripción gratuita de evaluación gratuita de 30 días de Azure](#), que incluye créditos de Azure gratuitos de USD200 para usarlos en los servicios de Azure. Estos créditos de Azure no son aptos para usarse en proveedores de hardware cuántico. Después de 30 días de suscripción o una vez que consuma los 200 USD de créditos gratuitos de Azure (lo que ocurra primero), debe actualizar a una [suscripción de pago por uso](#) para seguir usando los servicios de Azure Quantum. Una vez que tenga una suscripción activa, Azure Portal le permite acceder al formulario de creación del área de trabajo.

Para ver una lista de las suscripciones y los roles asociados, consulte [Comprobación de las suscripciones](#).

### Problema: La opción Creación rápida no está disponible

Debe ser propietario de la suscripción que seleccione para poder usar la **opción Creación rápida**. Para ver una lista de las suscripciones y los roles asociados, consulte [Comprobación de las suscripciones](#). Si es colaborador de la suscripción, puede usar la **opción Creación avanzada** para crear un área de trabajo.

## **Problema: No se puede crear o seleccionar un grupo de recursos o una cuenta de almacenamiento**

Este problema se produce porque no tiene la autorización necesaria en el nivel de suscripción, grupo de recursos o cuenta de almacenamiento. Para obtener más información sobre los niveles de acceso necesarios, consulte [Requisitos de rol para crear un área de trabajo](#).

## **Problema: aparece el mensaje de error "Error de validación de implementación" después de seleccionar Crear.**

Este mensaje de error puede incluir más detalles, como "El cliente no tiene autorización para realizar la acción".

Este problema se produce porque no tiene la autorización necesaria en el nivel de suscripción, grupo de recursos o cuenta de almacenamiento. Para obtener más información sobre los niveles de acceso necesarios, consulte [Requisitos de rol para crear un área de trabajo](#).

Si se concedió el acceso recientemente, es posible que tenga que actualizar la página. A veces, las nuevas asignaciones de roles pueden tardar hasta una hora en surtir efecto sobre los permisos almacenados en caché a lo largo del sistema.

## **Problema: no ve un proveedor de hardware cuántico específico en la pestaña Proveedores**

Este problema se produce porque el proveedor no admite la región de facturación en la que está establecida la suscripción. Por ejemplo, si la suscripción está establecida en Israel, la pestaña Proveedores no enumera Rigetti como proveedor disponible. Para obtener una lista de proveedores y su disponibilidad por país o región, consulte [Disponibilidad global de proveedores](#) de Azure Quantum.

## **Problema: Fallo en la creación del área de trabajo o al añadir/eliminar proveedores con "ResourceDeploymentFailure" o "ProviderDeploymentFailure".**

Este problema puede incluir más detalles, como "ResourceDeploymentFailure - La operación de recurso 'AzureAsyncOperationWaiting' se completó con el estado de aprovisionamiento terminal 'Fallido'", o "ProviderDeploymentFailure - Falló al crear un plan para el proveedor: <Nombre del proveedor>".

Este error se produce porque el inquilino no habilitó las compras de Azure Marketplace. Siga los pasos descritos en [Habilitación de las compras](#) de Azure Marketplace para habilitar las compras de Azure Marketplace.

## Problema: la implementación de un área de trabajo cuántica o la implementación de una cuenta de almacenamiento produce uno de los errores siguientes.

- **Área de trabajo:** "La operación de escritura de recursos no se pudo completar correctamente, porque alcanzó el estado de aprovisionamiento de terminal 'Failed'".
- **Cuenta de almacenamiento:** "Error en la implementación de la plantilla debido a una infracción de directiva".

Este problema puede producirse si la directiva de seguridad de la suscripción bloquea la creación de cuentas de almacenamiento que tienen habilitado el acceso público. El servicio Azure Quantum solo admite cuentas de almacenamiento a través del acceso público a Internet.

Para resolverlo, trabaje con el administrador de la suscripción para obtener una excepción para la cuenta de almacenamiento que desea usar.

# Contribución al Kit de desarrollo de Azure Quantum

Artículo • 16/10/2024

Gracias por ser parte de la comunidad de Azure Quantum, nos complacen sus contribuciones.

- Si desea contribuir al código al Kit de desarrollo de Azure Quantum o notificar un error, consulte [Contribución a Q#](#).
- Si desea contribuir a la documentación del Kit de desarrollo de Azure Quantum, consulte [documentación sobre cómo contribuir](#).

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#) | [Obtener ayuda en Microsoft Q&A](#)

# Lecturas adicionales: Recursos de aprendizaje de computación cuántica

Artículo • 10/09/2024

Este artículo compila algunos de los recursos más populares que podría encontrar útil al aprender computación cuántica.

## Recursos de computación cuántica de Microsoft

Aprenda a desarrollar y aplicar soluciones de computación cuántica con los Quantum Development Kit servicios y Azure Quantum.

- [Ruta](#) de aprendizaje de Azure Quantum: una ruta de aprendizaje interactiva, gratuita y práctica. En estos módulos, obtendrá información sobre la computación cuántica y cómo desarrollar soluciones cuánticas mediante Q# y Azure Quantum Development Kit.
- [Quantum Katas](#) [↗](#): una colección de tutoriales de programación cuántica autodirigidos Q#.
- [Vídeos](#) [↗](#) de Azure Quantum: una lista de reproducción con vídeos de anuncios, demostraciones y discusiones de *Azure Quantum Quantum Quantum*.
- [Q# ejemplos de código](#) [↗](#): empiece a compilar su primera solución cuántica con esta colección de ejemplos de código listos para usar.
- [Q# blog](#) [↗](#): un blog escrito por desarrolladores para desarrolladores. Puede leer sobre el QDK y Q# la información más reciente, y conocer los desafíos cuánticos y los anuncios de hackathons.
- [Publicaciones de investigación](#) [↗](#): lea sobre el último avance en hardware cuántico y los algoritmos desarrollados por los investigadores de Microsoft.

Estos y más recursos de computación cuántica se pueden encontrar en la [página de aprendizaje cuántico de Microsoft](#) [↗](#).

## Q# contenido creado por la comunidad

La comunidad cuántica crea y desarrolla los siguientes recursos y se complace en la programación cuántica.

### Libros creados por la comunidad

- [Introducción a la computación cuántica con Q# y QDK](#), Filip Wojcieszyn, Springer, 2022.
- [Q# Pocket Guide](#), Mariia Mykhailova, O'Reilly, 2021. Q# Los ejemplos actualizados están [aquí](#).
- [Presentación de Microsoft Quantum Computing para desarrolladores](#), Johnny Hooyberghs, Springer, 2021.
- [Aprenda computación cuántica con Python y Q# : un enfoque](#) práctico, S. Kaiser y C. Granade. Manning Publications, 2021.

## Blogs creados por la comunidad

- [Awesome qsharp](#): lista de código abierto del código y los recursos de Q#.
- [Q# Comunidad](#): un espacio de GitHub para proyectos controlados por la comunidad.

## Foros y comunidades para desarrolladores cuánticos

[Quantum Computing StackExchange](#): comunidad en línea para que los desarrolladores cuánticos aprendan y compartan sus conocimientos.

## Cursos de computación cuántica

Consulte los siguientes cursos de aprendizaje de computación cuántica.

- [Computación cuántica con Microsoft QDK](#): una serie de liveProjects que le ayudarán a aprender el desarrollo de software cuántico mediante la creación de proyectos de un extremo a otro. Explorará todo el potencial de quantum para la criptografía, la transmisión de datos, la reconstrucción de datos, etc.
- [Computación cuántica: curso de Brilliant](#): aprenda a compilar algoritmos cuánticos desde cero con un equipo cuántico simulado en el explorador en este curso, creado en colaboración con investigadores cuánticos y profesionales de Microsoft, X e IQIM de Caltech.
- [Computación cuántica a través de cómics: clases de HackadayU](#): aprenda sobre conceptos de computación cuántica y programación de algoritmos a través de la discusión en el aula e cómics intuitivos.

## Bibliografía

La siguiente bibliografía es una colección de publicaciones que abarcan una amplia gama de temas de computación cuántica.

## Computación cuántica para principiantes

Si es entusiasta del mundo cuántico y quiere empezar a aprender la teoría detrás de la computación cuántica, las siguientes publicaciones le darán instrucción sobre temas como la física cuántica, la informática y el álgebra lineal.

- Nielsen, M. A. y Chuang, I. L. *Computación cuántica e información cuántica*. Reino Unido: Cambridge University Press, 2010.
- Kaye, P., Laflamme, R., & Mosca, M. *Introducción a la computación cuántica*. Oxford University Press, 2007.
- Rieffel, E. G., & Polak, W. H. *Computación cuántica: una introducción suave*. MIT Press, 2011.

## Diferentes tipos de cúbits

- Sergio Bravyi, Oliver Dial, Jay M. Gambetta, Dario Gil y Zaira Nazario. *El futuro de la computación cuántica con cúbits superconductores*, 2022.
- Microsoft Quantum. *InAs-Al Hybrid Devices Passing the Topological Gap Protocol*, arXiv:2207.02472 [cond-mat.mes-hall], (2022).
- M Saffman. *Computación cuántica con cúbits atómicos y interacciones de rydberg: progreso y desafíos*, Journal of Physics B: Atomic, Molecular y Optical Physics, 49(20):202001, (2016).
- J. I. Cirac y P. Zoller. *Cálculos cuánticos con iones atrapados en frío*, Phys. Rev. Lett., 74:4091–4094 (1995).

## Corrección de errores cuánticos

- Michael Beverland, Vadym Kliuchnikov y Eddie Schoute. *Compilación de código surface a través de rutas de acceso separadas por bordes*, PRX Quantum, 3:020342, (2022) .
- Adam Paetznick, Christina Knapp, Nicolas Delfosse, Bela Bauer, Jeongwan Haah, Matthew B. Hastings y Marcus P. da Silva. *Rendimiento de códigos de floquet planar con cúbits basados en majorana*, 2022.
- Austin G. Fowler, Mateo Mariantoni, John M. Martinis y Andrew N. Cleland. *Códigos de superficie: hacia el cálculo cuántico práctico a gran escala*, Phys. Rev. A, 86:032324, (2012).

- Daniel Gottesman. *Introducción a la corrección cuántica de errores y cálculo cuántico* tolerante a errores. En ciencia de la información cuántica y sus contribuciones a las matemáticas, Actas de Symposia en Matemáticas aplicadas, volumen 68, páginas 13–58, (2010).

## Estimación de recursos

- M. E. Beverland, P. Murali, M. Troyer, K. M. Svore, T. Hoefler, V. Kliuchnikov, G. H. Low, M. Soeken, A. Sundaram y A. Vashillo. *Evaluación de los requisitos para escalar a ventaja cuántica* práctica, arXiv:2211.07629v1, 2022.
- Isaac H. Kim, Ye-Hua Liu, Sam Pallister, William Pol, Sam Roberts y Eunseok Lee. Estimación de recursos tolerantes a errores para simulaciones químicas cuánticas: caso práctico sobre moléculas de electrolitos de batería de iones de li-ion. *Phys. Rev. Research*, 4:023019, abr 2022.
- Giulia Meuli, Mathias Soeken, Martin Roetteler y Thomas Häner. *Habilitación de compiladores cuánticos con reconocimiento de precisión mediante la estimación de recursos simbólicos*, Proc. Programa ACM. Lang., 4(OOPSLA), 2020.

## Computación cuántica tolerante a errores

- Héctor Bombin, Chris Dawson, Ryan V. Mishmash, Naomi Nickerson, Fernando Pastawski y Sam Roberts. *Bloques lógicos para el cálculo cuántico topológico* tolerante a errores, 2021.
- Antonio D. Córcoles, Abhinav Kandala, Ali Javadi-Abhari, Douglas T. McClure, Andrew W. Cross, Kristan Temme, Paul D. Nation, Matthias Steffen y Jay M. Gambetta. *Desafíos y oportunidades de sistemas de computación cuántica a corto plazo*, Procedimientos del IEEE, 108(8):1338–1352 (2020).
- Michael Edward Beverland. *Hacia equipos cuánticos realizables*, tesis de doctorado, Instituto de Tecnología de California, 2016.
- Peter W Shor. *Cálculo cuántico tolerante a errores*. En actas de la 37ª conferencia sobre fundamentos de la informática, páginas 56–65. IEEE (1996).

## Química cuántica

- J. Tilly, Hongxiang Chen, Shuxiang Cao, D. Picozzi, K. Setia, Ying Li, E. Grant, L. Wossnig, I. Rungger, G. Booth, J. Tennyson. *The Variational Quantum Eigensolver: una revisión de métodos y procedimientos* recomendados, arXiv:2111.05176v3 [quant-ph], 2022.
- V. von Burg, Guang Hao Low, T. Häner, D.S. Steiger, M. Reiher, M. Roetteler y M. Troyer. *Computación cuántica mejorada catalysis* computacional. *Phys. Rev.*



Research 3, 033055 (2021).

- Bela Bauer, Sergio Bravyi, Mario Motta y Garnet Kin-Lic Chan. *Algoritmos cuánticos para la química cuántica y la ciencia de materiales cuánticos*, Revisiones químicas, 120(22):12685–12717 (2020).

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)

# Notas de la Quantum Development Kit versión de (QDK) y Azure Quantum

Artículo • 16/10/2024

En las notas de la versión se describen las actualizaciones de [Quantum Development Kit \(QDK\)](#) y el [servicio](#) Azure Quantum.

Para obtener instrucciones sobre cómo "empezar a trabajar", consulte [Configuración de un entorno de desarrollo local para Azure Quantum](#). Para obtener instrucciones sobre cómo actualizar el QDK a la versión más reciente, consulte [Actualización de \( Quantum Development Kit QDK\) a la versión](#) más reciente.

- Para obtener las notas más recientes de la versión de QDK y Azure Quantum, consulte el [repositorio](#)  qsharp de GitHub.
- Para obtener las notas más recientes de la versión del paquete de Python de Azure-Quantum, consulte el [repositorio](#)  azure-quantum-python de GitHub.

---

## Comentarios

¿Le ha resultado útil esta página?



Sí



No

[Proporcionar comentarios sobre el producto](#)  | [Obtener ayuda en Microsoft Q&A](#)