aws

Guía para desarrolladores

Amazon Braket

Amazon Braket: Guía para desarrolladores

Las marcas comerciales y la imagen comercial de Amazon no se pueden utilizar en relación con ningún producto o servicio que no sea de Amazon, de ninguna manera que pueda causar confusión entre los clientes y que menosprecie o desacredite a Amazon. Todas las demás marcas registradas que no son propiedad de Amazon son propiedad de sus respectivos propietarios, que pueden o no estar afiliados, conectados o patrocinados por Amazon.

Table of Contents

¿Qué es Amazon Braket?

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con AWS! Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Amazon Braket es una plataforma totalmente gestionada Servicio de AWS que ayuda a los investigadores, científicos y desarrolladores a empezar con la computación cuántica. La computación cuántica tiene el potencial de resolver problemas computacionales que están fuera del alcance de las computadoras clásicas, ya que emplea las leyes de la mecánica cuántica para procesar la información de nuevas formas.

Acceder al hardware de computación cuántica puede resultar caro e inconveniente. El acceso limitado dificulta la ejecución de algoritmos, la optimización de diseños, la evaluación del estado actual de la tecnología y la planificación de cuándo invertir sus recursos para obtener el máximo beneficio. Braket le ayuda a superar estas dificultades.

Braket ofrece un único punto de acceso a una variedad de tecnologías de computación cuántica. Con Braket, puede:

- Explorar y diseñar algoritmos cuánticos e híbridos.
- Probar algoritmos en diferentes simuladores de circuitos cuánticos.
- Ejecutar algoritmos en diferentes tipos de computadoras cuánticas.
- Crear aplicaciones de prueba de concepto.

Definir problemas cuánticos y programar computadoras cuánticas para resolverlos requiere un nuevo conjunto de habilidades. Para ayudarle a adquirir estas habilidades, Braket ofrece diferentes entornos para simular y ejecutar sus algoritmos cuánticos. Puede encontrar el enfoque que mejor se adapte a sus necesidades y empezar rápidamente con un conjunto de entornos de ejemplo denominados cuadernos.

El desarrollo de Braket consta de tres etapas:

- Desarrollo: Braket proporciona entornos de cuaderno de Jupyter totalmente administrados que facilitan la puesta en marcha. Los cuadernos de Braket vienen preinstalados con ejemplos de algoritmos, recursos y herramientas para desarrolladores, incluido el SDK de Amazon Braket. Con el SDK de Amazon Braket, puede desarrollar algoritmos cuánticos y, después, probarlos y ejecutarlos en diferentes simuladores y computadoras cuánticas cambiando una sola línea de código.
- Prueba: Braket proporciona acceso a simuladores de circuitos cuánticos de alto rendimiento y totalmente administrados. Puede probar y validar sus circuitos. Braket administra todos los componentes de software subyacentes y los clústeres de Amazon Elastic Compute Cloud (Amazon EC2) para eliminar la carga de simular circuitos cuánticos en la infraestructura clásica de computación de alto rendimiento (HPC).
- Ejecución: Braket proporciona un acceso seguro y bajo demanda a diferentes tipos de computadoras cuánticas. Tiene acceso a ordenadores cuánticos basados en puertas desdeAQT,, y IonQ IQMRigetti, así como a un simulador hamiltoniano analógico desde. QuEra Por otro lado, no tiene ningún compromiso inicial ni necesita obtener acceso a través de proveedores individuales.

Acerca de la computación cuántica y Braket

La computación cuántica se encuentra en su fase inicial de desarrollo. Es importante entender que en la actualidad no existe ninguna computadora cuántica universal y con tolerancia a errores. Por lo tanto, ciertos tipos de hardware cuántico se adaptan mejor a cada caso de uso y es crucial tener acceso a una variedad de hardware de computación. Braket ofrece una variedad de hardware a través de proveedores externos.

El hardware cuántico existente está limitado debido al ruido, que ocasiona errores. La industria se encuentra en la era de la cuántica de escala intermedia ruidosa (NISQ). En la era de la NISQ, los dispositivos de computación cuántica son demasiado ruidosos para soportar algoritmos cuánticos puros, como el algoritmo de Shor o el algoritmo de Grover. Hasta que se disponga de una mejor corrección de los errores cuánticos, la computación cuántica más práctica requiere la combinación de recursos de computación clásicos (tradicionales) con computadoras cuánticas para crear algoritmos híbridos. Braket le ayuda a trabajar con algoritmos cuánticos híbridos.

En los algoritmos cuánticos híbridos, las unidades de procesamiento cuántico (QPUs) se utilizan como coprocesadores CPUs, lo que acelera los cálculos específicos de un algoritmo clásico. Estos algoritmos utilizan el procesamiento iterativo, en el que la computación se mueve entre computadoras clásicas y cuánticas. Por ejemplo, las aplicaciones actuales de la computación cuántica en química, optimización y machine learning se basan en algoritmos

cuánticos variacionales, que son un tipo de algoritmo cuántico híbrido. En los algoritmos cuánticos variacionales, las rutinas de optimización clásicas ajustan los parámetros de un circuito cuántico parametrizado de forma iterativa, del mismo modo que los pesos de una red neuronal se ajustan de forma iterativa en función del error de un conjunto de entrenamiento de machine learning. Braket ofrece acceso a la biblioteca de software de código PennyLane abierto, que le ayuda con los algoritmos cuánticos variacionales.

La computación cuántica está ganando terreno en el campo de la computación en cuatro áreas principales:

- Teoría de números: incluye la factorización y la criptografía (por ejemplo, el algoritmo de Shor es el principal método cuántico para los cómputos de la teoría de números).
- Optimización: incluye la satisfacción de las restricciones, la resolución de sistemas lineales y el machine learning.
- Computación oracular: incluye búsqueda, subgrupos ocultos y localización de órdenes (por ejemplo, el algoritmo de Grover es un método cuántico primario para cómputos oraculares).
- Simulación: incluye aplicaciones de simulación directa, invariantes de nudo y algoritmos de optimización cuántica aproximada (QAOA).

Las aplicaciones de estas categorías de cómputos se encuentran en los servicios financieros, la biotecnología, la industria manufacturera y los productos farmacéuticos, entre otros. Braket ofrece capacidades y ejemplos de cuadernos que ya se pueden aplicar a muchos problemas de prueba de concepto, además de a algunos problemas prácticos.

En esta sección:

Funcionamiento de Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

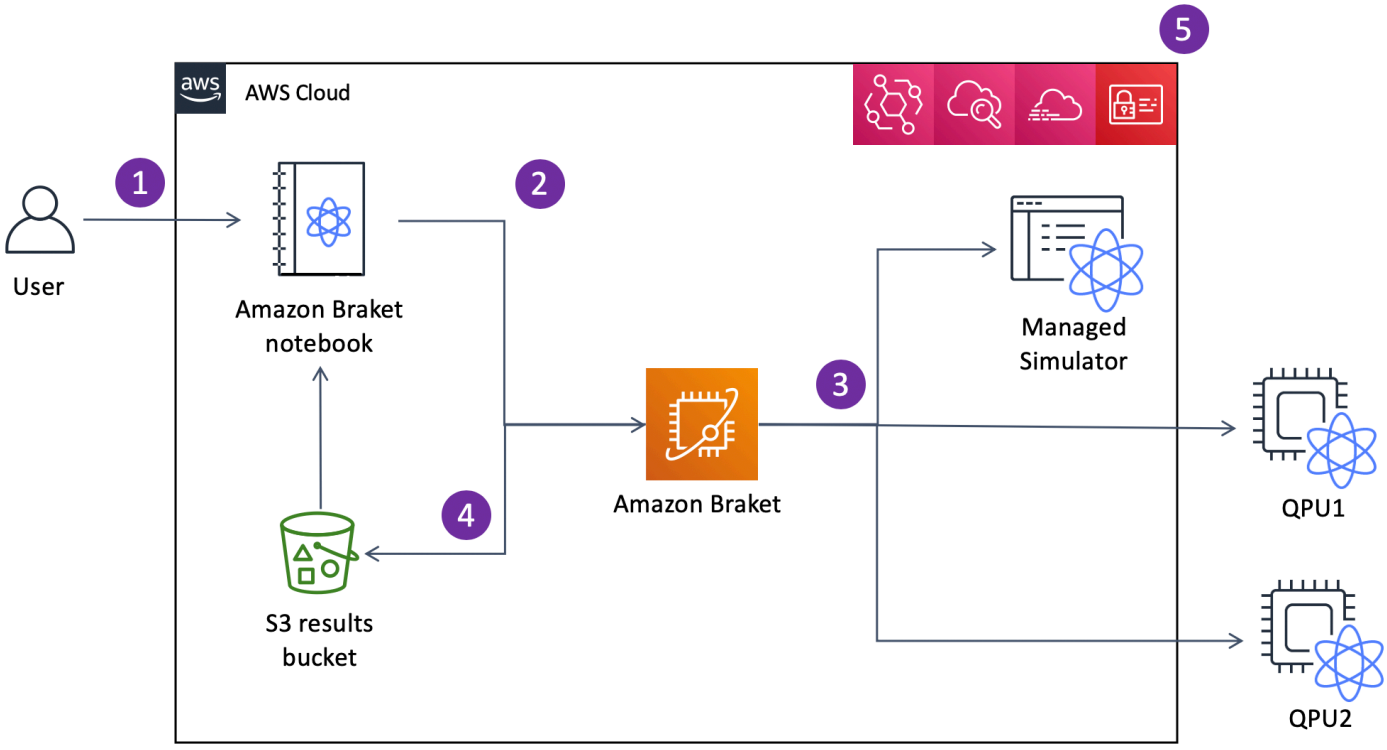
Amazon Braket proporciona acceso bajo demanda a dispositivos de computación cuántica, incluidos simuladores de circuitos bajo demanda y diferentes tipos de unidades de procesamiento cuántico (). QPUs En Amazon Braket, la solicitud atómica a un dispositivo es una tarea cuántica. En el caso de los dispositivos basados en puertas, esta solicitud incluye el circuito cuántico (incluidas las instrucciones de medición y el número de disparos) y otros metadatos de la solicitud. En el caso de los simuladores hamiltonianos analógicos, la tarea cuántica incluye la disposición física del registro cuántico y la dependencia temporal y espacial de los campos de manipulación.

Braket Direct es un programa que amplía la forma de explorar la computación cuántica y acelera la AWS investigación y la innovación. Puede reservar capacidad específica en varios dispositivos cuánticos, contactar directamente con especialistas en computación cuántica y tener acceso anticipado a capacidades de próxima generación, incluido el último dispositivo de iones atrapados de IonQ, Forte.

En esta sección, ofrecemos conocimientos sobre el flujo de alto nivel de ejecución de tareas cuánticas en Amazon Braket.

En esta sección:

Flujo de tareas cuánticas de Amazon Braket



Con las Jupyter libretas, puede definir, enviar y supervisar sus tareas cuánticas desde la consola Amazon Braket o mediante el Amazon Braket SDK. Puede desarrollar sus circuitos cuánticos directamente en el SDK. Sin embargo, en el caso de los simuladores hamiltonianos analógicos, usted define el diseño del registro y los campos de control (1). Una vez definida la tarea cuántica, puede elegir un dispositivo en el que ejecutarla y enviarla a la API de Amazon Braket (2). Según el dispositivo que elija, la tarea cuántica se pone en cola hasta que el dispositivo esté disponible y se envía a la QPU o al simulador para su implementación (3). Amazon Braket le brinda acceso a una variedad de dispositivos cuánticos compatibles QPUs, incluidos simuladores bajo demanda, simuladores locales y un simulador integrado.

Tras procesar la tarea cuántica, Amazon Braket devuelve los resultados a un bucket de Amazon S3, donde los datos se almacenan en su Cuenta de AWS (4). Al mismo tiempo, el SDK consulta los resultados en segundo plano y los carga en el cuaderno de Jupyter al completarse la tarea cuántica. También puede ver y gestionar sus tareas cuánticas en la página Quantum Tasks de la consola Amazon Braket o mediante el `GetQuantumTask` funcionamiento de Amazon Braket. API

Amazon Braket está integrado con AWS Identity and Access Management (IAM), Amazon CloudWatch y AWS CloudTrail Amazon EventBridge para la gestión, el monitoreo y el registro del acceso de los usuarios, así como para el procesamiento basado en eventos (5).

Procesamiento de datos de terceros

Las tareas cuánticas que se envían a un dispositivo QPU se procesan en computadoras cuánticas ubicadas en instalaciones operadas por proveedores externos. Para obtener más información sobre la seguridad y el procesamiento por parte de terceros en Amazon Braket, consulte Seguridad de los proveedores de hardware de Amazon Braket.

Términos y conceptos de Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

En Braket se usan los siguientes conceptos y términos:

Simulación hamiltoniana analógica

La simulación hamiltoniana analógica (AHS) es un paradigma de computación cuántica distinto para la simulación directa de la dinámica cuántica dependiente del tiempo de sistemas de muchos cuerpos. En la AHS, los usuarios especifican directamente un hamiltoniano dependiente del tiempo y la computadora cuántica se ajusta de tal manera que emula directamente la evolución continua del tiempo bajo este hamiltoniano. Los dispositivos de AHS suelen ser dispositivos de uso especial y no computadoras cuánticas universales como los dispositivos basados en puerta. Están limitados a una clase de hamiltonianos que puedan simular. Sin embargo, dado que estos hamiltonianos se implementan de forma natural en el dispositivo, la AHS no sufre la sobrecarga que supone formular algoritmos como circuitos e implementar operaciones de puerta.

Braket

Hemos denominado al servicio Braket por la notación bra-ket, una notación estándar en mecánica cuántica. Fue introducida por Paul Dirac en 1939 para describir el estado de los sistemas cuánticos y también se conoce como notación de Dirac.

Braket Direct

Con Braket Direct, puede reservar acceso exclusivo a los diferentes dispositivos cuánticos de su elección, ponerse en contacto con especialistas en computación cuántica para recibir orientación sobre su carga de trabajo y obtener acceso anticipado a las capacidades de próxima generación, como los nuevos dispositivos cuánticos con disponibilidad limitada.

Trabajo híbrido de Braket

Amazon Braket tiene una característica denominada Trabajos híbridos de Amazon Braket que proporciona ejecuciones totalmente administradas de algoritmos híbridos. Un trabajo híbrido de Braket consta de tres componentes:

1. La definición de su algoritmo, que se puede proporcionar como un script, un módulo de Python o un contenedor de Docker.
2. La instancia de trabajo híbrido, basada en Amazon EC2, en la que se ejecutará su algoritmo. El valor predeterminado es una instancia de ml.m5.xlarge.
3. El dispositivo cuántico en el que ejecutar las tareas cuánticas que forman parte de su algoritmo. Un único trabajo híbrido normalmente contiene un conjunto de muchas tareas cuánticas.

¿Dispositivo

En Amazon Braket, un dispositivo es un backend que puede ejecutar tareas cuánticas. Un dispositivo puede ser una QPU o un simulador de circuitos cuánticos. Para obtener más información, consulte Dispositivos compatibles con Amazon Braket.

Mitigación de errores

La mitigación de errores implica ejecutar varios circuitos físicos y combinar sus mediciones para obtener un resultado mejorado. Para obtener más información, consulte Técnicas de mitigación de errores.

Computación cuántica basada en puertas

En la computación cuántica (QC) basada en puertas, también denominada QC basada en circuitos, los cómputos se dividen en operaciones elementales (puertas). Ciertos conjuntos de puertas son universales, lo que significa que cada cómputo se puede expresar como una secuencia finita de esas puertas. Las puertas son los componentes básicos de los circuitos cuánticos y son análogas a las puertas lógicas de los circuitos digitales clásicos.

Límite de gateshot

El límite de gateshot se refiere al número total de entradas por shot (la suma de todos los tipos de puertas) y al recuento de shots por tarea. Matemáticamente, el límite de gateshot se puede expresar de la siguiente manera:

```
Gateshot limit = (Gate count per shot) * (Shot count per task)
```

Hamiltoniano

La dinámica cuántica de un sistema físico viene determinada por su hamiltoniano, que codifica toda la información sobre las interacciones entre los componentes del sistema y los efectos de las fuerzas impulsoras exógenas. El hamiltoniano de un sistema de N qubits se representa habitualmente como una matriz de 2^N por 2^N de números complejos en máquinas clásicas. Al ejecutar una simulación hamiltoniana analógica en un dispositivo cuántico, puede evitar estos requisitos exponenciales de recursos.

Pulso

Un pulso es una señal física transitoria que se transmite a los qubits. Se describe mediante una forma de onda reproducida en un marco que sirve de soporte para la señal portadora y está vinculada al canal o puerto de hardware. Los clientes pueden diseñar sus propios pulsos proporcionando el sobre análogo que modula la señal portadora sinusoidal de alta frecuencia. El marco se describe de forma única mediante una frecuencia y una fase que a menudo se eligen para que estén en resonancia con la separación de energía entre los niveles de energía para |0# y |1# del qubit. De este modo, las puertas se activan como pulsos con una forma predeterminada y parámetros calibrados, como su amplitud, frecuencia y duración. Los casos de uso que no estén cubiertos por formas de onda de plantilla se habilitarán mediante formas de onda personalizadas que se especificarán en la resolución de muestra única proporcionando una lista de valores separados por un tiempo de ciclo físico fijo.

Circuito cuántico

Un circuito cuántico es el conjunto de instrucciones que define un cómputo en una computadora cuántica basada en puertas. Un circuito cuántico es una secuencia de puertas cuánticas, que son transformaciones reversibles en un registro de qubit, junto con instrucciones de medición.

Simulador de circuitos cuánticos

Un simulador de circuitos cuánticos es un programa de computación que se ejecuta en computadoras cuánticas y calcula los resultados de medición de un circuito cuántico. En el caso de los circuitos generales, los requisitos de recursos de una simulación cuántica aumentan

exponencialmente con el número de qubits a simular. Braket proporciona acceso a simuladores de circuitos cuánticos administrados (a los que se accede a través de la API de Braket) y locales (parte del SDK de Amazon Braket).

Computadora cuántica

Una computadora cuántica es un dispositivo físico que utiliza fenómenos de la mecánica cuántica, como la superposición y el entrelazamiento, para realizar cómputos. Existen diferentes paradigmas en la computación cuántica (QC), como la QC basada en puertas.

Unidad de procesamiento cuántico (QPU)

Una QPU es un dispositivo físico de computación cuántica que puede ejecutarse en una tarea cuántica. QPUs puede basarse en diferentes paradigmas de control de calidad, como el control de calidad basado en puertas. Para obtener más información, consulte Dispositivos compatibles con Amazon Braket.

Puertas nativas de QPU

Las puertas nativas de QPU pueden asignarse directamente a pulsos de control mediante el sistema de control de la QPU. Las puertas nativas se pueden ejecutar en el dispositivo QPU sin necesidad de realizar más compilaciones. Subconjunto de puertas compatibles con QPU. Puede encontrar las puertas nativas de un dispositivo en la página Dispositivos de la consola de Amazon Braket y a través del SDK de Braket.

Puertas compatibles con QPU

Las puertas compatibles con QPU son las puertas aceptadas por el dispositivo QPU. Es posible que estas puertas no se ejecuten directamente en la QPU, lo que significa que podrían tener que descomponerse en puertas nativas. Puede encontrar las puertas compatibles de un dispositivo en la página Dispositivos de la consola de Amazon Braket y a través del SDK de Amazon Braket.

Tarea cuántica

En Braket, una tarea cuántica es la solicitud atómica a un dispositivo. En el caso de los dispositivos de QC basada en puertas, esta incluye el circuito cuántico (incluidas las instrucciones de medición y el número de shots) y otros metadatos de la solicitud. Puede crear tareas cuánticas a través del SDK de Amazon Braket o utilizando la operación CreateQuantumTask de la API directamente. Después de crear una tarea cuántica, se pondrá en cola hasta que el dispositivo solicitado esté disponible. Puede ver sus tareas cuánticas en la página Tareas cuánticas de la consola de Amazon Braket o mediante la GetQuantumTask o las operaciones SearchQuantumTasks de la API.

Qubit

La unidad básica de información de una computadora cuántica se denomina qubit (bit cuántico) y es muy parecido a un bit en la computación clásica. Un qubit es un sistema cuántico de dos niveles que se puede realizar mediante diferentes implementaciones físicas, como circuitos superconductores o iones y átomos individuales. Otros tipos de qubit se basan en fotones, espines electrónicos o nucleares o sistemas cuánticos más exóticos.

Queue depth

La Queue depth se refiere a la cantidad de tareas cuánticas y trabajos híbridos en cola para un dispositivo en particular. Se puede acceder a las tareas cuánticas y al recuento de la cola de trabajos híbridos de un dispositivo a través del Braket Software Development Kit (SDK) o de la Amazon Braket Management Console.

1. La profundidad de la cola de tareas se refiere al número total de tareas cuánticas que están esperando para ejecutarse con una prioridad normal.
2. La profundidad de la cola de tareas prioritarias se refiere al número total de tareas cuánticas enviadas que están esperando para ejecutarse a través de Amazon Braket Hybrid Jobs. Estas tareas tienen prioridad sobre las tareas independientes una vez que se inicia un trabajo híbrido.
3. La profundidad de la cola de trabajos híbridos se refiere al número total de trabajos híbridos que están actualmente en cola en un dispositivo. Las Quantum tasks enviadas como parte de un trabajo híbrido tienen prioridad y se añaden en la Priority Task Queue.

Queue position

Queue position se refiere a la posición actual de su tarea cuántica o trabajo híbrido en la cola del dispositivo correspondiente. Esta se puede obtener para tareas cuánticas o trabajos híbridos a través del Braket Software Development Kit (SDK) o de la Amazon Braket Management Console.

Shots

Dado que la computación cuántica es intrínsecamente probabilística, cualquier circuito debe evaluarse varias veces para obtener un resultado preciso. La ejecución y medición de un solo circuito se denomina shot. El número de shots (ejecuciones repetidas) de un circuito se elige en función de la precisión deseada para el resultado.

AWS terminología y consejos para Amazon Braket

Políticas de IAM

Una política de IAM es un documento que permite o deniega permisos para recursos y Servicios de AWS . Las políticas de IAM permiten personalizar los niveles de acceso de los usuarios a los recursos. Por ejemplo, puede permitir que los usuarios accedan a todos los buckets de Amazon S3 incluidos en el Cuenta de AWS suyo o solo a uno específico.

- Práctica recomendada: siga el principio de seguridad de privilegios mínimos al dar permisos. Esto ayudará a evitar que los usuarios o los roles tengan más permisos de los necesarios para realizar sus tareas cuánticas. Por ejemplo, si un empleado solo necesita acceder a un bucket concreto, especifique el bucket en la política de IAM en lugar de conceder al empleado acceso a todos los buckets de su Cuenta de AWS.

Funciones de IAM

Un rol de IAM es una identidad que puede adoptar para obtener acceso temporal a permisos. Antes de que un usuario, una aplicación o un servicio puedan adoptar un rol de IAM, se les debe proporcionar permisos para cambiar a dicho rol. Cuando alguien adopta un rol de IAM, abandona todos los permisos anteriores que tenía en el rol anterior y adopta los permisos del nuevo rol.

- Práctica recomendada: los roles de IAM son ideales para situaciones en las que el acceso a los servicios o recursos debe concederse de forma temporal, en lugar de a largo plazo.

Bucket de Amazon S3

Amazon Simple Storage Service (Amazon S3) es Servicio de AWS un servicio que permite almacenar datos como objetos en cubos. Los buckets de Amazon S3 ofrecen espacio de almacenamiento ilimitado. El tamaño máximo de objeto en un bucket de Amazon S3 es de 5 TB. Puede cargar cualquier tipo de archivo de datos en un bucket de Amazon S3, como imágenes, vídeos, archivos de texto, archivos de copia de seguridad, archivos multimedia para un sitio web, documentos archivados y los resultados de sus tareas cuánticas de Braket.

- Práctica recomendada: puede establecer permisos para controlar el acceso a su bucket de S3. Para obtener más información, consulte las políticas de bucket en la documentación de Amazon S3.

Seguimiento y ahorro de costos

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Con Amazon Braket, tiene acceso a los recursos de computación cuántica bajo demanda sin compromiso previo. Solo paga por lo que utiliza. Para obtener más información acerca de los precios, visite la página de precios.

En esta sección:

- Establecer límites de gasto para Amazon Braket QPUs
- Seguimiento de costos casi en tiempo real
- Prácticas recomendadas para el ahorro costos

Establecer límites de gasto para Amazon Braket QPUs

Los límites de gasto de Amazon Braket proporcionan controles de costes opcionales por dispositivo para las unidades de procesamiento cuántico (). QPUs

Cómo funcionan los límites de gasto: Amazon Braket realiza un seguimiento de tus gastos acumulados y valida todas las solicitudes de creación de tareas con respecto al límite configurado. Si el coste estimado de una tarea supera el límite de gastos restante, Amazon Braket rechaza la tarea inmediatamente con un error de validación. Si lo desea, puede configurar un período de tiempo para su límite de gasto. Al configurar un período de tiempo, puedes asegurarte de que las tareas solo se puedan enviar en ese período especificado. Se rechazarán las tareas enviadas fuera del período de tiempo.

Diseño opcional: los flujos de trabajo existentes no se verán afectados a menos que habilites los controles de forma explícita. Puedes eliminar todas las restricciones suprimiendo el límite de gasto.

> **Note**
>
> Los límites de gasto se aplican solo a las tareas de QPU híbridas y bajo demanda. Excluyen los simuladores, los cuadernos gestionados, los costes de las instancias EC2 de Hybrid Job

y las reservas de Braket Direct. Para una gestión integral de los costes de todos los servicios de AWS, siga utilizándolos AWS Budgets.

Lista de medidas para limitar el gasto

Busca

Con el siguiente comando de la CLI de AWS, puede buscar y enumerar los límites de gasto en una región de AWS específica y para un dispositivo Braket específico.

```
aws --region {device_region} braket search-spending-limits --filters
 name=deviceArn,operator=EQUAL,values={device_arn}
```

Crear

Con el siguiente comando de la AWS CLI, puede crear un nuevo límite de gasto para un dispositivo cuántico específico en una región específica. La solicitud se rechaza si ya existe un límite de gasto para el dispositivo.

```
aws --region {device_region} braket create-spending-limit --device-arn {device_arn}
 --spending-limit {max_spend}
```

Actualización

Con el siguiente comando de la AWS CLI, puede actualizar un límite de gasto existente a un nuevo valor máximo de gasto. La solicitud se rechaza si la suma del gasto actual y el gasto en cola ya es superior al nuevo gasto máximo solicitado.

```
aws --region {device_region} braket update-spending-limit --spending-limit-arn
 {spending_limit_arn} --spending-limit {new_max_spend}
```

Puedes proporcionar un período de tiempo en lugar del nuevo gasto máximo o además de él, como en el ejemplo anterior.

Eliminar

Con el siguiente comando de la AWS CLI, puede eliminar un límite de gasto existente.

```
aws --region {device_region} braket delete-spending-limit --spending-limit-arn
 {spending_limit_arn}
```

Puede proporcionar un período de tiempo en lugar del nuevo gasto máximo o además de él, como en el ejemplo anterior.

Si bien es opcional, especifique siempre el parámetro de región como práctica recomendada. Los comandos ejecutados en una región diferente a la del dispositivo fallarán o, en su caso`SearchSpendingLimits`, devolverán resultados incorrectos.

Para ver más ejemplos sobre cómo usar los límites de gasto, consulta el cuaderno de ejemplo.

Cómo funciona la validación de tareas

Cuando la cuenta de AWS envía una `CreateQuantumTask` solicitud que de otro modo sería válida, está sujeta al siguiente comportamiento de bloqueo. Nota: El presupuesto restante es la diferencia entre el límite de gasto y la suma del gasto pendiente y el gasto actual. (Consulta la siguiente sección)

- Caso 1: No hay límite de gasto para el dispositivo de tareas: se crea la tarea.
- Caso 2: Hay un límite de gasto para el dispositivo de destino y la hora actual se encuentra dentro del período de tiempo del límite de gasto:
  - Si el coste estimado de la tarea es inferior o igual al presupuesto restante: si se CreateQuantumTask realiza correctamente, se crea la tarea.
  - Si el costo estimado es superior al presupuesto restante: se produce un `CreateQuantumTask` error y no se crea ninguna tarea.
- Caso 3: hay un límite de gasto para el dispositivo de destino y la hora actual está fuera del período de tiempo del límite de gasto: `CreateQuantumTask` se produce un error y no se crea ninguna tarea.

¿Cómo se calcula el presupuesto restante

El presupuesto restante es la diferencia entre el límite de gasto y la suma del gasto actual y el gasto en cola.

Cuando se crea una tarea para un dispositivo con un límite de gasto, el gasto en cola se incrementa en función del coste estimado de la tarea. Este evento aparece en la primera fila de la tabla siguiente. En la siguiente tabla se muestra lo que ocurre con el gasto en cola y el gasto actual, en función del progreso de la tarea.

Estado antiguo de una tarea cuántica	Nuevo estado de tarea cuántica	Cambiar a gasto en cola	Cambiar al gasto actual
-	CREATED	Incrementado según el costo estimado	Sin cambios
CREATED	QUEUED	Sin cambios	Sin cambios
Cualquiera	RUNNING	Sin cambios	Sin cambios
Cualquiera	CANCELADO	Sin cambios	Sin cambios
CANCELADO	CANCELLED	Reducido según el costo estimado	Sin cambios
Cualquiera	ERROR	Reducido según el costo estimado	Sin cambios
RUNNING	COMPLETED	Reducido según el costo estimado	Incrementado según el costo estimado (ajustado en consecuencia para las tareas parcialmente completadas)

Casos perimetrales

P: Al crear un límite de gastos, ¿se tienen en cuenta las tareas que ya están en cola para el gasto en cola?

R: No. Las tareas que ya están creadas, en cola o en curso no se tienen en cuenta para el gasto en cola de un límite de gastos recién creado.

P: Si se reduce el límite de gasto mediante su actualización, ¿se termina anticipadamente una tarea cuántica creada, en cola o en curso?

R: No.

P: ¿Llegar a la hora de finalización del límite de gasto provoca la finalización anticipada de una tarea cuántica creada, en cola o en curso?

R: No. Las tareas creadas, en cola o en curso se pueden completar independientemente del estado del límite de gasto.

P: ¿En qué se diferencia la falta de límite de gastos de un límite de gasto de cero dólares?

R: No tener límite de gasto permite crear tareas cuánticas sin restricciones. Un límite de gasto de cero dólares bloquea todas las tareas cuánticas.

P: ¿Un límite de gasto de cero en el pasado o en el futuro bloquea toda creación de tareas cuánticas?

R: Sí.

P: Al crear un límite de gastos, ¿se tendrá en cuenta el coste estimado de las tareas que ya están en espera para el gasto actual una vez finalizadas dichas tareas?

R: No. Solo las tareas enviadas mientras haya un límite de gasto activo se tendrán en cuenta para el gasto acumulado.

Seguimiento de costos casi en tiempo real

El SDK de Braket le ofrece la opción de añadir un seguimiento de costos casi en tiempo real a sus cargas de trabajo cuánticas. Cada uno de nuestros cuadernos de ejemplo incluye un código de seguimiento de costes para proporcionarle una estimación del coste máximo de las unidades de procesamiento cuántico de Braket () QPUs y de los simuladores bajo demanda. Las estimaciones de costos máximos se mostrarán en USD y no incluyen ningún crédito ni descuento.

> **Note**
>
> Los cargos que se muestran son estimaciones basadas en el uso de tareas del simulador de la unidad de procesamiento cuántico (QPU) y el simulador de Amazon Braket. Los cargos estimados que se muestran pueden diferir de los cargos reales. En los cargos estimados no se incluyen descuentos ni créditos y es posible que se le apliquen cargos adicionales en función del uso de otros servicios, como Amazon Elastic Compute Cloud (Amazon EC2).

Seguimiento de costes para SV1

Para demostrar cómo se puede utilizar la función de seguimiento de costes, construiremos un circuito Bell State y lo ejecutaremos en nuestro SV1 simulador. Empezaremos importando los módulos del SDK de Braket, definiendo un Bell State y añadiendo la función `Tracker()` a nuestro circuito:

```
#import any required modules
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.tracking import Tracker

#create our bell circuit
circ = Circuit().h(0).cnot(0,1)
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
with Tracker() as tracker:
    task = device.run(circ, shots=1000).result()

#Your results
print(task.measurement_counts)
```

```
Counter({'00': 500, '11': 500})
```

Al utilizar su cuaderno, puede esperar el siguiente resultado para su simulación de Bell State. La función de seguimiento le mostrará el número de shots enviados, las tareas cuánticas completadas, la duración de la ejecución, la duración de la ejecución facturada y su costo máximo en USD. El tiempo de ejecución puede variar en cada simulación.

```
import datetime

tracker.quantum_tasks_statistics()
{'arn:aws:braket:::device/quantum-simulator/amazon/sv1':
 {'shots': 1000,
  'tasks': {'COMPLETED': 1},
  'execution_duration': datetime.timedelta(microseconds=4000),
  'billed_execution_duration': datetime.timedelta(seconds=3)}}

tracker.simulator_tasks_cost()
```

```
Decimal('0.0037500000')
```

Uso del rastreador de costos para establecer los costos máximos

Puede usar el rastreador de costos para establecer los costos máximos de un programa. Es posible que tenga un límite máximo de cuánto desea gastar en un programa determinado. De esta forma, puede usar el rastreador de costos para desarrollar una lógica de control de costos en su código de ejecución. El siguiente ejemplo utiliza el mismo circuito en una QPU de Rigetti y limita el costo a 1 USD. El costo de ejecutar una iteración del circuito en nuestro código es de 0,30 USD. Hemos establecido la lógica para repetir las iteraciones hasta que el costo total supere 1 USD; de este modo, el fragmento de código se ejecutará tres veces hasta que la siguiente iteración supere 1 USD. Por lo general, un programa seguiría iterando hasta alcanzar el costo máximo deseado, en este caso, tres iteraciones.

```
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
with Tracker() as tracker:
    while tracker.qpu_tasks_cost() < 1:
        result = device.run(circ, shots=200).result()
print(tracker.quantum_tasks_statistics())
print(tracker.qpu_tasks_cost(), "USD")
```

```
{'arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3': {'shots': 600, 'tasks':
 {'COMPLETED': 3}}}
1.4400000000 USD
```

> **Note**
>
> El rastreador de costos no registrará la duración de las tareas cuánticas de TN1 fallidas. Durante una simulación de TN1, si se completa el ensayo, pero la fase de contracción falla, los gastos del ensayo no se mostrarán en el registro de costos.

Prácticas recomendadas para el ahorro costos

Tenga en cuenta las siguientes prácticas recomendadas al usar Amazon Braket. Ahorre tiempo, minimice los costos y evite errores comunes.

Verificar con simuladores

- Verifique sus circuitos con un simulador antes de ejecutarlo en una QPU, de modo que pueda refinar su circuito sin incurrir en cargos por el uso de la QPU.

- Si bien es posible que los resultados de ejecutar el circuito en un simulador no sean idénticos a los de ejecutar el circuito en una QPU, puede identificar los errores de codificación o los problemas de configuración con un simulador.

Restringir el acceso de los usuarios a determinados dispositivos

- Puede configurar restricciones que impidan que usuarios no autorizados envíen tareas cuánticas en determinados dispositivos. El método recomendado para restringir el acceso es con AWS IAM. Para obtener más información sobre cómo hacerlo, consulte Restringir el acceso.
- Le recomendamos que no utilice su cuenta de administrador para conceder o restringir el acceso de los usuarios a los dispositivos de Amazon Braket.

Configurar alarmas de facturación

- Puede configurar una alarma de facturación para que le avise cuando su factura alcance un límite preestablecido. La forma recomendada de configurar una alarma es mediante AWS Budgets. Puede establecer presupuestos personalizados y recibir alertas cuando sus costos o su uso puedan superar la cantidad presupuestada. Encontrará información disponible en AWS Budgets.

Probar tareas cuánticas de TN1 con números bajos de shots

- Los simuladores cuestan menos que QPUs, pero algunos simuladores pueden resultar caros si las tareas cuánticas se ejecutan con un alto número de disparos. Le recomendamos que pruebe sus tareas de TN1 con un número bajo de shot. El número de Shot no afecta al costo de SV1 ni al de las tareas de simulador local.

Comprobar las tareas cuánticas en todas las regiones

- La consola muestra las tareas cuánticas solo para las actuales. Región de AWS Cuando busque tareas cuánticas facturables que se hayan enviado, asegúrese de comprobar todas las regiones.
- Puede ver una lista de dispositivos y sus regiones asociadas en la página de dispositivos compatibles de la documentación.

Repositorios y referencias de la API para Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con nosotros! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Amazon Braket proporciona APIs SDKs, y una interfaz de línea de comandos que puede usar para crear y administrar instancias de notebook y entrenar e implementar modelos.

- SDK de Python de Amazon Braket (recomendado)
- Referencia de la API de Amazon Braket
- AWS Command Line Interface
- AWS SDK para .NET
- AWS SDK para C++
- AWS SDK para GoAPI Reference
- AWS SDK para Java
- AWS SDK para JavaScript
- AWS SDK para PHP
- AWS SDK para Python (Boto)
- AWS SDK para Ruby

También puede obtener ejemplos de código del repositorio Amazon Braket Tutorials GitHub .

- Tutoriales de Braket GitHub

Repositorios principales

A continuación se muestra una lista de repositorios principales que contienen paquetes clave que se utilizan para Braket:

- SDK de Python de Braket: utilice el SDK de Python de Braket para configurar su código en los cuadernos de Jupyter en el lenguaje de programación Python. Una vez configurados cuadernos de Jupyter, puede ejecutar el código en dispositivos y simuladores de Braket.
- Esquemas de Braket: el contrato entre el SDK de Braket y el servicio de Braket.
- Simulador predeterminado de Braket: todos nuestros simuladores cuánticos locales para Braket (vector de estado y matriz de densidad).

Plugins

Luego están los diversos complementos que se utilizan junto con varios dispositivos y herramientas de programación. Estos incluyen los complementos compatibles con Braket, así como los complementos compatibles con terceros, como se muestra a continuación.

Compatibilidad con Amazon Braket:

- Biblioteca de algoritmos de Amazon Braket: catálogo de algoritmos cuánticos prediseñados escritos en Python. Ejecútelos tal como están o utilícelos como punto de partida para desarrollar algoritmos más complejos.
- PennyLane Plugin Braket: utilícelo PennyLane como marco QML en Braket.

Terceros (el equipo de Braket supervisa y contribuye):

- Proveedor de Qiskit-Braket: utilice el SDK de Qiskit para acceder a los recursos de Braket.
- SDK de Braket-Julia SDK: (EXPERIMENTAL) una versión nativa de Julia del SDK de Braket.

Regiones y dispositivos compatibles con Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

En Amazon Braket, un dispositivo representa una unidad de procesamiento cuántico (QPU) o un simulador al que se puede llamar para ejecutar tareas cuánticas. Amazon Braket proporciona

acceso a dispositivos QPU desdeAQT,, IonQIQM, QuEra y. Rigetti Además, AWS ofrece acceso a simuladores bajo demanda, locales e integrados. Para obtener más información sobre los simuladores integrados, consulte Acerca de los simuladores integrados.

Para obtener información sobre los proveedores de hardware cuántico compatibles, consulte Enviar tareas cuánticas a. QPUs Para obtener información sobre los simuladores disponibles, consulte Envío de tareas cuánticas a simuladores. En la siguiente tabla, se muestra la lista de dispositivos y simuladores disponibles.

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Region
AQT	IBEX-Q1	basado en puertas	QPU	arn:aws:braket:eu-north-1:: -Q1 device/qpu/aqt/Ibex	eu-north-1
IonQ	Forte-1	basado en puertas	QPU	arn:aws:braket:us-east-1:: -1 device/qpu/ionq/Forte	us-east-1
IonQ	Forte-Enterprise-1	basado en puertas	QPU	arn:aws:braket:us-east-1:: -Enterprise-1 device/qpu/ionq/Forte	us-east-1
IQM	Garnet	basado en puertas	QPU	arn:aws:braket:eu-north-1:: device/qpu/iqm/Garnet	eu-north-1
IQM	Emerald	basado en puertas	QPU	arn:aws:braket:eu-north-1:: device/qpu/iqm/Emerald	eu-north-1
QuEra	Aquila	Simulación hamiltoniana analógica	QPU	arn:aws:braket:us-east-1:: device/qpu/quera/Aquila	us-east-1

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Region
Rigetti	Ankaa-3	basado en puertas	QPU	arn:aws:braket:us-west-1:: -3 device/qpu/rigetti/Ankaa	us-west-1
AWS	braket_sv	basado en puertas	Simulador local	N/A (simulador local en SDK de Braket)	N/A
AWS	braket_dm	basado en puertas	Simulador local	N/A (simulador local en SDK de Braket)	N/A
AWS	braket_ahs	Simulación hamiltoniana analógica	Simulador local	N/A (simulador local en SDK de Braket)	N/A
AWS	SV1	basado en puertas	Simulador bajo demanda	arn:aws:braket::: 1 device/quantum-simulator/amazon/sv	us-east-1, us-west-1, us-west-2, eu-west-2
AWS	DM1	basado en puertas	Simulador bajo demanda	arn:aws:braket::: device/quantum-simulator/amazon/dm 1	us-east-1, us-west-1, us-west-2, eu-west-2

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Region
AWS	TN1	basado en puertas	Simulador bajo demanda	arn:aws:braket:::device/quantum-simulator/amazon/tn 1	us-east-1, us-west-2, and eu-west-2

> **Note**
>
> El dispositivo distingue entre mayúsculas y minúsculas. ARNs Por ejemplo, cuando utilice el AQT IBEX-Q1 dispositivo, compruebe que el ARN del dispositivo contiene. 'Ibex-Q1'

Para ver detalles adicionales sobre QPUs lo que puedes usar con Amazon Braket, consulta Amazon Braket Quantum Computers.

Propiedades del dispositivo

Para todos los dispositivos, puede encontrar más propiedades, como la topología del dispositivo, los datos de calibración y los conjuntos de puertas nativas, en la pestaña Dispositivos de la consola de Amazon Braket o mediante la API de `GetDevice`. Al construir un circuito con los simuladores, Amazon Braket requiere que utilice qubits contiguos o índices. Al trabajar con el SDK, en el siguiente ejemplo de código se muestra cómo acceder a las propiedades de cada dispositivo y simulador disponibles.

```
from braket.aws import AwsDevice
from braket.devices import LocalSimulator

device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/sv1')
 # SV1
# device = LocalSimulator()
 # Local State Vector Simulator
# device = LocalSimulator("default")
 # Local State Vector Simulator
# device = LocalSimulator(backend="default")
 # Local State Vector Simulator
# device = LocalSimulator(backend="braket_sv")
 # Local State Vector Simulator
```

```
# device = LocalSimulator(backend="braket_dm")
 # Local Density Matrix Simulator
# device = LocalSimulator(backend="braket_ahs")
 # Local Analog Hamiltonian Simulation
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/tn1')
 # TN1
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/dm1')
 # DM1
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1')
 # AQT IBEX-Q1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
 # IonQ Forte-1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1')
 # IonQ Forte-Enterprise-1
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet')
 # IQM Garnet
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald')
 # IQM Emerald
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/quera/Aquila')
 # QuEra Aquila
# device = AwsDevice('arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3')
 # Rigetti Ankaa-3

# Get device properties
device.properties
```

Regiones y puntos de conexión de Amazon Braket

Para ver una lista de las regiones y los puntos de conexión, consulte la Referencia general de AWS.

Las tareas de Quantum que se ejecutan en un dispositivo QPU se pueden ver en la consola de Amazon Braket de la región de ese dispositivo. Al utilizar el SDK de Amazon Braket, puede enviar tareas cuánticas a cualquier dispositivo QPU, independientemente de la región en la que trabaje. El SDK crea automáticamente una sesión en la región para la QPU especificada.

Amazon Braket está disponible en las siguientes versiones: Regiones de AWS

Nombre de la región	Region	Puntos de conexión de Braket
Este de EE. UU. (Norte de Virginia)	us-east-1	braket.us-east-1.amazonaws.com (IPv4 únicamente)

Nombre de la región	Region	Puntos de conexión de Braket
		braket.us-east-1.api.aws (doble pila)
Oeste de EE. UU. (Norte de California)	us-west-1	braket.us-west-1.amazonaws.com (IPv4 únicamente) braket.us-west-1.api.aws (doble pila)
Oeste de EE. UU. 2 (Oregón)	us-west-2	braket.us-west-2.amazonaws.com (IPv4 únicamente) braket.us-west-2.api.aws (doble pila)
Norte de Europa 1 (Estocolmo)	eu-north-1	braket.eu-north-1.amazonaws.com (IPv4 únicamente) braket.eu-north-1.api.aws (doble pila)
Oeste de Europa 2 (Londres)	eu-west-2	braket.eu-west-2.amazonaws.com (IPv4 únicamente) braket.eu-west-2.api.aws (doble pila)

> **Note**
>
> El SDK de Amazon Braket no es compatible únicamente con redes IPv6.

Introducción a Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con AWS! Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Una vez que haya seguido las instrucciones de la sección Activación de Amazon Braket, podrá empezar a utilizar Amazon Braket.

Los pasos para comenzar incluyen lo siguiente:

- Activación de Amazon Braket
- Creación de una instancia de cuaderno de Amazon Braket
- Crea una instancia de Braket notebook usando CloudFormation

Activación de Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Puede activar Amazon Braket en su cuenta a través de la consola de AWS.

En esta sección:

- Requisitos previos
- Pasos para la activación de Amazon Braket

Requisitos previos

Para activar y ejecutar Amazon Braket, debe tener un usuario o rol con permiso para iniciar acciones de Amazon Braket. Estos permisos se incluyen en la política de IAM (arn:aws:iam: :aws:policy/). AmazonBraketFullAccess AmazonBraketFullAccess

> **Note**
>
> Si es un administrador:
> Para permitir que otros usuarios accedan a Amazon Braket, concede permisos a los usuarios adjuntando la AmazonBraketFullAccesspolítica o adjuntando una política personalizada que tú crees. Para obtener más información sobre los permisos necesarios para utilizar Amazon Braket, consulte Administración del acceso a Amazon Braket.

Pasos para la activación de Amazon Braket

1. Inicia sesión en la consola Amazon Braket con tu. Cuenta de AWS
2. Abra la consola de Amazon Braket.
3. En la página de inicio de Braket, haga clic en Comenzar para ir a la página Panel de servicio. La alerta que aparece en la parte superior del panel de servicio le guiará a través de los tres pasos siguientes:
   a. Creación de los roles vinculados a servicios (SLR)
   b. Habilitación del acceso a computadoras cuánticas de terceros
   c. Creación de una nueva instancia de cuaderno de Jupyter

Para utilizar dispositivos cuánticos de terceros, debes aceptar ciertas condiciones relacionadas con la transferencia de datos entre tú y esos dispositivos. AWS Los términos y condiciones de este acuerdo se proporcionan en la pestaña General de la página Permisos y configuración en la consola de Amazon Braket.

> **Note**
>
> Los dispositivos cuánticos que no implican a terceros, como los simuladores locales de Braket o los simuladores bajo demanda, pueden utilizarse sin necesidad de aceptar el acuerdo de Habilitar dispositivos de terceros.

Solo es necesario aceptar estos términos para permitir el uso de dispositivos de terceros una vez por cuenta si accede a hardware de terceros.

Creación de una instancia de cuaderno de Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con AWS! Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Amazon Braket ofrece cuadernos de Jupyter totalmente administrados para que pueda empezar. Las instancias de bloc de notas Amazon Braket se basan en las instancias de bloc de notas Amazon SageMaker AI. En los pasos siguientes se explica cómo crear una nueva instancia de cuaderno para los clientes nuevos y existentes.

Clientes nuevos de Amazon Braket:

1. Abra la consola de Amazon Braket y vaya a la página Panel en el panel izquierdo.
2. Haga clic en Comenzar en el modal Bienvenido a Amazon Braket en el centro de la página de su panel. Proporcione un nombre de cuaderno para crear un cuaderno de Jupyter predeterminado.
   a. La creación de su cuaderno puede tardar varios minutos.
   b. Su cuaderno aparecerá en la página Cuadernos con el estado Pendiente.
   c. Cuando la instancia de su notebook esté lista para usarse, el estado cambiará a InService.
   d. Actualice la página para mostrar el estado actualizado del cuaderno.

Clientes existentes de Amazon Braket:

1. Abra la consola de Amazon Braket y seleccione Cuadernos en el panel izquierdo.
2. Seleccione Crear instancia de cuaderno.
   a. Si no tiene cuadernos, seleccione la configuración estándar para crear un cuaderno de Jupyter predeterminado.
3. Introduzca un Nombre de instancia del cuaderno utilizando solo caracteres alfanuméricos y guiones, y seleccione su Modo visual preferido.

4. Active o desactive el gestor de inactividad de cuadernos para su cuaderno.
   a. Si está activado, seleccione el tiempo de inactividad deseado antes de restablecer el cuaderno. Cuando se reinicia un cuaderno, los gastos de procesamiento dejan de generarse, pero los gastos de almacenamiento se mantienen.
   b. Para comprobar cuánto tiempo de inactividad le queda a la instancia de su cuaderno, vaya a la barra de comandos, seleccione la pestaña Braket y, a continuación, la pestaña Gestor de inactividad.

> **Note**
>
> Para guardar tu trabajo, integra tu instancia de bloc de notas de SageMaker IA con un repositorio de Git. Como alternativa, mueva su trabajo fuera de las carpetas `/Braket Algorithms` y `/Braket Examples` para que no se sobrescriban al reiniciar la instancia del cuaderno.

5. (Opcional) Con la configuración avanzada, puede crear un cuaderno con permisos de acceso, configuraciones adicionales y ajustes de acceso a la red:
   a. En la Configuración del cuaderno, elija el tipo de instancia.
      i. De forma predeterminada, se elige el tipo de instancia estándar y rentable, ml.t3.medium. Para obtener más información sobre los precios de las instancias, consulta los precios de Amazon SageMaker AI.
   b. Para asociar un repositorio público de GitHub a tu instancia de cuaderno, haga clic en el menú desplegable del repositorio de Git y seleccione Clonar un repositorio Git público desde una URL en el menú desplegable Repositorio. Introduzca la URL del repositorio en la barra de texto URL del repositorio Git.
   c. En Permisos de acceso, configure los roles de IAM, el acceso raíz y las claves de cifrado opcionales.
   d. En Acceso a la red, configure los ajustes de red y acceso personalizados para la instancia de Jupyter Notebook.
6. Revise la configuración y establezca cualquier etiqueta para identificar la instancia del cuaderno. Haga clic en Lanzar.

> **Note**
>
> Vea y gestione las instancias de su bloc de notas Amazon Braket en las consolas Amazon Braket y SageMaker Amazon AI. Los ajustes adicionales del portátil Amazon Braket están disponibles a través de la SageMaker consola.

Si está trabajando en la consola de Amazon Braket, el SDK de Amazon Braket y AWS los complementos vienen precargados en las libretas que ha creado. Para ejecutarlo en su propia máquina, instale el SDK y los complementos cuando ejecute el comando `pip install amazon-braket-sdk` o cuando ejecute el comando para los complementos. `pip install amazon-braket-pennylane-plugin` PennyLane

Crea una instancia de Braket notebook usando CloudFormation

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital de Amazon Braket y obtenga su propia insignia digital tras completar una serie de cursos de aprendizaje y una evaluación digital.

Puedes usarlo CloudFormation para gestionar las instancias de tu bloc de notas Amazon Braket. Las instancias Braket Notebook se basan en Amazon SageMaker AI. Con CloudFormation, puede aprovisionar una instancia de bloc de notas con un archivo de plantilla que describa la configuración prevista. El archivo de plantilla está escrito en formato JSON o YAML. Puede crear, actualizar y eliminar instancias de forma ordenada y repetible. Esto puede resultarle útil cuando gestione varias instancias de cuaderno de Braket en su Cuenta de AWS.

Después de crear una CloudFormation plantilla para un bloc de notas Braket, se utiliza CloudFormation para implementar el recurso. Para obtener más información, consulte Crear una pila en la CloudFormation consola en la guía del CloudFormation usuario.

Para crear una instancia de Braket Notebook mediante CloudFormation, siga estos tres pasos:

1. Cree un script de configuración del ciclo de vida de la SageMaker IA.
2. Cree un rol AWS Identity and Access Management (de IAM) para que lo asuma la SageMaker IA.
3. Cree una instancia de bloc de notas de SageMaker IA con el prefijo **`amazon-braket-`**

Puede reutilizar la configuración del ciclo de vida de todos los cuadernos de Braket que cree. También puede reutilizar el rol de IAM para los cuadernos de Braket a los que asigne los mismos permisos de ejecución.

En esta sección:

- Paso 1: Cree un script de configuración del ciclo de vida de la SageMaker IA
- Paso 2: Crear la función de IAM que asume Amazon AI SageMaker
- Paso 3: Crea una instancia de bloc de notas de SageMaker IA con el prefijo amazon-braket-

Paso 1: Cree un script de configuración del ciclo de vida de la SageMaker IA

Utilice la siguiente plantilla para crear un script de configuración del ciclo de vida de la SageMaker IA. El script personaliza una instancia de bloc de notas de SageMaker IA para Braket. Para ver las opciones de configuración del CloudFormation recurso del ciclo de vida, consulte AWS::SageMaker::NotebookInstanceLifecycleConfigla guía del CloudFormation usuario.

```
BraketNotebookInstanceLifecycleConfig:
    Type: "AWS::SageMaker::NotebookInstanceLifecycleConfig"
    Properties:
      NotebookInstanceLifecycleConfigName: BraketLifecycleConfig-${AWS::StackName}
      OnStart:
      - Content:
          Fn::Base64: |
            #!/usr/bin/env bash
            sudo -u ec2-user -i #EOS
            curl -o braket-notebook-lcc.zip https://d3ded4lzb1lnme.cloudfront.net/
notebook/braket-notebook-lcc.zip
            unzip braket-notebook-lcc.zip
            ./install.sh
            EOS

            exit 0
```

Paso 2: Crear la función de IAM que asume Amazon AI SageMaker

Cuando utiliza una instancia de Braket Notebook, la SageMaker IA realiza operaciones en su nombre. Por ejemplo, supongamos que ejecuta un cuaderno de Braket utilizando un circuito de

un dispositivo compatible. En la instancia de bloc de notas, la SageMaker IA ejecuta la operación en Braket por ti. La función de ejecución del cuaderno define las operaciones exactas que la SageMaker IA puede ejecutar en tu nombre. Para obtener más información, consulte las funciones de SageMaker IA en la guía para desarrolladores de Amazon SageMaker AI.

Utilice el siguiente ejemplo para crear un rol de ejecución de cuaderno de Braket con los permisos requeridos. Puede modificar las políticas en función de sus necesidades.

> **Note**
>
> Asegúrese de que el rol tenga permiso para las operaciones `s3:ListBucket` y `s3:GetObject` en los buckets de Amazon S3 con el prefijo `braketnotebookcdk-`". El script de configuración del ciclo de vida requiere estos permisos para copiar el script de instalación del cuaderno de Braket.

```
ExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: !Sub AmazonBraketNotebookRole-${AWS::StackName}
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "sagemaker.amazonaws.com"
          Action:
          - "sts:AssumeRole"
      Path: "/service-role/"
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonBraketFullAccess
      Policies:
        -
          PolicyName: "AmazonBraketNotebookPolicy"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
```

```
                - s3:GetObject
                - s3:PutObject
                - s3:ListBucket
              Resource:
                - arn:aws:s3:::amazon-braket-*
                - arn:aws:s3:::braketnotebookcdk-*
            - Effect: "Allow"
              Action:
                - "logs:CreateLogStream"
                - "logs:PutLogEvents"
                - "logs:CreateLogGroup"
                - "logs:DescribeLogStreams"
              Resource:
                - !Sub "arn:aws:logs:*:${AWS::AccountId}:log-group:/aws/sagemaker/*"
            - Effect: "Allow"
              Action:
                - braket:*
              Resource: "*"
```

Paso 3: Crea una instancia de bloc de notas de SageMaker IA con el prefijo **`amazon-braket-`**

Utilice el script del ciclo de vida de la SageMaker IA y el rol de IAM creados en los pasos 1 y 2 para crear una instancia de bloc de notas de SageMaker IA. La instancia del cuaderno está personalizada para Braket y se puede acceder a ella desde la consola de Amazon Braket. Para obtener más información sobre las opciones de configuración de este CloudFormation recurso, consulta AWS::SageMaker::NotebookInstancela guía del CloudFormation usuario.

```
BraketNotebook:
    Type: AWS::SageMaker::NotebookInstance
    Properties:
      InstanceType: ml.t3.medium
      NotebookInstanceName: !Sub amazon-braket-notebook-${AWS::StackName}
      RoleArn: !GetAtt ExecutionRole.Arn
      VolumeSizeInGB: 30
      LifecycleConfigName: !GetAtt
 BraketNotebookInstanceLifecycleConfig.NotebookInstanceLifecycleConfigName
```

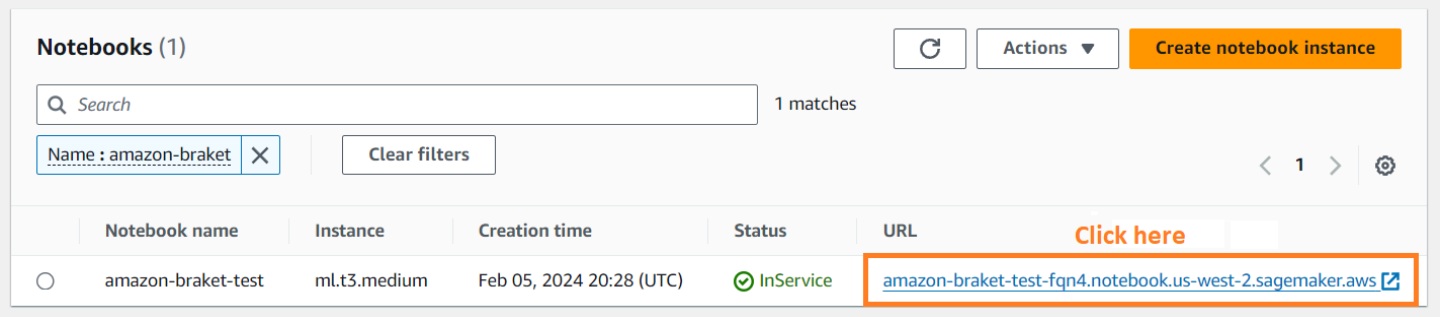
Desarrollo de tareas cuánticas con Amazon Braket

Braket proporciona entornos de cuaderno de Jupyter totalmente administrados que facilitan la puesta en marcha. Los cuadernos de Braket vienen preinstalados con ejemplos de algoritmos, recursos y herramientas para desarrolladores, incluido el SDK de Amazon Braket. Con el SDK de Amazon Braket, puede desarrollar algoritmos cuánticos y, después, probarlos y ejecutarlos en diferentes simuladores y computadoras cuánticas cambiando una sola línea de código.

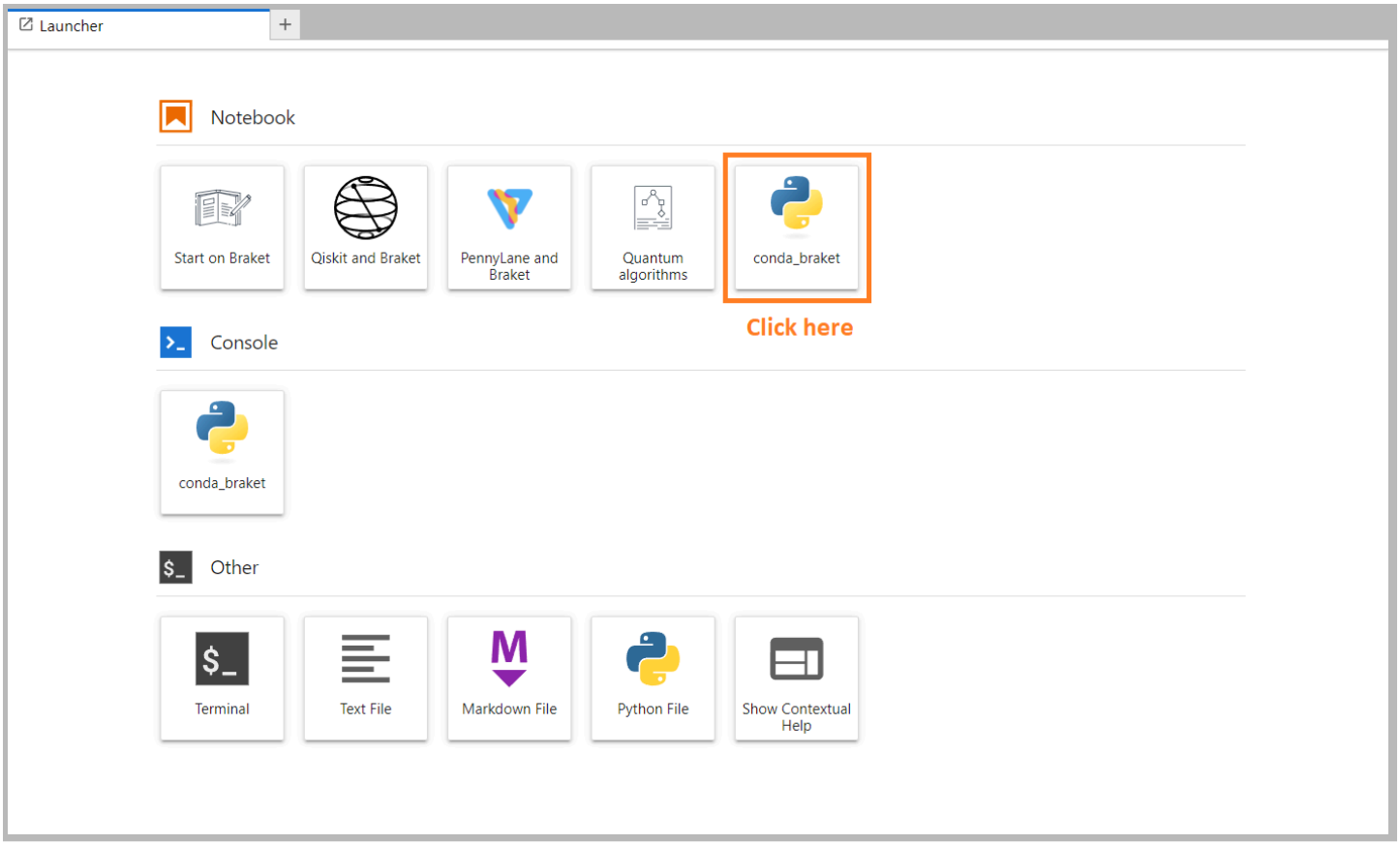
En esta sección:

Desarrollo de su primer circuito

Una vez lanzada la instancia de cuaderno, ábrala con una interfaz de Jupyter estándar. Para ello, seleccione el cuaderno que acaba de crear.



Las instancias de cuaderno de Amazon Braket vienen preinstaladas con el SDK de Amazon Braket y todas sus dependencias. Comience por crear un nuevo cuaderno con kernel de `conda_braket`.



Por ejemplo, puede empezar con un simple “¡Hola, mundo!” . Primero, desarrolle un circuito que prepare un estado de Bell y, a continuación, ejecute ese circuito en diferentes dispositivos para obtener los resultados.

Comience por importar el Comience por importar los módulos del SDK de Amazon Braket y defina un circuito simpleBRAKETlong; los módulos SDK y defina un circuito básico de Bell State.

```
import boto3
from braket.aws import AwsDevice
from braket.devices import LocalSimulator
from braket.circuits import Circuit

# Create the circuit
bell = Circuit().h(0).cnot(0, 1)
```

Puede ver el circuito con este comando:

```
print(bell)
```

```
T  : #  0  #  1  #
      #####
q0 : ## H ########
      #####   #
            #####
q1 : ######## X ##
            #####
T  : #  0  #  1  #
```

Ejecución del circuito en el simulador local

A continuación, elija el dispositivo cuántico en el que ejecutar el circuito. El SDK de Amazon Braket incluye un simulador local para la creación rápida de prototipos y pruebas. Recomendamos utilizar el simulador local para circuitos más pequeños, que pueden ser de hasta 25 qubits (según el hardware local).

Para crear una instancia de simulador local:

```
# Instantiate the local simulator
local_sim = LocalSimulator()
```

Para ejecutar el circuito:

```
# Run the circuit
result = local_sim.run(bell, shots=1000).result()
counts = result.measurement_counts
print(counts)
```

Debería ver un resultado similar al siguiente:

```
Counter({'11': 503, '00': 497})
```

El estado de Bell específico que ha preparado es una superposición igual de |00# y |11#, y una distribución aproximadamente igual (hasta el ruido de shot) de 00 y 11 como resultados de medición, según se espera.

Ejecución del circuito en un simulador bajo demanda

Amazon Braket también proporciona acceso a un simulador de alto rendimiento bajo demanda, SV1, para ejecutar circuitos más grandes. SV1 es un simulador de vector de estado bajo demanda que permite simular circuitos cuánticos de hasta 34 qubits. Encontrará más información SV1 en la

sección Dispositivos compatibles y en la AWS consola. Cuando ejecuta tareas cuánticas en SV1 (y en TN1 o en cualquier QPU), los resultados de la tarea cuántica se almacenan en un bucket de S3 en su cuenta. Si no especifica ningún bucket, el SDK de Braket crea un bucket predeterminado `amazon-braket-{region}-{accountID}` para usted. Para obtener más información, consulte Administración del acceso a Amazon Braket.

> **Note**
>
> Rellene el nombre real y existente de su bucket en donde el siguiente ejemplo muestra `amazon-braket-s3-demo-bucket` como el nombre del bucket. Los nombres de los buckets de Amazon Braket siempre comienzan por `amazon-braket-` seguido de otros caracteres identificativos que añada. Si necesita información sobre cómo configurar un bucket de S3, consulte Introducción a Amazon S3.

```
# Get the account ID
aws_account_id = boto3.client("sts").get_caller_identity()["Account"]

# The name of the bucket
my_bucket = "amazon-braket-s3-demo-bucket"

# The name of the folder in the bucket
my_prefix = "simulation-output"
s3_folder = (my_bucket, my_prefix)
```

Para ejecutar un circuito en SV1, debe proporcionar la ubicación del bucket de S3 que seleccionó previamente como argumento posicional en la llamada de `.run()`.

```
# Choose the cloud-based on-demand simulator to run your circuit
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")

# Run the circuit
task = device.run(bell, s3_folder, shots=100)

# Display the results
print(task.result().measurement_counts)
```

La consola de Amazon Braket proporciona más información sobre su tarea cuántica. Diríjase a la pestaña Tareas cuánticas de la consola y su tarea cuántica debería aparecer en la parte superior de

la lista. Como alternativa, puede buscar su tarea cuántica utilizando el identificador único de la tarea cuántica u otros criterios.

> **Note**
>
> Transcurridos 90 días, Amazon Braket elimina automáticamente todas las tareas cuánticas IDs y otros metadatos asociados a sus tareas cuánticas. Para obtener más información consulte Retención de datos.

Ejecución en una QPU

Con Amazon Braket, puede ejecutar el ejemplo anterior de circuito cuántico en una computadora cuántica física simplemente cambiando una línea de código. Amazon Braket proporciona acceso a una variedad de dispositivos con unidades de procesamiento cuántico (QPU). Puede encontrar información sobre los distintos dispositivos y las ventanas de disponibilidad en la sección Dispositivos compatibles y en la AWS consola, en la pestaña Dispositivos. En el siguiente ejemplo se muestra cómo crear instancias de un dispositivo de IQM.

```
# Choose the IQM hardware to run your circuit
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")
```

O elija un dispositivo de IonQ con este código:

```
# Choose the Ionq device to run your circuit
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1")
```

Después de seleccionar un dispositivo y antes de ejecutar su carga de trabajo, puede consultar la profundidad de la cola del dispositivo con el siguiente código para determinar el número de tareas cuánticas o trabajos híbridos. Además, los clientes pueden ver las profundidades de cola específicas de cada dispositivo en la página Dispositivos de la Amazon Braket Management Console.

```
# Print your queue depth
print(device.queue_depth().quantum_tasks)
# Returns the number of quantum tasks queued on the device
# {<QueueType.NORMAL: 'Normal'>: '0', <QueueType.PRIORITY: 'Priority'>: '0'}

print(device.queue_depth().jobs)
# Returns the number of hybrid jobs queued on the device
```

```
# '2'
```

Al ejecutar la tarea, el SDK de Amazon Braket sondea para obtener un resultado (con un tiempo de espera predeterminado de 5 días). Puede cambiar este valor predeterminado modificando el parámetro `poll_timeout_seconds` en el comando `.run()`, como se muestra en el siguiente ejemplo. Tenga en cuenta que si el tiempo de espera de sondeo es demasiado corto, es posible que los resultados no se devuelvan dentro del tiempo de sondeo, como cuando una QPU no está disponible y se devuelve un error de tiempo de espera local. Puede reiniciar el sondeo llamando a la función `task.result()`.

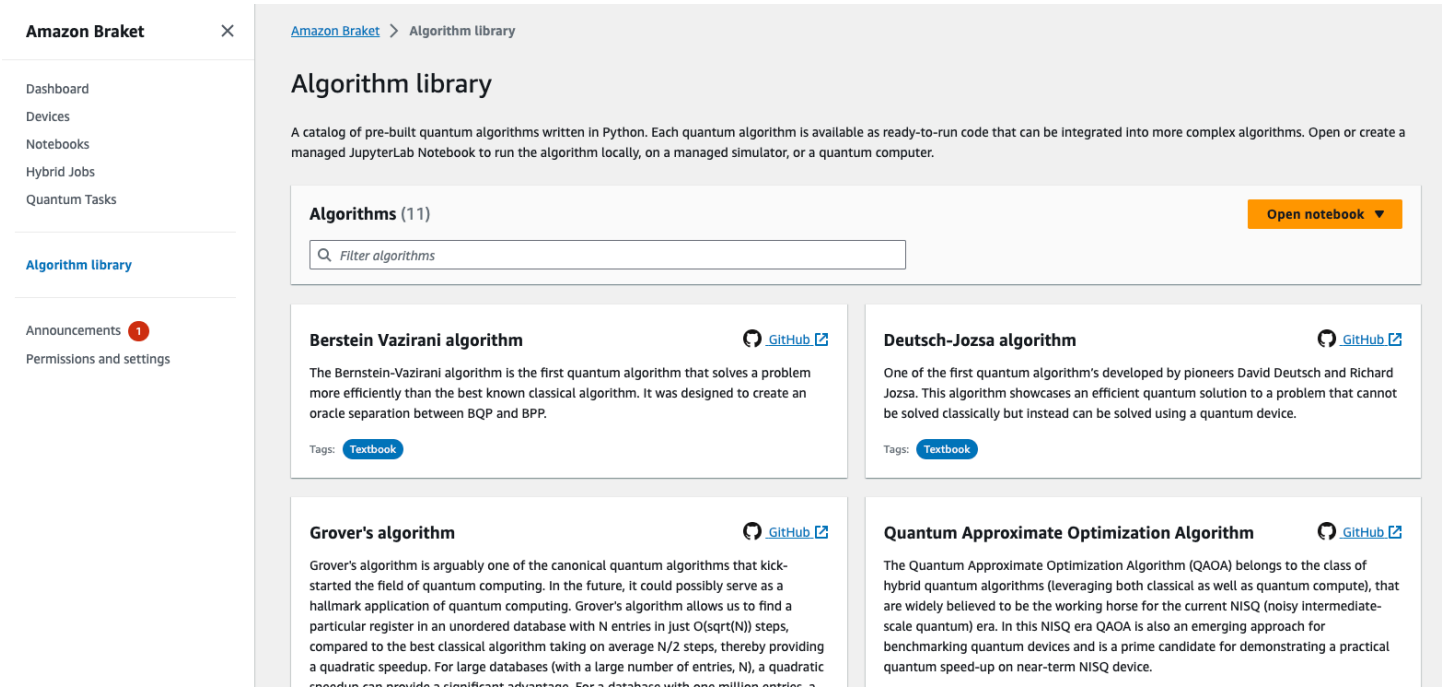
```
# Define quantum task with 1 day polling timeout
task = device.run(bell, s3_folder, poll_timeout_seconds=24*60*60)
print(task.result().measurement_counts)
```

Además, después de enviar su tarea cuántica o trabajo híbrido, puede llamar a la función `queue_position()` para comprobar su posición en la cola.

```
print(task.queue_position().queue_position)
# Return the number of quantum tasks queued ahead of you
# '2'
```

Desarrollo de sus primeros algoritmos cuánticos

La biblioteca de algoritmos de Amazon Braket es un catálogo de algoritmos cuánticos preconstruidos escritos en Python. Ejecute estos algoritmos tal como están o utilícelos como punto de partida para desarrollar algoritmos más complejos. Puede acceder a la biblioteca de algoritmos desde la consola de Braket. Para obtener más información, consulte la Biblioteca de algoritmos del GitHub de Braket.



La consola de Braket proporciona una descripción de cada algoritmo disponible en la biblioteca de algoritmos. Elija un GitHub enlace para ver los detalles de cada algoritmo o seleccione Abrir bloc de notas para abrir o crear un bloc de notas que contenga todos los algoritmos disponibles. Si elige la opción de cuaderno, podrá encontrar la biblioteca de algoritmos de Braket en la carpeta raíz de su cuaderno.

Construcción de circuitos en el SDK

En esta sección se proporcionan ejemplos sobre cómo definir un circuito, ver las puertas disponibles, extender un circuito y ver las puertas compatibles con cada dispositivo. También contiene instrucciones sobre cómo realizar la asignación manual de qubits, indicar al compilador que ejecute los circuitos exactamente como se ha definido y construir circuitos ruidosos con un simulador de ruido.

También puedes trabajar al nivel del pulso en Braket para varias puertas, con algunas QPUs. Para obtener más información, consulte Control de pulsos en Amazon Braket.

En esta sección:

- Puertas y circuitos
- Conjuntos de programas
- Medición parcial
- Asignación manual de qubit

- Compilación verbatim
- Simulación de ruido

Puertas y circuitos

Las puertas y circuitos cuánticos se definen en la clase de `braket.circuits` del SDK de Python de Amazon Braket. Desde el SDK, puede crear una instancia de un nuevo objeto de circuito llamando a `Circuit()`.

Ejemplo: Definir un circuito

El ejemplo comienza definiendo un circuito de muestra de cuatro qubits (etiquetados `q0`, `q1`, `q2` y `q3`) que consisten en puertas Hadamard estándar de un solo qubit y puertas CNOT de dos qubits. Puede visualizar este circuito llamando a la función `print`, como se muestra en el siguiente ejemplo.

```
# Import the circuit module
from braket.circuits import Circuit

# Define circuit with 4 qubits
my_circuit = Circuit().h(range(4)).cnot(control=0, target=2).cnot(control=1, target=3)
print(my_circuit)
```

```
T  : #  0  #     1     #
      #####
q0 : ## H ##############
      #####   #
      #####   #
q1 : ## H ##############
      #####   #     #
      ##### #####   #
q2 : ## H ### X ########
      ##### #####   #
      #####       #####
q3 : ## H ######### X ##
      #####       #####
T  : #  0  #     1     #
```

Ejemplo: Definir un circuito parametrizado

En este ejemplo, definimos un circuito con puertas que dependen de parámetros libres. Podemos especificar los valores de estos parámetros para crear un circuito nuevo o, al enviar el circuito, para que se ejecute como una tarea cuántica en determinados dispositivos.

```
from braket.circuits import Circuit, FreeParameter

# Define a FreeParameter to represent the angle of a gate
alpha = FreeParameter("alpha")

# Define a circuit with three qubits
my_circuit = Circuit().h(range(3)).cnot(control=0, target=2).rx(0, alpha).rx(1, alpha)
print(my_circuit)
```

Puede crear un circuito nuevo no parametrizado a partir de uno parametrizado proporcionando un único argumento `float` (que será el valor que tomarán todos los parámetros libres) o argumentos de palabra clave que especifiquen el valor de cada parámetro del circuito, como se indica a continuación.

```
my_fixed_circuit = my_circuit(1.2)
my_fixed_circuit = my_circuit(alpha=1.2)
print(my_fixed_circuit)
```

Tenga en cuenta que `my_circuit` no está modificado, por lo que puede usarlo para crear instancias de muchos circuitos nuevos con valores de parámetros fijos.

Ejemplo: Modificar las puertas de un circuito

En el siguiente ejemplo se define un circuito con puertas que utilizan modificadores de control y potencia. Puede utilizar estas modificaciones para crear nuevas puertas, como la puerta de Ry controlada.

```
from braket.circuits import Circuit

# Create a bell circuit with a controlled x gate
my_circuit = Circuit().h(0).x(control=0, target=1)

# Add a multi-controlled Ry gate of angle .13
my_circuit.ry(angle=.13, target=2, control=(0, 1))

# Add a 1/5 root of X gate
my_circuit.x(0, power=1/5)
```

```
print(my_circuit)
```

Los modificadores de puerta solo son compatibles con el simulador local.

Ejemplo: Consultar todas las puertas disponibles

En el siguiente ejemplo se muestra cómo ver todas las puertas disponibles en Amazon Braket.

```
from braket.circuits import Gate
# Print all available gates in Amazon Braket
gate_set = [attr for attr in dir(Gate) if attr[0].isupper()]
print(gate_set)
```

El resultado de este código muestra una lista de todas las puertas.

```
['CCNot', 'CNot', 'CPhaseShift', 'CPhaseShift00', 'CPhaseShift01', 'CPhaseShift10',
 'CSwap', 'CV', 'CY', 'CZ', 'ECR', 'GPhase', 'GPi', 'GPi2', 'H', 'I', 'ISwap', 'MS',
 'PRx', 'PSwap', 'PhaseShift', 'PulseGate', 'Rx', 'Ry', 'Rz', 'S', 'Si', 'Swap', 'T',
 'Ti', 'U', 'Unitary', 'V', 'Vi', 'X', 'XX', 'XY', 'Y', 'YY', 'Z', 'ZZ']
```

Cualquiera de estas puertas se puede añadir a un circuito llamando al método correspondiente a ese tipo de circuito. Por ejemplo, llamar a `circ.h(0)` para añadir una puerta Hadamard a al primer qubit.

> **Note**
>
> Las puertas están colocadas en su sitio y en el siguiente ejemplo se añaden todas las puertas enumeradas en el ejemplo anterior al mismo circuito.

```
circ = Circuit()
# toffoli gate with q0, q1 the control qubits and q2 the target.
circ.ccnot(0, 1, 2)
# cnot gate
circ.cnot(0, 1)
# controlled-phase gate that phases the |11> state, cphaseshift(phi) =
 diag((1,1,1,exp(1j*phi))), where phi=0.15 in the examples below
circ.cphaseshift(0, 1, 0.15)
# controlled-phase gate that phases the |00> state, cphaseshift00(phi) =
 diag([exp(1j*phi),1,1,1])
```

```
circ.cphaseshift00(0, 1, 0.15)
# controlled-phase gate that phases the |01> state, cphaseshift01(phi) =
 diag([1,exp(1j*phi),1,1])
circ.cphaseshift01(0, 1, 0.15)
# controlled-phase gate that phases the |10> state, cphaseshift10(phi) =
 diag([1,1,exp(1j*phi),1])
circ.cphaseshift10(0, 1, 0.15)
# controlled swap gate
circ.cswap(0, 1, 2)
# swap gate
circ.swap(0,1)
# phaseshift(phi)= diag([1,exp(1j*phi)])
circ.phaseshift(0,0.15)
# controlled Y gate
circ.cy(0, 1)
# controlled phase gate
circ.cz(0, 1)
# Echoed cross-resonance gate applied to q0, q1
circ = Circuit().ecr(0,1)
# X rotation with angle 0.15
circ.rx(0, 0.15)
# Y rotation with angle 0.15
circ.ry(0, 0.15)
# Z rotation with angle 0.15
circ.rz(0, 0.15)
# Hadamard gates applied to q0, q1, q2
circ.h(range(3))
# identity gates applied to q0, q1, q2
circ.i([0, 1, 2])
# iswap gate, iswap = [[1,0,0,0],[0,0,1j,0],[0,1j,0,0],[0,0,0,1]]
circ.iswap(0, 1)
# pswap gate, PSWAP(phi) = [[1,0,0,0],[0,0,exp(1j*phi),0],[0,exp(1j*phi),0,0],
[0,0,0,1]]
circ.pswap(0, 1, 0.15)
# X gate applied to q1, q2
circ.x([1, 2])
# Y gate applied to q1, q2
circ.y([1, 2])
# Z gate applied to q1, q2
circ.z([1, 2])
# S gate applied to q0, q1, q2
circ.s([0, 1, 2])
# conjugate transpose of S gate applied to q0, q1
circ.si([0, 1])
```

```
# T gate applied to q0, q1
circ.t([0, 1])
# conjugate transpose of T gate applied to q0, q1
circ.ti([0, 1])
# square root of not gate applied to q0, q1, q2
circ.v([0, 1, 2])
# conjugate transpose of square root of not gate applied to q0, q1, q2
circ.vi([0, 1, 2])
# exp(-iXX theta/2)
circ.xx(0, 1, 0.15)
# exp(i(XX+YY) theta/4), where theta=0.15 in the examples below
circ.xy(0, 1, 0.15)
# exp(-iYY theta/2)
circ.yy(0, 1, 0.15)
# exp(-iZZ theta/2)
circ.zz(0, 1, 0.15)
# IonQ native gate GPi with angle 0.15 applied to q0
circ.gpi(0, 0.15)
# IonQ native gate GPi2 with angle 0.15 applied to q0
circ.gpi2(0, 0.15)
# IonQ native gate MS with angles 0.15, 0.15, 0.15 applied to q0, q1
circ.ms(0, 1, 0.15, 0.15, 0.15)
```

Además del conjunto de puertas predefinido, también puede aplicar puertas unitarias autodefinidas al circuito. Pueden ser puertas de un solo qubit (como se muestra en el siguiente código fuente) o puertas de varios qubit aplicadas a los qubits definidos por el parámetro `targets`.

```
import numpy as np

# Apply a general unitary
my_unitary = np.array([[0, 1],[1, 0]])
circ.unitary(matrix=my_unitary, targets=[0])
```

Ejemplo: Ampliar los circuitos existentes

Puede ampliar los circuitos existentes añadiendo instrucciones. Una `Instruction` es una directiva cuántica que describe la tarea cuántica que se debe realizar en un dispositivo cuántico. Los operadores de la `Instruction` incluyen objetos del tipo `Gate` únicamente.

```
# Import the Gate and Instruction modules
from braket.circuits import Gate, Instruction
```

```
# Add instructions directly.
circ = Circuit([Instruction(Gate.H(), 4), Instruction(Gate.CNot(), [4, 5])])

# Or with add_instruction/add functions
instr = Instruction(Gate.CNot(), [0, 1])
circ.add_instruction(instr)
circ.add(instr)

# Specify where the circuit is appended
circ.add_instruction(instr, target=[3, 4])
circ.add_instruction(instr, target_mapping={0: 3, 1: 4})

# Print the instructions
print(circ.instructions)
# If there are multiple instructions, you can print them in a for loop
for instr in circ.instructions:
     print(instr)

# Instructions can be copied
new_instr = instr.copy()
# Appoint the instruction to target
new_instr = instr.copy(target=[5, 6])
new_instr = instr.copy(target_mapping={0: 5, 1: 6})
```

Ejemplo: Ver las puertas que admite cada dispositivo

Los simuladores admiten todas las puertas del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puede encontrar las puertas compatibles de un dispositivo en las propiedades del dispositivo. A continuación se muestra un ejemplo con un dispositivo IonQ:

```
# Import the device module
from braket.aws import AwsDevice

device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1")

# Get device name
device_name = device.name
# Show supportedQuantumOperations (supported gates for a device)
device_operations = device.properties.dict()['action']['braket.ir.openqasm.program']
['supportedOperations']
print('Quantum Gates supported by {}:\n {}'.format(device_name, device_operations))
```

```
Quantum Gates supported by Aria 1:
```

```
 ['x', 'y', 'z', 'h', 's', 'si', 't', 'ti', 'v', 'vi', 'rx', 'ry', 'rz', 'cnot',
 'swap', 'xx', 'yy', 'zz']
```

Es posible que las puertas compatibles deban compilarse en puertas nativas antes de que puedan ejecutarse en hardware cuántico. Cuando envía un circuito, Amazon Braket realiza esta compilación automáticamente.

Ejemplo: Recuperar mediante programación la fidelidad de las puertas nativas compatibles con un dispositivo

Puede ver la información de fidelidad en la página Dispositivos de la consola de Braket. A veces resulta útil acceder a la misma información de forma programática. El siguiente código muestra cómo extraer la fidelidad de las dos puertas de qubit de una QPU.

```
# Import the device module
from braket.aws import AwsDevice

device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

# Specify the qubits
a=10
b=11
edge_properties_entry =
 device.properties.standardized.twoQubitProperties['10-11'].twoQubitGateFidelity
gate_name = edge_properties_entry[0].gateName
fidelity = edge_properties_entry[0].fidelity
print(f"Fidelity of the {gate_name} gate between qubits {a} and {b}: {fidelity}")
```

Conjuntos de programas

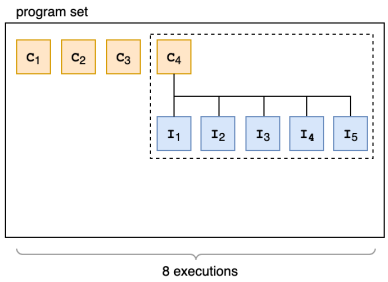
Los conjuntos de programas ejecutan de manera eficiente varios circuitos cuánticos en una sola tarea cuántica. En esa tarea, puede enviar hasta 100 circuitos cuánticos o un solo circuito paramétrico con hasta 100 conjuntos de parámetros diferentes. Esta operación minimiza el tiempo entre las ejecuciones subsiguientes de los circuitos y reduce la sobrecarga de procesamiento de las tareas cuánticas. Actualmente, los conjuntos de programas son compatibles con Amazon Braket Local Simulator y otros AQT dispositivosIQM. Rigetti

Definir un ProgramSet

El siguiente primer ejemplo de código muestra cómo desarrollar un `ProgramSet` utilizando circuitos parametrizados y circuitos sin parámetros.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit, FreeParameter
from braket.program_sets.circuit_binding import CircuitBinding
from braket.program_sets import ProgramSet

# Initialize the quantum device
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")

# Define circuits
circ1 = Circuit().h(0).cnot(0, 1)
circ2 = Circuit().rx(0, 0.785).ry(1, 0.393).cnot(1, 0)
circ3 = Circuit().t(0).t(1).cz(0, 1).s(0).cz(1, 2).s(1).s(2)
parameterize_circuit = Circuit().rx(0, FreeParameter("alpha")).cnot(0, 1).ry(1,
 FreeParameter("beta"))

# Create circuit bindings with different parameters
circuit_binding = CircuitBinding(
    circuit=parameterize_circuit,
    input_sets={
            'alpha': (0.10, 0.11, 0.22, 0.34, 0.45),
            'beta': (1.01, 1.01, 1.03, 1.04, 1.04),
    })

# Creating the program set
program_set_1 = ProgramSet([
    circ1,
    circ2,
    circ3,
    circuit_binding,
])
```

Este conjunto de programas contiene cuatro programas únicos: `circ1`, `circ2`, `circ3` y `circuit_binding`. El programa `circuit_binding` se ejecuta con cinco vínculos de parámetros diferentes, lo que crea cinco ejecutables. Los otros tres programas sin parámetros crean un ejecutable cada uno. Esto da como resultado un total de ocho ejecutables, como se muestra en la imagen siguiente.



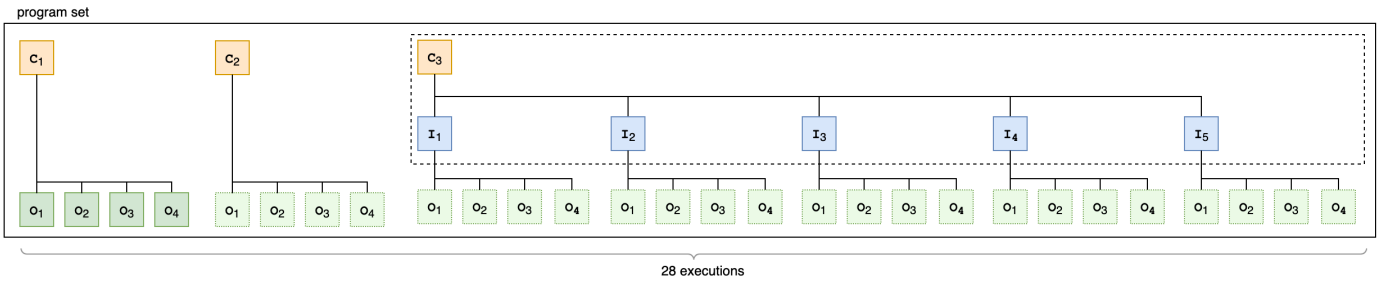

```
ProgramSet(
    [c1,
     c2,
     c3,
     CircuitBinding(c4, [i1, i2, i3, i4, i5])]
)
```

El siguiente segundo ejemplo de código muestra cómo utilizar el método `product()` para adjuntar el mismo conjunto de observables a cada ejecutable del conjunto de programas.

```
from braket.circuits.observables import I, X, Y, Z

observables = [Z(0) @ Z(1), X(0) @ X(1), Z(0) @ X(1), X(0) @ Z(1)]

program_set_2 = ProgramSet.product(
    circuits=[circ1, circ2, circuit_binding],
    observables=observables
)
```

En el caso de los programas sin parámetros, cada observable se mide para cada circuito. En el caso de los programas paramétricos, cada observable se mide para cada conjunto de entradas, como se muestra en la siguiente imagen.

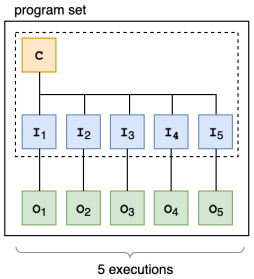
```
ProgramSet.product(
    circuits=[c1, c2, CircuitBinding(c3, [i1, i2, i3, i4, i5])],
    observables=[o1, o2, o3, o4]
)
```



En el tercer ejemplo de código siguiente se muestra cómo utilizar el método `zip()` para emparejar observables individuales con conjuntos de parámetros específicos en el `ProgramSet`.

```
program_set_3 = ProgramSet.zip(
    circuits=circuit_binding,
```

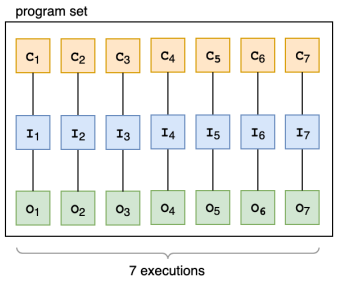
```
    observables=observables + [Y(0) @ Y(1)]
)
```




```
ProgramSet.zip(
    circuits=CicruitBinding(
        circuit=c,
        input_sets=[i1, i2, i3, i4, i5]
    ),
    observables=[o1, o2, o3, o4, o5]
)
```

En lugar de `CircuitBinding()`, puede comprimir directamente una lista de observables con una lista de circuitos y conjuntos de entradas.

```
program_set_4 = ProgramSet.zip(
    circuits=[circ1, circ2, circ3],
    input_sets=[{}, {}, {}],
    observables=observables[:3]
)
```




```
ProgramSet.zip(
    circuits=[c1, c2, c3, c4, c5, c6, c7],
    input_sets=[i1, i2, i3, i4, i5, i6, i7],
    observables=[o1, o2, o3, o4, o5, o6, o7]
)
```

Para obtener más información y ejemplos de conjuntos de programas, consulta la carpeta de conjuntos de programas en Github. amazon-braket-examples

Cómo inspeccionar y ejecutar un conjunto de programas en un dispositivo

El número de ejecutables de un conjunto de programas es igual al número de circuitos vinculados a parámetros únicos. Calcule el número total de shots y ejecutables de circuitos mediante el siguiente ejemplo de código.

```
# Number of shots per executable
```

```
shots = 10
num_executables = program_set_1.total_executables

# Calculate total number of shots across all executables
total_num_shots = shots*num_executables
```

> **Note**
>
> Con los conjuntos de programas, se paga una tarifa única por tarea y una tarifa por shot basada en el número total de shots en todos los circuitos de un conjunto de programas.

Para ejecutar el conjunto de programas, use el siguiente ejemplo de código.

```
# Run the program set
task = device.run(
   program_set_1, shots=total_num_shots,
)
```

Al utilizar dispositivos Rigetti, su conjunto de programas puede permanecer en el estado `RUNNING` mientras las tareas estén parcialmente terminadas y parcialmente en cola. Para obtener resultados más rápidos, considere enviar su conjunto de programas como Trabajo híbrido.

Analizando los resultados

Ejecute el siguiente código para analizar y medir los resultados de los ejecutables en un `ProgramSet`.

```
# Get the results from a program set
result = task.result()

# Get the first executbable
first_program = result[0]
first_executable = first_program[0]

# Inspect the results of the first executable
measurements_from_first_executable = first_executable.measurements
print(measurements_from_first_executable)
```

Medición parcial

En lugar de medir todos los qubits de un circuito cuántico, utilice la medición parcial para medir qubits individuales o un subconjunto de qubits.

> **Note**
>
> Las características adicionales, como la medición del circuito medio y las operaciones de prealimentación, están disponibles como capacidades experimentales. Consulte Acceso a circuitos dinámicos en dispositivos IQM.

Ejemplo: Medir un subconjunto de qubits

El siguiente ejemplo de código muestra una medición parcial midiendo solo qubit 0 en un circuito de estado de Bell.

```
from braket.devices import LocalSimulator
from braket.circuits import Circuit

# Use the local state vector simulator
device = LocalSimulator()

# Define an example bell circuit and measure qubit 0
circuit = Circuit().h(0).cnot(0, 1).measure(0)

# Run the circuit
task = device.run(circuit, shots=10)

# Get the results
result = task.result()

# Print the circuit and measured qubits
print(circuit)
print()
print("Measured qubits: ", result.measured_qubits)
```

Asignación manual de qubit

Cuando ejecuta un circuito cuántico en computadoras cuánticas desde Rigetti, puede utilizar opcionalmente la asignación manual de qubit para controlar qué qubits se utilizan para su algoritmo.

La consola de Amazon Braket y el SDK de Amazon Braket le ayudan a inspeccionar los datos de calibración más recientes de la unidad de procesamiento cuántico (QPU) que haya seleccionado, de modo que pueda seleccionar los mejores qubits para su experimento.

La asignación manual de qubit le permite ejecutar los circuitos con mayor precisión e investigar las propiedades de qubit individuales. Los investigadores y usuarios avanzados optimizan el diseño de sus circuitos basándose en los últimos datos de calibración de dispositivos y pueden obtener resultados más precisos.

En el siguiente ejemplo se muestra cómo asignar qubits de forma explícita.

```
# Import the device module
from braket.aws import AwsDevice

device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
circ = Circuit().h(0).cnot(0, 7)  # Indices of actual qubits in the QPU

# Set up S3 bucket (where results are stored)
my_bucket = "amazon-braket-s3-demo-bucket" # The name of the bucket
my_prefix = "your-folder-name" # The name of the folder in the bucket
s3_location = (my_bucket, my_prefix)

my_task = device.run(circ, s3_location, shots=100, disable_qubit_rewiring=True)
```

Para obtener más información, consulte los ejemplos de Amazon Braket sobre GitHub, o más específicamente, este cuaderno: Asignación de qubits en dispositivos QPU.

Compilación verbatim

Cuando ejecuta un circuito cuántico en computadoras cuánticas basadas en puertas, puede indicar al compilador que ejecute sus circuitos exactamente como se ha definido sin ninguna modificación. Mediante la compilación verbatim, puede especificar que un circuito completo se conserve exactamente como se ha especificado o que solo se conserven partes específicas del mismo (solo compatible con Rigetti). Al desarrollar algoritmos para la evaluación comparativa de hardware o protocolos de mitigación de errores, es necesario tener la opción de especificar con exactitud las puertas y los diseños de circuitos que se ejecutan en el hardware. La compilación verbatim le permite controlar directamente el proceso de compilación desactivando determinados pasos de optimización, lo que garantiza que sus circuitos funcionen exactamente según lo diseñado.

La compilación literal es compatible con los Rigetti dispositivosAQT, IonQIQM, y requiere el uso de puertas nativas. Cuando utilice la compilación verbatim, es recomendable comprobar la topología del

dispositivo para asegurarse de que las puertas se llaman sobre qubits conectados y que el circuito utiliza las puertas nativas compatibles con el hardware. En el siguiente ejemplo se muestra cómo acceder mediante programación a la lista de puertas nativas admitidas por un dispositivo.

```
device.properties.paradigm.nativeGateSet
```

Para Rigetti, el recableado de qubit debe desactivarse configurando `disableQubitRewiring=True` para el uso con la compilación verbatim. Si `disableQubitRewiring=False` se establece cuando se utilizan cuadros verbatim en una compilación, el circuito cuántico no pasa la validación y no se ejecuta.

Si la compilación verbatim se habilita para un circuito y se ejecuta en una QPU que no la admite, se genera un error que indica que una operación no compatible ha provocado el fallo de la tarea. A medida que más hardware cuántico admita de forma nativa las funciones del compilador, se ampliará esta característica para incluir estos dispositivos. Los dispositivos que admiten la compilación verbatim la incluyen como operación compatible cuando se consulta con el siguiente código.

```
from braket.aws import AwsDevice
from braket.device_schema.device_action_properties import DeviceActionType
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
device.properties.action[DeviceActionType.OPENQASM].supportedPragmas
```

No hay ningún costo adicional asociado al uso de la compilación verbatim. Se le seguirán cobrando las tareas cuánticas ejecutadas en dispositivos QPU de Braket, instancias de cuaderno y simuladores bajo demanda según las tarifas actuales especificadas en la página de precios de Amazon Braket. Para obtener más información, consulte el cuaderno de ejemplos de compilación verbatim.

> **Note**
>
> Si utilizas OpenQASM para escribir los circuitos de los IonQ dispositivos AQT y, además, deseas mapear tu circuito directamente a los cúbits físicos, tendrás que usar el parámetro, `#pragma braket verbatim` ya que OpenQasm ignora el `disableQubitRewiring` indicador.

Simulación de ruido

Para crear una instancia de simulador de ruido local, puede cambiar el backend de la siguiente manera.

```
# Import the device module
from braket.aws import AwsDevice

device = LocalSimulator(backend="braket_dm")
```

Puede desarrollar circuitos ruidosos de dos maneras:

1. Desarrolle el circuito ruidoso de abajo hacia arriba.
2. Tome un circuito existente sin ruido e inyecte ruido en él.

En el siguiente ejemplo se muestran los enfoques utilizando un circuito básico con ruido despolarizante y un canal Kraus personalizado.

```
import scipy.stats
import numpy as np

# Bottom up approach
# Apply depolarizing noise to qubit 0 with probability of 0.1
circ = Circuit().x(0).x(1).depolarizing(0, probability=0.1)

# Create an arbitrary 2-qubit Kraus channel
E0 = scipy.stats.unitary_group.rvs(4) * np.sqrt(0.8)
E1 = scipy.stats.unitary_group.rvs(4) * np.sqrt(0.2)
K = [E0, E1]

# Apply a two-qubit Kraus channel to qubits 0 and 2
circ = circ.kraus([0, 2], K)
```

```
from braket.circuits import Noise

# Inject noise approach
# Define phase damping noise
noise = Noise.PhaseDamping(gamma=0.1)
# The noise channel is applied to all the X gates in the circuit
circ = Circuit().x(0).y(1).cnot(0, 2).x(1).z(2)
circ_noise = circ.copy()
```

```
circ_noise.apply_gate_noise(noise, target_gates=Gate.X)
```

Ejecutar un circuito ofrece la misma experiencia de usuario que antes, como se muestra en los dos ejemplos siguientes.

Ejemplo 1

```
task = device.run(circ, shots=100)
```

O

Ejemplo 2

```
task = device.run(circ_noise, shots=100)
```

Para ver más ejemplos, consulte el ejemplo del simulador de ruido introductorio de Braket.

Inspección del circuito

Los circuitos cuánticos de Amazon Braket tienen un concepto de pseudotiempo denominado `Moments`. Cada qubit puede experimentar una sola puerta por `Moment`. El objetivo de `Moments` es facilitar el direccionamiento de los circuitos y sus puertas y proporcionar una estructura temporal.

> **Note**
>
> Por lo general, los Momentos no se corresponden con el tiempo real en el que se ejecutan las puertas en una QPU.

La profundidad de un circuito viene dada por el número total de Momentos en ese circuito. Puede ver la profundidad del circuito al llamar al método `circuit.depth`, como se muestra en el siguiente ejemplo.

```
from braket.circuits import Circuit

# Define a circuit with parametrized gates
circ = Circuit().rx(0, 0.15).ry(1, 0.2).cnot(0, 2).zz(1, 3, 0.15).x(0)
print(circ)
print('Total circuit depth:', circ.depth)
```

```
T  : #     0      #        1         #  2  #
      ############                    #####
q0 : ## Rx(0.15) ###################### X ##
      ############   #                #####
      ############   #   ############
q1 : ## Ry(0.20) ######### ZZ(0.15) ########
      ############   #   ############
                  #####      #
q2 : ############### X ######################
                  #####      #
                       ############
q3 : ###################### ZZ(0.15) ########
                       ############
T  : #     0      #        1         #  2  #
Total circuit depth: 3
```

La profundidad de circuito total del circuito anterior es 3 (mostrados como Momentos `0`, 1 y2). Puede comprobar la operación de la puerta en cada Momento.

`Moments` funciona como un diccionario de pares de clave-valor.

- La clave es `MomentsKey()`, la cual contiene pseudotiempo e información de qubit.
- El valor se asigna en el tipo de `Instructions()`.

```
moments = circ.moments
for key, value in moments.items():
    print(key)
    print(value, "\n")
```

```
MomentsKey(time=0, qubits=QubitSet([Qubit(0)]), moment_type=<MomentType.GATE: 'gate'>,
 noise_index=0, subindex=0)
Instruction('operator': Rx('angle': 0.15, 'qubit_count': 1), 'target':
 QubitSet([Qubit(0)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=0, qubits=QubitSet([Qubit(1)]), moment_type=<MomentType.GATE: 'gate'>,
 noise_index=0, subindex=0)
Instruction('operator': Ry('angle': 0.2, 'qubit_count': 1), 'target':
 QubitSet([Qubit(1)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=1, qubits=QubitSet([Qubit(0), Qubit(2)]), moment_type=<MomentType.GATE:
 'gate'>, noise_index=0, subindex=0)
```

```
Instruction('operator': CNot('qubit_count': 2), 'target': QubitSet([Qubit(0),
 Qubit(2)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=1, qubits=QubitSet([Qubit(1), Qubit(3)]), moment_type=<MomentType.GATE:
 'gate'>, noise_index=0, subindex=0)
Instruction('operator': ZZ('angle': 0.15, 'qubit_count': 2), 'target':
 QubitSet([Qubit(1), Qubit(3)]), 'control': QubitSet([]), 'control_state': (), 'power':
 1)

MomentsKey(time=2, qubits=QubitSet([Qubit(0)]), moment_type=<MomentType.GATE: 'gate'>,
 noise_index=0, subindex=0)
Instruction('operator': X('qubit_count': 1), 'target': QubitSet([Qubit(0)]), 'control':
 QubitSet([]), 'control_state': (), 'power': 1)
```

También puede añadir puertas a un circuito que a través de `Moments`.

```
from braket.circuits import Instruction, Gate

new_circ = Circuit()
instructions = [Instruction(Gate.S(), 0),
                Instruction(Gate.CZ(), [1, 0]),
                Instruction(Gate.H(), 1)
                ]

new_circ.moments.add(instructions)
print(new_circ)
```

```
T  : #  0  #  1  #  2  #
      ##### #####
q0 : ## S ### Z ########
      ##### #####
               #   #####
q1 : ############## H ##
                   #####
T  : #  0  #  1  #  2  #
```

Lista de tipos de resultados

Amazon Braket puede devolver diferentes tipos de resultados cuando se mide un circuito utilizando `ResultType`. Un circuito puede devolver los siguientes tipos de resultados.

- `AdjointGradient` devuelve el gradiente (derivado de vector) del valor esperado de un observable proporcionado. Este observable actúa sobre un objetivo determinado con respecto a parámetros específicos utilizando el método de diferenciación adjunta. Solo puede usar este método cuando shots = 0.
- `Amplitude` devuelve la amplitud de los estados cuánticos especificados en la función de onda de salida. Solo está disponible en los simuladores SV1 y locales.
- `Expectation` devuelve el valor esperado de un observable dado, que se puede especificar con la clase `Observable`, que se presenta más adelante en este capítulo. Para medir el observable, se debe especificar el objetivo qubits utilizado y el número de objetivos especificados debe ser igual al número de qubits sobre los que actúa el observable. Si no se especifican objetivos, el observable debe operar solo en 1 qubit y se aplica a todos los qubits en paralelo.
- `Probability` devuelve las probabilidades de medir estados básicos computacionales. Si no se especifica ningún objetivo, `Probability` devuelve la probabilidad de medir todos los estados base. Si se especifican los objetivos, solo se devuelven las probabilidades marginales de los vectores base en los qubits especificados. Los simuladores gestionados QPUs están limitados a un máximo de 15 qubits y los simuladores locales están limitados al tamaño de la memoria del sistema.
- `Reduced density matrix` devuelve una matriz de densidad para un subsistema de los qubits objetivo especificados de un sistema de qubits. Para limitar el tamaño de este tipo de resultado, Braket limita el número de qubits objetivo a un máximo de 8.
- `StateVector` devuelve el vector de estado completo. Está disponible en el simulador local.
- `Sample` devuelve los recuentos de mediciones de un conjunto de qubit objetivo especificado y observable. Si no se especifican objetivos, el observable debe operar solo en 1 qubit y se aplica a todos los qubits en paralelo. Si se especifican objetivos, el número de objetivos especificados debe ser igual al número de qubits sobre el que actúa el observable.
- `Variance` devuelve la varianza (`mean([x-mean(x)]`^{2}`)`) del conjunto de qubit objetivo especificado y el observable como el tipo de resultado solicitado. Si no se especifican objetivos, el observable debe operar solo en 1 qubit y se aplica a todos los qubits en paralelo. De lo contrario, el número de objetivos especificados debe ser igual al número de qubits objetivo a los que se puede aplicar el observable.

Los tipos de resultados admitidos para los distintos proveedores son los siguientes:

	Sim local	SV1	DM1	TN1	AQT	IonQ	IQM	Rigetti
Gradient adjunto	N	Y	N	N	N	N	N	N
Amplitud	Y	Y	N	N	N	N	N	N
Expectativa	Y	Y	Y	Y	Y	Y	Y	Y
Probability	Y	Y	Y	N	Y	Y	Y	Y
Matriz de densidad reducida	Y	N	Y	N	N	N	N	N
Vector de estado	Y	N	N	N	N	N	N	N
Muestra	Y	Y	Y	Y	Y	Y	Y	Y
Varianza	Y	Y	Y	Y	Y	Y	Y	Y

Puede comprobar los tipos de resultados compatibles examinando las propiedades del dispositivo, como se muestra en el siguiente ejemplo.

```
from braket.aws import AwsDevice

device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

# Print the result types supported by this device
for iter in
 device.properties.action['braket.ir.openqasm.program'].supportedResultTypes:
    print(iter)
```

```
name='Sample' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Expectation' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Variance' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Probability' observables=None minShots=10 maxShots=50000
```

Para llamar a `ResultType`, añádalo a un circuito, como se muestra en el siguiente ejemplo.

```
from braket.circuits import Circuit, Observable

circ = Circuit().h(0).cnot(0, 1).amplitude(state=["01", "10"])
circ.probability(target=[0, 1])
circ.probability(target=0)
circ.expectation(observable=Observable.Z(), target=0)
circ.sample(observable=Observable.X(), target=0)
circ.state_vector()
circ.variance(observable=Observable.Z(), target=0)

# Print one of the result types assigned to the circuit
print(circ.result_types[0])
```

> **Note**
>
> Los diferentes dispositivos cuánticos proporcionan resultados en varios formatos. Por ejemplo, los dispositivos Rigetti devuelven mediciones, mientras que los dispositivos IonQ ofrecen probabilidades. El SDK de Amazon Braket ofrece una propiedad de medición para todos los resultados. Sin embargo, en el caso de los dispositivos que devuelven probabilidades, estas mediciones se calculan posteriormente y se basan en las probabilidades, ya que las mediciones por shot no están disponibles. Para determinar si un resultado se ha computado posteriormente, compruebe las `measurements_copied_from_device` en el objeto de resultado. Esta operación se detalla en el archivo gate_model_quantum_task_result.py del repositorio del SDK GitHub de Amazon Braket.

Observables

La clase `Observable` de Amazon Braket le permite medir un observable específico.

Solo puede aplicar un único observable no identitario a cada qubit. Se produce un error si se especifican dos o más observables no identitarios diferentes para el mismo qubit. Para este

propósito, cada factor de un producto tensorial cuenta como un observable individual. Esto significa que puede tener varios productos tensoriales en el mismo qubit, siempre y cuando los factores que actúen en el qubit sigan siendo los mismos.

Es posible escalar un observable y añadir otros observables (escalados o no). Esto crea una `Sum` que se puede usar en el tipo de resultado `AdjointGradient`.

La clase `Observable` incluye los siguientes observables.

```
import numpy as np

Observable.I()
Observable.H()
Observable.X()
Observable.Y()
Observable.Z()

# Get the eigenvalues of the observable
print("Eigenvalue:", Observable.H().eigenvalues)
# Or rotate the basis to be computational basis
print("Basis rotation gates:", Observable.H().basis_rotation_gates)

# Get the tensor product of the observable for the multi-qubit case
tensor_product = Observable.Y() @ Observable.Z()
# View the matrix form of an observable by using
print("The matrix form of the observable:\n", Observable.Z().to_matrix())
print("The matrix form of the tensor product:\n", tensor_product.to_matrix())

# Factorize an observable in the tensor form
print("Factorize an observable:", tensor_product.factors)

# Self-define observables, given it is a Hermitian
print("Self-defined Hermitian:", Observable.Hermitian(matrix=np.array([[0, 1], [1,
 0]])))

print("Sum of other (scaled) observables:", 2.0 * Observable.X() @ Observable.X() + 4.0
 * Observable.Z() @ Observable.Z())
```

```
Eigenvalue: [ 1. -1.]
Basis rotation gates: (Ry('angle': -0.7853981633974483, 'qubit_count': 1),)
The matrix form of the observable:
 [[ 1.+0.j  0.+0.j]
```

```
 [ 0.+0.j -1.+0.j]]
The matrix form of the tensor product:
 [[ 0.+0.j  0.+0.j  0.-1.j  0.+0.j]
 [ 0.+0.j -0.+0.j  0.+0.j  0.+1.j]
 [ 0.+1.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.-1.j  0.+0.j -0.+0.j]]
Factorize an observable: (Y('qubit_count': 1), Z('qubit_count': 1))
Self-defined Hermitian: Hermitian('qubit_count': 1, 'matrix': [[0.+0.j 1.+0.j], [1.+0.j
 0.+0.j]])
Sum of other (scaled) observables: Sum(TensorProduct(X('qubit_count': 1),
 X('qubit_count': 1)), TensorProduct(Z('qubit_count': 1), Z('qubit_count': 1)))
```

Parameters

Los circuitos pueden incorporar parámetros libres. Estos parámetros libres solo deben construirse una vez para ejecutarse varias veces y pueden utilizarse para calcular gradientes.

Cada parámetro libre utiliza un nombre codificado en cadena que se utiliza para lo siguiente:

- Establecer valores de parámetros
- Identificar qué parámetros utilizar

```
from braket.circuits import Circuit, FreeParameter, observables
from braket.parametric import FreeParameter

theta = FreeParameter("theta")
phi = FreeParameter("phi")
circ = Circuit().h(0).rx(0, phi).ry(0, phi).cnot(0, 1).xx(0, 1, theta)
```

Gradiente adjunto

El dispositivo SV1 calcula el gradiente adjunto de un valor esperado observable, incluido el hamiltoniano multitérmino. Para diferenciar los parámetros, especifique su nombre (en formato de cadena) o por referencia directa.

```
from braket.aws import AwsDevice
from braket.devices import Devices

device = AwsDevice(Devices.Amazon.SV1)
```

```
circ.adjoint_gradient(observable=3 * Observable.Z(0) @ Observable.Z(1) - 0.5 *
 observables.X(0), parameters = ["phi", theta])
```

Al transferir valores de parámetros fijos como argumentos a un circuito parametrizado, se eliminarán los parámetros libres. La ejecución de este circuito con `AdjointGradient` produce un error, ya que los parámetros libres ya no existen. El siguiente ejemplo de código demuestra el uso correcto e incorrecto:

```
# Will error, as no free parameters will be present
#device.run(circ(0.2), shots=0)

# Will succeed
device.run(circ, shots=0, inputs={'phi': 0.2, 'theta': 0.2})
```

Asesoramiento de expertos

Póngase en contacto con expertos en computación cuántica directamente en la consola de administración de Braket para obtener orientación adicional sobre sus cargas de trabajo.

Para explorar las opciones de asesoramiento de expertos a través de Braket Direct, abra la consola de Braket, seleccione Braket Direct en el panel izquierdo y vaya a la sección Asesoramiento de expertos. Hay disponibles las siguientes opciones de asesoramiento de expertos:

- Horario de oficina de Braket: el horario de oficina de Braket consiste en sesiones individuales, por orden de llegada, y se lleva a cabo todos los meses. Cada horario de oficina disponible es de 30 minutos y es gratuito. Hablar con los expertos de Braket puede ayudarle a pasar más rápido de la idea a la ejecución, ya que explora use-case-to-device las opciones para utilizar mejor Braket para su algoritmo y recibe recomendaciones sobre cómo utilizar determinadas funciones de Braket, como Amazon Braket Hybrid Jobs, Braket Pulse o Analog Hamiltonian Simulation.
  - Para registrarse en el horario de oficina de Braket, seleccione Registrarse y rellene la información de contacto, los detalles de la carga de trabajo y los temas que desea tratar.
  - Recibirá una invitación de calendario para la próxima franja horaria disponible a través de su correo electrónico.

> **Note**
>
> Para cuestiones urgentes o preguntas rápidas sobre solución de problemas, recomendamos ponerse en contacto con el servicio técnico de AWS. Para preguntas

que no sean urgentes, también puede utilizar el foro re:Post de AWS o el sitio Quantum Computing Stack Exchange, donde puede consultar preguntas ya respondidas y formular otras nuevas.

- Ofertas de proveedores de hardware cuántico: IonQ, QuEra, y Rigetti proporcionan servicios profesionales a través de AWS Marketplace.
  - Para explorar sus ofertas, seleccione Ponerse en contacto y explore sus listados.
  - Para obtener más información sobre las ofertas de servicios profesionales disponibles en, consulte los productos de servicios profesionales. AWS Marketplace
- Laboratorio de soluciones cuánticas de Amazon (QSL): QSL es un equipo de investigación colaborativa y servicios profesionales formado por expertos en computación cuántica que pueden ayudarle a explorar eficazmente la computación cuántica y evaluar el rendimiento actual de esta tecnología.
  - Para contactar con QSL, seleccione Ponerse en contacto y complete la información de contacto y los detalles del caso de uso.
  - El equipo de QSL se pondrá en contacto con usted por correo electrónico para explicarle los siguientes pasos.

Ejecución de circuitos con OpenQASM 3.0

Amazon Braket es ahora compatible con OpenQASM 3.0 para dispositivos y simuladores cuánticos basados en puertas. En esta guía del usuario se proporciona información sobre el subconjunto de OpenQASM 3.0 compatible con Braket. Ahora, los clientes de Braket tienen la opción de enviar circuitos de Braket con el SDK o proporcionando directamente cadenas de OpenQASM 3.0 a todos los dispositivos basados en puertas con la API de Amazon Braket y el SDK de Python de Amazon Braket.

Los temas de esta guía le muestran varios ejemplos de cómo completar las siguientes tareas cuánticas.

- Creación y envío de tareas cuánticas de OpenQASM en diferentes dispositivos de Braket
- Acceso a operaciones compatibles y tipos de resultados
- Simulación de ruido con OpenQASM
- Uso de compilación verbatim con OpenQASM
- Solución de problemas de OpenQASM

Esta guía también proporciona una introducción a determinadas características específicas del hardware que se pueden implementar con OpenQASM 3.0 en Braket y enlaces a otros recursos.

En esta sección:

¿Qué es OpenQASM 3.0?

OpenQASM (Open Quantum Assembly Language) es una representación intermedia para instrucciones cuánticas. OpenQASM es un marco de código abierto muy utilizado para la especificación de programas cuánticos para dispositivos basados en puertas. Con OpenQASM, los usuarios pueden programar las puertas cuánticas y las operaciones de medición que constituyen los componentes básicos de la computación cuántica. La versión anterior de OpenQASM (2.0) fue utilizada por varias bibliotecas de programación cuántica para describir programas básicos.

La nueva versión de OpenQASM (3.0) amplía la versión anterior para incluir más características, como el control a nivel de pulso, la temporización de las puertas y el flujo de control clásico, para cerrar la brecha entre la interfaz de usuario final y el lenguaje de descripción del hardware. Los detalles y las especificaciones de la versión 3.0 actual están disponibles en la especificación GitHub OpenQASM 3.x Live. El futuro desarrollo de OpenQASM está gobernado por el Comité Directivo

Técnico de OpenQASM 3.0, del que AWS forma parte junto con IBM, Microsoft y la Universidad de Innsbruck.

¿Cuándo se debe usar OpenQASM 3.0?

OpenQASM proporciona un marco expresivo para especificar programas cuánticos a través de controles de bajo nivel que no son específicos de la arquitectura, lo que lo hace muy adecuado como representación en varios dispositivos basados en puertas. La compatibilidad de Braket con OpenQASM fomenta su adopción como un enfoque coherente para desarrollar algoritmos cuánticos basados en puertas, lo que reduce la necesidad de que los usuarios aprendan y mantengan bibliotecas en varios marcos.

Si dispone de bibliotecas de programas en OpenQASM 3.0, puede adaptarlas para utilizarlas con Braket en lugar de reescribir completamente estos circuitos. Los investigadores y desarrolladores también deberían beneficiarse del creciente número de bibliotecas de terceros disponibles que admiten el desarrollo de algoritmos en OpenQASM.

Cómo funciona OpenQASM 3.0

La compatibilidad con OpenQASM 3.0 de Braket proporciona paridad de características con la representación intermedia actual. Esto significa que todo lo que puede hacer hoy en día en dispositivos de hardware y simuladores bajo demanda con Braket, lo puede hacer con OpenQASM utilizando la API de Braket. Puede ejecutar programas de OpenQASM 3.0 suministrando directamente cadenas de OpenQASM a todos los dispositivos basados en puertas, de forma similar a como se suministran actualmente los circuitos a los dispositivos en Braket. Los usuarios de Braket también pueden integrar bibliotecas de terceros compatibles con OpenQASM 3.0. En el resto de esta guía se detalla cómo desarrollar representaciones de OpenQASM para usarlas con Braket.

Requisitos previos

Para utilizar OpenQASM 3.0 en Amazon Braket, debe tener la versión v1.8.0 de los esquemas de Python de Amazon Braket y la versión v1.17.0 o superior del SDK de Python de Amazon Braket.

Si es la primera vez que utiliza Amazon Braket, tiene que activar Amazon Braket. Para obtener instrucciones, consulte Activación de Amazon Braket.

¿Qué características de OpenQASM admite Braket?

En la siguiente sección se enumeran los tipos de datos, las declaraciones y las instrucciones pragmáticas de OpenQASM 3.0 compatibles con Braket.

En esta sección:

Tipos de datos de OpenQASM admitidos

Amazon Braket admite lo siguientes tipos de datos de OpenQASM.

- Los números enteros no negativos se utilizan para los índices de qubits (virtuales y físicos):
  - `cnot q[0], q[1];`
  - `h $0;`
- Se pueden usar números o constantes de punto flotante para los ángulos de rotación de la puerta:
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi es una constante integrada en OpenQASM y no se puede utilizar como nombre de parámetro.

- Se admiten matrices de números complejos (con la notación `im` de OpenQASM para la parte imaginaria) en los pragmas de tipo resultado para definir observables hermitianos generales y en los pragmas unitarios:
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

Declaraciones de OpenQASM admitidas

Amazon Braket admite las siguientes declaraciones de OpenQASM.

- `Header: OPENQASM 3;`
- Declaraciones de bit clásicas:
  - `bit b1;` (de forma equivalente, `creg b1;`)
  - `bit[10] b2;` (de forma equivalente, `creg b2[10];`)
- Declaraciones de Qubit:
  - `qubit b1;` (de forma equivalente, `qreg b1;`)
  - `qubit[10] b2;` (de forma equivalente, `qreg b2[10];`)
- Indexación en matrices: `q[0]`
- Entrada: `input float alpha;`
- especificación física qubits: `$0`
- Puertas y operaciones admitidas en un dispositivo:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> Las puertas admitidas en un dispositivo se encuentran en las propiedades del dispositivo para las acciones de OpenQASM; no se necesitan definiciones de puerta para utilizar esas puertas.

- Declaraciones de cuadro verbatim Actualmente, no admitimos la notación de duración de cuadro. Las puertas nativas y los qubits físicos son obligatorios en los cuadros verbatim.

```
#pragma braket verbatim
box{
    rx(0.314) $0;
}
```

- Medición y asignación de mediciones en qubits o en un registro completo de qubit.
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`

  - `b = measure q;`
  - `measure q # b;`
- Las instrucciones de barrera proporcionan un control explícito sobre la compilación y ejecución de los circuitos al impedir que las puertas se reordenen y optimicen más allá de los límites de las barreras. También imponen un orden temporal estricto durante la ejecución, lo que garantiza que todas las operaciones ante una barrera se completen antes de que comiencen las siguientes.
  - `barrier;`
  - `barrier q[0], q[1];`
  - `barrier $3, $6;`

Pragmas de Braket OpenQASM

Amazon Braket admite las siguientes instrucciones de pragma OpenQASM.

- Pragmas de ruido
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- Pragmas verbatim
  - `#pragma braket verbatim`
- Pragmas de tipo de resultado
  - Tipos de resultados invariantes de base:
    - Vector de estado: `#pragma braket result state_vector`
    - Matriz de densidad: `#pragma braket result density_matrix`
  - Pragmas de cómputo de gradiente:
    - Gradiente adjunto: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Tipos de resultados de base Z:
    - Amplitud: `#pragma braket result amplitude "01"`
    - Probabilidad: `#pragma braket result probability q[0], q[1]`
  - Tipos de resultados rotados de base
    - Expectativa: `#pragma braket result expectation x(q[0]) @ y([q1])`

- Varianza: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
- Ejemplo: `#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0 es compatible con OpenQASM 2.0, por lo que los programas escritos con la versión 2.0 pueden ejecutarse en Braket. Sin embargo, las características de OpenQASM 3.0 compatibles con Braket presentan algunas pequeñas diferencias de sintaxis, como `qreg` frente a `creg` y `qubit` frente a `bit`. También hay diferencias en la sintaxis de medición y es necesario admitirlas con su sintaxis correcta.

Compatibilidad con características avanzadas para OpenQASM en el simulador local

El `LocalSimulator` admite características avanzadas de OpenQASM que no se ofrecen como parte de las QPU de Braket ni de los simuladores bajo demanda. La siguiente lista de características solo se admite en el `LocalSimulator`:

- Modificadores de puertas
- Puertas integradas de OpenQASM
- Variables clásicas
- Operaciones clásicas
- Puertas personalizadas
- Control clásico
- Archivos QASM
- Subrutinas

Para ver ejemplos de cada característica avanzada, consulte este cuaderno de ejemplo. Para ver la especificación completa de OpenQASM, consulte el sitio web de OpenQASM.

Operaciones y gramática compatibles con OpenPulse

Tipos OpenPulse de datos compatibles

Bloques de cal:

```
cal {
    ...
}
```

Bloques de defcal:

```
// 1 qubit
defcal x $0 {
...
}

// 1 qubit w. input parameters as constants
defcal my_rx(pi) $0 {
...
}

// 1 qubit w. input parameters as free parameters
defcal my_rz(angle theta) $0 {
...
}

// 2 qubit (above gate args are also valid)
defcal cz $1, $0 {
...
}
```

Marcos:

```
frame my_frame = newframe(port_0, 4.5e9, 0.0);
```

Formas de onda:

```
// prebuilt
waveform my_waveform_1 = constant(1e-6, 1.0);

//arbitrary
waveform my_waveform_2 = {0.1 + 0.1im, 0.1 + 0.1im, 0.1, 0.1};
```

Ejemplo de calibración de puerta personalizada:

```
cal {
```

```
    waveform wf1 = constant(1e-6, 0.25);
}

defcal my_x $0 {
   play(wf1, q0_rf_frame);
}

defcal my_cz $1, $0 {
    barrier q0_q1_cz_frame, q0_rf_frame;
    play(q0_q1_cz_frame, wf1);
    delay[300ns] q0_rf_frame
    shift_phase(q0_rf_frame, 4.366186381749424);
    delay[300ns] q0_rf_frame;
    shift_phase(q0_rf_frame.phase, 5.916747563126659);
    barrier q0_q1_cz_frame, q0_rf_frame;
    shift_phase(q0_q1_cz_frame, 2.183093190874712);
}

bit[2] ro;
my_x $0;
my_cz $1,$0;
c[0] = measure $0;
```

Ejemplo de pulso arbitrario:

```
bit[2] ro;
cal {
    waveform wf1 = {0.1 + 0.1im, 0.1 + 0.1im, 0.1, 0.1};
    barrier q0_drive, q0_q1_cross_resonance;
    play(q0_q1_cross_resonance, wf1);
    delay[300ns] q0_drive;
    shift_phase(q0_drive, 4.366186381749424);
    delay[300dt] q0_drive;
   barrier q0_drive, q0_q1_cross_resonance;
   play(q0_q1_cross_resonance, wf1);
    ro[0] = capture_v0(r0_measure);
    ro[1] = capture_v0(r1_measure);
}
```

Creación y envío de un ejemplo de tarea cuántica de OpenQASM 3.0

Puede utilizar el SDK Amazon Braket Python, Boto3 o el para enviar tareas cuánticas de OpenQASM 3.0 AWS CLI a un dispositivo Amazon Braket.

En esta sección:

Un ejemplo de programa OpenQASM 3.0

Para crear una tarea de OpenQASM 3.0, puede empezar con un programa OpenQASM 3.0 básico (ghz.qasm) que prepare un estado de GHZ, como se muestra en el siguiente ejemplo.

```
// ghz.qasm
// Prepare a GHZ state
OPENQASM 3;

qubit[3] q;
bit[3] c;

h q[0];
cnot q[0], q[1];
cnot q[1], q[2];

c = measure q;
```

Uso del SDK de Python para crear tareas cuánticas de OpenQASM 3.0

Puede usar el SDK de Python de Amazon Braket para enviar este programa a un dispositivo de Amazon Braket con el siguiente código. Asegúrese de reemplazar la ubicación del bucket de Amazon S3 de ejemplo, «amzn-s3-demo-bucket», por el nombre del bucket de Amazon S3.

```
with open("ghz.qasm", "r") as ghz:
    ghz_qasm_string = ghz.read()

# Import the device module
from braket.aws import AwsDevice
# Choose the Rigetti device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
from braket.ir.openqasm import Program

program = Program(source=ghz_qasm_string)
```

```
my_task = device.run(program)

# Specify an optional s3 bucket location and number of shots
s3_location = ("amzn-s3-demo-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
)
```

Uso de Boto3 para crear tareas cuánticas de OpenQASM 3.0

También puede usar el SDK de Python de AWS para Braket (Boto3) para crear las tareas cuánticas mediante cadenas de OpenQASM 3.0, como se muestra en el siguiente ejemplo. El siguiente fragmento de código hace referencia a ghz.qasm, que prepara un estado de GHZ, como se muestra anteriormente.

```
import boto3
import json

my_bucket = "amzn-s3-demo-bucket"
s3_prefix = "openqasm-tasks"

with open("ghz.qasm") as f:
    source = f.read()

action = {
    "braketSchemaHeader": {
        "name": "braket.ir.openqasm.program",
        "version": "1"
    },
    "source": source
}
device_parameters = {}
device_arn = "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3"
shots = 100

braket_client = boto3.client('braket', region_name='us-west-1')
rsp = braket_client.create_quantum_task(
    action=json.dumps(
        action
    ),
    deviceParameters=json.dumps(
```

```
        device_parameters
    ),
    deviceArn=device_arn,
    shots=shots,
    outputS3Bucket=my_bucket,
    outputS3KeyPrefix=s3_prefix,
)
```

Úselo para crear tareas de OpenQASM 3.0 AWS CLI

La AWS Command Line Interface (CLI) también se puede utilizar para enviar programas OpenQASM 3.0, como se muestra en el siguiente ejemplo.

```
aws braket create-quantum-task \
    --region "us-west-1" \
    --device-arn "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3" \
    --shots 100 \
    --output-s3-bucket "amzn-s3-demo-bucket" \
    --output-s3-key-prefix "openqasm-tasks" \
    --action '{
        "braketSchemaHeader": {
            "name": "braket.ir.openqasm.program",
            "version": "1"
        },
        "source": $(cat ghz.qasm)
    }'
```

Compatibilidad con OpenQASM en diferentes dispositivos de Braket

En el caso de los dispositivos compatibles con OpenQASM 3.0, el campo `action` admite una nueva acción a través de la respuesta `GetDevice`, como se muestra en el siguiente ejemplo para los dispositivos Rigetti y IonQ.

```
//OpenQASM as available with the Rigetti device capabilities
{
    "braketSchemaHeader": {
        "name": "braket.device_schema.rigetti.rigetti_device_capabilities",
        "version": "1"
    },
    "service": {...},
    "action": {
        "braket.ir.jaqcd.program": {...},
```

```
        "braket.ir.openqasm.program": {
            "actionType": "braket.ir.openqasm.program",
            "version": [
                "1"
            ],
            ….
        }
    }
}

//OpenQASM as available with the IonQ device capabilities
{
    "braketSchemaHeader": {
        "name": "braket.device_schema.ionq.ionq_device_capabilities",
        "version": "1"
    },
    "service": {...},
    "action": {
        "braket.ir.jaqcd.program": {...},
        "braket.ir.openqasm.program": {
            "actionType": "braket.ir.openqasm.program",
            "version": [
                "1"
            ],
            ….
        }
    }
}
```

En el caso de los dispositivos que admiten el control de pulsos, el campo `pulse` se muestra en la respuesta `GetDevice`. En el siguiente ejemplo, se muestra este campo `pulse` para el dispositivo Rigetti.

```
// Rigetti
{
  "pulse": {
    "braketSchemaHeader": {
      "name": "braket.device_schema.pulse.pulse_device_action_properties",
      "version": "1"
    },
    "supportedQhpTemplateWaveforms": {
      "constant": {
        "functionName": "constant",
```

```
        "arguments": [
          {
            "name": "length",
            "type": "float",
            "optional": false
          },
          {
            "name": "iq",
            "type": "complex",
            "optional": false
          }
        ]
      },
      ...
    },
    "ports": {
      "q0_ff": {
        "portId": "q0_ff",
        "direction": "tx",
        "portType": "ff",
        "dt": 1e-9,
        "centerFrequencies": [
          375000000
        ]
      },
      ...
    },
    "supportedFunctions": {
      "shift_phase": {
        "functionName": "shift_phase",
        "arguments": [
          {
            "name": "frame",
            "type": "frame",
            "optional": false
          },
          {
            "name": "phase",
            "type": "float",
            "optional": false
          }
        ]
      },
     ...
```

```
    },
    "frames": {
      "q0_q1_cphase_frame": {
        "frameId": "q0_q1_cphase_frame",
        "portId": "q0_ff",
        "frequency": 462475694.24460185,
        "centerFrequency": 375000000,
        "phase": 0,
        "associatedGate": "cphase",
        "qubitMappings": [
          0,
          1
        ]
      },
      ...
    },
    "supportsLocalPulseElements": false,
    "supportsDynamicFrames": false,
    "supportsNonNativeGatesWithPulses": false,
    "validationParameters": {
      "MAX_SCALE": 4,
      "MAX_AMPLITUDE": 1,
      "PERMITTED_FREQUENCY_DIFFERENCE": 400000000
    }
  }
}
```

Los campos anteriores detallan lo siguiente:

Puertos:

Describe los puertos de dispositivos externos (`extern`) previamente creados y declarados en la QPU, además de las propiedades asociadas al puerto en cuestión. Todos los puertos enumerados en esta estructura se declaran previamente como identificadores válidos en el programa `OpenQASM 3.0` enviado por el usuario. Las propiedades adicionales de un puerto incluyen:

- ID de puerto (portId)
  - El nombre del puerto declarado como identificador en OpenQASM 3.0.
- Dirección (direction)
  - La dirección del puerto. Los puertos de accionamiento transmiten impulsos (dirección «tx»), mientras que los puertos de medición reciben impulsos (dirección «rx»).

- Tipo de puerto (portType)
  - El tipo de acción de la que es responsable este puerto (por ejemplo, accionar, capturar o ff - fast-flux).
- Dt (dt)
  - El tiempo en segundos que representa un único intervalo de muestreo en el puerto determinado.
- Asignaciones de qubits (qubitMappings)
  - Los qubits asociados al puerto determinado.
- Frecuencias centrales (centerFrequencies)
  - Una lista de las frecuencias centrales asociadas para todos los marcos predeclarados o definidos por el usuario en el puerto. Para obtener más información, consulte «Marcos».
- Propiedades específicas de QHP () qhpSpecificProperties
  - Un mapa opcional que detalla las propiedades existentes sobre el puerto específico del QHP.

Marcos:

Describe los marcos externos previamente creados y declarados en la QPU, además de las propiedades asociadas a los marcos. Todos los marcos enumerados en esta estructura se declaran previamente como identificadores válidos en el programa `OpenQASM 3.0` enviado por el usuario. Las propiedades adicionales de un marco incluyen:

- ID de marco (frameId)
  - El nombre del marco declarado como identificador en OpenQASM 3.0.
- ID de puerto (portId)
  - El puerto de hardware asociado para el marco.
- Frecuencia (frequency)
  - La frecuencia inicial predeterminada del marco.
- Frecuencia central (centerFrequency)
  - El centro del ancho de banda de frecuencia del marco. Por lo general, los marcos solo se pueden ajustar a un determinado ancho de banda en torno a la frecuencia central. Como resultado, los ajustes de frecuencia deben mantenerse dentro de un delta determinado de la frecuencia central. Encontrará el valor del ancho de banda en los parámetros de validación.
- Fase (phase)
  - La frecuencia inicial predeterminada del marco.

- Puerta asociada (associatedGate)
  - Las puertas asociadas al marco determinado.
- Asignaciones de qubits (qubitMappings)
  - Los qubits asociados al marco determinado.
- Propiedades específicas de QHP () qhpSpecificProperties
  - Un mapa opcional que detalla las propiedades existentes sobre el marco específico del QHP.

SupportsDynamicFrames:

Describe si un marco puede declararse en los bloques de `cal` o `defcal` a través de la función `newframe` de OpenPulse. En caso negativo, solo se podrán utilizar en el programa los marcos que figuran en la estructura de marcos.

SupportedFunctions:

Describe las funciones de OpenPulse que son compatibles con el dispositivo, además de los argumentos asociados, los tipos de argumentos y los tipos de retorno para las funciones determinadas. Para ver ejemplos del uso de las OpenPulse funciones, consulte la OpenPulseespecificación. En este momento, Braket admite:

- shift_phase
  - Cambia la fase de un marco por un valor especificado.
- set_phase
  - Establece la fase del marco en el valor especificado.
- swap_phases
  - Intercambia las fases entre dos marcos.
- shift_frequency
  - Cambia la frecuencia de un marco por un valor especificado.
- set_frequency
  - Establece la frecuencia del marco en el valor especificado.
- jugar
  - Programa una forma de onda.
- capture_v0
  - Devuelve el valor de un marco de captura a un registro de bits.

SupportedQhpTemplateWaveforms:

Describe las funciones de forma de onda predefinidas disponibles en el dispositivo y los argumentos y tipos asociados. De forma predeterminada, Braket Pulse ofrece rutinas de forma de onda prediseñadas en todos los dispositivos, que son:

Constant

$$Constant(t, \tau, iq) = iq$$

`τ` es la longitud de la forma de onda e `iq` es un número complejo.

```
def constant(length, iq)
```

Gaussian

$$Gaussian(t, \tau, \sigma, A = 1, ZaE = 0) =$$
$$\frac{A}{1 - ZaE * \exp\left(-\frac{1}{2}\left(\frac{\tau}{2\sigma}\right)^2\right)} \left[\exp\left(-\frac{1}{2}\left(\frac{t - \frac{\tau}{2}}{\sigma}\right)^2\right) - ZaE * \exp\left(-\frac{1}{2}\left(\frac{\tau}{2\sigma}\right)^2\right)\right]$$

`τ` es la longitud de la onda, σ es la anchura del gaussiano y `A` es la amplitud. Si se establece `ZaE` en`True`, el gaussiano se desplaza y se reescala de manera que sea igual a cero al principio y al final de la forma de onda y alcance `A` en su máximo.

```
def gaussian(length, sigma, amplitude=1, zero_at_edges=False)
```

DRAG Gaussian

$$DRAG\_Gaussian(t, \tau, \sigma, \beta, A = 1, ZaE = 0) =$$
$$\frac{A}{1 - ZaE * \exp\left(-\frac{1}{2}\left(\frac{\tau}{2\sigma}\right)^2\right)} \left(1 - i\beta\frac{t - \frac{\tau}{2}}{\sigma^2}\right) \left[\exp\left(-\frac{1}{2}\left(\frac{t - \frac{\tau}{2}}{\sigma}\right)^2\right) - ZaE * \exp\left(-\frac{1}{2}\left(\frac{\tau}{2\sigma}\right)^2\right)\right]$$

`τ` es la longitud de la onda, σ es la anchura del gaussiano, β es un parámetro libre y `A` es la amplitud. Si `ZaE` se establece en `True`, La gaussiana de eliminación de derivadas mediante puerta adiabática (DRAG) se compensa y se reescala de modo que sea igual a cero al inicio y al final de la forma de onda, y la parte real alcance `A` en su máximo. Para obtener más información sobre la forma de onda DRAG, consulte el artículo Impulsos simples para la eliminación de fugas en qubits débilmente no lineales.

```
def drag_gaussian(length, sigma, beta, amplitude=1, zero_at_edges=False)
```

Erf Square

$$\text{Erf\_Square}(t, L, W, \sigma, A = 1, ZaE = 0) =$$

$$A \times \frac{\text{erf}((t - t_1)/\sigma) + \text{erf}(-(t - t_2)/\sigma)}{2 \times \text{erf}(W/2\sigma))}$$

Dónde `L` es la longitud, `W` es la anchura de la forma de onda, σ define la rapidez con la que suben y bajan los bordes, t_1`=(L-W)/2` y t_2`2=(L+W)/2`, y `A` es la amplitud. Si se establece `ZaE` en`True`, el gaussiano se desplaza y se reescala de manera que sea igual a cero al principio y al final de la forma de onda y alcance `A` en su máximo. La siguiente ecuación es la versión reescalada de la forma de onda.

$$\text{Erf\_Square}(..., ZaE = 1) = (a \times \text{Erf\_Square}(..., ZaE = 0) - bA)/(a - b)$$

Donde `a=erf(W/2σ)` y `b=erf(-`t_1`/σ)/2+erf(t` $_2$`/σ)/2` .

```
def erf_square(length, width, sigma, amplitude=1, zero_at_edges=False)
```

SupportsLocalPulseElements:

Describe si los elementos de pulso, como puertos, marcos y formas de onda, se pueden definir localmente en bloques de `defcal`. Si el valor es `false`, los elementos se deben definir en bloques de `cal`.

SupportsNonNativeGatesWithPulses:

Describe si podemos o no usar puertas no nativas en combinación con programas de pulsos.
Por ejemplo, no puede usar una puerta no nativa como una puerta `H` en un programa sin definir primero la puerta a través de `defcal` para el qubit utilizado. Puede encontrar la lista de claves `nativeGateSet` de puertas nativas en las capacidades del dispositivo.

ValidationParameters:

Describe los límites de validación de los elementos de pulso, que incluyen:

- Valores máximos de escala/amplitud para formas de onda (arbitrarias y prediseñadas)

- Ancho de banda de frecuencia máxima a partir de la frecuencia central suministrada en Hz
- Pulso mínimo length/duration en segundos
- Pulso máximo length/duration en segundos

Operaciones, resultados y tipos de resultados compatibles con OpenQASM

Para saber qué característica de OpenQASM 3.0 admite cada dispositivo, consulte la clave `braket.ir.openqasm.program` que aparece en el campo `action` de la salida de capacidades del dispositivo. Por ejemplo, a continuación se muestran las operaciones y los tipos de resultados compatibles disponibles para el simulador de vector de estado de Braket SV1.

```
...
  "action": {
    "braket.ir.jaqcd.program": {
      ...
    },
 "braket.ir.openqasm.program": {
      "version": [
        "1.0"
      ],
      "actionType": "braket.ir.openqasm.program",
      "supportedOperations": [
        "ccnot",
        "cnot",
        "cphaseshift",
        "cphaseshift00",
        "cphaseshift01",
        "cphaseshift10",
        "cswap",
        "cy",
        "cz",
        "h",
        "i",
        "iswap",
        "pswap",
        "phaseshift",
        "rx",
        "ry",
        "rz",
        "s",
        "si",
```

```
        "swap",
        "t",
        "ti",
        "v",
        "vi",
        "x",
        "xx",
        "xy",
        "y",
        "yy",
        "z",
        "zz"
      ],
      "supportedPragmas": [
        "braket_unitary_matrix"
      ],
      "forbiddenPragmas": [],
      "maximumQubitArrays": 1,
      "maximumClassicalArrays": 1,
      "forbiddenArrayOperations": [
        "concatenation",
        "negativeIndex",
        "range",
        "rangeWithStep",
        "slicing",
        "selection"
      ],
      "requiresAllQubitsMeasurement": true,
      "supportsPhysicalQubits": false,
      "requiresContiguousQubitIndices": true,
      "disabledQubitRewiringSupported": false,
      "supportedResultTypes": [
        {
          "name": "Sample",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
          ],
          "minShots": 1,
          "maxShots": 100000
```

```
        },
        {
          "name": "Expectation",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
          ],
          "minShots": 0,
          "maxShots": 100000
        },
        {
          "name": "Variance",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
          ],
          "minShots": 0,
          "maxShots": 100000
        },
        {
          "name": "Probability",
          "minShots": 1,
          "maxShots": 100000
        },
        {
          "name": "Amplitude",
          "minShots": 0,
          "maxShots": 0
        }
        {
          "name": "AdjointGradient",
          "minShots": 0,
          "maxShots": 0
        }
      ]
    }
```

```
  },
...
```

Simulación de ruido con OpenQASM 3.0

Para simular ruido con OpenQASM3, utilice instrucciones pragmáticas para añadir operadores de ruido. Por ejemplo, para simular la versión ruidosa del programa GHZ proporcionada anteriormente, puede enviar el siguiente programa OpenQASM.

```
// ghz.qasm
// Prepare a GHZ state
OPENQASM 3;

qubit[3] q;
bit[3] c;

h q[0];
#pragma braket noise depolarizing(0.75) q[0] cnot q[0], q[1];
#pragma braket noise depolarizing(0.75) q[0]
#pragma braket noise depolarizing(0.75) q[1] cnot q[1], q[2];
#pragma braket noise depolarizing(0.75) q[0]
#pragma braket noise depolarizing(0.75) q[1]

c = measure q;
```

Las especificaciones de todos los operadores de ruido pragma admitidos se proporcionan en la siguiente lista.

```
#pragma braket noise bit_flip(<float in [0,1/2]>) <qubit>
#pragma braket noise phase_flip(<float in [0,1/2]>) <qubit>
#pragma braket noise pauli_channel(<float>, <float>, <float>)  <qubit>
#pragma braket noise depolarizing(<float in [0,3/4]>) <qubit>
#pragma braket noise two_qubit_depolarizing(<float in [0,15/16]>) <qubit>, <qubit>
#pragma braket noise two_qubit_dephasing(<float in [0,3/4]>) <qubit>, <qubit>
#pragma braket noise amplitude_damping(<float in [0,1]>) <qubit>
#pragma braket noise generalized_amplitude_damping(<float in [0,1]> <float in [0,1]>)
 <qubit>
#pragma braket noise phase_damping(<float in [0,1]>) <qubit>
#pragma braket noise kraus([[<complex m0_00>, ], ...], [[<complex m1_00>, ], ...], ...)
 <qubit>[, <qubit>]     // maximum of 2 qubits and maximum of 4 matrices for 1 qubit,
 16 for 2
```

Operador de Kraus

Para generar un operador de Kraus, puede recorrer en iteración una lista de matrices e imprimir cada elemento de la matriz como una expresión compleja.

Cuando utilice operadores de Kraus, recuerde lo siguiente:

- El número de qubits no debe ser superior a 2. La definición actual de los esquemas establece este límite.
- La longitud de la lista de argumentos debe ser un múltiplo de 8. Esto significa que debe estar compuesta únicamente por matrices de 2x2.
- La longitud total no supera las matrices de 2^{2*num_qubits}. Esto significa 4 matrices para 1 qubit y 16 para 2 qubits.
- Todas las matrices suministradas tienen conservación de traza completamente positiva (CPTP).
- El producto de los operadores de Kraus con sus conjugados de transposición debe sumarse a una matriz de identidad.

Recableado de Qubit con OpenQASM 3.0

Amazon Braket admite la notación de qubit físico dentro de OpenQASM en los dispositivos Rigetti (para obtener más información, consulte esta página). Cuando utilice qubits físicos con la estrategia de recableado nativo, asegúrese de que los qubits están conectados en el dispositivo seleccionado. Como alternativa, si se utilizan registros de qubit en su lugar, la estrategia de recableado PARCIAL está habilitada de forma predeterminada en los dispositivos Rigetti.

```
// ghz.qasm
// Prepare a GHZ state
OPENQASM 3;

h $0;
cnot $0, $1;
cnot $1, $2;

measure $0;
measure $1;
measure $2;
```

Compilación verbatim con OpenQASM 3.0

Cuando ejecuta un circuito cuántico en computadoras cuánticas suministradas por proveedores como Rigetti y IonQ, puede indicar al compilador que ejecute sus circuitos exactamente tal y como están definidos, sin ninguna modificación. Esta característica se conoce como compilación verbatim. Con los dispositivos Rigetti, puede especificar con precisión lo que se debe conservar, ya sea un circuito completo o solo partes específicas del mismo. Para conservar solo partes específicas de un circuito, deberá usar puertas nativas dentro de las regiones preservadas. Actualmente, IonQ solo admite la compilación verbatim de todo el circuito, por lo que todas las instrucciones del circuito deben estar incluidas en un cuadro verbatim.

Con OpenQASM, puede especificar explícitamente un pragma verbatim alrededor de un cuadro de código que luego se deja intacto y no se optimiza mediante la rutina de compilación de bajo nivel del hardware. En el siguiente ejemplo de código, se muestra cómo utilizar la directiva de `#pragma braket verbatim` para lograr esto.

```
OPENQASM 3;

bit[2] c;

#pragma braket verbatim
box{
    rx(0.314159) $0;
    rz(0.628318) $0, $1;
    cz $0, $1;
}

c[0] = measure $0;
c[1] = measure $1;
```

Para obtener información más detallada sobre el proceso de compilación literal, incluidos ejemplos y mejores prácticas, consulta el cuaderno de ejemplos de compilación literal disponible en el repositorio de github. amazon-braket-examples

La consola de Braket

Las tareas de OpenQASM 3.0 están disponibles y se pueden gestionar en la consola de Amazon Braket. En la consola, tendrá la misma experiencia en el envío de tareas cuánticas en OpenQASM 3.0 que en el envío de tareas cuánticas existentes.

Recursos adicionales

OpenQASM está disponible en todas las regiones de Amazon Braket.

Para ver un ejemplo de bloc de notas para empezar a usar OpenQASM en Amazon Braket, consulta Braket Tutorials. GitHub

Cómputo de gradientes con OpenQASM 3.0

Amazon Braket admite el cómputo de gradientes en simuladores locales y simuladores bajo demanda cuando se ejecuta en el modo `shots=0` (exacto). Esto se logra mediante el uso del método de diferenciación adjunta. Para especificar el gradiente que desea computar, puede proporcionar el pragma adecuado, tal y como se muestra en el código del siguiente ejemplo.

```
OPENQASM 3.0;
input float alpha;

bit[2] b;
qubit[2] q;

h q[0];
h q[1];
rx(alpha) q[0];
rx(alpha) q[1];
b[0] = measure q[0];
b[1] = measure q[1];

#pragma braket result adjoint_gradient h(q[0]) @ i(q[1]) alpha
```

En lugar de enumerar todos los parámetros individuales de forma explícita, también puede especificar la palabra clave `all` dentro del pragma. Esto calculará el gradiente con respecto a todos los parámetros de `input` enumerados, lo que puede ser una opción conveniente cuando el número de parámetros es muy grande. En este caso, el pragma se verá como el código del siguiente ejemplo.

```
#pragma braket result adjoint_gradient h(q[0]) @ i(q[1]) all
```

Todos los tipos observables son compatibles con la implementación de OpenQASM 3.0 de Amazon Braket, incluidos los operadores individuales, los productos tensoriales, los observables hermitianos

y los observables `Sum`. El operador específico que le interesa utilizar al computar gradientes debe estar envuelto dentro de la función `expectation()` y los qubits sobre los que actúa cada término del observable deben especificarse explícitamente.

Medición de qubits específicos con OpenQASM 3.0

El simulador de vector de estado local y el simulador de matriz de densidad local proporcionados por Amazon Braket admiten el envío de programas OpenQASM en los que se puede medir de forma selectiva un subconjunto de qubits del circuito. Esta capacidad, con frecuencia denominada «medición parcial», permite realizar cómputos cuánticos más específicos y eficientes. Por ejemplo, en el siguiente fragmento de código, puede crear un circuito de dos qubits y elegir medir solo el primer qubit y dejar el segundo qubit sin medir.

```
partial_measure_qasm = """
OPENQASM 3.0;
bit[1] b;
qubit[2] q;
h q[0];
cnot q[0], q[1];
b[0] = measure q[0];
"""
```

En este ejemplo, tenemos un circuito cuántico con dos qubits, `q[0]` y `q[1]`, pero solo nos interesa medir el estado del primer qubit. Esto se consigue mediante la línea `b[0] = measure q[0]`, que mide el estado del qubit[0] y almacena el resultado en el bit clásico b[0]. Para ejecutar este escenario de medición parcial, podemos ejecutar el siguiente código en el simulador de vector de estado local proporcionado por Amazon Braket.

```
from braket.devices import LocalSimulator

local_sim = LocalSimulator()
partial_measure_local_sim_task =
 local_sim.run(OpenQASMProgram(source=partial_measure_qasm), shots = 10)
partial_measure_local_sim_result = partial_measure_local_sim_task.result()
print(partial_measure_local_sim_result.measurement_counts)
print("Measured qubits: ", partial_measure_local_sim_result.measured_qubits)
```

Puede comprobar si un dispositivo admite la medición parcial inspeccionando el campo `requiresAllQubitsMeasurement` en sus propiedades de acción; si es `False`, entonces se admite la medición parcial.

```
from braket.devices import Devices

AwsDevice(Devices.Rigetti.Ankaa3).properties.action['braket.ir.openqasm.program'].requiresAllQu
```

Aquí, `requiresAllQubitsMeasurement` es `False`, lo que indica que no se deben medir todos los qubits.

Exploración de las capacidades experimentales

Las capacidades experimentales proporcionan acceso a hardware con disponibilidad limitada y a nuevas funciones de software emergentes. Estas funciones pueden afectar al rendimiento del dispositivo más allá de las especificaciones estándar. Puede activar automáticamente las capacidades del software experimental para cada tarea a través del Amazon Braket SDK.

Para utilizar las capacidades experimentales, especifique el `experimental_capabilities` parámetro al crear tareas cuánticas. Establezca este parámetro en `"ALL"` para habilitar todas las funciones experimentales disponibles para esa tarea. El siguiente ejemplo muestra cómo habilitar las capacidades experimentales al ejecutar un circuito en un dispositivo:

```
from braket.aws import AwsDevice

device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/quera/Aquila")

task = device.run(
   circuit,
   shots=1000,
   experimental_capabilities="ALL"
)
```

> **Note**
>
> Estas funciones son experimentales y pueden cambiar sin previo aviso. El rendimiento del dispositivo puede diferir de las especificaciones publicadas y los resultados pueden variar de los de las operaciones estándar. Debe habilitar de forma explícita las capacidades experimentales para cada tarea. Las tareas sin este parámetro utilizarán únicamente las capacidades estándar del dispositivo.

En esta sección:

Acceso a la desafinación local en Aquila QuEra

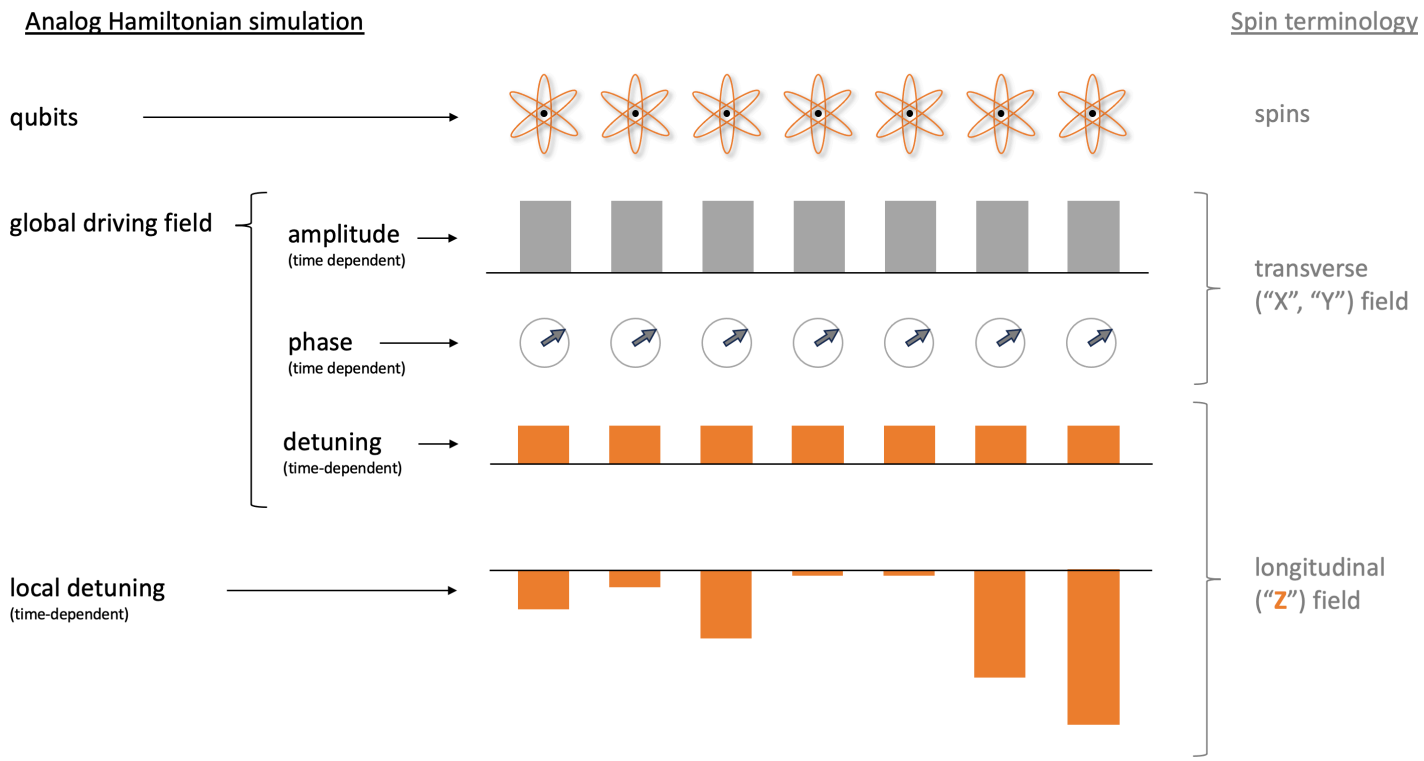
La desintonización local (LD) es un nuevo campo de control dependiente del tiempo con un patrón espacial personalizable. El campo LD afecta a los qubits según un patrón espacial personalizable, lo que permite obtener diferentes hamiltonianos para diferentes qubits más allá de lo que pueden crear el campo de accionamiento uniforme y la interacción Rydberg-Rydberg.

Restricciones:

El patrón espacial del campo de desintonización local se puede personalizar para cada programa AHS, pero se mantiene constante a lo largo del programa. La serie temporal del campo de desintonización local debe comenzar y terminar en cero, y todos los valores deben ser menores o iguales a cero. Además, los parámetros del campo de desintonización local están limitados por restricciones numéricas, que pueden visualizarse a través del SDK de Braket en la sección de propiedades específicas del dispositivo: `aquila_device.properties.paradigm.rydberg.rydbergLocal`.

Limitaciones:

Al ejecutar programas cuánticos que utilizan el campo de desintonización local (incluso si su magnitud se establece en cero constante en el hamiltoniano), el dispositivo experimenta una decoherencia más rápida que el tiempo T2 enumerado en la sección de rendimiento de las propiedades de Aquila. Cuando no sea necesario, se recomienda omitir el campo de desintonización local del hamiltoniano del programa AHS.



Ejemplos:

1. Simulación del efecto de un campo magnético longitudinal no uniforme en sistemas de espín

   Mientras que la amplitud y la fase del campo de accionamiento tienen el mismo efecto sobre los qubits que el campo magnético transversal sobre los espines, la suma de la desintonización del campo de accionamiento y la desintonización local produce el mismo efecto sobre los qubits que el campo longitudinal sobre los espines. Con el control espacial del campo de desintonización local, se pueden simular sistemas de espín más complejos.
2. Preparación de estados iniciales de no equilibrio

   El cuaderno de ejemplo Simulating lattice gauge theory with Rydberg atoms muestra cómo suprimir la excitación del átomo central de una disposición lineal de 9 átomos al recocer el sistema hacia la fase Z2 ordenada. Tras la fase de preparación, el campo de desintonización local se reduce gradualmente y el programa AHS continúa simulando la evolución temporal del sistema a partir de este estado concreto de no equilibrio.
3. Resolución de problemas de optimización ponderada

   El cuaderno de ejemplo Maximum weight independent set (MWIS) muestra cómo resolver un problema de conjunto independiente de peso máximo (MWIS) en Aquila. El campo de

desintonización local se utiliza para definir los pesos en los nodos del gráfico del disco unitario, cuyos bordes se realizan mediante el efecto de bloqueo de Rybderg. Partiendo del estado fundamental uniforme y aumentando gradualmente el campo de desintonización local, el sistema pasa al estado fundamental del hamiltoniano de MWIS para encontrar soluciones al problema.

Acceso a geometrías altas en Aquila QuEra

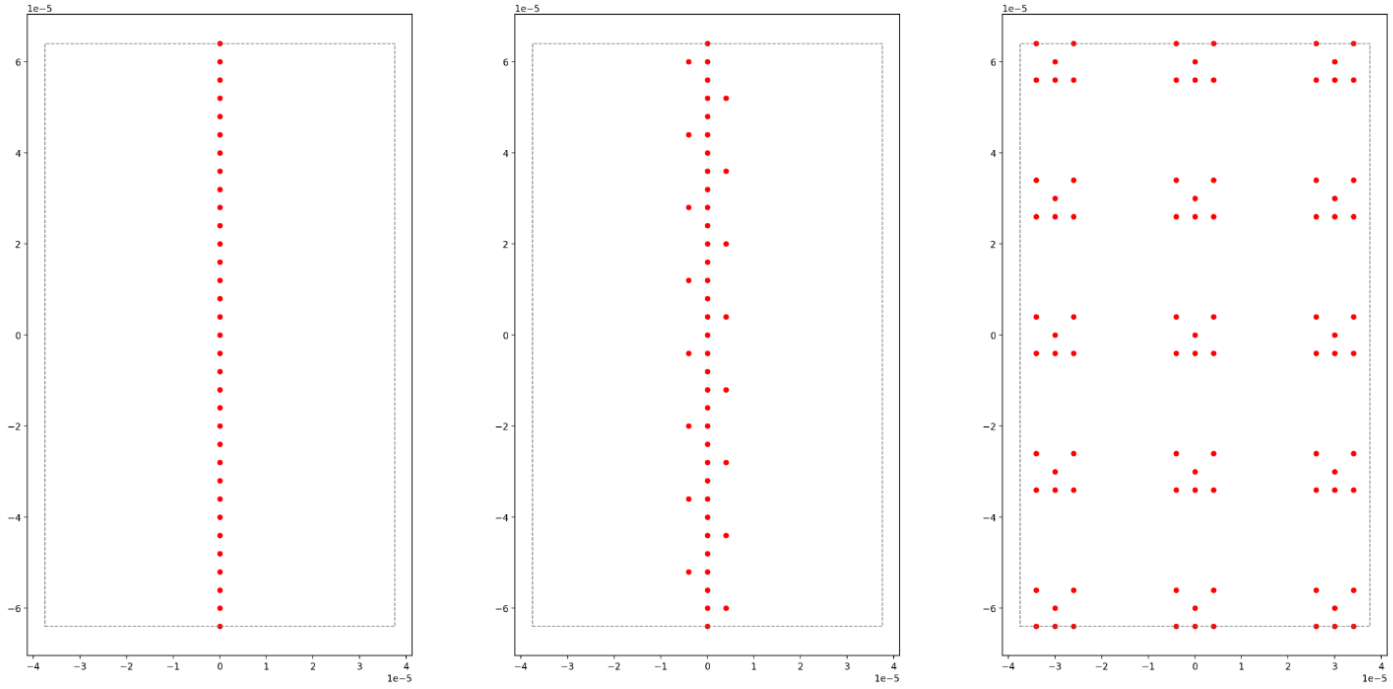
La característica de geometrías altas le permite especificar geometrías con mayor altura. Con esta capacidad, las disposiciones de átomos de sus programas AHS pueden abarcar una longitud adicional en la dirección «y» más allá de las capacidades habituales de Aquila.

Restricciones:

La altura máxima para geometrías altas es de 0,000128 m (128 um).

Limitaciones:

Cuando esta función experimental esté habilitada para su cuenta, las funciones que se muestran en la página de propiedades del dispositivo y la llamada a `GetDevice` seguirán reflejando el límite inferior habitual en altura. Cuando un programa AHS utiliza disposiciones de átomos que superan las capacidades normales, es de esperar que el error de relleno aumente. Encontrará un número elevado de ceros inesperados en la parte `pre_sequence` del resultado de la tarea, lo que a su vez reducirá la posibilidad de obtener una disposición perfectamente inicializada. Este efecto es más fuerte en filas con muchos átomos.



Ejemplos:

1. Disposiciones 1d y cuasi 1d más grandes

   Las cadenas de átomos y las disposiciones en forma de escalera se pueden extender a números de átomos más altos. Al orientar la dirección larga en paralelo a «y», se pueden programar instancias más largas de estos modelos.

2. Más espacio para multiplexar la ejecución de tareas con geometrías pequeñas

   El cuaderno de ejemplo Parallel quantum tasks on Aquila, muestra cómo aprovechar al máximo el área disponible: colocando copias multiplexadas de la geometría en cuestión en una disposición de átomos. Al aumentar el área disponible, se pueden colocar más copias.

Acceso a geometrías reducidas en Aquila QuEra

La característica de geometrías ajustadas le permite especificar geometrías con un espacio más corto entre las filas vecinas. En un programa AHS, los átomos están dispuestos en filas, separados por un espaciado vertical mínimo. La coordenada y de dos sitios atómicos cualesquiera debe ser cero (misma fila) o diferir en más del espaciado mínimo entre filas (filas diferentes). Gracias a la capacidad de geometrías estrechas, se reduce la distancia mínima entre filas, lo que permite crear disposiciones de átomos más compactas. Aunque esta extensión no modifica el requisito de

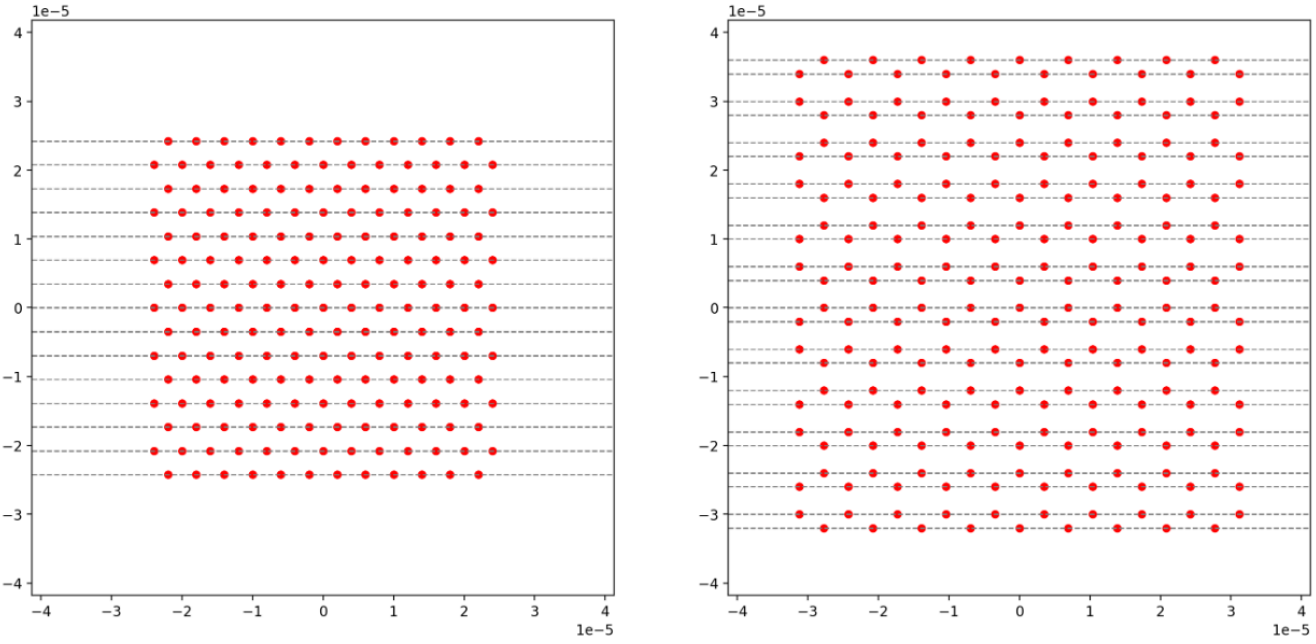
distancia euclídea mínima entre átomos, permite la creación de retículas en las que átomos distantes ocupan filas vecinas más cercanas entre sí, como es el caso, por ejemplo, de la retícula triangular.

Restricciones:

El espacio mínimo entre filas para geometrías estrechas es de 0,000002 m (2 um).

Limitaciones:

Cuando esta función experimental esté habilitada para su cuenta, las funciones que se muestran en la página de propiedades del dispositivo y la llamada a `GetDevice` seguirán reflejando el límite inferior habitual en altura. Cuando un programa AHS utiliza disposiciones de átomos que superan las capacidades normales, es de esperar que el error de relleno aumente. Los clientes encontrarán un número elevado de ceros inesperados en la parte `pre_sequence` del resultado de la tarea, lo que a su vez reducirá la posibilidad de obtener una disposición perfectamente inicializada. Este efecto es más fuerte en filas con muchos átomos.



Ejemplos:

1. Retículas no rectangulares con constantes de retículas pequeñas

   Un espaciado entre filas más estrecho permite la creación de retículas en las que el vecino más cercano a algunos átomos está en la dirección diagonal. Algunos ejemplos notables son las retículas triangulares, hexagonales y de Kagome, así como algunos cuasicristales.

2. Familia de retículas ajustables

   En los programas AHS, las interacciones se ajustan modificando la distancia entre pares de átomos. Un espaciado entre filas más estrecho permite ajustar las interacciones de los diferentes pares de átomos entre sí con mayor libertad, ya que los ángulos y distancias que definen la estructura atómica están menos limitados por la restricción del espaciado mínimo entre filas. Un ejemplo notable es la familia de retículas Shastry-Sutherland con diferentes longitudes de enlace.

Circuitos dinámicos en dispositivos IQM

Los circuitos dinámicos de los dispositivos IQM permiten realizar mediciones del circuito medio (MCM) y realizar operaciones de prealimentación. Estas características permiten a los investigadores y desarrolladores cuánticos implementar algoritmos cuánticos avanzados con lógica condicional y capacidades de reutilización de qubits. Esta característica experimental ayuda a explorar algoritmos cuánticos con una mayor eficiencia de recursos y a estudiar esquemas de mitigación y corrección de errores cuánticos.

Instrucciones clave:

- `measure_ff`: implementa la medición para el control de prealimentación midiendo un qubit y almacenando el resultado con una tecla de retroalimentación.
- `cc_prx`: implementa una rotación controlada clásicamente que solo se aplica cuando el resultado asociado con la clave de retroalimentación determinada mide un estado |1#.

Amazon Braket admite circuitos dinámicos a través de OpenQASM, el Amazon Braket SDK, el Amazon Braket Qiskit Provider.

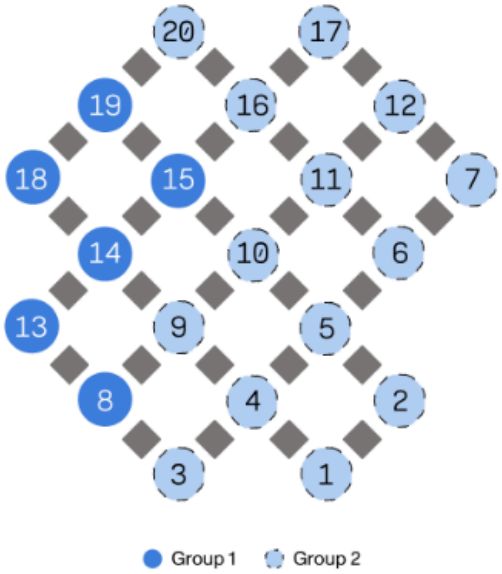
Restricciones:

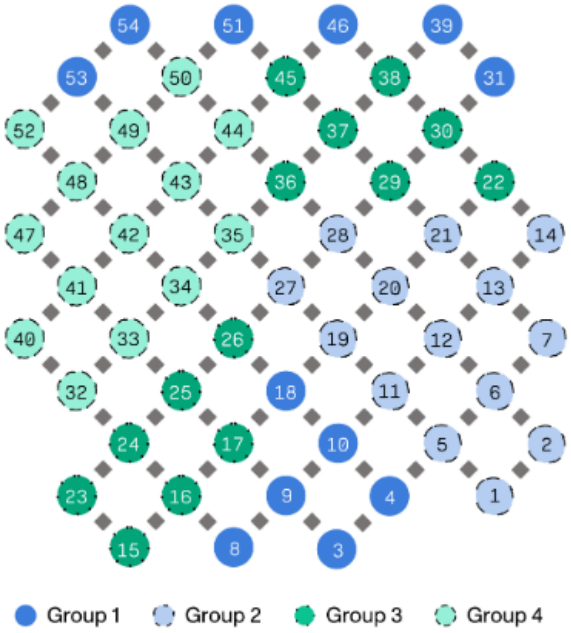
1. Las claves de retroalimentación de las instrucciones de `measure_ff` deben ser únicas.
2. Un `cc_prx` debe suceder después de un `measure_ff` con la misma clave de retroalimentación.
3. En un circuito único, la prealimentación de un qubit solo puede ser controlada por un qubit, ya sea por sí mismo o por otro qubit. En diferentes circuitos, puede tener diferentes pares de control.
   a. Por ejemplo, si el qubit 1 está controlado por el qubit 2, no puede ser controlado por el qubit 3 en el mismo circuito. No hay ninguna restricción en cuanto al número de veces que se aplica el control entre el qubit 1 y el qubit 2. El qubit 2 puede ser controlado por el qubit 3 (o el qubit 1), a menos que se haya realizado un restablecimiento activo en el qubit 2.

4. El control solo se puede aplicar a qubits del mismo grupo. Los grupos de qubits de los dispositivos Emerald y IQM Garnet se muestran en las siguientes imágenes.
5. Los programas con estas capacidades deben enviarse como programas verbatim. Para obtener más información sobre los programas verbatim, consulte Compilación verbatim con OpenQASM 3.0.

Limitaciones:

La MCM solo se puede usar para controlar la prealimentación en un programa. Los resultados de la MCM (0 o 1) no se devuelven como parte del resultado de una tarea.





Estas imágenes muestran las agrupaciones de qubits de ambos dispositivos IQM. El dispositivo Garnet de 20 qubits contiene 2 grupos de qubits, mientras que el dispositivo Emerald de 54 qubits contiene 4 grupos de qubits.

Ejemplos:

1. Reutilización de qubits mediante un restablecimiento activo

   La MCM con operaciones de restablecimiento condicional permite la reutilización de los qubits en una sola ejecución de circuito. Esto reduce los requisitos de profundidad de circuito y mejora la utilización de los recursos de los dispositivos cuánticos.

2. Protección activa contra inversión de bits

   Los circuitos dinámicos detectan errores de inversión de bits y aplican operaciones correctivas basadas en los resultados de las mediciones. Esta implementación sirve como un experimento de detección de errores cuánticos.

3. Experimentos de teletransportación

   La teletransportación estatal transfiere estados de cúbits mediante operaciones cuánticas locales e información clásica de. MCMs La teletransportación de puertas implementa puertas entre qubits sin operaciones cuánticas directas. Estos experimentos demuestran subrutinas fundamentales en tres áreas clave: corrección de errores cuánticos, computación cuántica basada en mediciones y comunicación cuántica.

4. Simulación de sistemas cuánticos abiertos

   Los circuitos dinámicos modelan el ruido en los sistemas cuánticos a través de los qubit de datos y el entrelazamiento del entorno, y las mediciones ambientales. Este enfoque utiliza qubits específicos para representar los datos y los elementos del entorno. Se puede diseñar un canal de ruido mediante las puertas y las mediciones aplicadas al entorno.

Para obtener más información sobre el uso de circuitos dinámicos, consulte ejemplos adicionales en el repositorio de cuadernos de Amazon Braket.

Control de pulsos en Amazon Braket

Los pulsos son las señales analógicas que controlan los qubits en una computadora cuántica. Con algunos dispositivos de Amazon Braket, puede acceder a la característica de control de pulsos para enviar circuitos mediante pulsos. Puedes acceder al control de pulsos a través del SDK de Braket, mediante OpenQASM 3.0, o directamente a través del Braket. APIs En primer lugar, introduzca algunos conceptos clave para el control de pulsos en Braket.

En esta sección:

Marcos

Un marco es una abstracción de software que actúa como un reloj dentro del programa cuántico y como una fase. La hora del reloj se incrementa con cada uso y con cada señal portadora con estado que se define mediante una frecuencia. Al transmitir señales al qubit, un marco determina la frecuencia portadora del qubit, el desplazamiento de fase y el momento en que se emite la envolvente de la forma de onda. En Braket Pulse, la construcción de marcos depende del dispositivo, la frecuencia y la fase. Según el dispositivo, puede elegir un marco predefinido o crear instancias de marcos nuevos proporcionando un puerto.

```
from braket.aws import AwsDevice
from braket.pulse import Frame, Port

# Predefined frame from a device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
drive_frame = device.frames["Transmon_5_charge_tx"]

# Create a custom frame
readout_frame = Frame(frame_id="r0_measure", port=Port("channel_0", dt=1e-9),
 frequency=5e9, phase=0)
```

Puertos

Un puerto es una abstracción de software que representa cualquier componente de input/output hardware que controle los qubits. Ayuda a los proveedores de hardware a proporcionar una interfaz con la que los usuarios pueden interactuar para manipular y observar los qubits. Los puertos se caracterizan por una cadena única que representa el nombre del conector. Esta cadena también expone un incremento de tiempo mínimo que especifica la precisión con la que podemos definir las formas de onda.

```
from braket.pulse import Port

Port0 = Port("channel_0", dt=1e-9)
```

Formas de onda

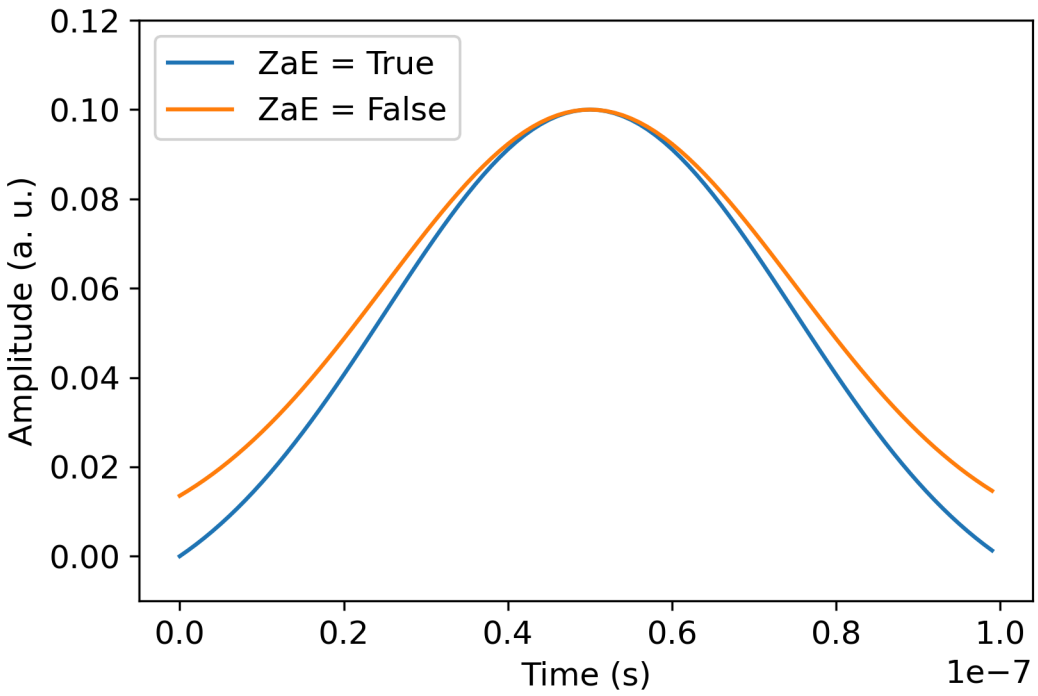
Una forma de onda es una envolvente dependiente del tiempo que podemos utilizar para emitir señales en un puerto de salida o capturar señales a través de un puerto de entrada. Puede

especificar sus formas de onda directamente, ya sea mediante una lista de números complejos o utilizando una plantilla de forma de onda para generar una lista del proveedor de hardware.

```
from braket.pulse import ArbitraryWaveform, ConstantWaveform
import numpy as np

cst_wfm = ConstantWaveform(length=1e-7, iq=0.1)
arb_wf = ArbitraryWaveform(amplitudes=np.linspace(0, 100))
```

Braket Pulse proporciona una biblioteca estándar de formas de onda, que incluye una forma de onda constante, una forma de onda gaussiana y una forma de onda de eliminación de derivadas mediante puerta adiabática (DRAG). Puede recuperar los datos de la forma de onda mediante la función `sample` para dibujar la forma de la onda, como se muestra en el siguiente ejemplo.

```
from braket.pulse import GaussianWaveform
import numpy as np
import matplotlib.pyplot as plt

zero_at_edge1 = GaussianWaveform(1e-7, 25e-9, 0.1, True)
# or zero_at_edge1 = GaussianWaveform(1e-7, 25e-9, 0.1)
zero_at_edge2 = GaussianWaveform(1e-7, 25e-9, 0.1, False)

times_1 = np.arange(0, zero_at_edge1.length, drive_frame.port.dt)
times_2 = np.arange(0, zero_at_edge2.length, drive_frame.port.dt)

plt.plot(times_1, zero_at_edge1.sample(drive_frame.port.dt))
plt.plot(times_2, zero_at_edge2.sample(drive_frame.port.dt))
```



La imagen anterior muestra las formas de onda gaussianas creadas a partir de `GaussianWaveform`. Elegimos una longitud de pulso de 100 ns, una anchura de 25 ns y una amplitud de 0,1 (unidades arbitrarias). Las formas de onda se centran en la ventana de pulsos. `GaussianWaveform` acepta un argumento booleano `zero_at_edges` (ZaE en la leyenda). Cuando se establece en `True`, este argumento desplaza la forma de onda gaussiana de manera que los puntos en t=0 y t=`length` estén en cero y reescala la amplitud de manera que el valor máximo se corresponda con el argumento de `amplitude`.

Trabajar con Hello Pulse

En esta sección, aprenderá a caracterizar y construir una única puerta de qubit directamente mediante pulso en un dispositivo Rigetti. Al aplicar un campo electromagnético a un qubit, se produce una oscilación de Rabi, que cambia los qubits entre su estado 0 y 1. Con la longitud y la fase del pulso calibradas, la oscilación de Rabi puede calcular las puertas de un solo qubit. Aquí, determinaremos la longitud de pulso óptima para medir un pulso de pi/2, un bloque elemental que se utiliza para construir secuencias de pulsos más complejas.

En primer lugar, para crear una secuencia de pulsos, importe la clase `PulseSequence`.

```
from braket.aws import AwsDevice
```

```
from braket.circuits import FreeParameter
from braket.devices import Devices
from braket.pulse import PulseSequence, GaussianWaveform

import numpy as np
```

A continuación, cree una instancia de un nuevo dispositivo Braket utilizando el Amazon Resource Name (ARN) de la QPU. El siguiente bloque de comandos usa Rigetti Ankaa-3.

```
device = AwsDevice(Devices.Rigetti.Ankaa3)
```

La siguiente secuencia de pulsos incluye dos componentes: reproducir una forma de onda y medir un qubit. La secuencia de pulsos normalmente se puede aplicar a marcos. Con algunas excepciones, como la barrera y el retardo, que se pueden aplicar a los qubits. Antes de construir la secuencia de pulsos, debe recuperar los marcos disponibles. El marco de accionamiento se utiliza para aplicar el pulso para la oscilación de Rabi y el marco de lectura sirve para medir el estado del qubit. En este ejemplo, se utilizan los marcos del qubit 25.

```
drive_frame = device.frames["Transmon_25_charge_tx"]
readout_frame = device.frames["Transmon_25_readout_rx"]
```

Ahora, cree la forma de onda que se reproducirá en el marco de accionamiento. El objetivo es caracterizar el comportamiento de los qubits para diferentes longitudes de pulso. Reproducirá una forma de onda con diferentes longitudes cada vez. En lugar de crear una nueva forma de onda cada vez, utilice el `FreeParameter` compatible con Braket en la secuencia de pulsos. Puede crear la forma de onda y la secuencia de pulsos una vez con parámetros libres y, a continuación, ejecutar la misma secuencia de pulsos con valores de entrada diferentes.

```
waveform = GaussianWaveform(FreeParameter("length"), FreeParameter("length") * 0.25,
 0.2, False)
```

Por último, póngalos juntos como una secuencia de pulsos. En la secuencia de pulsos, `play` reproduce la forma de onda especificada en el marco de accionamiento y, a continuación, `capture_v0` mide el estado desde el marco de lectura.

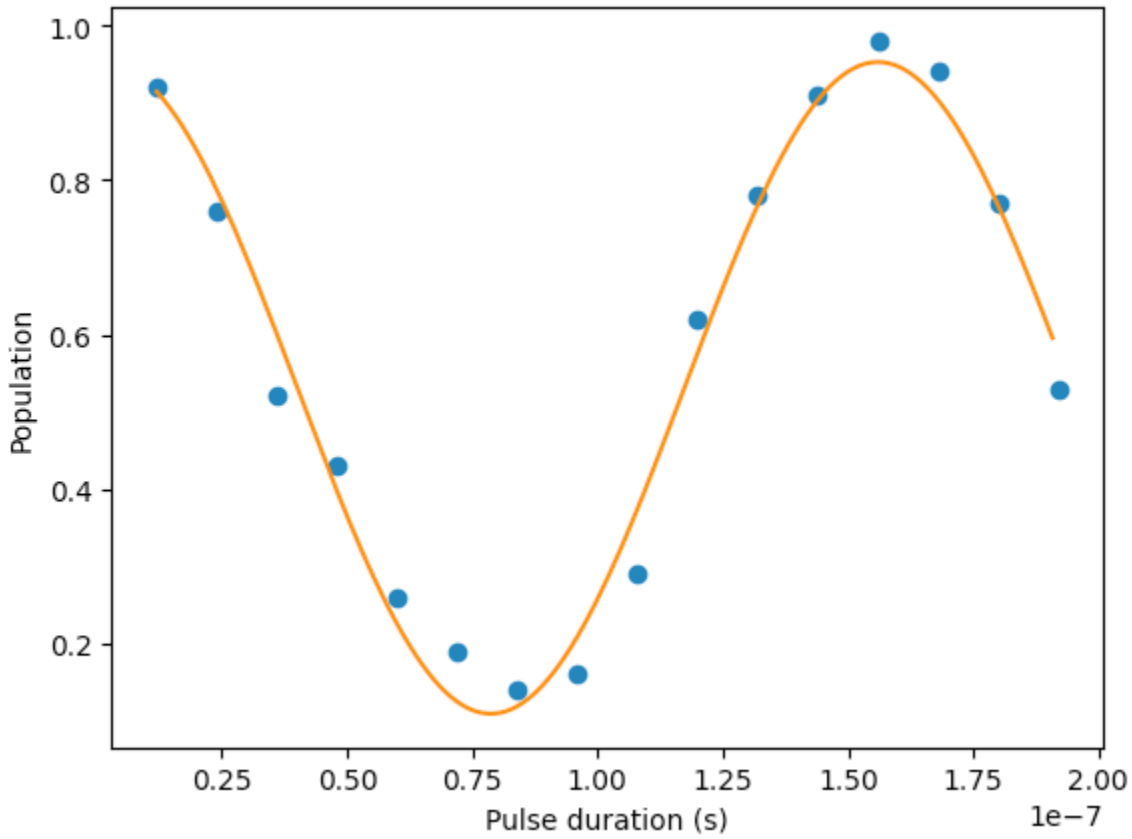
```
pulse_sequence = (
    PulseSequence()
    .play(drive_frame, waveform)
```

```
    .capture_v0(readout_frame)
)
```

Escanee un intervalo de longitud de pulso y envíelo a la QPU. Antes de ejecutar las secuencias de pulsos en una QPU, vincule el valor de los parámetros libres.

```
start_length = 12e-9
end_length = 2e-7
lengths = np.arange(start_length, end_length, 12e-9)
N_shots = 100

tasks = [
    device.run(pulse_sequence(length=length), shots=N_shots)
    for length in lengths
]

probability_of_zero = [
    task.result().measurement_counts['0']/N_shots
    for task in tasks
]
```

Las estadísticas de la medición de qubits muestran la dinámica oscilatoria del qubit que oscila entre el estado 0 y el estado 1. A partir de los datos de medición, puede extraer la frecuencia de Rabi y refinar la longitud del pulso para implementar una puerta de 1 qubit concreta. Por ejemplo, a partir de los datos de la figura siguiente, la periodicidad es de aproximadamente 154 ns. Por lo tanto, una puerta de rotación pi/2 correspondería a la secuencia de pulsos con una longitud = 38,5 ns.



Hello Pulse usando OpenPulse

OpenPulsees un lenguaje para especificar el control a nivel de pulso de un dispositivo cuántico general y forma parte de la especificación OpenQASM 3.0. Amazon Braket admite OpenPulse para la programación directa de pulsos mediante la representación OpenQASM 3.0.

Braket utiliza OpenPulse como representación intermedia subyacente para expresar los pulsos en instrucciones nativas. OpenPulse admite la adición de calibraciones de instrucción en forma de declaraciones `defcal` (abreviatura de «definir calibración»). Con estas declaraciones, puede especificar la implementación de una instrucción de puerta dentro de una gramática de control de nivel inferior.

Puede ver el OpenPulse programa de un `PulseSequence` Braket con el siguiente comando.

```
print(pulse_sequence.to_ir())
```

También puede crear un OpenPulse programa directamente.

```
from braket.ir.openqasm import Program
```

```
openpulse_script = """
OPENQASM 3.0;
cal {
    bit[1] psb;
    waveform my_waveform = gaussian(12.0ns, 3.0ns, 0.2, false);
    play(Transmon_25_charge_tx, my_waveform);
    psb[0] = capture_v0(Transmon_25_readout_rx);
}
"""
```

Cree un objeto `Program` con su script. A continuación, envíe el programa a una QPU.

```
from braket.aws import AwsDevice
from braket.devices import Devices
from braket.ir.openqasm import Program

program = Program(source=openpulse_script)

device = AwsDevice(Devices.Rigetti.Ankaa3)
task = device.run(program, shots=100)
```

Acceso a puertas nativas mediante pulsos

Los investigadores a menudo necesitan saber exactamente cómo se implementan como pulsos las puertas nativas compatibles con una QPU en particular. Las secuencias de pulsos son cuidadosamente calibradas por los proveedores de hardware, pero acceder a ellas brinda a los investigadores la oportunidad de diseñar mejores puertas o explorar protocolos para la mitigación de errores, como la extrapolación de ruido cero mediante el alargamiento de los pulsos de puertas específicas.

Amazon Braket admite el acceso programático a las puertas nativas de Rigetti.

```
import math
from braket.aws import AwsDevice
from braket.circuits import Circuit, GateCalibrations, QubitSet
from braket.circuits.gates import Rx

device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

calibrations = device.gate_calibrations
```

```
print(f"Downloaded {len(calibrations)} calibrations.")
```

> **Note**
>
> Los proveedores de hardware calibran periódicamente la QPU, a menudo más de una vez al día. El SDK de Braket le permite obtener las calibraciones de puertas más recientes.

```
device.refresh_gate_calibrations()
```

Para recuperar una puerta nativa determinada, como la puerta RX o XY, debe pasar el objeto `Gate` y los qubits de interés. Por ejemplo, puede inspeccionar la implementación de pulso del RX (π /2) aplicado a qubit 0.

```
rx_pi_2_q0 = (Rx(math.pi/2), QubitSet(0))

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

Puede crear un conjunto filtrado de calibraciones mediante la función `filter`. Pasa una lista de puertas o una lista de `QubitSet`. El siguiente código crea dos conjuntos que contienen todas las calibraciones de RX (π /2) y de qubit 0.

```
rx_calibrations = calibrations.filter(gates=[Rx(math.pi/2)])
q0_calibrations = calibrations.filter(qubits=QubitSet([0]))
```

Ahora puede proporcionar o modificar la acción de las puertas nativas adjuntando un conjunto de calibración personalizado. Por ejemplo, considere el siguiente circuito:

```
bell_circuit = (
    Circuit()
    .rx(0, math.pi/2)
    .rx(1, math.pi/2)
    .iswap(0, 1)
    .rx(1, -math.pi/2)
)
```

Puede ejecutarlo con una calibración de puerta personalizada para la puerta `rx` en `qubit 0` pasando un diccionario de objetos `PulseSequence` al argumento de la palabra clave `gate_definitions`. Puede construir un diccionario a partir del atributo `pulse_sequences` del

objeto `GateCalibrations`. Todas las puertas no especificadas se sustituyen por la calibración de pulso del proveedor de hardware cuántico.

```
nb_shots = 50
custom_calibration = GateCalibrations({rx_pi_2_q0: pulse_sequence_rx_pi_2_q0})
task = device.run(bell_circuit, gate_definitions=custom_calibration.pulse_sequences,
 shots=nb_shots)
```

Simulación hamiltoniana analógica

La simulación hamiltoniana analógica (AHS) es un paradigma emergente en la computación cuántica que difiere significativamente del modelo de circuito cuántico tradicional. En lugar de una secuencia de puertas, en la que cada circuito actúa solo sobre un par de qubits a la vez. Un programa de AHS se define por los parámetros dependientes del tiempo y del espacio del hamiltoniano en cuestión. El hamiltoniano de un sistema codifica sus niveles de energía y los efectos de las fuerzas externas, que juntos gobiernan la evolución temporal de sus estados. Para un sistema de N qubits, el hamiltoniano puede representarse mediante una matriz cuadrada $2^N X 2^N$ de números complejos.

Los dispositivos cuánticos capaces de realizar AHS están diseñados para aproximarse con gran precisión a la evolución temporal de un sistema cuántico bajo un hamiltoniano personalizado, ajustando cuidadosamente sus parámetros de control internos. Por ejemplo, ajustando los parámetros de amplitud y desintonización de un campo de excitación coherente. El paradigma AHS es muy adecuado para simular las propiedades estáticas y dinámicas de los sistemas cuánticos con muchas partículas en interacción, como en la física de la materia condensada o la química cuántica. Las unidades de procesamiento cuántico diseñadas específicamente (QPUs), como el dispositivo Aquila deQuEra, se han desarrollado para utilizar el poder del AHS y abordar problemas que están fuera del alcance de los enfoques convencionales de computación cuántica digital de manera innovadora.

En esta sección:

- Hola AHS: ejecución de la primera simulación hamiltoniana analógica
- Envíe un programa analógico con Aquila QuEra

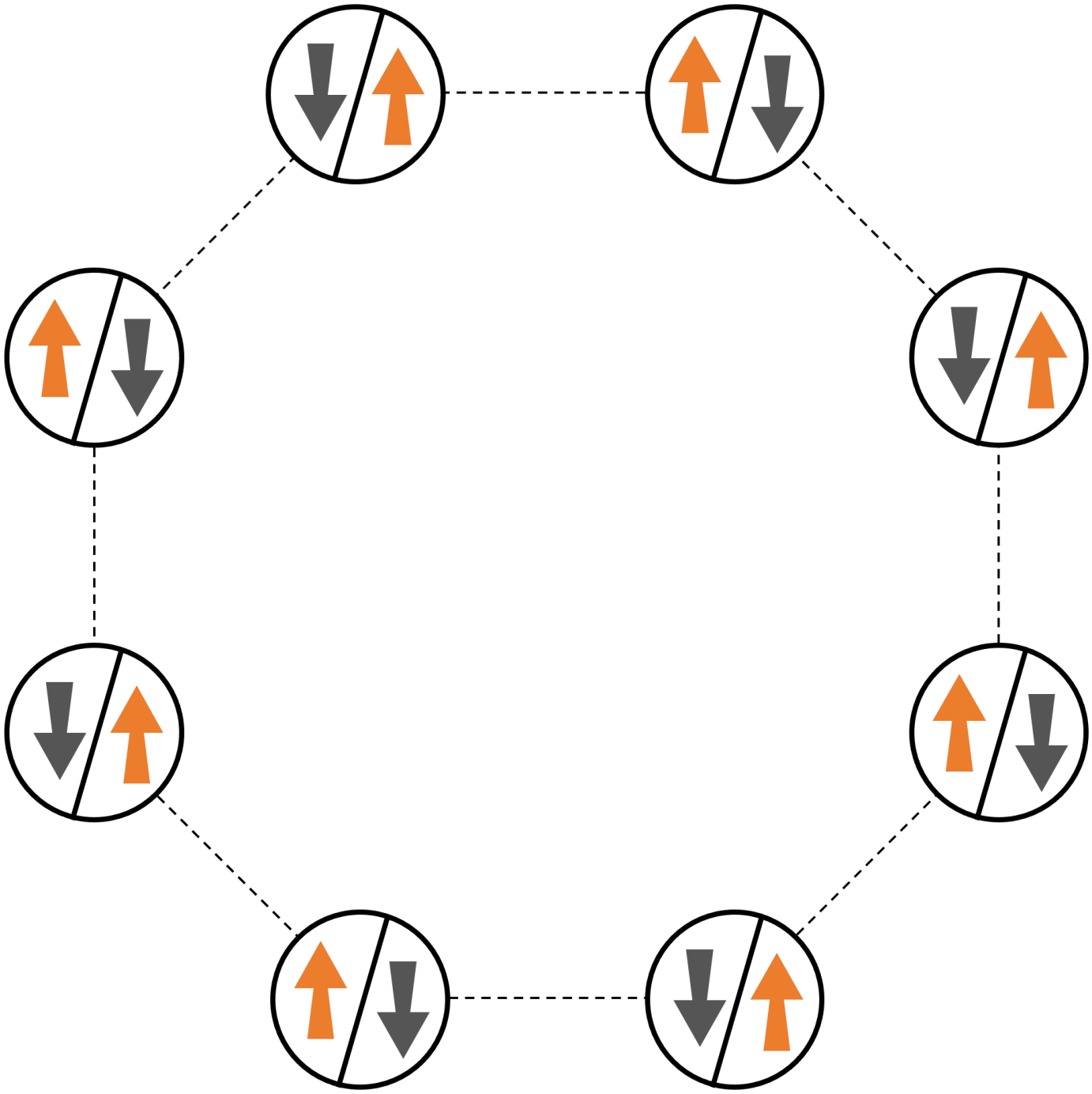
Hola AHS: ejecución de la primera simulación hamiltoniana analógica

En esta sección se proporciona información sobre cómo ejecutar su primera simulación hamiltoniana analógica.

En esta sección:

Cadena de espín interactiva

Como ejemplo canónico de un sistema de muchas partículas que interactúan, consideremos un anillo de ocho espines (cada uno de los cuales puede estar en estados «arriba» y «abajo»). Aunque pequeño, este sistema modelo ya muestra una serie de fenómenos interesantes propios de los materiales magnéticos naturales. En este ejemplo, mostraremos cómo preparar un orden denominado antiferromagnético, en el que los espines consecutivos apuntan en direcciones opuestas.



Disposición

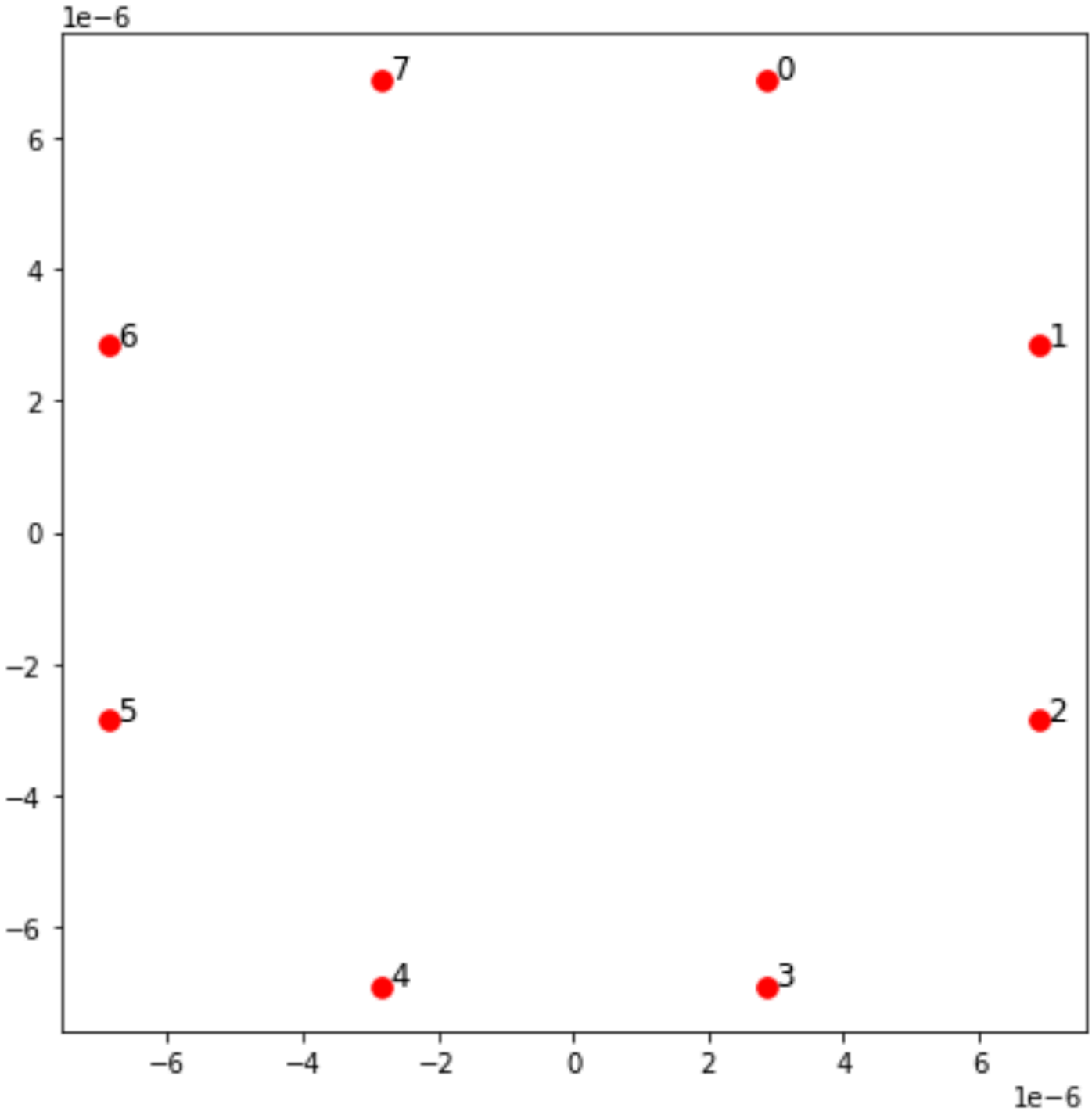
Utilizaremos un átomo neutro para representar cada espín, y los estados de espín «arriba» y «abajo» se codificarán en el estado excitado de Rydberg y el estado fundamental de los átomos, respectivamente. En primer lugar, crearemos la disposición 2-D. Podemos programar el anillo de espines anterior con el siguiente código.

Requisitos previos: es necesario instalar el SDK de Braket. (Si utiliza una instancia de cuaderno alojada en Braket, este SDK viene preinstalado con los cuadernos). Para reproducir los gráficos, también debe instalar matplotlib por separado con el intérprete de comandos `pip install matplotlib`.

```
from braket.ahs.atom_arrangement import AtomArrangement
import numpy as np
import matplotlib.pyplot as plt  # Required for plotting

a = 5.7e-6  # Nearest-neighbor separation (in meters)

register = AtomArrangement()
register.add(np.array([0.5, 0.5 + 1/np.sqrt(2)]) * a)
register.add(np.array([0.5 + 1/np.sqrt(2), 0.5]) * a)
register.add(np.array([0.5 + 1/np.sqrt(2), - 0.5]) * a)
register.add(np.array([0.5, - 0.5 - 1/np.sqrt(2)]) * a)
register.add(np.array([-0.5, - 0.5 - 1/np.sqrt(2)]) * a)
register.add(np.array([-0.5 - 1/np.sqrt(2), - 0.5]) * a)
register.add(np.array([-0.5 - 1/np.sqrt(2), 0.5]) * a)
register.add(np.array([-0.5, 0.5 + 1/np.sqrt(2)]) * a)
```

que también podemos representar gráficamente con:

```
fig, ax = plt.subplots(1, 1, figsize=(7, 7))
xs, ys = [register.coordinate_list(dim) for dim in (0, 1)]
ax.plot(xs, ys, 'r.', ms=15)

for idx, (x, y) in enumerate(zip(xs, ys)):
    ax.text(x, y, f" {idx}", fontsize=12)

plt.show()  # This will show the plot below in an ipython or jupyter session
```



Interacción

Para preparar la fase antiferromagnética, necesitamos inducir interacciones entre espines vecinos. Para ello utilizamos la interacción de van der Waals, que se implementa de forma nativa mediante dispositivos de átomos neutros (como el dispositivo Aquila de QuEra). Utilizando la representación de

espines, el término hamiltoniano para esta interacción puede expresarse como una suma de todos los pares de espines (j, k).

$$H_{\text{interaction}} = \sum_{j=1}^{N-1} \sum_{k=j+1}^{N} V_{j,k} n_j n_k$$

Aquí, nj=|↑$_j$##↑$_j$|es un operador que toma el valor 1 solo si el espín j está en el estado «arriba», y 0 en caso contrario. La fuerza es $V_{j,k}=C_6/(d_{j,k})^6$, donde C_6 es el coeficiente fijo y $d_{j,k}$ es la distancia euclidiana entre los espines j y k. El efecto inmediato de este término de interacción es que cualquier estado en el que tanto el espín j como el espín k estén «arriba» tienen una energía elevada (con la cantidad $V_{j,k}$). Al diseñar cuidadosamente el resto del programa de AHS, esta interacción evitará que los espines vecinos se encuentren ambos en el estado «arriba», un efecto conocido comúnmente como «bloqueo de Rydberg».

Campo de accionamiento

Al inicio del programa de AHS, todos los espines (por defecto) comienzan en su estado «abajo», es decir, se encuentran en la denominada fase ferromagnética. Con la mirada puesta en nuestro objetivo de preparar la fase antiferromagnética, especificamos un campo de excitación coherente dependiente del tiempo que hace que los espines pasen suavemente de este estado a un estado de muchos cuerpos en el que se prefieren los estados «arriba». El hamiltoniano correspondiente se puede escribir como:

$$H_{\text{drive}}(t) = \sum_{k=1}^{N} \frac{1}{2}\Omega(t)\left[e^{i\phi(t)} S_{-,k} + e^{-i\phi(t)} S_{+,k}\right] - \sum_{k=1}^{N} \Delta(t) n_k$$

donde Ω(t),ϕ(t),Δ(t) son la amplitud global dependiente del tiempo (también conocida como frecuencia de Rabi), la fase y la desintonización del campo de accionamiento que afecta a todos los espines de manera uniforme. Aquí, $S_{-,k}$=|↓$_k$##↑$_k$|and $S_{+,k}=(S_{-,k})^{\dagger}$=|↑$_k$##↓$_k$|son los operadores de bajada y subida del espín k, respectivamente, y n_k=|↑$_k$##↑$_k$|es el mismo operador que antes. La parte Ω del campo de accionamiento acopla de forma coherente los estados «abajo» y «arriba» de todos los espines simultáneamente, mientras que la parte Δ controla la recompensa energética para los estados «arriba».

Para programar una transición suave de la fase ferromagnética a la fase antiferromagnética, especificamos el campo de excitación con el siguiente código.

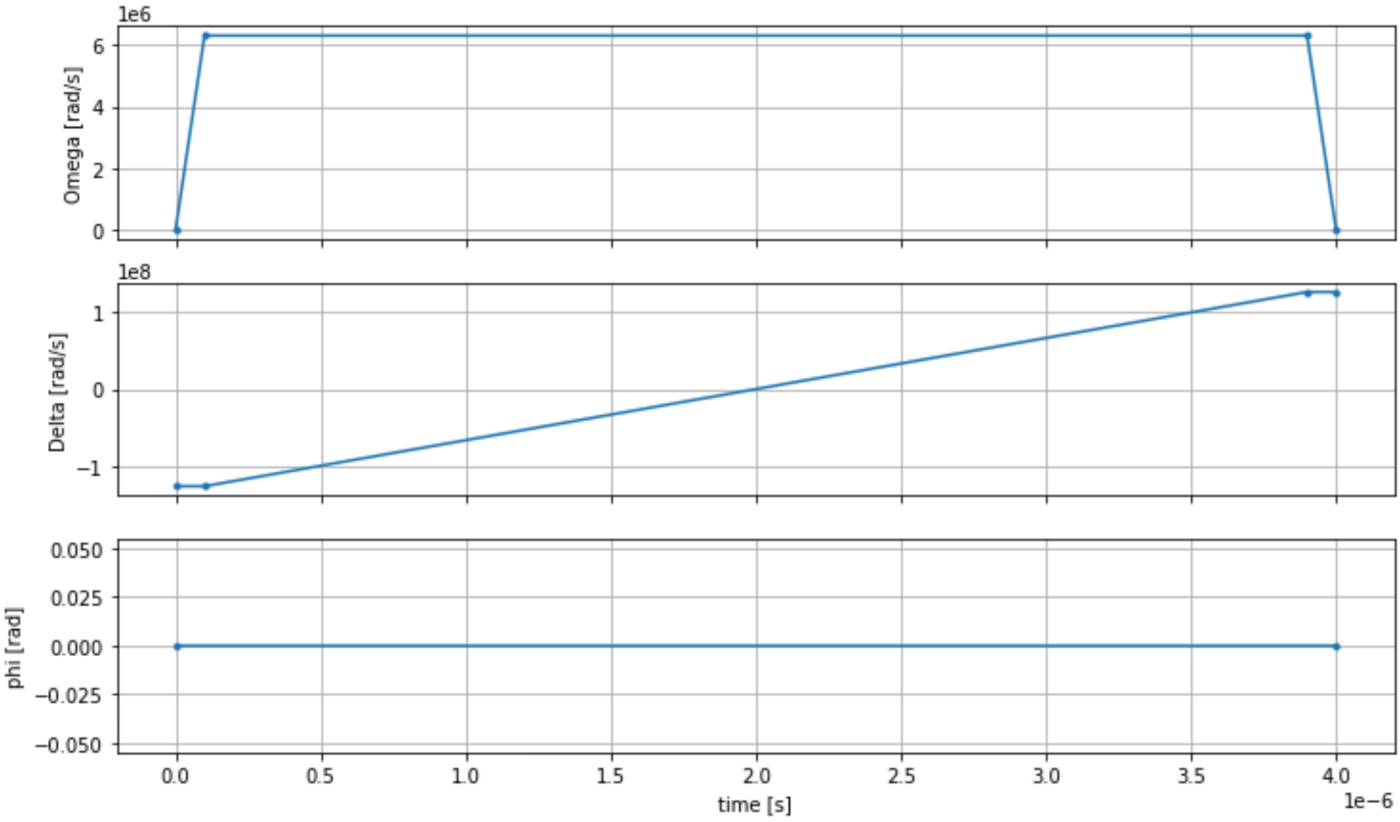
```
from braket.timings.time_series import TimeSeries
```

```
from braket.ahs.driving_field import DrivingField

# Smooth transition from "down" to "up" state
time_max = 4e-6  # seconds
time_ramp = 1e-7  # seconds
omega_max = 6300000.0  # rad / sec
delta_start = -5 * omega_max
delta_end = 5 * omega_max

omega = TimeSeries()
omega.put(0.0, 0.0)
omega.put(time_ramp, omega_max)
omega.put(time_max - time_ramp, omega_max)
omega.put(time_max, 0.0)

delta = TimeSeries()
delta.put(0.0, delta_start)
delta.put(time_ramp, delta_start)
delta.put(time_max - time_ramp, delta_end)
delta.put(time_max, delta_end)

phi = TimeSeries().put(0.0, 0.0).put(time_max, 0.0)

drive = DrivingField(
   amplitude=omega,
   phase=phi,
   detuning=delta
)
```

Podemos visualizar la serie temporal del campo de accionamiento con el siguiente script.

```
fig, axes = plt.subplots(3, 1, figsize=(12, 7), sharex=True)

ax = axes[0]
time_series = drive.amplitude.time_series
ax.plot(time_series.times(), time_series.values(), '.-')
ax.grid()
ax.set_ylabel('Omega [rad/s]')

ax = axes[1]
time_series = drive.detuning.time_series
ax.plot(time_series.times(), time_series.values(), '.-')
ax.grid()
```

```
ax.set_ylabel('Delta [rad/s]')

ax = axes[2]
time_series = drive.phase.time_series
# Note: time series of phase is understood as a piecewise constant function
ax.step(time_series.times(), time_series.values(), '.-', where='post')
ax.set_ylabel('phi [rad]')
ax.grid()
ax.set_xlabel('time [s]')

plt.show()  # This will show the plot below in an ipython or jupyter session
```



Programa de AHS

El registro, el campo de accionamiento (y las interacciones implícitas de van der Waals) conforman el programa de simulación hamiltoniana analógica `ahs_program`.

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

ahs_program = AnalogHamiltonianSimulation(
   register=register,
```

```
   hamiltonian=drive
)
```

Ejecución en un simulador local

Dado que este ejemplo es pequeño (menos de 15 espines), antes de ejecutarlo en una QPU compatible con AHS, podemos ejecutarlo en el simulador AHS local que viene con el SDK de Braket. Puesto que el simulador local está disponible de forma gratuita con el SDK de Braket, esta es la mejor práctica para garantizar que nuestro código se ejecute correctamente.

Aquí, podemos establecer el número de shots en un valor alto (por ejemplo, 1 millón) porque el simulador local realiza un rastreo de la evolución temporal del estado cuántico y extrae muestras del estado final; por lo tanto, al aumentar el número de shots, el tiempo total de ejecución solo aumenta ligeramente.

```
from braket.devices import LocalSimulator

device = LocalSimulator("braket_ahs")

result_simulator = device.run(
   ahs_program,
   shots=1_000_000
).result()  # Takes about 5 seconds
```

Análisis de resultados de simuladores

Podemos agregar los resultados de los lanzamientos con la siguiente función que infiere el estado de cada espín (que puede ser «d» para «abajo», «u» para «arriba» o «e» para sitio vacío) y cuenta cuántas veces se produjo cada configuración en los shots.

```
from collections import Counter


def get_counts(result):
    """Aggregate state counts from AHS shot results

    A count of strings (of length = # of spins) are returned, where
    each character denotes the state of a spin (site):
      e: empty site
      u: up state spin
```

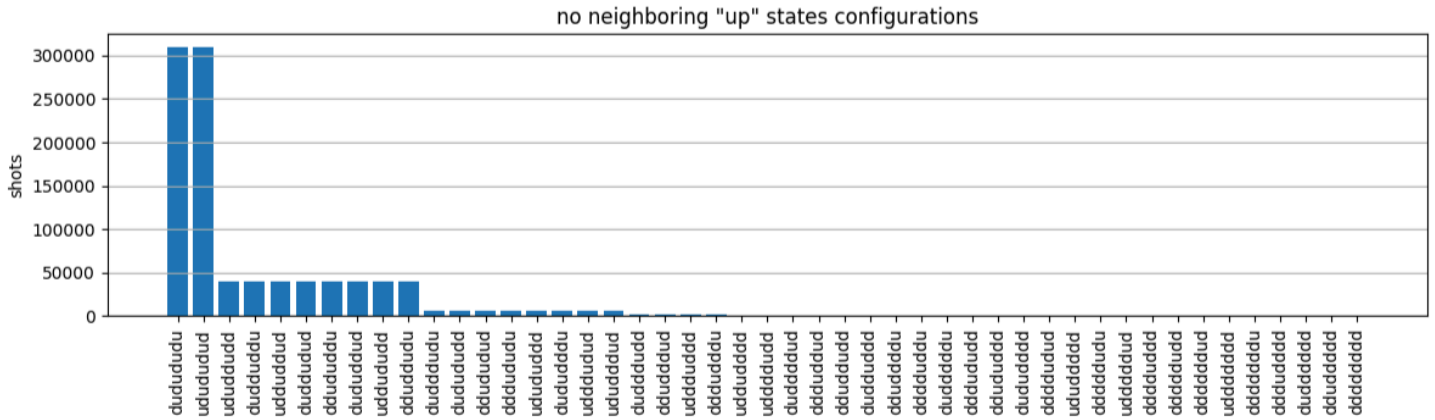
```
      d: down state spin

    Args:
      result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

    Returns
       dict: number of times each state configuration is measured

    """
    state_counts = Counter()
    states = ['e', 'u', 'd']
    for shot in result.measurements:
        pre = shot.pre_sequence
        post = shot.post_sequence
        state_idx = np.array(pre) * (1 + np.array(post))
        state = "".join(map(lambda s_idx: states[s_idx], state_idx))
        state_counts.update((state,))
    return dict(state_counts)


counts_simulator = get_counts(result_simulator)  # Takes about 5 seconds
print(counts_simulator)
```

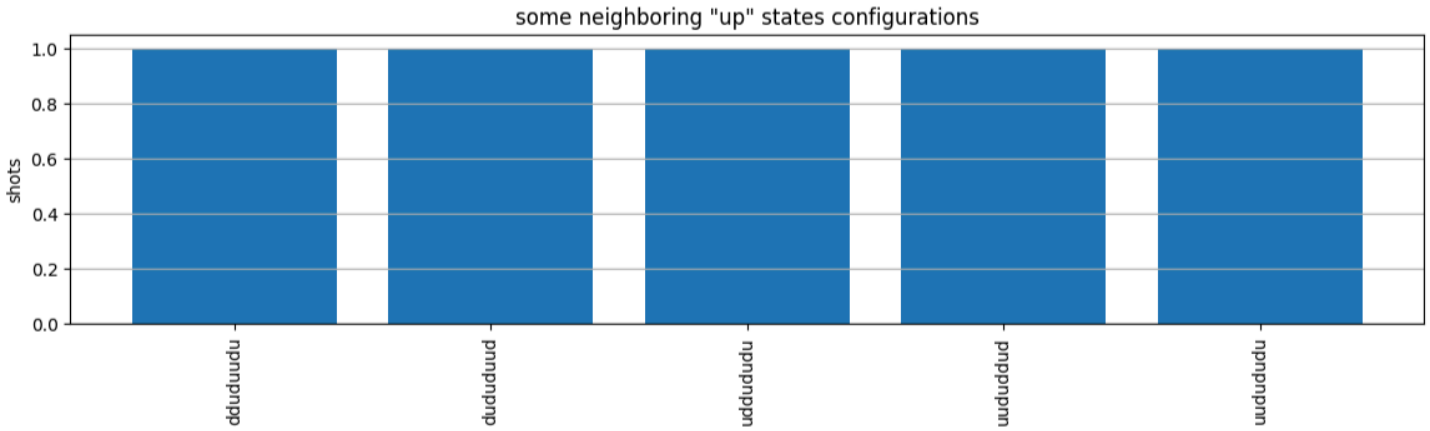
```
*[Output]*
{'dddddddd': 5, 'dddddddu': 12, 'ddddddud': 15, ...}
```

Este `counts` es un diccionario que cuenta el número de veces que se observa cada configuración de estado en los shots. También podemos visualizarlos con el siguiente código.

```
from collections import Counter


def has_neighboring_up_states(state):
    if 'uu' in state:
        return True
    if state[0] == 'u' and state[-1] == 'u':
        return True
    return False


def number_of_up_states(state):
    return Counter(state)['u']
```

```
def plot_counts(counts):
    non_blockaded = []
    blockaded = []
    for state, count in counts.items():
        if not has_neighboring_up_states(state):
            collection = non_blockaded
        else:
            collection = blockaded
        collection.append((state, count, number_of_up_states(state)))

    blockaded.sort(key=lambda _: _[1], reverse=True)
    non_blockaded.sort(key=lambda _: _[1], reverse=True)

    for configurations, name in zip((non_blockaded,
                                     blockaded),
                                    ('no neighboring "up" states',
                                     'some neighboring "up" states')):
        plt.figure(figsize=(14, 3))
        plt.bar(range(len(configurations)), [item[1] for item in configurations])
        plt.xticks(range(len(configurations)))
        plt.gca().set_xticklabels([item[0] for item in configurations], rotation=90)
        plt.ylabel('shots')
        plt.grid(axis='y')
        plt.title(f'{name} configurations')
        plt.show()


plot_counts(counts_simulator)
```





A partir de los gráficos, podemos extraer las siguientes observaciones que confirman que hemos preparado correctamente la fase antiferromagnética.

1. Por lo general, los estados no bloqueados (en los que no hay dos espines vecinos en estado «arriba») son más comunes que los estados en los que al menos un par de espines vecinos se encuentran ambos en estado «arriba».
2. Normalmente, se prefieren los estados con más excitaciones «arriba», a menos que la configuración esté bloqueada.
3. Los estados más comunes son, de hecho, los estados antiferromagnéticos perfectos `"dudududu"` y `"udududud"`.
4. Los segundos estados más comunes son aquellos en los que solo hay tres excitaciones «arriba» con separaciones consecutivas de 1, 2, 2. Esto demuestra que la interacción de van der Waals también afecta (aunque en menor medida) a los vecinos más cercanos.

Se ejecuta en la QuEra QPU Aquila

Requisitos previos: además de instalar el SDK de Braket con pip, si es nuevo en Amazon Braket, asegúrese de haber completado los pasos de introducción.

> **Note**
>
> Si utiliza una instancia de cuaderno alojada en Braket, el SDK de Braket viene preinstalado con la instancia.

Con todas las dependencias instaladas, podemos conectarnos a la QPU de Aquila.

```
from braket.aws import AwsDevice

aquila_qpu = AwsDevice("arn:aws:braket:us-east-1::device/qpu/quera/Aquila")
```

Para que nuestro programa de AHS sea adecuado para la máquina de QuEra, necesitamos redondear todos los valores para que cumplan con los niveles de precisión permitidos por la QPU de Aquila. (Estos requisitos se rigen por los parámetros del dispositivo con «Resolución» en su nombre. Podemos verlos al ejecutar `aquila_qpu.properties.dict()` en un cuaderno. Para obtener más detalles sobre las capacidades y requisitos de Aquila, consulte el cuaderno Introducción a Aquila). Podemos hacerlo llamando al método `discretize`.

```
discretized_ahs_program = ahs_program.discretize(aquila_qpu)
```

Ahora podemos ejecutar el programa (ejecutando solo 100 shots de momento) en la QPU de Aquila.

> **Note**
>
> La ejecución de este programa en el procesador Aquila tendrá un costo. El SDK de Amazon Braket incluye un Rastreador de costos que permite a los clientes establecer límites de costos y realizar el seguimiento de sus costos casi en tiempo real.

```
task = aquila_qpu.run(discretized_ahs_program, shots=100)

metadata = task.metadata()
task_arn = metadata['quantumTaskArn']
task_status = metadata['status']

print(f"ARN: {task_arn}")
print(f"status: {task_status}")
```

```
*[Output]*
ARN: arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef
status: CREATED
```

Debido a la gran variación en el tiempo que puede tardar en ejecutarse una tarea cuántica (dependiendo de los periodos de disponibilidad y la utilización de la QPU), es recomendable

anotar el ARN de la tarea cuántica, para poder comprobar su estado más adelante con el siguiente fragmento de código.

```
# Optionally, in a new python session
from braket.aws import AwsQuantumTask

SAVED_TASK_ARN = "arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef"

task = AwsQuantumTask(arn=SAVED_TASK_ARN)
metadata = task.metadata()
task_arn = metadata['quantumTaskArn']
task_status = metadata['status']

print(f"ARN: {task_arn}")
print(f"status: {task_status}")
```

```
*[Output]*
ARN: arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef
status: COMPLETED
```

Una vez que el estado sea COMPLETADO (lo cual también se puede comprobar desde la página de tareas cuánticas de la consola de Amazon Braket), podemos consultar los resultados con:

```
result_aquila = task.result()
```

Análisis de los resultados de la QPU

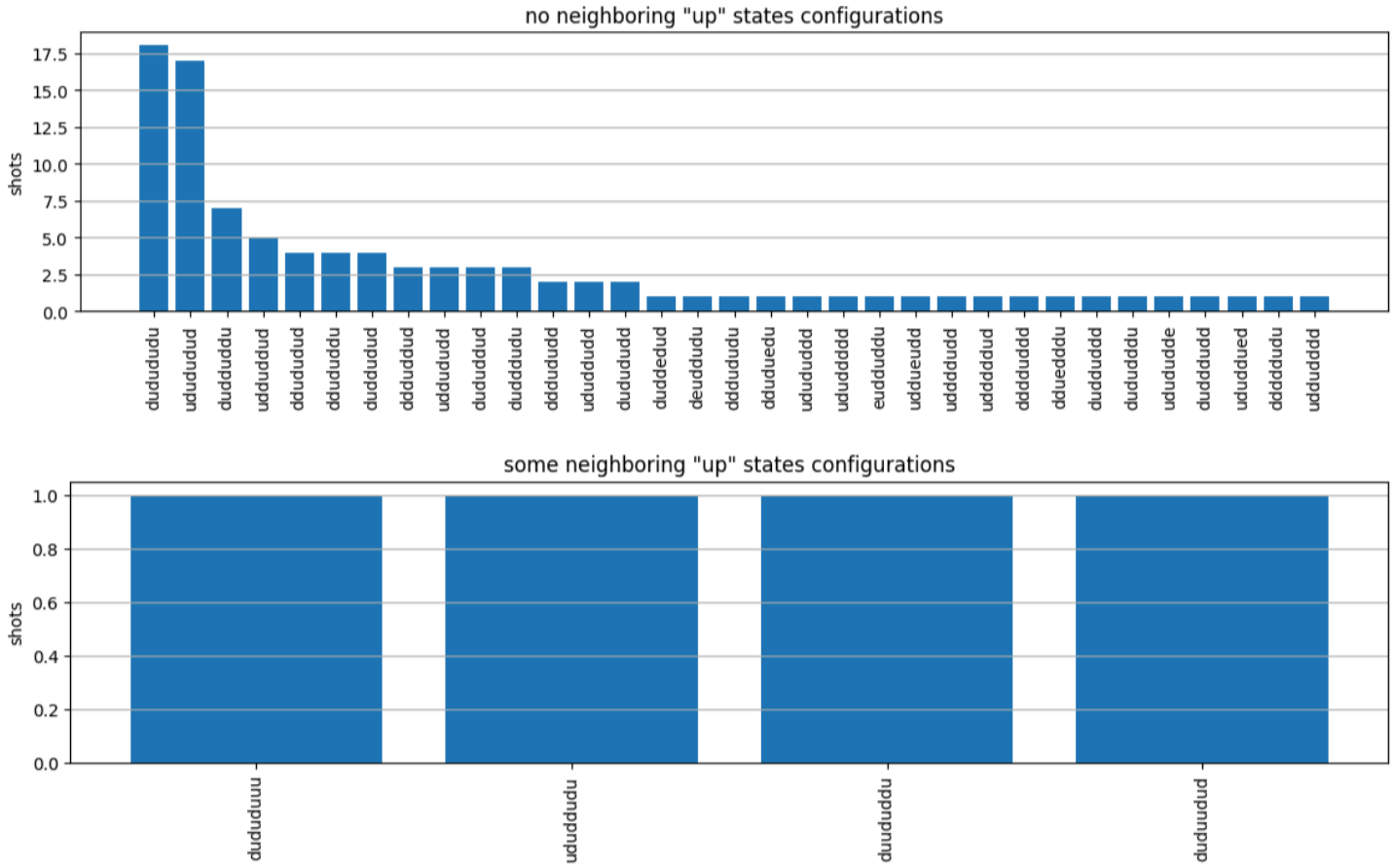
Usando las mismas funciones `get_counts` que antes, podemos computar los recuentos:

```
counts_aquila = get_counts(result_aquila)
   print(counts_aquila)
```

```
*[Output]*
{'ddddudud': 2, 'dudududu': 18, 'dddududud': 4, ...}
```

y representarlos gráficamente con `plot_counts`:

```
plot_counts(counts_aquila)
```



Tenga en cuenta que una pequeña fracción de los shots tiene sitios vacíos (marcados con una «e»). Esto se debe a que la QPU de Aquila presenta imperfecciones en la preparación de un 1-2 % por átomo. Además, los resultados coinciden con los de la simulación dentro de la fluctuación estadística esperada debido al reducido número de shots.

Siguientes pasos

Enhorabuena, ya ha ejecutado su primera carga de trabajo de AHS en Amazon Braket con el simulador AHS local y la QPU de Aquila.

Para obtener más información sobre la física de Rydberg, la simulación hamiltoniana analógica y el dispositivo Aquila, consulte nuestros cuadernos de ejemplos.

Envíe un programa analógico con Aquila QuEra

Esta página incluye documentación completa sobre las capacidades de la máquina Aquila de QuEra. Los detalles que se tratan aquí son los siguientes:

1. El hamiltoniano parametrizado simulado por Aquila

2. Parámetros del programa de AHS
3. Contenido de resultados de AHS
4. Parámetro de capacidades de Aquila

En esta sección:

- Hamiltoniano
- Esquema del programa de AHS de Braket
- Esquema de resultados de tareas de AHS de Braket
- QuEra esquema de propiedades del dispositivo

Hamiltoniano

La máquina Aquila de QuEra simula el siguiente hamiltoniano (dependiente del tiempo) de forma nativa:

$$H(t) = \sum_{k=1}^{N} H_{\text{drive},k}(t) + \sum_{k=1}^{N} H_{\text{local detuning},k}(t) + \sum_{k=1}^{N-1} \sum_{l=k+1}^{N} V_{\text{vdw},k,l}$$

> **Note**
>
> El acceso a la desintonización local es una capacidad experimental y está disponible previa solicitud a través de Braket Direct.

donde

- $H_{\text{drive},k}(t) = (\,{}^{1}/_{2}\,\Omega(t)e^{i\phi(t)}S_{-,k} + {}^{1}/_{2}\,\Omega(t)e^{-i\phi(t)}\,S_{+,k}) + (-\Delta_{\text{global}}(t)n_k)$,
  - Ω(t) es la amplitud de excitación global dependiente del tiempo (también conocida como frecuencia de Rabi), en unidades de (rad / s)
  - φ (t) es la fase global dependiente del tiempo, medida en radianes
  - $S_{-,k}$ y $S_{+,k}$ son los operadores de reducción y aumento del espín del átomo k (en la base |↓#=|g#, |↑#=|r#, son S_{-}=|g##r|, $S_{+}=(S_{-})^{\dagger}$=|r##g|)
  - $\Delta_{\text{global}}(t)$ es la desintonización global dependiente del tiempo
  - n_k es el operador de proyección sobre el estado de Rydberg del átomo k (es decir, n=|r##r|)
- $H_{\text{local detuning},k}(t) = -\Delta_{\text{local}}(t)h_k n_k$

  - $\Delta_{local}(t)$ es el factor dependiente del tiempo del desplazamiento de frecuencia local, en unidades de (rad / s)
  - h_k es el factor dependiente del emplazamiento, un número adimensional entre 0,0 y 1,0
- $V_{vdw,k,l}=C_6/(d_{k,l})^6 n_k n_l$,
  - C_6 es el coeficiente de van der Waals en unidades de (rad / s) * (m)^6
  - $d_{k,l}$ es la distancia euclidiana entre los átomos k y l, medida en metros.

Los usuarios tienen control sobre los siguientes parámetros a través del esquema del programa de AHS de Braket.

- Disposición de átomos en 2-d (coordenadas x_k e y_k de cada átomo k, en unidades de um), que controla las distancias atómicas por pares $d_{k,l}$ con k,l=1,2,…N
- $\Omega(t)$, la frecuencia Rabi global dependiente del tiempo en unidades de (rad / s)
- $\phi(t)$, la fase global dependiente del tiempo en unidades de (rad)
- $\Delta_{global}(t)$, la desintonización global dependiente del tiempo en unidades de (rad / s)
- $\Delta_{local}(t)$, el factor (global) dependiente del tiempo de la magnitud de la desintonización local en unidades de (rad / s)
- h_k, el factor (estático) dependiente del emplazamiento de la magnitud de la desintonización local, un número adimensional entre 0,0 y 1,0

> **Note**
>
> El usuario no puede controlar qué niveles intervienen (es decir, los operadores S_-, S_+ y n son fijos) ni la intensidad del coeficiente de interacción Rydberg-Rydberg (C_6).

Esquema del programa de AHS de Braket

Objeto braket.ir.ahs.program_v1.Program (ejemplo)

> **Note**
>
> Si la característica desintonización local no está habilitada para su cuenta, use `localDetuning=[]` en el siguiente ejemplo.

```
Program(
    braketSchemaHeader=BraketSchemaHeader(
        name='braket.ir.ahs.program',
        version='1'
    ),
    setup=Setup(
        ahs_register=AtomArrangement(
            sites=[
                [Decimal('0'), Decimal('0')],
                [Decimal('0'), Decimal('4e-6')],
                [Decimal('4e-6'), Decimal('0')]
            ],
            filling=[1, 1, 1]
        )
    ),
    hamiltonian=Hamiltonian(
        drivingFields=[
            DrivingField(
                amplitude=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('15700000.0'),
 Decimal('15700000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
 Decimal('0.000003')]
                    ),
                    pattern='uniform'
                ),
                phase=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000003')]
                    ),
                    pattern='uniform'
                ),
                detuning=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('-54000000.0'), Decimal('54000000.0')],
                        times=[Decimal('0'), Decimal('0.000003')]
                    ),
                    pattern='uniform'
                )
            )
        ],
```

```
        localDetuning=[
            LocalDetuning(
                magnitude=PhysicalField(
                    times_series=TimeSeries(
                        values=[Decimal('0'), Decimal('25000000.0'),
 Decimal('25000000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
 Decimal('0.000003')]
                    ),
                    pattern=Pattern([Decimal('0.8'), Decimal('1.0'), Decimal('0.9')])
                )
            )
        ]
    )
)
```

JSON (ejemplo)

> **Note**
>
> Si la característica desintonización local no está habilitada para su cuenta, use `"localDetuning": []` en el siguiente ejemplo.

```
{
    "braketSchemaHeader": {
        "name": "braket.ir.ahs.program",
        "version": "1"
    },
    "setup": {
        "ahs_register": {
            "sites": [
                [0E-7, 0E-7],
                [0E-7, 4E-6],
                [4E-6, 0E-7]
            ],
            "filling": [1, 1, 1]
        }
    },
    "hamiltonian": {
        "drivingFields": [
            {
```

```
                "amplitude": {
                    "time_series": {
                        "values": [0.0, 15700000.0, 15700000.0, 0.0],
                        "times": [0E-9, 0.000001000, 0.000002000, 0.000003000]
                    },
                    "pattern": "uniform"
                },
                "phase": {
                    "time_series": {
                        "values": [0E-7, 0E-7],
                        "times": [0E-9, 0.000003000]
                    },
                    "pattern": "uniform"
                },
                "detuning": {
                    "time_series": {
                        "values": [-54000000.0, 54000000.0],
                        "times": [0E-9, 0.000003000]
                    },
                    "pattern": "uniform"
                }
            }
        ],
        "localDetuning": [
            {
                "magnitude": {
                    "time_series": {
                        "values": [0.0, 25000000.0, 25000000.0, 0.0],
                        "times": [0E-9, 0.000001000, 0.000002000, 0.000003000]
                    },
                    "pattern": [0.8, 1.0, 0.9]
                }
            }
        ]
    }
}
```

Campos principales

Campo de programa	type	description
setup.ahs_register.sites	List[List[Decimal]]	Lista de coordenadas 2-d donde las

Campo de programa	type	description
		pinzas atrapan átomos
setup.ahs_register.filling	List[int]	Marca los átomos que ocupan los emplazamientos de la trampa con 1 y los emplazami entos vacíos con 0
hamiltonian.drivingFields[].amplitude.time_series.times	List[Decimal]	puntos temporales de la amplitud de accionamiento, Omega(t)
hamiltonian.drivingFields[].amplitude.time_series.values	List[Decimal]	valores de la amplitud de accionamiento, Omega(t)
hamiltonian.drivingFields[].amplitude.pattern	str	patrón espacial de la amplitud de conducción, Omega(t); debe ser «uniform»
hamiltonian.drivingFields[].phase.time_series.times	List[Decimal]	puntos temporales de la fase de accionamiento, phi(t)
hamiltonian.drivingFields[].phase.time_series.values	List[Decimal]	valores de la fase de accionamiento, phi(t)

Campo de programa	type	description
hamiltonian.drivingFields[].phase.pattern	str	patrón espacial de la fase de accionamiento, phi(t); debe ser «uniform»
hamiltonian.drivingFields[].detuning.time_series.times	List[Decimal]	puntos temporales de desintonización de accionamiento, Delta_global(t)
hamiltonian.drivingFields[].detuning.time_series.values	List[Decimal]	valores de desintonización de accionamiento, Delta_global(t)
hamiltonian.drivingFields[].detuning.pattern	str	patrón espacial de desintonización de conducción, Delta_global(t); debe ser «uniforme»
hamiltonian.localDetuning[].magnitude.time_series.times	List[Decimal]	puntos temporales del factor dependiente del tiempo de la magnitud de desintonización local, Delta_local(t)

Campo de programa	type	description
hamiltonian.localDetuning[].magnitude.time_series.values	List[Decimal]	valores del factor dependiente del tiempo de la magnitud de desintonización local, Delta_local(t)
hamiltonian.localDetuning[].magnitude.pattern	List[Decimal]	factor dependiente del emplazamiento de la magnitud de desintonización local, h_k (los valores corresponden a los emplazamientos en setup.ahs_register.sites)

Campos de metadatos

Campo de programa	type	description
braketSchemaHeader.name	str	nombre del esquema; debe ser «braket.ir.ahs.program»
braketSchemaHeader.versión	str	versión del esquema

Esquema de resultados de tareas de AHS de Braket

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQuantumTaskResult(ejemplo)

```
AnalogHamiltonianSimulationQuantumTaskResult(
    task_metadata=TaskMetadata(
        braketSchemaHeader=BraketSchemaHeader(
```

```
            name='braket.task_result.task_metadata',
            version='1'
        ),
        id='arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef',
        shots=2,
        deviceId='arn:aws:braket:us-east-1::device/qpu/quera/Aquila',
        deviceParameters=None,
        createdAt='2022-10-25T20:59:10.788Z',
        endedAt='2022-10-25T21:00:58.218Z',
        status='COMPLETED',
        failureReason=None
    ),
    measurements=[
        ShotResult(
            status=<AnalogHamiltonianSimulationShotStatus.SUCCESS: 'Success'>,

            pre_sequence=array([1, 1, 1, 1]),
            post_sequence=array([0, 1, 1, 1])
        ),

        ShotResult(
            status=<AnalogHamiltonianSimulationShotStatus.SUCCESS: 'Success'>,

            pre_sequence=array([1, 1, 0, 1]),
            post_sequence=array([1, 0, 0, 0])
        )
    ]
)
```

JSON (ejemplo)

```
{
    "braketSchemaHeader": {
        "name": "braket.task_result.analog_hamiltonian_simulation_task_result",
        "version": "1"
    },
    "taskMetadata": {
        "braketSchemaHeader": {
            "name": "braket.task_result.task_metadata",
            "version": "1"
        },
```

```
        "id": "arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef",
        "shots": 2,
        "deviceId": "arn:aws:braket:us-east-1::device/qpu/quera/Aquila",

        "createdAt": "2022-10-25T20:59:10.788Z",
        "endedAt": "2022-10-25T21:00:58.218Z",
        "status": "COMPLETED"

    },
    "measurements": [
        {
            "shotMetadata": {"shotStatus": "Success"},
            "shotResult": {
                "preSequence": [1, 1, 1, 1],
                "postSequence": [0, 1, 1, 1]
            }
        },
        {
            "shotMetadata": {"shotStatus": "Success"},
            "shotResult": {
                "preSequence": [1, 1, 0, 1],
                "postSequence": [1, 0, 0, 0]
            }
        }
    ],
    "additionalMetadata": {
        "action": {...}
        "queraMetadata": {
            "braketSchemaHeader": {
                "name": "braket.task_result.quera_metadata",
                "version": "1"
            },
            "numSuccessfulShots": 100
        }
    }
}
```

Campos principales

Campo de resultados de la tarea	type	description
measurements[].shotResult.preSequence	List[int]	Bits de medición previos a la secuencia (uno por cada emplazamiento atómico) para cada shot: 0 si el emplazamiento está vacío, 1 si el emplazami ento está lleno, medidos antes de las secuencias de pulsos que ejecutan la evolución cuántica
measurements[].shotResult.postSequence	List[int]	Bits de medición posteriores a la secuencia para cada shot: 0 si el átomo se encuentra en estado de Rydberg o el emplazamiento está vacío, 1 si el átomo se encuentra en estado fundamental, medido al final de las secuencias de pulsos que ejecutan la evolución cuántica

Campos de metadatos

Campo de resultados de la tarea	type	description
braketSchemaHeader.name	str	nombre del esquema; debe ser «braket.t ask_resul t.analog_ hamiltoni an_simula tion_task _result»

Campo de resultados de la tarea	type	description
braketSchemaHeader.versión	str	versión del esquema
Metadatos de tareas. braketSchemaHeader.nombre	str	nombre del esquema; debe ser «braket.task_result.task_metadata»
Metadatos de la tarea. braketSchemaHeader.versión	str	versión del esquema
taskMetadata.id	str	El ID de la tarea cuántica. Para las tareas AWS cuánticas, esta es la tarea cuántica ARN.
taskMetadata.shots	int	El número de shots de la tarea cuántica
taskMetadata.shots.deviceId	str	El ID del dispositivo en el que se ejecutó la tarea cuántica. Para AWS los dispositivos, este es el ARN del dispositivo.

Campo de resultados de la tarea	type	description
taskMetadata.shots.createdAt	str	La marca de tiempo de la creación; el formato debe estar en el formato RFC3339 ISO-8601/ de cadena: MM:SS.SSSZ. YYYY-MM-D DTHH El valor predeterm inado es Ninguno.
taskMetadata.shots.endedAt	str	La marca de tiempo en la que finalizó la tarea cuántica; el formato debe estar en el formato ISO-8601/ de cadena: MM:SS.SSS Z. RFC3339 YYYY-MM-D DTHH El valor predeterm inado es Ninguno.

Campo de resultados de la tarea	type	description
taskMetadata.shots.status	str	El estado de la tarea cuántica (CREADA, EN COLA, EN EJECUCIÓN, COMPLETADA, FALLIDA). El valor predeterminado es Ninguno.
taskMetadata.shots.failureReason	str	El motivo del error de la tarea cuántica. El valor predeterminado es Ninguno.
additionalMetadata.action	braket.ir.ahs.program_v1.Program	(Consulte la sección Esquema del programa de AHS de Braket)
Metadatos adicionales. Acción. braketSchemaHeaderMetadatos.Nombre de quera.Metadatos	str	nombre del esquema; debe ser «braket.task_result.quera_metadata»

Campo de resultados de la tarea	type	description
Metadatos adicionales. Acción. braketSchemaHeader.Quera Metadata.Versión	str	versión del esquema
Metadatos adicionales. Acción. numSuccessfulShots	int	número de shots totalmente de éxito; debe ser igual al número de shots solicitado
measurements[].shotMetadata.shotStatus	int	El estado del shot (éxito, éxito parcial, fracaso) debe ser «Éxito».

QuEra esquema de propiedades del dispositivo

braket.device_schema.quera.quera_device_capabilities_v1. QueraDeviceCapabilities(ejemplo)

```
QueraDeviceCapabilities(
    service=DeviceServiceProperties(
        braketSchemaHeader=BraketSchemaHeader(
            name='braket.device_schema.device_service_properties',
            version='1'
            ),
            executionWindows=[
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.MONDAY: 'Monday'>,
                    windowStartHour=datetime.time(1, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.TUESDAY: 'Tuesday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
                ),
                DeviceExecutionWindow(
```

```
                    executionDay=<ExecutionDay.WEDNESDAY: 'Wednesday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.FRIDAY: 'Friday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.SATURDAY: 'Saturday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.SUNDAY: 'Sunday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
                )
            ],
            shotsRange=(1, 1000),
            deviceCost=DeviceCost(
                price=0.01,
                unit='shot'
            ),
            deviceDocumentation=
                DeviceDocumentation(
                    imageUrl='https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png',
                    summary='Analog quantum processor based on neutral atom arrays',
                    externalDocumentationUrl='https://www.quera.com/aquila'
                ),
                deviceLocation='Boston, USA',
                updatedAt=datetime.datetime(2024, 1, 22, 12, 0,
 tzinfo=datetime.timezone.utc),
                getTaskPollIntervalMillis=None
    ),
    action={
        <DeviceActionType.AHS: 'braket.ir.ahs.program'>: DeviceActionProperties(
                version=['1'],
                actionType=<DeviceActionType.AHS: 'braket.ir.ahs.program'>
            )
    },
```

```
    deviceParameters={},
    braketSchemaHeader=BraketSchemaHeader(
        name='braket.device_schema.quera.quera_device_capabilities',
        version='1'
    ),
    paradigm=QueraAhsParadigmProperties(
        ...
        # See https://github.com/amazon-braket/amazon-braket-schemas-python/blob/main/
src/braket/device_schema/quera/quera_ahs_paradigm_properties_v1.py
        ...
    )
)
```

JSON (ejemplo)

```
{
    "service": {
        "braketSchemaHeader": {
            "name": "braket.device_schema.device_service_properties",
            "version": "1"
        },
        "executionWindows": [
            {
                "executionDay": "Monday",
                "windowStartHour": "01:00:00",
                "windowEndHour": "23:59:59"
            },
            {
                "executionDay": "Tuesday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            },
            {
                "executionDay": "Wednesday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            },
            {
                "executionDay": "Friday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "23:59:59"
            },
            {
```

```
                "executionDay": "Saturday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "23:59:59"
            },
            {
                "executionDay": "Sunday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            }
        ],
        "shotsRange": [
            1,
            1000
        ],
        "deviceCost": {
            "price": 0.01,
            "unit": "shot"
        },
        "deviceDocumentation": {
            "imageUrl": "https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png",
            "summary": "Analog quantum processor based on neutral atom arrays",
            "externalDocumentationUrl": "https://www.quera.com/aquila"
        },
        "deviceLocation": "Boston, USA",
        "updatedAt": "2024-01-22T12:00:00+00:00"
    },
    "action": {
        "braket.ir.ahs.program": {
            "version": [
                "1"
            ],
            "actionType": "braket.ir.ahs.program"
        }
    },
    "deviceParameters": {},
    "braketSchemaHeader": {
        "name": "braket.device_schema.quera.quera_device_capabilities",
        "version": "1"
    },
    "paradigm": {
        ...
        # See Aquila device page > "Calibration" tab > "JSON" page
```

```
        ...
    }
}
```

Campos de propiedades de servicio

Campos de propiedades de servicio	type	description
service.executionWindows[].executionDay	ExecutionDay	Días del periodo de ejecución; debe ser «Todos los días», «Días laborables», «Fin de semana», «Lunes», «Martes», «Miércoles», «Jueves», «Viernes», «Sábado» o «Domingo»
Service.ExecutionWindows []. windowStartHour	datetime.time	Formato UTC de 24 horas de la hora en que comienza el periodo de ejecución
Service.ExecutionWindows []. windowEndHour	datetime.time	Formato UTC de 24 horas de la hora en que finaliza el periodo de ejecución
service.qpu_capabilities.service.shotsRange	Tuple[int, int]	Número mínimo y máximo de shots del dispositivo
service.qpu_capabilities.service.deviceCost.price	float	Precio del dispositivo en dólares estadounidenses
service.qpu_capabilities.service.deviceCost.unit	str	unidad para cobrar el precio, por ejemplo: «minuto», «hora», «disparo», «tarea»

Campos de metadatos

Campo de metadatos	type	description
action[].version	str	versión del esquema del programa de AHS
action[].actionType	ActionType	nombre del esquema del programa de AHS; debe ser «braket.ir.ahs.program»
servicio. braketSchemaHeader.nombre	str	nombre del esquema; debe ser «braket.device_schema.device_service_properties»
servicio. braketSchemaHeader.versión	str	versión del esquema
service.deviceDocumentation.imageUrl	str	URL de la imagen del dispositivo
service.deviceDocumentation.summary	str	breve descripción del dispositivo
Servicio. Documentación del dispositivo. externalDocumentationUrl	str	URL de documentación externa
service.deviceLocation	str	ubicación geográfica del dispositivo
service.updatedAt	datetime	hora en que se actualizaron las propiedades del dispositivo por última vez

Trabajando con AWS Boto3

Boto3 es el AWS SDK para Python. Con Boto3, los desarrolladores de Python pueden crear, configurar y administrar Servicios de AWS, como Amazon Braket. Boto3 proporciona una API orientada a objetos, así como acceso de bajo nivel a Amazon Braket.

Siga las instrucciones de la guía de inicio rápido de Boto3 para aprender a instalar y configurar Boto3.

Boto3 proporciona la funcionalidad básica que funciona junto con el SDK de Python de Amazon Braket para ayudarle a configurar y ejecutar sus tareas cuánticas. Los clientes de Python siempre deben instalar Boto3 porque esa es la implementación principal. Si desea utilizar métodos auxiliares adicionales, también debe instalar el SDK de Amazon Braket.

Por ejemplo, cuando llama a `CreateQuantumTask`, el SDK de Amazon Braket envía la solicitud a Boto3, que a continuación llama a la API de AWS .

En esta sección:

- Activar el cliente de Boto3 de Amazon Braket
- Configure los AWS CLI perfiles para Boto3 y el SDK de Braket

Activar el cliente de Boto3 de Amazon Braket

Para utilizar Boto3 con Amazon Braket, debe importar Boto3 y, a continuación, definir un cliente que utilice para conectarse a la API de Amazon Braket. En el siguiente ejemplo, el cliente de Boto3 se denomina `braket`.

```
import boto3
import botocore

braket = boto3.client("braket")
```

> **Note**
>
> Soportes para Braket. IPv6 Si utiliza una red IPv6 exclusiva o desea asegurarse de que su carga de trabajo utilice IPv6 tráfico, utilice los terminales de doble pila tal y como se describe en la guía de terminales de doble pila y FIPS.

Ahora que tiene un cliente de `braket` establecido, puede realizar solicitudes y procesar respuestas desde el servicio Amazon Braket. Puede obtener más detalles sobre los datos de solicitud y respuesta en la referencia de la API.

Los siguientes ejemplos muestran cómo trabajar con dispositivos y tareas cuánticas.

- Búsqueda de dispositivos
- Recuperación de un dispositivo
- Creación de una tarea cuántica
- Recuperación de una tarea cuántica
- Búsqueda de tareas cuánticas
- Cancelación de tarea cuántica

Búsqueda de dispositivos

- `search_devices(**kwargs)`

Buscar los dispositivos mediante los filtros especificados.

```
# Pass search filters and optional parameters when sending the
# request and capture the response
response = braket.search_devices(filters=[{
    'name': 'deviceArn',
    'values': ['arn:aws:braket:::device/quantum-simulator/amazon/sv1']
}], maxResults=10)

print(f"Found {len(response['devices'])} devices")

for i in range(len(response['devices'])):
    device = response['devices'][i]
    print(device['deviceArn'])
```

Recuperación de un dispositivo

- `get_device(deviceArn)`

Recuperar los dispositivos disponibles en Amazon Braket.

```
# Pass the device ARN when sending the request and capture the repsonse
response = braket.get_device(deviceArn='arn:aws:braket:::device/quantum-simulator/
amazon/sv1')

print(f"Device {response['deviceName']} is {response['deviceStatus']}")
```

Creación de una tarea cuántica

- `create_quantum_task(**kwargs)`

Crear una tarea cuántica.

```
# Create parameters to pass into create_quantum_task()
kwargs = {
    # Create a Bell pair
    'action': '{"braketSchemaHeader": {"name": "braket.ir.jaqcd.program", "version":
 "1"}, "results": [], "basis_rotation_instructions": [], "instructions": [{"type": "h",
 "target": 0}, {"type": "cnot", "control": 0, "target": 1}]}',
    # Specify the SV1 Device ARN
    'deviceArn': 'arn:aws:braket:::device/quantum-simulator/amazon/sv1',
    # Specify 2 qubits for the Bell pair
    'deviceParameters': '{"braketSchemaHeader": {"name":
 "braket.device_schema.simulators.gate_model_simulator_device_parameters",
 "version": "1"}, "paradigmParameters": {"braketSchemaHeader": {"name":
 "braket.device_schema.gate_model_parameters", "version": "1"}, "qubitCount": 2}}',
    # Specify where results should be placed when the quantum task completes.
    # You must ensure the S3 Bucket exists before calling create_quantum_task()
    'outputS3Bucket': 'amazon-braket-examples',
    'outputS3KeyPrefix': 'boto-examples',
    # Specify number of shots for the quantum task
    'shots': 100
}

# Send the request and capture the response
response = braket.create_quantum_task(**kwargs)

print(f"Quantum task {response['quantumTaskArn']} created")
```

Recuperación de una tarea cuántica

- `get_quantum_task(quantumTaskArn)`

Recuperar la tarea cuántica especificada.

```
# Pass the quantum task ARN when sending the request and capture the response
response = braket.get_quantum_task(quantumTaskArn='arn:aws:braket:us-
west-1:123456789012:quantum-task/ce78c429-cef5-45f2-88da-123456789012')

print(response['status'])
```

Búsqueda de tareas cuánticas

- `search_quantum_tasks(**kwargs)`

Buscar tareas cuánticas que coincidan con los valores de filtro especificados.

```
# Pass search filters and optional parameters when sending the
# request and capture the response
response = braket.search_quantum_tasks(filters=[{
    'name': 'deviceArn',
    'operator': 'EQUAL',
    'values': ['arn:aws:braket:::device/quantum-simulator/amazon/sv1']
}], maxResults=25)

print(f"Found {len(response['quantumTasks'])} quantum tasks")

for n in range(len(response['quantumTasks'])):
    task = response['quantumTasks'][n]
    print(f"Quantum task {task['quantumTaskArn']} for {task['deviceArn']} is
 {task['status']}")
```

Cancelación de tarea cuántica

- `cancel_quantum_task(quantumTaskArn)`

Cancelar la tarea cuántica especificada.

```
# Pass the quantum task ARN when sending the request and capture the response
response = braket.cancel_quantum_task(quantumTaskArn='arn:aws:braket:us-
west-1:123456789012:quantum-task/ce78c429-cef5-45f2-88da-123456789012')

print(f"Quantum task {response['quantumTaskArn']} is {response['cancellationStatus']}")
```

Configure los AWS CLI perfiles para Boto3 y el SDK de Braket

El SDK de Amazon Braket se basa en las AWS CLI credenciales predeterminadas, a menos que especifiques lo contrario de forma explícita. Le recomendamos que mantenga la configuración predeterminada cuando ejecute un cuaderno administrado de Amazon Braket, ya que debe proporcionar un rol de IAM que tenga permisos para iniciar la instancia del cuaderno.

Si lo desea, si ejecuta el código localmente (en una instancia de Amazon EC2, por ejemplo), puede establecer perfiles con nombre AWS CLI . Puede asignar a cada perfil un conjunto de permisos diferente, en lugar de sobrescribir regularmente el perfil predeterminado.

En esta sección se explica brevemente cómo configurar dicho `profile` de CLI y cómo incorporar ese perfil en Amazon Braket para que las llamadas a la API se realicen con los permisos de ese perfil.

En esta sección:

Paso 1: Configurar una AWS CLI local **`profile`**

Explicar cómo crear un usuario y cómo configurar un perfil no predeterminado está fuera del alcance de este documento. Para obtener más información sobre estos temas, consulte:

- Introducción
- Configurar el que se AWS CLI va a utilizar AWS IAM Identity Center

Para utilizar Amazon Braket, debe proporcionar a este usuario, y al `profile` de CLI asociado, los permisos necesarios de Braket. Por ejemplo, puede adjuntar la AmazonBraketFullAccesspolítica.

Paso 2: Establecer un objeto de sesión de Boto3

Para establecer un objeto de sesión de Boto3, utilice el siguiente ejemplo de código.

```
from boto3 import Session

# Insert CLI profile name here
```

```
boto_sess = Session(profile_name=`profile`)
```

> **Note**
>
> Si las llamadas a la API previstas tienen restricciones basadas en la región que no coinciden con su región de `profile` predeterminada, puede especificar una región para la sesión de Boto3, como se muestra en el siguiente ejemplo.

```
# Insert CLI profile name _and_ region
boto_sess = Session(profile_name=`profile`, region_name=`region`)
```

Sustituya el argumento designado como `region` por un valor que corresponda a uno de los Regiones de AWS en los que Amazon Braket esté disponible, por ejemplo`us-east-1`,`us-west-1`, y así sucesivamente.

Paso 3: Incorpora la sesión de Boto3 al Braket AwsSession

En el siguiente ejemplo, se muestra cómo inicializar una sesión de Boto3 con Braket e instanciar un dispositivo en esa sesión.

```
from braket.aws import AwsSession, AwsDevice

# Initialize Braket session with Boto3 Session credentials
aws_session = AwsSession(boto_session=boto_sess)

# Instantiate any Braket QPU device with the previously initiated AwsSession
sim_arn = 'arn:aws:braket:::device/quantum-simulator/amazon/sv1'
device = AwsDevice(sim_arn, aws_session=aws_session)
```

Una vez completada esta configuración, puede enviar tareas cuánticas a ese objeto de `AwsDevice` instanciado (por ejemplo, llamando al comando `device.run(…)`). Todas las llamadas a la API realizadas por ese dispositivo pueden usar las credenciales de IAM asociadas al perfil de CLI que previamente designó como `profile`.

Pruebas de tareas cuánticas con Amazon Braket

Amazon Braket ofrece una serie de simuladores de circuitos cuánticos de alto rendimiento para ayudarlo a probar y validar sus algoritmos cuánticos antes de ejecutarlos en hardware cuántico real. Estos simuladores gestionan el complejo software y la infraestructura subyacentes, así como los clústeres de Amazon Elastic Compute Cloud (Amazon EC2), para eliminar la carga que supone simular circuitos cuánticos en una infraestructura clásica de computación de alto rendimiento (HPC). Estos recursos le permiten centrarse en desarrollar y optimizar sus aplicaciones cuánticas.

Con los simuladores de Braket, puede probar exhaustivamente sus circuitos y algoritmos cuánticos sin las restricciones y limitaciones de los dispositivos cuánticos físicos. Esto le permite explorar una amplia gama de conceptos de computación cuántica, desde puertas y circuitos cuánticos básicos hasta algoritmos cuánticos más avanzados y técnicas de mitigación de errores.

El SDK de Braket simplifica el envío de sus tareas cuánticas a los simuladores, ya que le permite controlar los parámetros de la simulación, como el número de shots y el modelo de ruido, para comprender mejor el comportamiento de sus algoritmos cuánticos. También puede utilizar las capacidades de los trabajos híbridos de Amazon Braket para combinar elementos de computación clásica y cuántica, ampliando aún más el alcance de sus pruebas y validaciones.

Al probar exhaustivamente sus tareas cuánticas en los simuladores de Braket, puede obtener información valiosa, perfeccionar sus algoritmos y garantizar su corrección antes de implementarlos en hardware cuántico real. Esto ayuda a reducir el tiempo de desarrollo, minimizar los errores y, en última instancia, acelerar su progreso en el campo de la computación cuántica.

En esta sección:

Envío de tareas cuánticas a simuladores

Amazon Braket proporciona acceso a varios simuladores que pueden probar sus tareas cuánticas. Puede enviar tareas cuánticas de forma individual o puede ejecutar varios programas.

Simuladores

- Simulador de matriz de densidad, DM1: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`

- Simulador de vector de estado, SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Simulador de red tensora, TN1: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- El simulador local: `LocalSimulator()`

> **Note**
>
> Puede cancelar las tareas cuánticas que se encuentran en el `CREATED` estado QPUs y en los simuladores bajo demanda. Puede cancelar las tareas cuánticas en el `QUEUED` estado haciendo todo lo posible para los simuladores bajo demanda y. QPUs Tenga en cuenta que es poco probable que las tareas cuánticas de la QPU con el estado `QUEUED` se cancelen correctamente durante los periodos de disponibilidad de la QPU.

En esta sección:

Simulador de vector de estado local (**`braket_sv`**)

El simulador de vector de estado local (`braket_sv`) forma parte del SDK de Amazon Braket que se ejecuta localmente en su entorno. Es ideal para la creación rápida de prototipos en circuitos pequeños (hasta 25 qubits), dependiendo de las especificaciones de hardware de su instancia de cuaderno de Braket o de su entorno local.

El simulador local admite todas las puertas del SDK de Amazon Braket, pero los dispositivos QPU solo admiten un subconjunto más reducido. Puede encontrar las puertas compatibles de un dispositivo en las propiedades del dispositivo.

> **Note**
>
> El simulador local admite las características avanzadas de OpenQASM que pueden no ser compatibles con dispositivos QPU u otros simuladores. Para obtener más información sobre las características compatibles, consulte los ejemplos proporcionados en el cuaderno Simulador local de OpenQASM.

Para obtener más información sobre cómo trabajar con simuladores, consulte los ejemplos de Amazon Braket.

Simulador de matriz de densidad local (`braket_dm`)

El simulador de matriz de densidad local (`braket_dm`) forma parte del SDK de Amazon Braket que se ejecuta localmente en su entorno. Es ideal para la creación rápida de prototipos en circuitos pequeños con ruido (hasta 12 qubits), dependiendo de las especificaciones de hardware de su instancia de cuaderno de Braket o de su entorno local.

Puede construir circuitos ruidosos comunes desde cero mediante operaciones de ruido de puerta, como el cambio de bits y el error de despolarización. También puede aplicar operaciones de ruido a qubits específicos y puertas de circuitos existentes que están diseñados para ejecutarse con ruido y sin él.

El simulador local de `braket_dm` puede proporcionar los siguientes resultados, teniendo en cuenta el número especificado de shots:

- Matriz de densidad reducida: Shots = 0

> **Note**
>
> El simulador local admite las características avanzadas de OpenQASM, que pueden no ser compatibles con dispositivos QPU u otros simuladores. Para obtener más información sobre las características compatibles, consulte los ejemplos proporcionados en el cuaderno Simulador local de OpenQASM.

Para obtener más información sobre el simulador de matriz de densidad local, consulte el ejemplo de simulador de ruido introductorio de Braket.

Simulador de AHS local (`braket_ahs`)

El simulador de AHS (simulación hamiltoniana analógica) (`braket_ahs`) forma parte del SDK de Amazon Braket que se ejecuta localmente en su entorno. Se puede utilizar para simular los resultados de un programa de AHS. Es ideal para crear prototipos en registros pequeños (hasta 10-12 átomos), dependiendo de las especificaciones de hardware de su instancia de cuaderno de Braket o de su entorno local.

El simulador local es compatible con los programas de AHS con un campo de accionamiento uniforme, un campo de desplazamiento (no uniforme) y disposiciones arbitrarias de átomos. Para obtener más información, consulte la clase de AHS y el esquema del programa de AHS DE Braket.

Para obtener más información sobre el simulador local de AHS, consulte la página Hola AHS: ejecución de la primera simulación hamiltoniana analógica y los cuadernos de ejemplos de simulación hamiltoniana analógica.

Simulador de vector de estado (SV1)

SV1 es un simulador de vector de estado universal, de alto rendimiento y bajo demanda. Puede simular circuitos de hasta 34 qubits. Es de esperar que un circuito denso y cuadrado de 34-qubit (profundidad del circuito = 34) tarde aproximadamente entre 1 y 2 horas en completarse, dependiendo del tipo de puertas utilizadas y otros factores. Los circuitos con all-to-all compuertas son muy adecuados paraSV1. Devuelven resultados en forma de vector de estado completo o matriz de amplitudes.

SV1 tiene un tiempo de ejecución máximo de 6 horas. Tiene un valor predeterminado de 35 tareas cuánticas simultáneas y un máximo de 100 (50 en us-west-1 y eu-west-2) tareas cuánticas simultáneas.

Resultados de SV1

SV1 puede proporcionar los siguientes resultados, teniendo en cuenta el número especificado de shots:

- Muestra: Shots > 0
- Expectativa: Shots >= 0
- Varianza: Shots >= 0

- Probabilidad: Shots > 0
- Amplitud: Shots = 0
- Gradiente adjunto: Shots = 0

Para obtener más información sobre los resultados, consulte Lista de tipos de resultados.

SV1 está siempre disponible, ejecuta sus circuitos bajo demanda y puede ejecutar varios circuitos en paralelo. El tiempo de ejecución se escala linealmente con el número de operaciones y exponencialmente con el número de qubits. El número de shots tiene un pequeño impacto en el tiempo de ejecución. Para obtener más información, visite Comparación de simuladores.

Los simuladores admiten todas las puertas del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puede encontrar las puertas compatibles de un dispositivo en las propiedades del dispositivo.

Simulador de matriz de densidad (DM1)

DM1 es un simulador de matriz de densidad de alto rendimiento y bajo demanda. Puede simular circuitos de hasta 17 qubits.

DM1 tiene un tiempo de ejecución máximo de 6 horas, un valor predeterminado de 35 tareas cuánticas simultáneas y un máximo de 50 tareas cuánticas simultáneas.

Resultados de DM1

DM1 puede proporcionar los siguientes resultados, teniendo en cuenta el número especificado de shots:

- Muestra: Shots > 0
- Expectativa: Shots >= 0
- Varianza: Shots >= 0
- Probabilidad: Shots > 0
- Matriz de densidad reducida: Shots = 0, hasta un máximo de 8 qubits

Para obtener más información sobre los resultados, consulte Lista de tipos de resultados.

DM1 está siempre disponible, ejecuta sus circuitos bajo demanda y puede ejecutar varios circuitos en paralelo. El tiempo de ejecución se escala linealmente con el número de operaciones y

exponencialmente con el número de qubits. El número de shots tiene un pequeño impacto en el tiempo de ejecución. Para obtener más información, consulte Comparación de simuladores.

Puertas de ruido y limitaciones

```
AmplitudeDamping
    Probability has to be within [0,1]
BitFlip
    Probability has to be within [0,0.5]
Depolarizing
    Probability has to be within [0,0.75]
GeneralizedAmplitudeDamping
    Probability has to be within [0,1]
PauliChannel
    The sum of the probabilities has to be within [0,1]
Kraus
    At most 2 qubits
    At most 4 (16) Kraus matrices for 1 (2) qubit
PhaseDamping
    Probability has to be within [0,1]
PhaseFlip
    Probability has to be within [0,0.5]
TwoQubitDephasing
    Probability has to be within [0,0.75]
TwoQubitDepolarizing
    Probability has to be within [0,0.9375]
```

Simulador de red tensora (TN1)

TN1es un simulador de red tensorial de alto rendimiento y bajo demanda. TN1puede simular ciertos tipos de circuitos con hasta 50 qubits y una profundidad de circuito de 100 o menos. TN1es particularmente potente para circuitos dispersos, circuitos con puertas locales y otros circuitos con una estructura especial, como los circuitos con transformada cuántica de Fourier (QFT). TN1funciona en dos fases. En primer lugar, la fase de ensayo intenta identificar una ruta computacional eficiente para su circuito, de modo que TN1 pueda estimar el tiempo de ejecución de la siguiente etapa, que se denomina fase de contracción. Si el tiempo de contracción estimado supera el límite de tiempo de ejecución de la simulación de TN1, TN1 no intenta la contracción.

TN1 tiene un tiempo de ejecución máximo de 6 horas. Está limitado a un máximo de 10 (5 en eu-west-2) tareas cuánticas simultáneas.

Resultados de TN1

La fase de contracción consta de una serie de multiplicaciones de matrices. La serie de multiplicaciones continúa hasta que se alcanza un resultado o hasta que se determina que no se puede alcanzar un resultado.

Nota: Shots debe ser > 0.

Los tipos de resultados incluyen:

- Muestra
- Expectativa
- Varianza

Para obtener más información sobre los resultados, consulte Lista de tipos de resultados.

TN1 está siempre disponible, ejecuta sus circuitos bajo demanda y puede ejecutar varios circuitos en paralelo. Para obtener más información, consulte Comparación de simuladores.

Los simuladores admiten todas las puertas del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puede encontrar las puertas compatibles de un dispositivo en las propiedades del dispositivo.

Visite el GitHub repositorio de Amazon Braket para ver un TN1 ejemplo de cuaderno que le ayudará a empezarTN1.

Mejores prácticas para trabajar con TN1

- Evite all-to-all los circuitos.
- Pruebe un circuito nuevo o una clase de circuitos con un número pequeño de shots para conocer la «dureza» del circuito para TN1.
- Divida simulaciones de shot grandes en varias tareas cuánticas.

Acerca de los simuladores integrados

Los simuladores integrados funcionan incorporando la simulación directamente en el código del algoritmo. Además, están incluidos en el mismo contenedor y ejecutan la simulación directamente en la instancia de trabajo híbrido. Este enfoque resulta útil para eliminar los cuellos de botella que suelen asociarse a la comunicación entre la simulación y un dispositivo remoto. Al mantener todos los cómputos en un único entorno cohesionado, los simuladores integrados pueden reducir considerablemente los requisitos de memoria y disminuir el número de ejecuciones de circuitos

necesarias para alcanzar el resultado deseado. Esto puede suponer una mejora sustancial del rendimiento, a menudo multiplicándolo por diez o más, en comparación con las configuraciones tradicionales que se basan en la simulación remota. Para obtener más información sobre cómo los simuladores integrados mejoran el rendimiento y permiten optimizar los trabajos híbridos, consulte la página de documentación Ejecutar un trabajo híbrido con los trabajos híbridos de Amazon Braket.

PennyLanesus simuladores de relámpagos

Puedes usar los simuladores PennyLane de relámpagos como simuladores integrados en Braket. Con PennyLane los simuladores de relámpagos, puedes usar métodos avanzados de cálculo de gradientes, como la diferenciación contigua, para evaluar los gradientes más rápido. El simulador lightning.qubit está disponible como dispositivo a través de Braket NBIs y como simulador integrado, mientras que el simulador lightning.gpu debe ejecutarse como un simulador integrado con una instancia de GPU. Consulte el cuaderno Embedded simulators in Braket Hybrid Jobs para ver un ejemplo del uso de lightning.gpu.

Comparación de los simuladores de Amazon Braket

Esta sección le ayuda a seleccionar el simulador de Amazon Braket más adecuado para su tarea cuántica mediante la descripción de algunos conceptos, limitaciones y casos de uso.

Elección entre simuladores locales y simuladores bajo demanda (SV1, TN1, DM1)

El rendimiento de los simuladores locales depende del hardware que aloja el entorno local, como una instancia de cuaderno de Braket utilizada para ejecutar el simulador. Los simuladores bajo demanda se ejecutan en la AWS nube y están diseñados para ampliarse más allá de los entornos locales típicos. Los simuladores bajo demanda están optimizados para circuitos más grandes, pero añaden cierta latencia adicional por cada tarea cuántica o lote de tareas cuánticas. Esto puede implicar una compensación si hay muchas tareas cuánticas involucradas. Dadas estas características generales de rendimiento, la siguiente guía puede ayudarle a elegir cómo ejecutar las simulaciones, incluidas las que presentan ruido.

Para simulaciones:

- Si emplea menos de 18 qubits, utilice un simulador local.
- Si emplea entre 18 y 24 qubits, elija un simulador en función de la carga de trabajo.
- Si emplea más de 24 qubits, utilice un simulador bajo demanda.

Para simulaciones de ruido:

- Si emplea menos de 9 qubits, utilice un simulador local.
- Si emplea entre 9 y 12 qubits, elija un simulador en función de la carga de trabajo.
- Cuando emplee más de 12 qubits, utilice DM1.

¿Qué es un simulador de vector de estado?

SV1 es un simulador de vector de estado universal. Almacena la función de onda completa del estado cuántico y aplica secuencialmente las operaciones de puerta al estado. Almacena todas las posibilidades, incluso las más improbables. El tiempo de ejecución del simulador SV1 para una tarea cuántica aumenta linealmente con el número de puertas en el circuito.

¿Qué es un simulador de matriz de densidad?

DM1 simula circuitos cuánticos con ruido. Almacena la matriz de densidad completa del sistema y aplica secuencialmente las puertas y las operaciones de ruido del circuito. La matriz de densidad final contiene información completa sobre el estado cuántico después de que el circuito se ejecute. Por lo general, el tiempo de ejecución se escala linealmente con el número de operaciones y exponencialmente con el número de qubits.

¿Qué es un simulador de red tensorial?

TN1 codifica circuitos cuánticos en un gráfico estructurado.

- Los nodos del gráfico consisten en puertas cuánticas o qubits.
- Los bordes del gráfico representan las conexiones entre las puertas.

Como resultado de esta estructura, TN1 puede encontrar soluciones simuladas para circuitos cuánticos relativamente grandes y complejos.

TN1 requiere dos fases

Por lo general, TN1 funciona con un método de dos fases para simular la computación cuántica.

- La fase de ensayo: en esta fase, TN1 presenta una forma de recorrer el gráfico de manera eficiente, lo que implica visitar cada nodo para que usted pueda obtener la medición que desea. Como cliente, no ve esta fase porque TN1 realiza ambas fases en conjunto. Completa la primera fase y determina si se debe realizar la segunda fase por su cuenta en función de las limitaciones prácticas. Usted no puede influir en esa decisión una vez que la simulación ha comenzado.

- La fase de contracción: esta fase es análoga a la fase de ejecución de un cómputo en una computadora clásica. La fase consta de una serie de multiplicaciones de matrices. El orden de estas multiplicaciones tiene un gran efecto en la dificultad de la computación. Por lo tanto, la fase de ensayo se lleva a cabo primero para encontrar las rutas de computación más eficaces a lo largo del gráfico. Cuando encuentre la ruta de contracción durante la fase de ensayo, TN1 contrae las puertas del circuito para obtener los resultados de la simulación.

Los gráficos de TN1 son análogos a un mapa.

Metafóricamente, se puede comparar el gráfico de TN1 subyacente con las calles de una ciudad. En una ciudad con una planificación cuadriculada, es fácil encontrar una ruta a su destino utilizando un mapa. En una ciudad con calles no planificadas, nombres de calles duplicados, etc., puede resultar difícil encontrar la ruta a su destino utilizando un mapa.

Si TN1 no realizara la fase de ensayo, sería como recorrer las calles de la ciudad para encontrar su destino, en lugar de consultar primero un mapa. Pasar más tiempo mirando el mapa puede dar sus frutos en términos de tiempo de recorrido. Del mismo modo, la fase de ensayo proporciona información valiosa.

Se podría decir que TN1 tiene cierta «conciencia» de la estructura del circuito subyacente que atraviesa. Adquiere esta conciencia durante la fase de ensayo.

Tipos de problemas más adecuados para cada uno de estos tipos de simuladores

SV1 es muy adecuado para cualquier clase de problemas que dependan principalmente de tener un número determinado de qubits y puertas. Por lo general, el tiempo necesario aumenta de forma lineal con el número de puertas, mientras que no depende del número de shots. Por lo general, SV1 es más rápido que TN1 para circuitos inferiores a 28 qubits.

SV1 puede ser más lento para números más altos de qubit porque en realidad simula todas las posibilidades, incluso las extremadamente improbables. No tiene forma de determinar qué resultados son probables. Por lo tanto, para una evaluación de 30-qubit, SV1 debe calcular 2^30 configuraciones. El límite de 34 qubits para el simulador SV1 de Amazon Braket es una restricción práctica debido a las limitaciones de memoria y almacenamiento. Puede pensar en ello de esta manera: cada vez que añade un qubit a SV1, el problema se vuelve dos veces más difícil.

Para muchas clases de problemas, TN1 puede evaluar circuitos mucho más grandes en tiempo realista que SV1 porque TN1 aprovecha la estructura del gráfico. Básicamente, realiza un seguimiento de la evolución de las soluciones desde su punto de partida y solo retiene

las configuraciones que contribuyen a un recorrido eficiente. En otras palabras, guarda las configuraciones para crear un orden de multiplicación de matrices que da como resultado un proceso de evaluación más sencillo.

Para TN1, el número de qubits y puertas son importantes, pero la estructura del gráfico es mucho más importante. Por ejemplo, TN1 es muy eficaz para evaluar circuitos (gráficos) en los que las puertas son de corto alcance (es decir, cada qubit está conectado por puertas solo a su qubits vecino más cercano) y circuitos (gráficos) en los que las conexiones (o puertas) tienen un alcance similar. Un alcance típico de TN1 es posibilitar que cada qubit se comunique solo con otros qubits que estén a 5 qubits de distancia. Si la mayor parte de la estructura se puede descomponer en relaciones más simples como estas, que se pueden representar en más matrices más pequeñas o más uniformes, TN1 realiza la evaluación de manera eficiente.

Limitaciones de TN1

TN1 puede ser más lento que SV1 dependiendo de la complejidad estructural del gráfico. En algunos gráficos, TN1 finaliza la simulación después de la fase de ensayo y muestra un estado `FAILED`, por una de estas dos razones:

- No se puede encontrar una ruta: si el gráfico es demasiado complejo, es muy difícil encontrar una buena ruta transversal y el simulador abandona el cómputo. TN1 no puede realizar la contracción. Es posible que aparezca un mensaje de error similar a este: `No viable contraction path found.`
- La etapa de contracción es demasiado difícil: en algunos gráficos, TN1 se puede encontrar una trayectoria transversal, pero su evaluación es muy larga y requiere mucho tiempo. En este caso, la contracción es tan cara que el costo sería prohibitivo y, en cambio, TN1 sale tras la fase de ensayo. Es posible que veas un mensaje de error similar a este: `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> Se le facturará la fase de ensayo TN1 aunque no se realice la contracción y verá un estado `FAILED`.

El tiempo de ejecución previsto también depende del recuento de shot. En el peor de los casos, el tiempo de contracción de TN1 depende linealmente del recuento de shot. El circuito puede ser contraíble con menos shots. Por ejemplo, podría enviar una tarea cuántica con 100 shots y TN1

decidirá que no es contraíble, pero si la vuelve a enviar con solo 10, la contracción se lleva a cabo. En este caso, para obtener 100 muestras, podría enviar 10 tareas cuánticas de 10 shots para el mismo circuito y combinar los resultados al final.

Como práctica recomendada, le aconsejamos que siempre pruebe su circuito o clase de circuito con pocos shots (por ejemplo, 10) para averiguar qué tan difícil es su circuito para TN1, antes de continuar con un número mayor de shots.

> **Note**
>
> La serie de multiplicaciones que forma la fase de contracción comienza con matrices pequeñas NxN. Por ejemplo, una puerta de 2-qubit requiere una matriz de 4x4. Las matrices intermedias que se requieren durante una contracción que se considera demasiado difícil son gigantescas. Un cómputo de este tipo tardaría días en completarse. Por eso Amazon Braket no intenta contracciones extremadamente complejas.

Concurrency (Simultaneidad)

Todos los simuladores de Braket le permiten ejecutar varios circuitos simultáneamente. Los límites de simultaneidad varían según el simulador y la región. Para obtener más información sobre los límites de concurrencia, consulte la página Cuotas.

Ejemplo de tareas cuánticas en Amazon Braket

En esta sección se explican las etapas de ejecución de una tarea cuántica de ejemplo, desde la selección del dispositivo hasta la visualización del resultado. Como práctica recomendada para Amazon Braket, le recomendamos que comience por ejecutar el circuito en un simulador, como SV1.

En esta sección:

- Especificación del dispositivo
- Envío de una de tarea cuántica de ejemplo
- Envío de una tarea parametrizada
- Especifique shots.
- Sondeo de resultados
- Visualización de los resultados de ejemplo

Especificación del dispositivo

Primero, seleccione y especifique el dispositivo para su tarea cuántica. En este ejemplo se muestra cómo elegir el simulador SV1.

```
from braket.aws import AwsDevice

# Choose the on-demand simulator to run the circuit
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
```

Puede ver algunas de las propiedades de este dispositivo de la siguiente manera:

```
print(device.name)
for iter in device.properties.action['braket.ir.jaqcd.program']:
    print(iter)
```

```
SV1
('version', ['1.0', '1.1'])
('actionType', 'braket.ir.jaqcd.program')
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
 'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', 'ecr', 'h', 'i', 'iswap',
 'pswap', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'unitary', 'v',
 'vi', 'x', 'xx', 'xy', 'y', 'yy', 'z', 'zz'])
('supportedResultTypes', [ResultType(name='Sample', observables=['x', 'y', 'z', 'h',
 'i', 'hermitian'], minShots=1, maxShots=100000), ResultType(name='Expectation',
 observables=['x', 'y', 'z', 'h', 'i', 'hermitian'], minShots=0, maxShots=100000),
 ResultType(name='Variance', observables=['x', 'y', 'z', 'h', 'i', 'hermitian'],
 minShots=0, maxShots=100000), ResultType(name='Probability', observables=None,
 minShots=1, maxShots=100000), ResultType(name='Amplitude', observables=None,
 minShots=0, maxShots=0)])
('disabledQubitRewiringSupported', None)
```

Envío de una de tarea cuántica de ejemplo

Envíe una tarea cuántica de ejemplo para ejecutarla en el simulador bajo demanda.

```
from braket.circuits import Circuit, Observable

# Create a circuit with a result type
circ = Circuit().rx(0, 1).ry(1, 0.2).cnot(0, 2).variance(observable=Observable.Z(),
 target=0)
# Add another result type
```

```
circ.probability(target=[0, 2])

# Set up S3 bucket (where results are stored)
my_bucket = "amazon-braket-s3-demo-bucket"  # The name of the bucket
my_prefix = "your-folder-name"  # The name of the folder in the bucket
s3_location = (my_bucket, my_prefix)

# Submit the quantum task to run
my_task = device.run(circ, s3_location, shots=1000, poll_timeout_seconds=100,
 poll_interval_seconds=10)
# The positional argument for the S3 bucket is optional if you want to specify a bucket
 other than the default

# Get results of the quantum task
result = my_task.result()
```

El comando `device.run()` crea una tarea cuántica a través de la API de `CreateQuantumTask`. Tras un breve período de inicialización, la tarea cuántica se pone en cola hasta que exista la capacidad de ejecutarla en un dispositivo. En este caso, el dispositivo es SV1. Una vez que el dispositivo completa el cómputo, Amazon Braket escribe los resultados en la ubicación de Amazon S3 especificada en la llamada. El argumento posicional `s3_location` es necesario para todos los dispositivos excepto para el simulador local.

> **Note**
>
> La acción de la tarea cuántica de Braket tiene un tamaño limitado de 3 MB.

Envío de una tarea parametrizada

Amazon Braket ofrece simuladores locales y bajo demanda, y QPUs también admite la especificación de valores de parámetros libres al enviar la tarea. Puede hacer esto utilizando el argumento `inputs` para `device.run()`, como se muestra en el siguiente ejemplo. Las `inputs` deben ser un diccionario de pares cadena-número flotante, donde las claves son los nombres de los parámetros.

La compilación paramétrica puede mejorar el rendimiento de la ejecución de ciertos circuitos paramétricos. QPUs Al enviar un circuito paramétrico como tarea cuántica a una QPU compatible, Braket compilará el circuito una vez y almacenará en caché el resultado. No se produce la recompilación en las actualizaciones posteriores de los parámetros del mismo circuito, lo que se

traduce en tiempos de ejecución más rápidos para las tareas que utilizan el mismo circuito. Braket utiliza automáticamente los datos de calibración actualizados del proveedor de hardware al compilar su circuito para garantizar resultados de la máxima calidad.

> **Note**
>
> La compilación paramétrica es compatible con todos los sistemas superconductores basados en compuertas, Rigetti Computing con la excepción de los QPUs programas de nivel de pulso.

```
from braket.circuits import Circuit, FreeParameter, Observable

# Create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')

# Create a circuit with a result type
circ = Circuit().rx(0, alpha).ry(1, alpha).cnot(0, 2).xx(0, 2, beta)
circ.variance(observable=Observable.Z(), target=0)

# Add another result type
circ.probability(target=[0, 2])

# Submit the quantum task to run
my_task = device.run(circ, inputs={'alpha': 0.1, 'beta': 0.2}, shots=100)
```

Especifique shots.

El argumento shots se refiere al número de shots de medición deseados. Los simuladores como SV1 admiten dos modos de simulación.

- Para shots = 0, el simulador realiza una simulación exacta y devuelve los valores verdaderos de todos los tipos de resultados. (No disponible en TN1).
- Para valores distintos de ceroshots, el simulador toma muestras de la distribución de salida para emular el ruido real. shot QPUs Los dispositivos QPU solo permiten shots > 0.

Para obtener información sobre el número máximo de shots por tarea cuántica, consulte la sección Cuotas de Braket.

Sondeo de resultados

Al ejecutar `my_task.result()`, el SDK comienza a sondear en busca de un resultado con los parámetros que usted define al crear la tarea cuántica:

- `poll_timeout_seconds` es el número de segundos que se tarda en sondear la tarea cuántica antes de que se agote el tiempo de espera al ejecutar la tarea cuántica en el simulador bajo demanda o en los dispositivos QPU. El valor predeterminado es 432 000 segundos, lo que equivale a 5 días.
- Nota: En el caso de los dispositivos QPU como Rigetti y IonQ, le recomendamos que espere unos días. Si el tiempo de espera del sondeo es demasiado corto, es posible que los resultados no se devuelvan dentro del tiempo de sondeo. Por ejemplo, cuando una QPU no está disponible, se devuelve un error de tiempo de espera local.
- `poll_interval_seconds` es la frecuencia con la que se sondea la tarea cuántica. Especifica la frecuencia con la que se llama a la API de Braket para obtener el estado cuando la tarea cuántica se ejecuta en el simulador bajo demanda y en dispositivos QPU. El valor predeterminado es 1 segundo.

Esta ejecución asíncrona facilita la interacción con dispositivos QPU que no siempre están disponibles. Por ejemplo, un dispositivo podría no estar disponible durante un período de mantenimiento regular.

El resultado devuelto contiene una serie de metadatos asociados a la tarea cuántica. Puede utilizar estos comandos para comprobar el resultado de medición:

```
print('Measurement results:\n', result.measurements)
print('Counts for collapsed states:\n', result.measurement_counts)
print('Probabilities for collapsed states:\n', result.measurement_probabilities)
```

```
Measurement results:
 [[1 0 1]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [1 0 1]]
Counts for collapsed states:
```

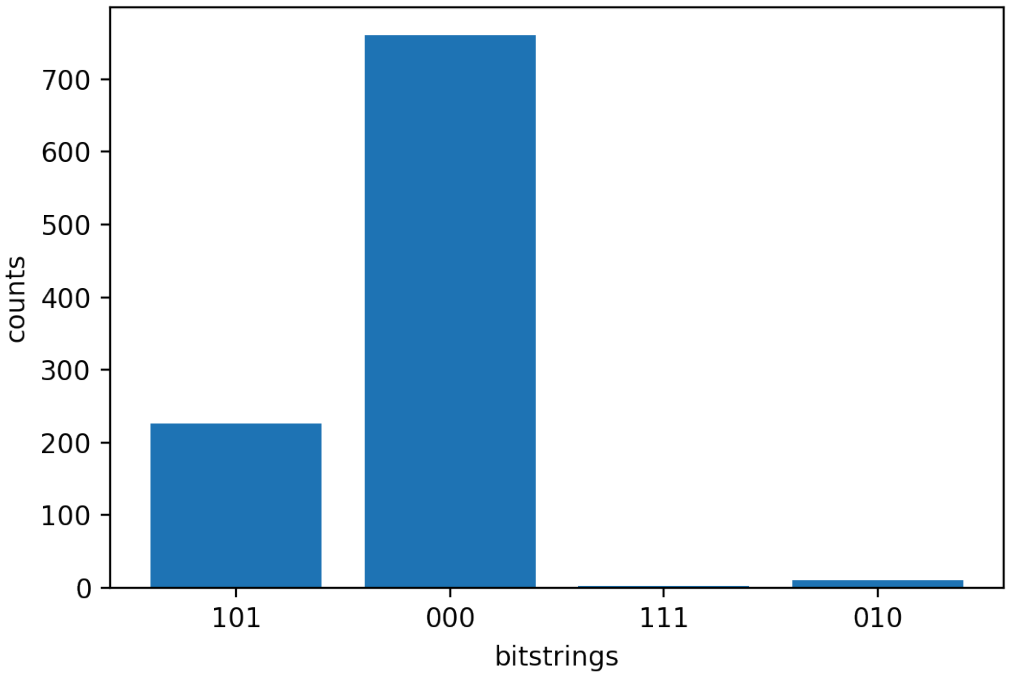
```
 Counter({'000': 766, '101': 220, '010': 11, '111': 3})
Probabilities for collapsed states:
 {'101': 0.22, '000': 0.766, '010': 0.011, '111': 0.003}
```

Visualización de los resultados de ejemplo

Como también especificó el `ResultType`, puede ver los resultados devueltos. Los tipos de resultados aparecen en el orden en que se añadieron al circuito.

```
print('Result types include:\n', result.result_types)
print('Variance=', result.values[0])
print('Probability=', result.values[1])

# Plot the result and do some analysis
import matplotlib.pyplot as plt
plt.bar(result.measurement_counts.keys(), result.measurement_counts.values())
plt.xlabel('bitstrings')
plt.ylabel('counts')
```

```
Result types include:
 [ResultTypeValue(type=Variance(observable=['z'], targets=[0], type=<Type.variance:
 'variance'>), value=0.693084), ResultTypeValue(type=Probability(targets=[0, 2],
 type=<Type.probability: 'probability'>), value=array([0.777, 0.   , 0.   , 0.223]))]
Variance= 0.693084
Probability= [0.777 0.    0.    0.223]
Text(0, 0.5, 'counts')
```



Prueba de una tarea cuántica con el simulador local

Puede enviar tareas cuánticas directamente a un simulador local para realizar prototipos y pruebas rápidas. Este simulador se ejecuta en su entorno local, por lo que no es necesario especificar una ubicación de Amazon S3. Los resultados se computan directamente en la sesión. Para ejecutar una tarea cuántica en el simulador local, solo debe especificar el parámetro shots.

> **Note**
>
> La velocidad de ejecución y el número máximo de qubits que puede procesar el simulador local dependen del tipo de instancia del cuaderno de Amazon Braket o de las especificaciones de su hardware local.

Los siguientes comandos son todos idénticos e instancian el simulador local de vector de estado (sin ruido).

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator
```

```
# The following are identical commands
device = LocalSimulator()
device = LocalSimulator("default")
device = LocalSimulator(backend="default")
device = LocalSimulator(backend="braket_sv")
```

A continuación, ejecute una tarea cuántica con lo siguiente.

```
my_task = device.run(circ, shots=1000)
```

Para instanciar el simulador de matriz de densidad local (ruido), los clientes deben cambiar el backend de la siguiente manera.

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

device = LocalSimulator(backend="braket_dm")
```

Medición de qubits específicos en el simulador local

El simulador de vector de estado local y el simulador de matriz de densidad local admiten el funcionamiento de circuitos en los que se puede medir un subconjunto de los qubits del circuito, lo que se suele denominar medición parcial.

Por ejemplo, en el siguiente código puede crear un circuito de dos qubits y medir solo el primer qubit añadiendo una instrucción de `measure` con los qubits de destino al final del circuito.

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

# Use the local simulator device
device = LocalSimulator()

# Define a bell circuit and only measure
circuit = Circuit().h(0).cnot(0, 1).measure(0)

# Run the circuit
task = device.run(circuit, shots=10)

# Get the results
```

```
result = task.result()

# Print the measurement counts for qubit 0
print(result.measurement_counts)
```

Emulador de dispositivo cuántico local

Con la herramienta de emulación local de Amazon Braket, puede emular sus programas cuánticos verbatim de forma local antes de ejecutarlos en hardware cuántico real. El emulador utiliza datos de calibración del dispositivo para validar los circuitos verbatim, lo que le permite detectar problemas de compatibilidad más pronto.

Además, el emulador local simula el ruido cuántico del hardware mediante el siguiente proceso:

- Uso de los datos de calibración del dispositivo para construir el modelo de ruido
- Aplicación de ruido despolarizante a cada puerta del circuito
- Aplicación del error de lectura al final del circuito
- Simulación del circuito ruidoso con un simulador de matriz de densidad local

Para obtener más información sobre el uso de un emulador local, consulte Emulación local para circuitos literales en Amazon Braket en el repositorio. amazon-braket-examples GitHub

En esta sección:

- Ventajas de la emulación local
- Creación de un emulador local

Ventajas de la emulación local

- Valide los circuitos verbatim frente a las restricciones del dispositivo utilizando datos de calibración en tiempo real o históricos.
- Depure los problemas antes de enviar tareas al hardware cuántico.
- Compare las emulaciones sin ruido y con ruido con los resultados del hardware para comprender los efectos del ruido.
- Optimice el flujo de trabajo del desarrollo de algoritmos cuánticos sensibles al ruido.

Creación de un emulador local

Se puede crear un emulador de dispositivo cuántico local directamente a partir de un dispositivo cuántico o un conjunto de propiedades del dispositivo. Al emular directamente un dispositivo, el emulador utiliza los datos de calibración más recientes del dispositivo instanciado. El siguiente ejemplo de código muestra cómo emular directamente el dispositivo Rigetti's Ankaa-3.

```
from braket.aws.aws_device import AwsDevice

ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_emulator = ankaa3.emulator()
```

El siguiente ejemplo muestra cómo crear un emulador de dispositivo local a partir de un conjunto de propiedades del dispositivo Ankaa-3 en formato JSON.

```
from braket.aws import AwsDevice
from braket.emulation.local_emulator import LocalEmulator
import json

# Instantiate the device
ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_properties = ankaa3.properties

# Put the Ankaa-3 properties in a file named ankaa3_device_properties.json
with open("ankaa3_device_properties.json", "w") as f:
    json.dump(ankaa3_properties.json(), f)

# Load the json into the ankaa3_data_json variable
with open("ankaa3_device_properties.json", "r") as json_file:
    ankaa3_data_json = json.load(json_file)

# Create the Ankaa-3 local emulator from the json file you created
ankaa3_emulator = LocalEmulator.from_json(ankaa3_data_json)
```

Puede personalizar el ejemplo de la siguiente manera:

- Uso de propiedades de un dispositivo QPU diferente
- Especificación de un archivo JSON diferente para el emulador
- Cambio de los valores de las propiedades del dispositivo antes de instanciar el emulador

Ejecución de las tareas cuánticas con Amazon Braket

Braket proporciona un acceso seguro y bajo demanda a diferentes tipos de computadoras cuánticas. Tiene acceso a ordenadores cuánticos basados en puertas desdeAQT,, y IonQ IQMRigetti, así como a un simulador hamiltoniano analógico desde. QuEra Por otro lado, no tiene ningún compromiso inicial ni necesita obtener acceso a través de proveedores individuales.

- La consola de Amazon Braket proporciona información y estado del dispositivo para ayudarlo a crear, administrar y monitorizar sus recursos y tareas cuánticas.
- Envíe y ejecute tareas cuánticas a través del SDK de Python de Amazon Braket, así como a través de la consola. Se puede acceder al SDK a través de cuadernos de Amazon Braket preconfigurados.
- Se puede acceder a la API de Amazon Braket a través del SDK de Python de Amazon Braket y de los cuadernos. Puede realizar llamadas directamente a la API si está desarrollando aplicaciones que funcionan con la computación cuántica mediante programación.

Los ejemplos de esta sección muestran cómo puede trabajar directamente con la API de Amazon Braket mediante el SDK de Python de Amazon Braket junto con el SDK de Python de AWS para Braket (Boto3).

Más información sobre el SDK de Python de Amazon Braket

Para trabajar con el SDK de Python de Amazon Braket, instale primero el SDK de AWS Python para Braket (Boto3) de forma que pueda comunicarse con el. AWS API Puede considerar el SDK de Python de Amazon Braket como un práctico envoltorio de Boto3 para los clientes cuánticos.

- Boto3 contiene interfaces que necesita utilizar. AWS API (Tenga en cuenta que Boto3 es un gran SDK de Python que habla con. AWS API La mayoría Servicios de AWS admite una interfaz Boto3.)
- El SDK de Python de Amazon Braket contiene módulos de software para circuitos, puertas, dispositivos, tipos de resultados y otras partes de una tarea cuántica. Cada vez que crea un programa, importa los módulos que necesita para esa tarea cuántica.
- Se puede acceder al SDK de Python de Amazon Braket a través de cuadernos, que vienen precargados con todos los módulos y dependencias necesarios para ejecutar tareas cuánticas.
- Si no desea trabajar con cuadernos, puede importar módulos del SDK de Python de Amazon Braket a cualquier script de Python.

Una vez que haya instalado Boto3, una visión general de los pasos para crear una tarea cuántica a través del SDK de Python de Amazon Braket es similar a esto:

1. (Opcional) Abra su cuaderno.
2. Importe los módulos del SDK que necesite para sus circuitos.
3. Especifique una QPU o un simulador.
4. Instancie el circuito.
5. Ejecute el circuito.
6. Recopile los resultados.

Los ejemplos de esta sección muestran los detalles de cada paso.

Para ver más ejemplos, consulte el repositorio Amazon Braket Examples en. GitHub

En esta sección:

- Enviar tareas cuánticas a QPUs
- Ejecución de varios programas
- ¿Cuándo se ejecutará mi tarea cuántica?
- Trabajar con reservas
- Técnicas de mitigación de errores

Enviar tareas cuánticas a QPUs

Amazon Braket proporciona acceso a varios dispositivos que pueden ejecutar tareas cuánticas. Puede enviar las tareas cuánticas de forma individual o puede configurar el procesamiento por lotes de tareas cuánticas.

Unidades de procesamiento cuántico (QPUs)

Puedes enviar tareas cuánticas QPUs en cualquier momento, pero la tarea se ejecuta dentro de determinadas ventanas de disponibilidad que se muestran en la página Dispositivos de la consola Amazon Braket. Puede recuperar los resultados de la tarea cuántica con el ID de la tarea cuántica, que se presenta en la siguiente sección.

- AQT IBEX-Q1 : `arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1`

- IonQ Forte-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1`
- IonQ Forte-Enterprise-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1`
- IQM Garnet : `arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet`
- IQM Emerald : `arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Ankaa-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3`

> **Note**
>
> Puede cancelar las tareas cuánticas en el `CREATED` estado QPUs y en los simuladores bajo demanda. Puede cancelar las tareas cuánticas en el `QUEUED` estado haciendo todo lo posible para los simuladores bajo demanda y. QPUs Tenga en cuenta que es poco probable que las tareas cuánticas de la QPU con el estado `QUEUED` se cancelen correctamente durante los periodos de disponibilidad de la QPU.

En esta sección:

AQT

AQTLa QPU del IBEX-Q1 se basa en un cristal de ^{40} iones Ca^{+} en una trampa macroscópica de radiofrecuencia situada en una cámara de vacío ultra alto. El dispositivo funciona a temperatura ambiente y cabe en dos racks de 19 pulgadas compatibles con centros de datos.

Las bajas velocidades de calentamiento de la trampa y el uso de una transición óptica directa para la rotación de los cúbits permiten la entrada de alta fidelidad. La transición de cúbits es impulsada por

un láser de ancho de línea estrecho con una estabilidad de frecuencia relativa muy alta. Los qubits también cuentan con una eficiente preparación y lectura del estado a través de una estantería óptica. All-to-allla conectividad se logra mediante la interacción de Coulomb de largo alcance en el cristal iónico. El direccionamiento y la lectura de un solo ion se logran mediante el uso de una lente de alta apertura numérica.

El AQT dispositivo admite las siguientes puertas cuánticas.

```
'ccnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01', 'cphaseshift10',
 'cswap', 'swap', 'iswap', 'pswap', 'ecr', 'cy', 'cz', 'xy', 'xx', 'yy', 'zz', 'h',
 'i', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 't', 'ti', 'v', 'vi', 'x', 'y', 'z',
 'prx'
```

Con la compilación literal, el AQT dispositivo admite las siguientes puertas nativas.

```
'prx', 'xx', 'rz'
```

> **Note**
>
> A continuación se describen las puertas equivalentes entre las puertas AQT nativas y Amazon Braket:
>
> - La puerta AQT Mølmer-Sørensen (MS o RXX) corresponde a la puerta de Braket `'xx'`
> - La puerta AQT R corresponde a la puerta de `'prx'` Braket
> - El nombre de la `'rz'` puerta es el mismo

IonQ

IonQofrece tecnología basada en compuertas QPUs y basada en trampas de iones. IonQ's QPUs los iones atrapados se forman sobre una cadena de iones 171Yb+ atrapados que están confinados espacialmente mediante una trampa de electrodos de superficie microfabricada dentro de una cámara de vacío.

Los dispositivos IonQ admiten las siguientes puertas cuánticas.

```
'x', 'y', 'z', 'rx', 'ry', 'rz', 'h', 'cnot', 's', 'si', 't', 'ti', 'v', 'vi', 'xx',
 'yy', 'zz', 'swap'
```

Con la compilación literal, admiten las siguientes puertas nativas. IonQ QPUs

```
'gpi', 'gpi2', 'ms'
```

Si solo especifica dos parámetros de fase al usar la puerta nativa de MS, se ejecuta una puerta de MS totalmente entrelazada. Una puerta de MS totalmente entrelazada siempre realiza una rotación π /2. Para especificar un ángulo diferente y ejecutar una puerta de MS parcialmente entrelazada, especifique el ángulo deseado añadiendo un tercer parámetro. Para obtener más información, consulte el módulo braket.circuits.gate.

Estas puertas nativas solo se pueden usar con la compilación verbatim. Para obtener más información sobre la compilación verbatim, consulte Compilación verbatim.

IQM

Los procesadores cuánticos de IQM son dispositivos de tipo puerta universal basados en qubits transmónicos superconductores. IQM Garnet es un dispositivo de 20 qubits, mientras que IQM Emerald es un dispositivo de 54 qubits. Ambos dispositivos utilizan una topología de retícula cuadrada, también conocida como topología de retícula de cristal.

Los dispositivos IQM admiten las siguientes puertas cuánticas.

```
"ccnot", "cnot", "cphaseshift", "cphaseshift00", "cphaseshift01", "cphaseshift10",
 "cswap", "swap", "iswap", "pswap", "ecr", "cy", "cz", "xy", "xx", "yy", "zz", "h",
 "i", "phaseshift", "rx", "ry", "rz", "s", "si", "t", "ti", "v", "vi", "x", "y", "z"
```

Con la compilación verbatim, los dispositivos IQM admiten las siguientes puertas nativas.

```
'cz', 'prx'
```

Rigetti

Los procesadores cuánticos de Rigetti son máquinas universales de tipo puerta basadas en qubits superconductores totalmente ajustables.

- El sistema Ankaa-3 es un dispositivo de 84 qubits que utiliza tecnología escalable de varios chips.

El dispositivo Rigetti admite las siguientes puertas cuánticas.

```
'cz', 'xy', 'ccnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01',
 'cphaseshift10', 'cswap', 'h', 'i', 'iswap', 'phaseshift', 'pswap', 'rx', 'ry', 'rz',
 's', 'si', 'swap', 't', 'ti', 'x', 'y', 'z'
```

Con la compilación verbatim, Ankaa-3 admite las siguientes puertas nativas.

```
'rx', 'rz', 'iswap'
```

Los procesadores cuánticos superconductores de Rigetti pueden ejecutar la puerta «rx» solo con los ángulos de ±π /2 o ±π.

El control a nivel de impulsos está disponible en los dispositivos Rigetti que admiten un conjunto de marcos predefinidos de los siguientes tipos para el sistema Ankaa-3.

```
`flux_tx`, `charge_tx`, `readout_rx`, `readout_tx`
```

El Ankaa-3 dispositivo tiene un límite máximo de 20 000 puertas por circuito. Los circuitos que superen este límite se rechazan con un error de validación. Se trata de un límite fijo que no se puede aumentar. El recuento de puertas se refiere al circuito compilado, que puede diferir del recuento de puertas del circuito no compilado original. Para estimar el recuento de puertas compilado antes de enviarlo a la QPU, puede utilizar la compilación literal de forma local o transpilar el circuito al conjunto de puertas nativo (,,). `rx` `rz` `iswap`

QuEra

QuEra ofrece dispositivos basados en átomos neutros que pueden ejecutar tareas cuánticas de simulación hamiltoniana analógica (AHS). Estos dispositivos especiales reproducen fielmente la dinámica cuántica dependiente del tiempo de cientos de qubits que interactúan simultáneamente.

Estos dispositivos se pueden programar en el paradigma de la simulación hamiltoniana analógica prescribiendo la disposición del registro de qubits y la dependencia temporal y espacial de los campos de manipulación. Amazon Braket proporciona utilidades para construir dichos programas a través del módulo AHS del SDK de Python, `braket.ahs`.

Para obtener más información, consulte los cuadernos de ejemplos de simulación hamiltoniana analógica o la página Enviar un programa analógico utilizando Aquila. QuEra

Ejemplo: Enviar una tarea cuántica a una QPU

Amazon Braket permite ejecutar un circuito cuántico en un dispositivo QPU. El siguiente ejemplo muestra cómo enviar una tarea cuántica a los dispositivos Rigetti o IonQ.

Elija el dispositivo Rigetti Ankaa-3 y, a continuación, mire el gráfico de conectividad asociado.

```
# import the QPU module
from braket.aws import AwsDevice
# choose the Rigetti device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

# take a look at the device connectivity graph
device.properties.dict()['paradigm']['connectivity']
```

```
{'fullyConnected': False,
 'connectivityGraph': {'0': ['1', '7'],
  '1': ['0', '2', '8'],
  '2': ['1', '3', '9'],
  '3': ['2', '4', '10'],
  '4': ['3', '5', '11'],
  '5': ['4', '6', '12'],
  '6': ['5', '13'],
  '7': ['0', '8', '14'],
  '8': ['1', '7', '9', '15'],
  '9': ['2', '8', '10', '16'],
  '10': ['3', '9', '11', '17'],
  '11': ['4', '10', '12', '18'],
  '12': ['5', '11', '13', '19'],
  '13': ['6', '12', '20'],
  '14': ['7', '15', '21'],
  '15': ['8', '14', '22'],
  '16': ['9', '17', '23'],
  '17': ['10', '16', '18', '24'],
  '18': ['11', '17', '19', '25'],
  '19': ['12', '18', '20', '26'],
  '20': ['13', '19', '27'],
  '21': ['14', '22', '28'],
  '22': ['15', '21', '23', '29'],
  '23': ['16', '22', '24', '30'],
  '24': ['17', '23', '25', '31'],
  '25': ['18', '24', '26', '32'],
  '26': ['19', '25', '33'],
```

```
    '27': ['20', '34'],
    '28': ['21', '29', '35'],
    '29': ['22', '28', '30', '36'],
    '30': ['23', '29', '31', '37'],
    '31': ['24', '30', '32', '38'],
    '32': ['25', '31', '33', '39'],
    '33': ['26', '32', '34', '40'],
    '34': ['27', '33', '41'],
    '35': ['28', '36', '42'],
    '36': ['29', '35', '37', '43'],
    '37': ['30', '36', '38', '44'],
    '38': ['31', '37', '39', '45'],
    '39': ['32', '38', '40', '46'],
    '40': ['33', '39', '41', '47'],
    '41': ['34', '40', '48'],
    '42': ['35', '43', '49'],
    '43': ['36', '42', '44', '50'],
    '44': ['37', '43', '45', '51'],
    '45': ['38', '44', '46', '52'],
    '46': ['39', '45', '47', '53'],
    '47': ['40', '46', '48', '54'],
    '48': ['41', '47', '55'],
    '49': ['42', '56'],
    '50': ['43', '51', '57'],
    '51': ['44', '50', '52', '58'],
    '52': ['45', '51', '53', '59'],
    '53': ['46', '52', '54'],
    '54': ['47', '53', '55', '61'],
    '55': ['48', '54', '62'],
    '56': ['49', '57', '63'],
    '57': ['50', '56', '58', '64'],
    '58': ['51', '57', '59', '65'],
    '59': ['52', '58', '60', '66'],
    '60': ['59'],
    '61': ['54', '62', '68'],
    '62': ['55', '61', '69'],
    '63': ['56', '64', '70'],
    '64': ['57', '63', '65', '71'],
    '65': ['58', '64', '66', '72'],
    '66': ['59', '65', '67'],
    '67': ['66', '68'],
    '68': ['61', '67', '69', '75'],
    '69': ['62', '68', '76'],
    '70': ['63', '71', '77'],
```

```
  '71': ['64', '70', '72', '78'],
  '72': ['65', '71', '73', '79'],
  '73': ['72', '80'],
  '75': ['68', '76', '82'],
  '76': ['69', '75', '83'],
  '77': ['70', '78'],
  '78': ['71', '77', '79'],
  '79': ['72', '78', '80'],
  '80': ['73', '79', '81'],
  '81': ['80', '82'],
  '82': ['75', '81', '83'],
  '83': ['76', '82']}}
```

El diccionario anterior `connectivityGraph` enumera los qubits vecinos de cada qubit en el dispositivo Rigetti.

Elija el dispositivo IonQ Forte-Enterprise-1.

En el caso del IonQ Forte-Enterprise-1 dispositivo, `connectivityGraph` está vacío, como se muestra en el siguiente ejemplo, porque el dispositivo ofrece conectividad. all-to-all Por lo tanto, no se necesita un `connectivityGraph` detallado.

```
# or choose the IonQ Forte-Enterprise-1 device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1")

# take a look at the device connectivity graph
device.properties.dict()['paradigm']['connectivity']
```

```
{'fullyConnected': True, 'connectivityGraph': {...}}
```

Como se muestra en el siguiente ejemplo, tiene la opción de ajustar los shots (por defecto = 1000), los `poll_timeout_seconds` (por defecto = 432 000 = 5 días), los `poll_interval_seconds` (por defecto = 1) y la ubicación del bucket de S3 (`s3_location`) en la que se almacenarán los resultados si decide especificar una ubicación distinta del bucket predeterminado.

```
my_task = device.run(circ, s3_location = 'amazon-braket-my-folder', shots=100,
 poll_timeout_seconds = 100, poll_interval_seconds = 10)
```

Los dispositivos Rigetti y IonQ compilan automáticamente el circuito proporcionado en sus respectivos conjuntos de puertas nativas y asignan los índices de qubit abstractos a qubits físicos en la QPU respectiva.

> **Note**
>
> Los dispositivos QPU tienen una capacidad limitada. Cuando se alcanza la capacidad, puede esperar tiempos de espera más largos.

Amazon Braket puede ejecutar tareas cuánticas de QPU dentro de determinados períodos de disponibilidad, pero puede enviar tareas cuánticas en cualquier momento (24 horas al día, 7 días a la semana), ya que todos los datos y metadatos correspondientes se almacenan de forma fiable en el bucket de S3 correspondiente. Como se muestra en la siguiente sección, puede recuperar su tarea cuántica utilizando `AwsQuantumTask` y su identificador único de tarea cuántica.

Inspección de circuitos compilados

Cuando es necesario ejecutar un circuito cuántico en un dispositivo de hardware, como una unidad de procesamiento cuántico (QPU), primero se debe compilar el circuito en un formato aceptable que el dispositivo pueda entender y procesar. Por ejemplo, transpilar el circuito cuántico de alto nivel a las puertas nativas específicas compatibles con el hardware de la QPU de destino. Inspeccionar la salida compilada real del circuito cuántico puede resultar extremadamente útil para fines de depuración y optimización. Este conocimiento puede ayudar a identificar posibles problemas, obstáculos u oportunidades para mejorar el rendimiento y la eficiencia de la aplicación cuántica. Puede ver y analizar la salida compilada de sus circuitos cuánticos, para los dispositivos de computación cuántica Rigetti y IQM utilizando el código que se proporciona a continuación.

```
task = AwsQuantumTask(arn=task_id, aws_session=session)
# After the task has finished running
task_result = task.result()
compiled_circuit = task_result.get_compiled_circuit()
```

> **Note**
>
> Actualmente, no se admite la visualización de la salida del circuito compilado para los dispositivos IonQ.

Ejecución de varios programas

Amazon Braket ofrece dos enfoques para ejecutar varios programas cuánticos de manera eficiente: conjuntos de programas y procesamiento por lotes de tareas cuánticas.

Los conjuntos de programas son la forma preferida de ejecutar cargas de trabajo con varios programas. Permiten empaquetar varios programas en una sola tarea cuántica de Amazon Braket. Los conjuntos de programas ofrecen mejoras en el rendimiento y un ahorro de costos en comparación con el envío de programas de forma individual, especialmente cuando el número de ejecuciones de programas se acerca a las 100.

Actualmente, los dispositivos IQM y Rigetti admiten conjuntos de programas. Antes de enviar los conjuntos de programas QPUs, se recomienda probarlos primero en el simulador local de Amazon Braket. Para comprobar si un dispositivo admite conjuntos de programas, puede ver las propiedades del dispositivo mediante el SDK de Amazon Braket o consultar la página del dispositivo en la consola de Amazon Braket.

El siguiente ejemplo muestra cómo ejecutar un conjunto de programas.

```
from math import pi
from braket.devices import LocalSimulator
from braket.program_sets import ProgramSet
from braket.circuits import Circuit

program_set = ProgramSet([
    Circuit().h(0).cnot(0,1),
    Circuit().rx(0, pi/4).ry(1, pi/8).cnot(1,0),
    Circuit().t(0).t(1).cz(0,1).s(0).cz(1,2).s(1).s(2),
])

device = LocalSimulator()
result = device.run(program_set, shots=300).result()
print(result[0][0].counts)  # The result of the first program in the program set
```

Para obtener más información sobre las diferentes formas de construir un conjunto de programas (por ejemplo, construir un conjunto de programas a partir de muchos observables o parámetros con un solo programa) y recuperar los resultados del conjunto de programas, consulte la sección de conjuntos de programas de la Guía para desarrolladores de Amazon Braket y la carpeta de conjuntos de programas del repositorio de GitHub de ejemplos de Braket.

El procesamiento por lotes de tareas cuánticas está disponible en todos los dispositivos de Amazon Braket. El procesamiento por lotes es especialmente útil para las tareas cuánticas que se ejecutan en los simuladores bajo demanda (SV1, DM1 o TN1) porque pueden procesar varias tareas cuánticas en paralelo. El procesamiento por lotes le permite iniciar tareas cuánticas en paralelo. Por ejemplo, si desea realizar un cálculo que requiera 10 tareas cuánticas y los programas de esas tareas cuánticas son independientes entre sí, se recomienda utilizar el procesamiento por lotes de tareas. Utilice el procesamiento de tareas cuánticas por lotes cuando ejecute cargas de trabajo con varios programas en un dispositivo que no admita conjuntos de programas.

El siguiente ejemplo muestra cómo ejecutar un lote de tareas cuánticas.

```
from braket.circuits import Circuit
from braket.devices import LocalSimulator

bell = Circuit().h(0).cnot(0, 1)
circuits = [bell for _ in range(5)]

device = LocalSimulator()
batch = device.run_batch(circuits, shots=100)
print(batch.results()[0].measurement_counts)  # The result of the first quantum task in
 the batch
```

Para obtener información más específica sobre el procesamiento por lotes, consulta los ejemplos de Amazon Braket en. GitHub

En esta sección:

- Acerca del conjunto de programas y los costos
- Acerca del procesamiento por lotes de tareas cuánticas y los costos
- Procesamiento por lotes de tareas cuánticas y PennyLane
- Procesamiento por lotes de tareas y circuitos parametrizados

Acerca del conjunto de programas y los costos

Los conjuntos de programas ejecutan varios programas cuánticos de manera eficiente al empaquetar hasta 100 programas o conjuntos de parámetros en una sola tarea cuántica. Con los conjuntos de programas, solo paga una tarifa por tarea más las tarifas por shot en función del total de shots de todos los programas, lo que reduce considerablemente los costos en comparación con el envío de los programas de forma individual. Este enfoque resulta especialmente ventajoso para las cargas de

trabajo con muchos programas y con un número reducido de shots por programa. Actualmente, los conjuntos de programas son compatibles con los dispositivos IQM y Rigetti y con el simulador local de Amazon Braket.

Para obtener más información, consulte la sección Conjuntos de programas para ver los pasos de implementación detallados, las mejores prácticas y los ejemplos de código.

Acerca del procesamiento por lotes de tareas cuánticas y los costos

Algunas advertencias a tener en cuenta con respecto a los costos de facturación y el procesamiento por lotes de tareas cuánticas:

- De forma predeterminada, el procesamiento por lotes de tareas cuánticas se reintenta cada vez que se agota el tiempo de espera o las tareas cuánticas fallan 3 veces.
- Un lote de tareas cuánticas de larga duración, como 34 qubits para SV1, puede generar grandes costos. Asegúrese de comprobar con cuidado los valores de las asignaciones de `run_batch` antes de iniciar un lote de tareas cuánticas. No recomendamos el uso de TN1 con `run_batch`.
- TN1pueden incurrir en costes si no se realizan tareas de la fase de ensayo (consulte la TN1 descripción para obtener más información). Los reintentos automáticos pueden aumentar el costo, por lo que recomendamos establecer «max_retries» en el procesamiento por lotes en 0 cuando se utilice TN1 (consulte Procesamiento por lotes de tareas cuánticas, línea 186).

Procesamiento por lotes de tareas cuánticas y PennyLane

Aprovecha el procesamiento por lotes cuando utilices PennyLane Amazon Braket `parallel = True` configurando cuándo instancias un dispositivo Amazon Braket, como se muestra en el siguiente ejemplo.

```
import pennylane as qml

# Define the number of wires (qubits) you want to use
wires = 2  # For example, using 2 qubits

# Define your S3 bucket
my_bucket = "amazon-braket-s3-demo-bucket"
my_prefix = "pennylane-batch-output"
s3_folder = (my_bucket, my_prefix)

device = qml.device("braket.aws.qubit",
```

```
                    device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
                    wires=wires,
                    s3_destination_folder=s3_folder,
                    parallel=True)
```

Para obtener más información sobre el procesamiento por lotes con PennyLane, consulte Optimización paralelizada de circuitos cuánticos.

Procesamiento por lotes de tareas y circuitos parametrizados

Al enviar un lote de tareas cuánticas que contenga circuitos parametrizados, puede proporcionar un diccionario de `inputs`, que se utiliza para todas las tareas cuánticas del lote o una `list` de diccionarios de entrada, en cuyo caso el diccionario `i`-th se empareja con la tarea `i`-th, como se muestra en el siguiente ejemplo.

```
from braket.circuits import Circuit, FreeParameter, Observable
from braket.aws import AwsQuantumTaskBatch, AwsDevice

# Define your quantum device
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")

# Create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')

# Create two circuits
circ_a = Circuit().rx(0, alpha).ry(1, alpha).cnot(0, 2).xx(0, 2, beta)
circ_a.variance(observable=Observable.Z(), target=0)

circ_b = Circuit().rx(0, alpha).rz(1, alpha).cnot(0, 2).zz(0, 2, beta)
circ_b.expectation(observable=Observable.Z(), target=2)

# Use the same inputs for both circuits in one batch
tasks = device.run_batch([circ_a, circ_b], inputs={'alpha': 0.1, 'beta': 0.2})

# Or provide each task its own set of inputs
inputs_list = [{'alpha': 0.3, 'beta': 0.1}, {'alpha': 0.1, 'beta': 0.4}]

tasks = device.run_batch([circ_a, circ_b], inputs=inputs_list)
```

También puede preparar una lista de diccionarios de entrada para un único circuito paramétrico y enviarlos como un lote de tareas cuánticas. Si hay N diccionarios de entrada en la lista, el

lote contiene N tareas cuánticas. La tarea cuántica `i`-th corresponde al circuito ejecutado con el diccionario de entrada `i`-th.

```
from braket.circuits import Circuit, FreeParameter

# Create a parametric circuit
circ = Circuit().rx(0, FreeParameter('alpha'))

# Provide a list of inputs to execute with the circuit
inputs_list = [{'alpha': 0.1}, {'alpha': 0.2}, {'alpha': 0.3}]

tasks = device.run_batch(circ, inputs=inputs_list, shots=100)
```

¿Cuándo se ejecutará mi tarea cuántica?

Cuando envía un circuito, Amazon Braket lo envía al dispositivo que especifique. Las tareas cuánticas de la unidad de procesamiento cuántico (QPU) y del simulador bajo demanda se ponen en cola y se procesan en el orden en que se reciben. El tiempo necesario para procesar una tarea cuántica después de enviarla varía en función del número y la complejidad de las tareas enviadas por otros clientes de Amazon Braket y de la disponibilidad de la QPU seleccionada.

En esta sección:

- Periodos de disponibilidad de la QPU y estado
- Visibilidad de las colas
- Configuración de notificaciones por correo electrónico o SMS

Periodos de disponibilidad de la QPU y estado

La disponibilidad de la QPU varía de un dispositivo a otro.

En la página Dispositivos de la consola de Amazon Braket puede ver los periodos de disponibilidad actuales y futuros, y el estado del dispositivo. Además, cada página del dispositivo muestra las profundidades de las colas individuales de las tareas cuánticas e híbridas.

Se considera que un dispositivo está desconectado si no está disponible para los clientes, independientemente del período de disponibilidad. Por ejemplo, podría estar desconectado debido a tareas de mantenimiento programadas, actualizaciones o problemas operativos.

Visibilidad de las colas

Antes de enviar una tarea cuántica o un trabajo híbrido, puede ver cuántas tareas cuánticas o trabajos híbridos tiene por delante comprobando la profundidad de la cola del dispositivo.

Profundidad de la cola

La Queue depth se refiere a la cantidad de tareas cuánticas y trabajos híbridos en cola para un dispositivo en particular. Se puede acceder a las tareas cuánticas y al recuento de la cola de trabajos híbridos de un dispositivo a través del Braket Software Development Kit (SDK) o de la Amazon Braket Management Console.

1. La profundidad de la cola de tareas se refiere al número total de tareas cuánticas que están actualmente esperando para ejecutarse con una prioridad normal.
2. La profundidad de la cola de tareas prioritarias se refiere al número total de tareas cuánticas enviadas que están esperando para ejecutarse a través de Amazon Braket Hybrid Jobs. Estas tareas se ejecutan antes que las tareas independientes.
3. La profundidad de la cola de trabajos híbridos se refiere al número total de trabajos híbridos que están actualmente en cola en un dispositivo. Las Quantum tasks enviadas como parte de un trabajo híbrido tienen prioridad y se añaden en la Priority Task Queue.

Los clientes que deseen ver la profundidad de la cola a través de Braket SDK pueden modificar el siguiente fragmento de código para obtener la posición en la cola de su tarea cuántica o trabajo híbrido:

```
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1")

# returns the number of quantum tasks queued on the device
print(device.queue_depth().quantum_tasks)
{<QueueType.NORMAL: 'Normal'>: '0', <QueueType.PRIORITY: 'Priority'>: '0'}


# returns the number of hybrid jobs queued on the device
print(device.queue_depth().jobs)
'3'
```

Enviar una tarea cuántica o un trabajo híbrido a una QPU puede provocar que su carga de trabajo se establezca en el estado `QUEUED`. Amazon Braket proporciona a los clientes visibilidad sobre la posición en la cola de sus tareas cuánticas y trabajos híbridos.

Posición en la cola

Queue position se refiere a la posición actual de su tarea cuántica o trabajo híbrido en la cola del dispositivo correspondiente. Esta se puede obtener para tareas cuánticas o trabajos híbridos a través del Braket Software Development Kit (SDK) o de la Amazon Braket Management Console.

Los clientes que deseen ver la posición en la cola a través de Braket SDK pueden modificar el siguiente fragmento de código para obtener la posición en la cola de su tarea cuántica o trabajo híbrido:

```
# choose the device to run your circuit
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")

#execute the circuit
task = device.run(bell, s3_folder, shots=100)

# retrieve the queue position information
print(task.queue_position().queue_position)

# Returns the number of Quantum Tasks queued ahead of you
'2'


from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=False
)

# retrieve the queue position information
print(job.queue_position().queue_position)
'3' # returns the number of hybrid jobs queued ahead of you
```

Configuración de notificaciones por correo electrónico o SMS

Amazon Braket envía eventos a Amazon EventBridge cuando cambia la disponibilidad de una QPU o cuando cambia el estado de una tarea cuántica. Siga estos pasos para recibir notificaciones de cambios en el estado del dispositivo y de las tareas cuánticas por correo electrónico o SMS:

1. Cree un tema de Amazon SNS y una suscripción a correo electrónico o SMS. La disponibilidad de correo electrónico o SMS depende de su región. Para obtener más información, consulte Introducción a Amazon SNS y Envío de mensajes SMS.
2. Cree una regla EventBridge que active las notificaciones a su tema de SNS. Para obtener más información, consulta Cómo monitorizar Amazon Braket con Amazon. EventBridge

(Opcional) Configuración de notificaciones SNS

Puede configurar notificaciones a través de Amazon Simple Notification Service (SNS) para que reciba una alerta cuando se complete su tarea cuántica de Amazon Braket. Las notificaciones activas son útiles si prevé un tiempo de espera prolongado; por ejemplo, cuando envía una tarea cuántica grande o cuando envía una tarea cuántica fuera del periodo de disponibilidad de un dispositivo. Si no desea esperar a que se complete la tarea cuántica, puede configurar una notificación de SNS.

Un cuaderno de Amazon Braket le guiará a través de los pasos de configuración. Para obtener más información, consulta los ejemplos de Amazon Braket GitHub y, específicamente, el ejemplo de bloc de notas para configurar las notificaciones.

Trabajar con reservas

Las reservas le dan acceso exclusivo al dispositivo cuántico que elija. Puede programar una reserva cuando le resulte más cómodo, de modo que sepa exactamente cuándo comienza y finaliza la ejecución de su carga de trabajo. Las reservas están disponibles en incrementos de 1 hora para todos los dispositivos Braket y se pueden cancelar con hasta 48 horas de antelación, sin coste adicional. Recomendamos poner en cola las tareas cuánticas y los trabajos híbridos para una próxima reserva con antelación, utilizar su ARN de reserva directa de Braket o enviar las cargas de trabajo durante la reserva.

El coste del acceso a un dispositivo dedicado se basa en la duración de tu reserva, independientemente del número de tareas cuánticas e híbridas que ejecutes en la unidad de procesamiento cuántico (QPU). Puedes encontrar una lista actualizada de ordenadores cuánticos

disponibles para reservas en nuestra página de precios o a través de la consola de administración de Amazon Braket.

> **Note**
>
> En una reserva, no hay límites de entradas. Además, en el caso de IonQ los dispositivos, el número mínimo de disparos para las tareas de mitigación de errores se reduce a 500 (frente a 2500 para las tareas bajo demanda).

Cuándo usar una reserva

Aprovechar el acceso mediante reserva le proporciona la comodidad y la previsibilidad de saber exactamente cuándo comienza y finaliza la ejecución de su carga de trabajo cuántica. En comparación con el envío de tareas y trabajos híbridos bajo demanda, no tendrá que esperar en una cola con las tareas de otros clientes. Como tiene acceso exclusivo al dispositivo durante la reserva, solo sus cargas de trabajo se ejecutarán en el dispositivo durante toda la reserva.

Recomendamos utilizar el acceso bajo demanda para la fase de diseño y creación de prototipos de su investigación, lo que permitirá una iteración rápida y rentable de los algoritmos. Cuando esté listo para producir los resultados finales del experimento, considere programar una reserva del dispositivo cuando le resulte conveniente para asegurarse de que puede cumplir con los plazos del proyecto o de publicación. También recomendamos utilizar reservas cuando desee ejecutar tareas en momentos específicos, como cuando realiza una demostración en directo o un taller sobre una computadora cuántica.

En esta sección:

Cómo crear una reserva

Para crear una reserva, póngase en contacto con el equipo de Braket siguiendo estos pasos:

1. Abra la consola de Amazon Braket.
2. Seleccione Braket Direct en el panel izquierdo y, a continuación, en la sección Reservas, seleccione Reservar dispositivo.
3. Seleccione el dispositivo que desea reservar.
4. Proporcione su información de contacto, incluidos el nombre y el correo electrónico. Asegúrese de proporcionar una dirección de correo electrónico válida que consulte con frecuencia.
5. En Cuéntenos sobre su carga de trabajo, proporcione todos los detalles sobre la carga de trabajo necesaria para utilizar su reserva. Por ejemplo, la duración de la reserva deseada, las restricciones relevantes o la programación deseada.

Tras enviar el formulario, recibirá un correo electrónico del equipo de Braket con los siguientes pasos. Una vez que se confirme su reserva, recibirá el ARN de reserva en su correo electrónico. Necesitará usar el ARN de reserva para crear tareas de reserva. Las tareas creadas sin el ARN de reserva se enviarán a la cola normal bajo demanda y NO se ejecutarán durante la reserva.

> **Note**
>
> Su reserva solo estará confirmada una vez que reciba el ARN de reserva.

Las reservas están disponibles en incrementos de 1 hora como mínimo y algunos dispositivos pueden tener restricciones de duración de reserva adicionales (incluidas las duraciones mínima y máxima de reserva). El equipo de Braket compartirá con usted cualquier información relevante antes de confirmar la reserva.

El equipo de Braket se pondrá en contacto contigo a través de tu correo electrónico para concertar una sesión de 30 minutos con un experto de Braket.

Ejecución de tareas cuánticas durante una reserva

Tras obtener un ARN de reserva válido con la opción Crear una reserva, puede crear tareas cuánticas para ejecutarlas durante la reserva. Las tareas de Quantum y los trabajos híbridos enviados con un ARN de reserva no aparecerán en la cola de dispositivos. Las tareas enviadas antes de la hora de inicio de la reserva permanecerán en ese `QUEUED` estado hasta que comience la reserva.

> **Note**
>
> Las reservas son específicas AWS de la cuenta y del dispositivo. Solo la AWS cuenta que creó la reserva puede usar su ARN de reserva.
> Durante una reserva, se pueden crear tanto tareas de reserva como tareas normales.
> Para comprobar que una tarea cuántica de Braket creada está asociada a una reserva, compruebe el campo «ARN de reserva» en la página de la tarea cuántica en la consola de Braket o consulte el mismo campo en los metadatos de la tarea mediante el SDK. En el resto de esta página, se describe cómo especificar qué tareas están asociadas a la reserva.

Puede crear tareas cuánticas con Python SDKsBraket,, CUDA-QPennyLaneQiskit, o directamente con boto3 (trabajar con Boto3). Para utilizar las reservas, debe tener la versión v1.79.0 o superior del Python SDK de Amazon Braket. Puede actualizar el SDK de Braket, el proveedor de Qiskit y el plugin de PennyLane de Braket a la versión más reciente con el siguiente código.

```
pip install --upgrade amazon-braket-sdk amazon-braket-pennylane-plugin qiskit-braket-provider
```

Ejecución de tareas con el administrador de contexto de **`DirectReservation`**

La forma recomendada de ejecutar una tarea en su reserva programada es utilizar el administrador de contexto de `DirectReservation`. Al especificar su dispositivo de destino y el ARN de reserva, el administrador de contexto garantiza que todas las tareas creadas en la declaración `with` de Python se ejecuten con acceso exclusivo al dispositivo.

Primero, defina un circuito cuántico y el dispositivo. A continuación, utilice el contexto de reserva y ejecute la tarea. Asegúrese de que toda su carga de trabajo se ejecute dentro del **`with`** bloque; ¡todo lo que se ejecute fuera del alcance del **`with`** bloque no se asociará a su reserva!

```
from braket.aws import AwsDevice, DirectReservation
from braket.circuits import Circuit
from braket.devices import Devices

bell = Circuit().h(0).cnot(0, 1)
device = AwsDevice(Devices.IonQ.ForteEnterprise1)

# run the circuit in a reservation
with DirectReservation(device, reservation_arn="<my_reservation_arn>"):
```

```
    task = device.run(bell, shots=100)
```

Puedes crear tareas cuánticas en una reserva mediante CUDA-QPennyLane, y Qiskit complementos, siempre que el `DirectReservation` contexto esté activo al crear tareas cuánticas. Por ejemplo, con el proveedor de Qiskit-Braket, puede ejecutar las tareas de la siguiente manera.

```
from braket.devices import Devices
from braket.aws import DirectReservation
from qiskit import QuantumCircuit
from qiskit_braket_provider import BraketProvider


qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)

qpu = BraketProvider().get_backend("Forte Enterprise 1")

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    qpu_task = qpu.run(qc, shots=10)
```

Del mismo modo, el siguiente código ejecuta un circuito durante una reserva mediante el complemento Braket-PennyLane.

```
from braket.devices import Devices
from braket.aws import DirectReservation
import pennylane as qml


dev = qml.device("braket.aws.qubit", device_arn=Devices.IonQ.ForteEnterprise1.value,
 wires=2, shots=10)

@qml.qnode(dev)
def bell_state():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    return qml.probs(wires=[0, 1])

# run the circuit in a reservation
```

```
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    probs = bell_state()
```

Configuración manual del contexto de reserva

También puede configurar el contexto de reserva de forma manual con el siguiente código.

```
# set reservation context
reservation_context = DirectReservation(device,
 reservation_arn="<my_reservation_arn>").start()

# run circuit during reservation
task = device.run(bell, shots=100)
```

Esto es ideal para los cuadernos de Jupyter, donde el contexto se puede ejecutar en la primera celda y todas las tareas subsiguientes se ejecutarán en la reserva.

> **Note**
>
> La celda que contiene la llamada a `.start()` solo debe ejecutarse una vez.

Para volver al modo bajo demanda: reinicie el cuaderno de Jupyter o llame a lo siguiente para volver a cambiar el contexto al modo bajo demanda.

```
reservation_context.stop()  # unset reservation context
```

> **Note**
>
> Las reservas tienen una hora de inicio y finalización predeterminadas (consulte Crear una reserva). Los métodos `reservation_context.start()` y `reservation_context.stop()` no inician ni finalizan una reserva. En cambio, mientras el contexto esté activo, cualquier tarea cuántica que crees se asociará a tu reserva y solo se ejecutará durante la reserva programada. El contexto de la reserva no afecta a la hora de reserva programada.

Transferencia explícita del ARN de reserva al crear la tarea

Otra forma de crear tareas durante una reserva es transferir explícitamente el ARN de reserva al llamar a `device.run()`.

```
task = device.run(bell, shots=100, reservation_arn="<my_reservation_arn>")
```

Este método asocia directamente la tarea cuántica con el ARN de reserva, lo que garantiza que se ejecute durante el período reservado. Para esta opción, añada el ARN de reserva a cada tarea que vaya a ejecutar durante una reserva. Sin embargo, tenga en cuenta que al utilizar bibliotecas de tercerosPennyLane, como Qiskit o, puede resultar difícil garantizar que las tareas enviadas utilicen el ARN de reserva correcto. Por este motivo, se recomienda utilizar el administrador de DirectReservation contexto.

Cuando utilice boto3 directamente, transfiera el ARN de reserva como una asociación al crear una tarea.

```
import boto3

braket_client = boto3.client("braket")


kwargs["associations"] = [
    {
        "arn": "<my_reservation_arn>",
        "type": "RESERVATION_TIME_WINDOW_ARN",
    }
]

response = braket_client.create_quantum_task(**kwargs)
```

Ejecución de trabajos híbridos durante una reserva

Una vez que tenga una función de Python para ejecutar como un trabajo híbrido, puede ejecutar el trabajo híbrido en una reserva transfiriendo el argumento de palabra clave `reservation_arn`. Todas las tareas del trabajo híbrido utilizan el ARN de reserva. Es importante destacar que el trabajo híbrido con `reservation_arn` solo activa el cómputo clásico una vez que se inicia la reserva.

> **Note**
>
> Un trabajo híbrido que se ejecuta durante una reserva solo ejecuta correctamente las tareas cuánticas en el dispositivo reservado. El intento de utilizar un dispositivo Braket bajo

demanda diferente generará un error. Si necesita ejecutar tareas tanto en un simulador bajo demanda como en el dispositivo reservado en el mismo trabajo híbrido, utilice `DirectReservation` en su lugar.

El siguiente código muestra cómo ejecutar un trabajo híbrido durante una reserva.

```
from braket.aws import AwsDevice
from braket.devices import Devices
from braket.jobs import get_job_device_arn, hybrid_job

@hybrid_job(device=Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>")
def example_hybrid_job():
    # declare AwsDevice within the hybrid job
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)

    task = device.run(bell, shots=10)
```

En el caso de los trabajos híbridos que utilizan un script de Python (consulte la sección acerca de crear su primer trabajo híbrido en la guía para desarrolladores), puede ejecutarlos en la reserva transfiriendo el argumento de palabra clave `reservation_arn` al crear el trabajo.

```
from braket.aws import AwsQuantumJob
from braket.devices import Devices

job = AwsQuantumJob.create(
    Devices.IonQ.ForteEnterprise1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="<my_reservation_arn>"
)
```

Qué ocurre al final de la reserva

Una vez finalizada la reserva, ya no tendrá acceso exclusivo al dispositivo. Las cargas de trabajo restantes que estén en cola con esta reserva se cancelarán automáticamente.

> **Note**
>
> Cuando la reserva finalice, se cancelarán los trabajos con el estado `RUNNING`. Recomendamos el uso de puntos de comprobación para guardar y reiniciar trabajos según lo necesite.

Una reserva en curso (después del inicio de la reserva y antes del final de la reserva) no se puede ampliar porque cada reserva representa un acceso independiente y exclusivo al dispositivo. Por ejemplo, dos back-to-back reservas se consideran independientes y las tareas pendientes de la primera reserva se cancelan automáticamente. No se reanuda en la segunda reserva.

> **Note**
>
> Las reservas representan un acceso exclusivo a un dispositivo para tu AWS cuenta. Incluso si el dispositivo permanece inactivo, ningún otro cliente puede usarlo. Por lo tanto, se le cobrará por la duración del tiempo reservado, independientemente del tiempo utilizado.

Cancelación o reprogramación de una reserva existente

Puede cancelar su reserva al menos 48 horas antes de la hora de inicio programada. Para cancelarla, solicítelo respondiendo al correo electrónico de confirmación de reserva que recibió.

Para reprogramarla, debe cancelar su reserva actual y, a continuación, crear una nueva.

Técnicas de mitigación de errores

La mitigación de errores cuánticos es un conjunto de técnicas destinadas a reducir los efectos de los errores en las computadoras cuánticas.

Los dispositivos cuánticos están sujetos al ruido ambiental que degrada la calidad de los cómputos realizados. Si bien la computación cuántica tolerante a errores promete una solución a este problema, los dispositivos cuánticos actuales están limitados por el número de qubits y por unas tasas de error relativamente altas. Para combatir esta situación a corto plazo, los investigadores están estudiando métodos para mejorar la precisión de la computación cuántica ruidosa. Este enfoque, conocido como mitigación de errores cuánticos, implica el uso de varias técnicas para extraer la mejor señal de los datos de medición ruidosos.

En esta sección:

Técnicas de mitigación de errores en dispositivos de IonQ

La mitigación de errores implica ejecutar varios circuitos físicos y combinar sus mediciones para obtener un resultado mejorado.

> **Note**
>
> Para todos los dispositivos de IonQ: cuando se utiliza un modelo bajo demanda, hay un límite de gateshot de 1 millón y un mínimo de 2500 shots para las tareas de mitigación de errores. Para una reserva directa, no hay límite de gateshot y hay un mínimo de 500 shots para tareas de mitigación de errores.

Eliminación de sesgos

Los dispositivos de IonQ incluyen un método de mitigación de errores denominado eliminación de sesgos.

La eliminación de sesgos mapea un circuito en múltiples variantes que actúan sobre diferentes permutaciones de qubits o con diferentes descomposiciones de puertas. Esto reduce el efecto de errores sistemáticos, como las rotaciones excesivas de las puertas o un solo qubit defectuoso, mediante el uso de diferentes implementaciones de un circuito que, de otro modo, podrían sesgar los resultados de las mediciones. Esto supone una sobrecarga adicional para calibrar varios qubits y puertas.

Para obtener más información sobre la eliminación de sesgos, consulte Mejora del rendimiento de las computadoras cuánticas mediante la simetrización.

> **Note**
>
> El uso de la eliminación de sesgos requiere un mínimo de 2500 shots.

Puede ejecutar una tarea cuántica con eliminación de sesgos en un dispositivo IonQ mediante el siguiente código:

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.error_mitigation import Debias

# choose an IonQ device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1")
circuit = Circuit().h(0).cnot(0, 1)

task = device.run(circuit, shots=2500, device_parameters={"errorMitigation": Debias()})

result = task.result()
print(result.measurement_counts)
>>> {"00": 1245, "01": 5, "10": 10 "11": 1240} # result from debiasing
```

Cuando se complete la tarea cuántica, podrá ver las probabilidades de medición y cualquier tipo de resultado de la tarea cuántica. Las probabilidades de medición y los recuentos de todas las variantes se agregan en una única distribución. Todos los tipos de resultados especificados en el circuito, como los valores esperados, se calculan utilizando los recuentos de mediciones agregados.

Definición

También puede acceder a las probabilidades de medición computadas con una estrategia de posprocesamiento diferente denominada definición. La definición compara los resultados de cada variante y descarta los shots incoherentes, lo que favorece el resultado de medición más probable en todas las variantes. Para obtener más información, consulte Mejora del rendimiento de las computadoras cuánticas mediante la simetrizacion.

Es importante destacar que la definición asume que la distribución de salida es escasa, con pocos estados de alta probabilidad y muchos estados de probabilidad cero. Si esta suposición no es válida, puede distorsionar la distribución de probabilidad.

Puede acceder a las probabilidades desde una distribución definida en el campo `additional_metadata` en el `GateModelTaskResult` del SDK de Python de Braket. Tenga en cuenta que la definición no devuelve los recuentos de mediciones, sino que devuelve una distribución de probabilidad renormalizada. El siguiente fragmento de código muestra cómo acceder a la distribución después de la definición.

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

Trabajar con trabajos híbridos de Amazon Braket

Amazon Braket Hybrid Jobs le ofrece una forma de ejecutar algoritmos híbridos cuántico-clásicos que requieren tanto AWS recursos clásicos como unidades de procesamiento cuántico (). QPUs Los trabajos híbridos están diseñados para activar los recursos clásicos solicitados, ejecutar su algoritmo y liberar las instancias una vez completados, de modo que solo pague por lo que use.

Los trabajos híbridos son ideales para algoritmos iterativos de larga duración que implican el uso tanto de recursos de computación clásica como de recursos de computación cuántica. Con los trabajos híbridos, después de enviar su algoritmo para su ejecución, Braket lo ejecutará en un entorno escalable y en contenedor. Una vez que el algoritmo se haya completado, puede recuperar los resultados.

Además, las tareas cuánticas que se crean a partir de un trabajo híbrido se benefician de una mayor prioridad al hacer cola en el dispositivo de QPU de destino. Esta priorización garantiza que sus cálculos cuánticos se procesen y ejecuten antes que otras tareas que esperan en la cola. Esto resulta especialmente ventajoso para los algoritmos híbridos iterativos, en los que los resultados de una tarea cuántica dependen de los resultados de tareas cuánticas anteriores. Algunos ejemplos de estos algoritmos incluyen el algoritmo de optimización cuántica aproximada (QAOA), el solucionador de problemas de autovalores cuánticos variacionales o la machine learning cuántica. También puede supervisar el progreso de su algoritmo casi en tiempo real, lo que le permite realizar un seguimiento de los costos, el presupuesto o métricas personalizadas, como la pérdida de entrenamiento o los valores esperados.

Puede acceder a los trabajos híbridos en Braket mediante:

- el SDK de Python de Amazon Braket;
- la consola de Amazon Braket; o
- la API de Amazon Braket.

En esta sección:

Cuándo utilizar los trabajos híbridos de Amazon Braket

Los trabajos híbridos de Amazon Braket le permiten ejecutar algoritmos híbridos cuánticos-clásicos, como el solucionador de problemas de autovalores cuánticos variacionales (VQE) y el algoritmo de optimización cuántica aproximada (QAOA), que combinan recursos de computación clásica con dispositivos de computación cuántica para optimizar el rendimiento de los sistemas cuánticos actuales. Los trabajos híbridos de Amazon Braket ofrecen tres ventajas principales:

1. Rendimiento: los trabajos híbridos de Amazon Braket ofrecen un mejor rendimiento que la ejecución de algoritmos híbridos desde su propio entorno. Mientras se ejecuta su trabajo, tiene acceso prioritario a la QPU de destino seleccionada. Las tareas de tu trabajo se ejecutan antes que las demás tareas que están en cola en el dispositivo. Esto se traduce en tiempos de ejecución más cortos y predecibles para los algoritmos híbridos. Los trabajos híbridos de Amazon Braket también admiten la compilación paramétrica. Puede enviar un circuito utilizando parámetros libres y Braket compila el circuito una vez, sin necesidad de recompilarlo para posteriores actualizaciones de parámetros en el mismo circuito, lo que se traduce en tiempos de ejecución aún más rápidos.
2. Comodidad: los trabajos híbridos de Amazon Braket simplifican la configuración y la gestión de su entorno de computación, y mantienen su funcionamiento mientras se ejecuta su algoritmo híbrido. Solo tiene que proporcionar el script de algoritmo y seleccionar un dispositivo cuántico (ya sea una unidad de procesamiento cuántico o un simulador) en el que ejecutarlo. Amazon Braket espera a que el dispositivo de destino esté disponible, activa los recursos clásicos, ejecuta la carga de trabajo en entornos de contenedores preconstruidos, devuelve los resultados a Amazon Simple Storage Service (Amazon S3) y libera los recursos de computación.
3. Métricas: Amazon Braket Hybrid Jobs proporciona on-the-fly información sobre la ejecución de los algoritmos y proporciona métricas de algoritmos personalizables prácticamente en tiempo real a Amazon CloudWatch y a la consola Amazon Braket para que pueda realizar un seguimiento del progreso de sus algoritmos.

Ejecución de un trabajo híbrido con los trabajos híbridos de Amazon Braket

Para ejecutar un trabajo híbrido con los trabajos híbridos de Amazon Braket, primero debe definir su algoritmo. Puede definirlo escribiendo el script del algoritmo y, si lo desea, otros archivos de dependencia mediante el Amazon Braket Python SDK o. PennyLane Si desea utilizar otras bibliotecas (de código abierto o patentadas), puede definir su propia imagen de contenedor personalizada mediante Docker, que incluye estas bibliotecas. Para obtener más información, consulte Utilice su propio contenedor (BYOC).

En cualquier caso, a continuación, cree un trabajo híbrido utilizando la API de Amazon Braket, donde proporcionará el contenedor o el script de algoritmo, seleccionará el dispositivo cuántico de destino que utilizará el trabajo híbrido y, a continuación, elegirá entre una serie de configuraciones opcionales. Los valores predeterminados proporcionados para estas configuraciones opcionales funcionan para la mayoría de los casos de uso. Para que el dispositivo de destino ejecute su trabajo híbrido, puede elegir entre una QPU, un simulador bajo demanda (como SV1, DM1 o TN1) o la propia instancia de trabajo híbrida clásica. Con un simulador bajo demanda o una QPU, su contenedor de trabajos híbridos realiza llamadas a la API a un dispositivo remoto. Con los simuladores integrados, el simulador se integra en el mismo contenedor que el script de algoritmo. Los simuladores Lightning de PennyLane vienen integrados en el contenedor de trabajos híbridos prediseñado por defecto para su uso. Si ejecuta el código con un PennyLane simulador integrado o un simulador personalizado, puede especificar un tipo de instancia, así como el número de instancias que desea utilizar. Consulte la página de precios de Amazon Braket para conocer los costos asociados a cada opción.

Si el dispositivo de destino es un simulador bajo demanda integrado, Amazon Braket comienza a ejecutar el trabajo híbrido de inmediato. Inicia la instancia de trabajo híbrido (puede personalizar el tipo de instancia en la llamada a la API), ejecuta su algoritmo, escribe los resultados en Amazon S3 y libera sus recursos. Esta liberación de recursos garantiza que solo pague por lo que usa.

El número total de trabajos híbridos simultáneos por unidad de procesamiento cuántico (QPU) está restringido. En la actualidad, solo se puede ejecutar un trabajo híbrido en una QPU en un momento determinado. Las colas se utilizan para controlar la cantidad de trabajos híbridos que se pueden ejecutar a fin de no superar el límite permitido. Si el dispositivo de destino es una QPU, el trabajo híbrido entra primero en la cola de trabajos de la QPU seleccionada. Amazon Braket activa la instancia de trabajo híbrido necesaria y ejecuta su trabajo híbrido en el dispositivo. Durante la duración de su algoritmo, su trabajo híbrido tiene acceso prioritario, lo que significa que las tareas cuánticas de su trabajo híbrido se ejecutan antes que otras tareas cuánticas de Braket en cola en el dispositivo, siempre que las tareas cuánticas del trabajo se envíen a la QPU una vez cada pocos minutos. Una vez completado el trabajo híbrido, los recursos se liberan, lo que significa que solo pagará por lo que utilice.

> **Note**
>
> Los dispositivos son regionales y su trabajo híbrido se ejecuta de la Región de AWS misma manera que su dispositivo principal.

Tanto en los escenarios de destino del simulador como de la QPU, tiene la opción de definir métricas de algoritmo personalizadas, como la energía de su hamiltoniano, como parte de su algoritmo. Estas métricas se notifican automáticamente a Amazon CloudWatch y, desde allí, se muestran casi en tiempo real en la consola Amazon Braket.

> **Note**
>
> Si desea utilizar una instancia basada en GPU, asegúrese de utilizar uno de los simuladores basados en GPU disponibles con los simuladores integrados en Braket (por ejemplo, `lightning.gpu`). Si elige uno de los simuladores integrados basados en CPU (por ejemplo, `lightning.qubit` o `braket:default-simulator`), la GPU no se utilizará y usted podría incurrir en gastos innecesarios.

Conceptos clave de los trabajos híbridos

En esta sección se explican los conceptos clave de la función `AwsQuantumJob.create` proporcionada por el SDK de Python de Amazon Braket y su asignación a la estructura del archivo contenedor.

Además del archivo o los archivos que componen el script completo del algoritmo, el trabajo híbrido puede tener entradas y salidas adicionales. Cuando se inicia el trabajo híbrido, Amazon Braket copia las entradas proporcionadas como parte de la creación del trabajo híbrido en el contenedor que ejecuta el script de algoritmo. Cuando se completa el trabajo híbrido, todos los resultados definidos durante el algoritmo se copian en la ubicación de Amazon S3 especificada.

> **Note**
>
> Las métricas del algoritmo se notifican en tiempo real y no siguen este procedimiento de salida.

Amazon Braket también proporciona varias variables de entorno y funciones auxiliares para simplificar las interacciones con las entradas y salidas de los contenedores. Para obtener más información, consulte el paquete braket.jobs del SDK de Amazon Braket.

En esta sección:

- Entradas

- Outputs
- Variables de entorno
- Funciones auxiliares

Entradas

Datos de entrada: los datos de entrada se pueden proporcionar al algoritmo híbrido especificando el archivo de datos de entrada, que está configurado como un diccionario, con el argumento `input_data`. El usuario define el argumento `input_data` dentro de la función `AwsQuantumJob.create` en el SDK. Esto copia los datos de entrada al sistema de archivo contenedor en la ubicación indicada por la variable de entorno `"AMZN_BRAKET_INPUT_DIR"`. Para ver un par de ejemplos de cómo se utilizan los datos de entrada en un algoritmo híbrido, consulte la QAOA con Amazon Braket Hybrid Jobs PennyLane y el aprendizaje automático cuántico en los cuadernos Amazon Braket Hybrid Jobs Jupyter.

> **Note**
>
> Si los datos de entrada son grandes (>1 GB), habrá un largo tiempo de espera antes de que se envíe el trabajo híbrido. Esto se debe a que los datos de entrada locales se cargarán primero en un bucket S3, luego se añadirá la ruta S3 a la solicitud de trabajo híbrido y, por último, la solicitud de trabajo híbrido se enviará al servicio de Braket.

Hiperparámetros: si transfiere`hyperparameters`, estarán disponibles en la variable de entorno `"AMZN_BRAKET_HP_FILE"`.

> **Note**
>
> Para obtener más información sobre cómo crear hiperparámetros y datos de entrada y, a continuación, pasar esta información al script de trabajo híbrido, consulte la sección Uso de hiperparámetros en esta página de GitHub.

Puntos de control: para especificar un `job-arn` cuyo punto de control desea utilizar en un nuevo trabajo híbrido, utilice el comando `copy_checkpoints_from_job`. Este comando copia los datos del punto de control al `checkpoint_configs3Uri` del nuevo trabajo híbrido, de forma que

estén disponibles en la ruta indicada por la variable de entorno `AMZN_BRAKET_CHECKPOINT_DIR` mientras se ejecuta el trabajo. El valor predeterminado es `None`, lo que significa que los datos del punto de control de otro trabajo híbrido no se utilizarán en el nuevo trabajo híbrido.

Outputs

Tareas cuánticas: los resultados de las tareas cuánticas se almacenan en la ubicación `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks` de S3.

Resultados del trabajo: todo lo que el script de algoritmo guarda en el directorio indicado por la variable de entorno `"AMZN_BRAKET_JOB_RESULTS_DIR"` se copia en la ubicación de S3 especificada en `output_data_config`. Si no se especifica el valor, el valor predeterminado es. `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data` Proporcionamos la función auxiliar del SDK **`save_job_result`** , que puede utilizar para almacenar los resultados cómodamente en forma de diccionario cuando se llame desde el script de algoritmo.

Puntos de control: si desea utilizar puntos de control, puede guardarlos en el directorio indicado por la variable de entorno `"AMZN_BRAKET_CHECKPOINT_DIR"`. También puede usar la función auxiliar del SDK `save_job_checkpoint` en su lugar.

Métricas de algoritmos: puedes definir las métricas de los algoritmos como parte del script de tu algoritmo que se emiten a Amazon CloudWatch y se muestran en tiempo real en la consola de Amazon Braket mientras se ejecuta tu trabajo híbrido. Para ver un ejemplo de cómo utilizar las métricas de algoritmo, consulte Uso de los trabajos híbridos de Amazon Braket para ejecutar un algoritmo QAOA.

Para obtener más información sobre cómo guardar los resultados de sus trabajos, consulte Guardar los resultados en la documentación de trabajos híbridos.

Variables de entorno

Amazon Braket proporciona varias variables de entorno para simplificar las interacciones con las entradas y salidas del contenedor. En el siguiente código se enumeran las variables de entorno que utiliza Braket.

- `AMZN_BRAKET_INPUT_DIR`— El directorio opt/braket/input/data de datos de entrada.
- `AMZN_BRAKET_JOB_RESULTS_DIR`— El directorio de salida en el opt/braket/model que se escriben los resultados del trabajo.

- `AMZN_BRAKET_JOB_NAME`: el nombre del trabajo.
- `AMZN_BRAKET_CHECKPOINT_DIR`: el directorio de puntos de control.
- `AMZN_BRAKET_HP_FILE`: el archivo que contiene los hiperparámetros.
- `AMZN_BRAKET_DEVICE_ARN`— El ARN (nombre del AWS recurso) del dispositivo.
- `AMZN_BRAKET_OUT_S3_BUCKET`: el bucket de Amazon S3 de salida, tal como se especifica en la `OutputDataConfig` de la solicitud de `CreateJob`.
- `AMZN_BRAKET_SCRIPT_ENTRY_POINT`: el punto de entrada, tal como se especifica en la `ScriptModeConfig` de la solicitud de `CreateJob`.
- `AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE`: el tipo de compresión, tal como se especifica en la `ScriptModeConfig` de la solicitud de `CreateJob`.
- `AMZN_BRAKET_SCRIPT_S3_URI`: la ubicación en Amazon S3 del script del usuario, tal como se especifica en la `ScriptModeConfig` de la solicitud de `CreateJob`.
- `AMZN_BRAKET_TASK_RESULTS_S3_URI`: la ubicación de Amazon S3 en la que el SDK almacenaría los resultados de las tareas cuánticas de forma predeterminada.
- `AMZN_BRAKET_JOB_RESULTS_S3_PATH`: la ubicación de Amazon S3 en la que se almacenarían los resultados del trabajo, tal como se especifica en la `OutputDataConfig` de la solicitud de `CreateJob`.
- `AMZN_BRAKET_JOB_TOKEN`: la cadena que se debe pasar al parámetro `jobToken` de `CreateQuantumTask` correspondiente a las tareas cuánticas creadas en el contenedor de tareas.

Funciones auxiliares

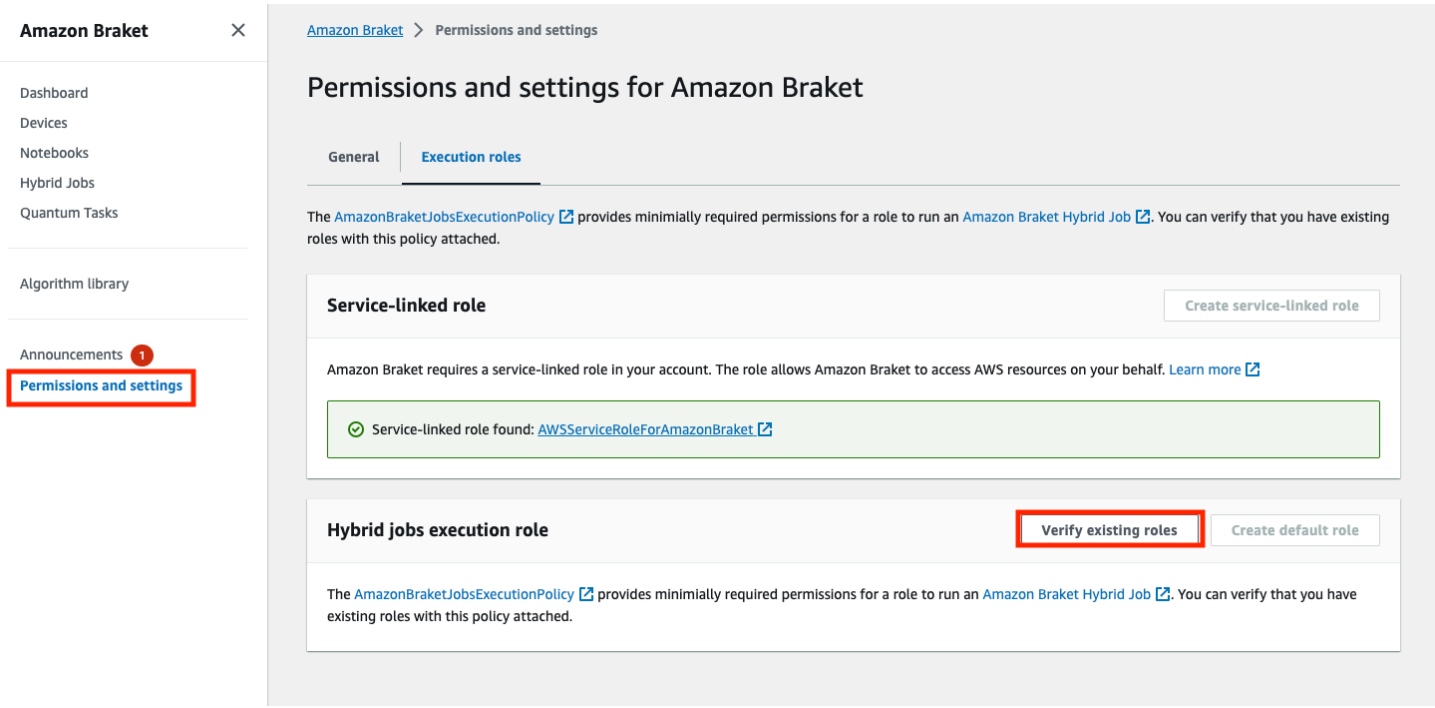
Amazon Braket proporciona varias funciones auxiliares para simplificar las interacciones con las entradas y salidas del contenedor. Estas funciones auxiliares se llamarían desde dentro del script de algoritmo que se utiliza para ejecutar su trabajo híbrido. En el siguiente ejemplo se muestra cómo utilizarlas.

```
from braket.jobs import get_checkpoint_dir, get_hyperparameters, get_input_data_dir,
 get_job_device_arn, get_job_name, get_results_dir, save_job_result,
 save_job_checkpoint, load_job_checkpoint

get_checkpoint_dir() # Get the checkpoint directory
get_hyperparameters() # Get the hyperparameters as strings
get_input_data_dir() # Get the input data directory
get_job_device_arn() # Get the device specified by the hybrid job
```

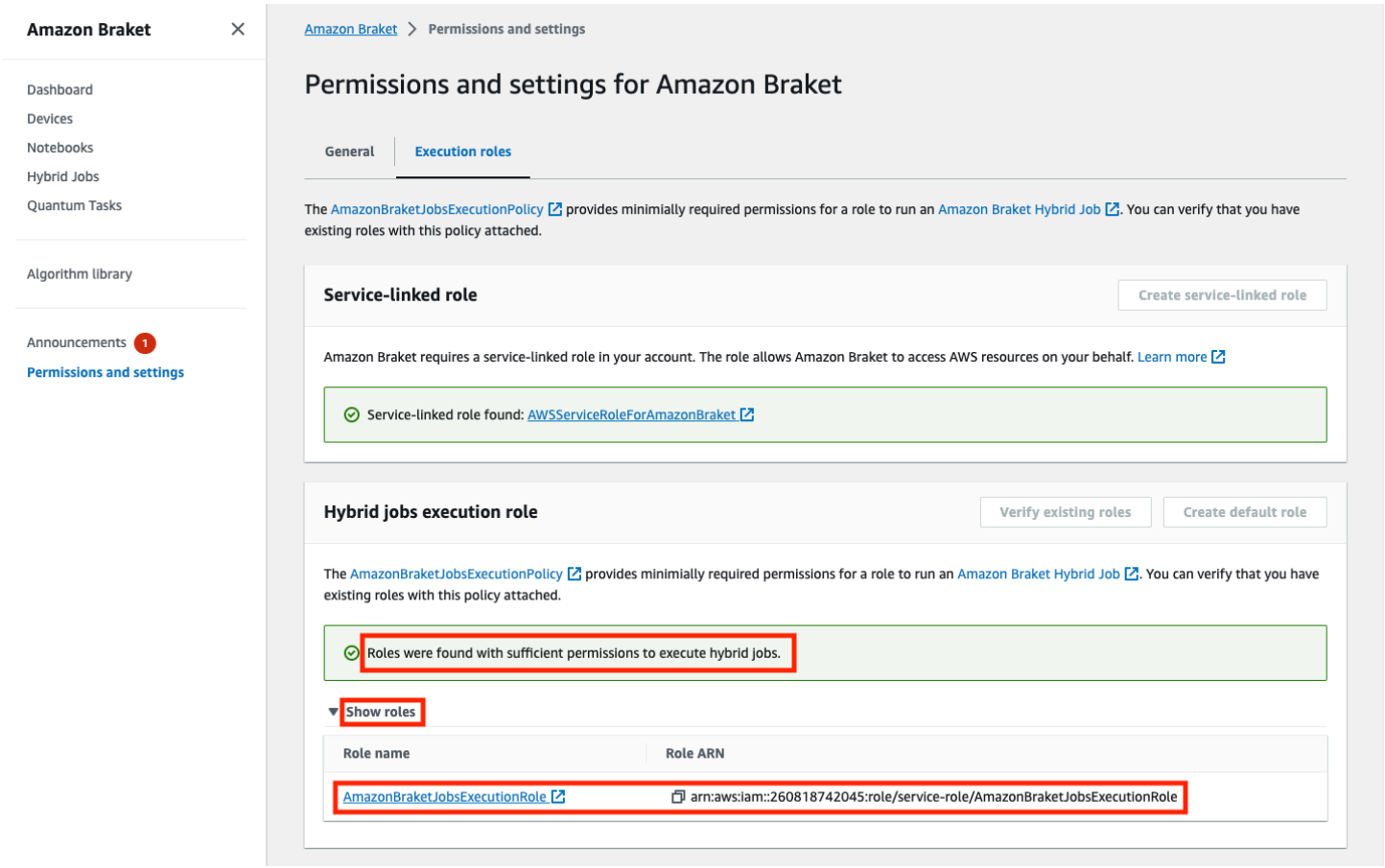
```
get_job_name() # Get the name of the hybrid job.
get_results_dir() # Get the path to a results directory
save_job_result(result_data='data') # Save hybrid job results
save_job_checkpoint(checkpoint_data={'key': 'value'}) # Save a checkpoint
load_job_checkpoint() # Load a previously saved checkpoint
```

Requisitos previos

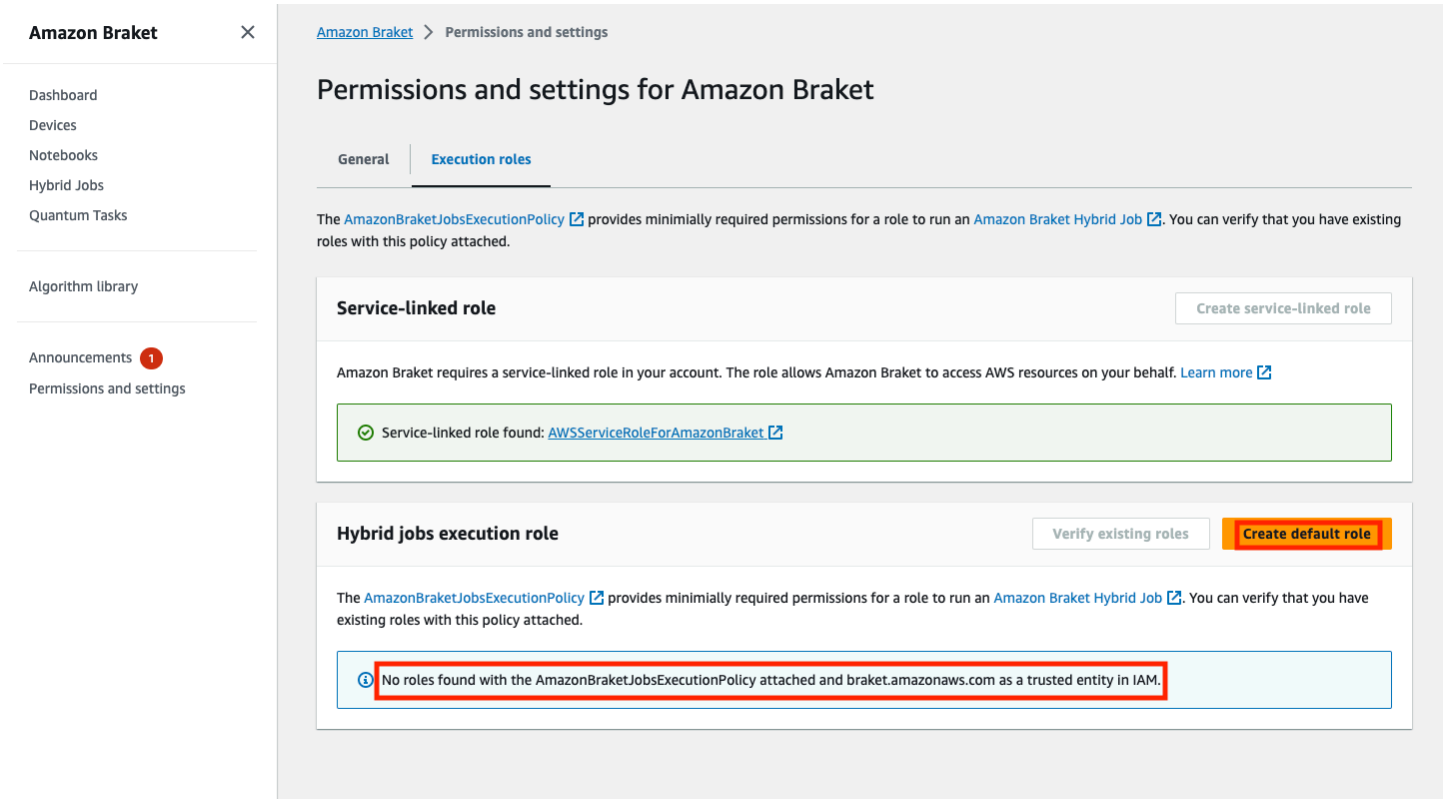
Antes de ejecutar su primer trabajo híbrido, debe asegurarse de que tiene permisos suficientes para continuar con esta tarea. Para determinar si tiene los permisos correctos, seleccione Permisos en el menú de la parte izquierda de la consola de Braket. La página Permisos de administración para Amazon Braket le ayuda a verificar si alguno de sus roles existentes tiene permisos suficientes para ejecutar su trabajo híbrido o le guía a través de la creación de un rol predeterminado que se puede utilizar para ejecutar su trabajo híbrido si aún no dispone de dicho rol.



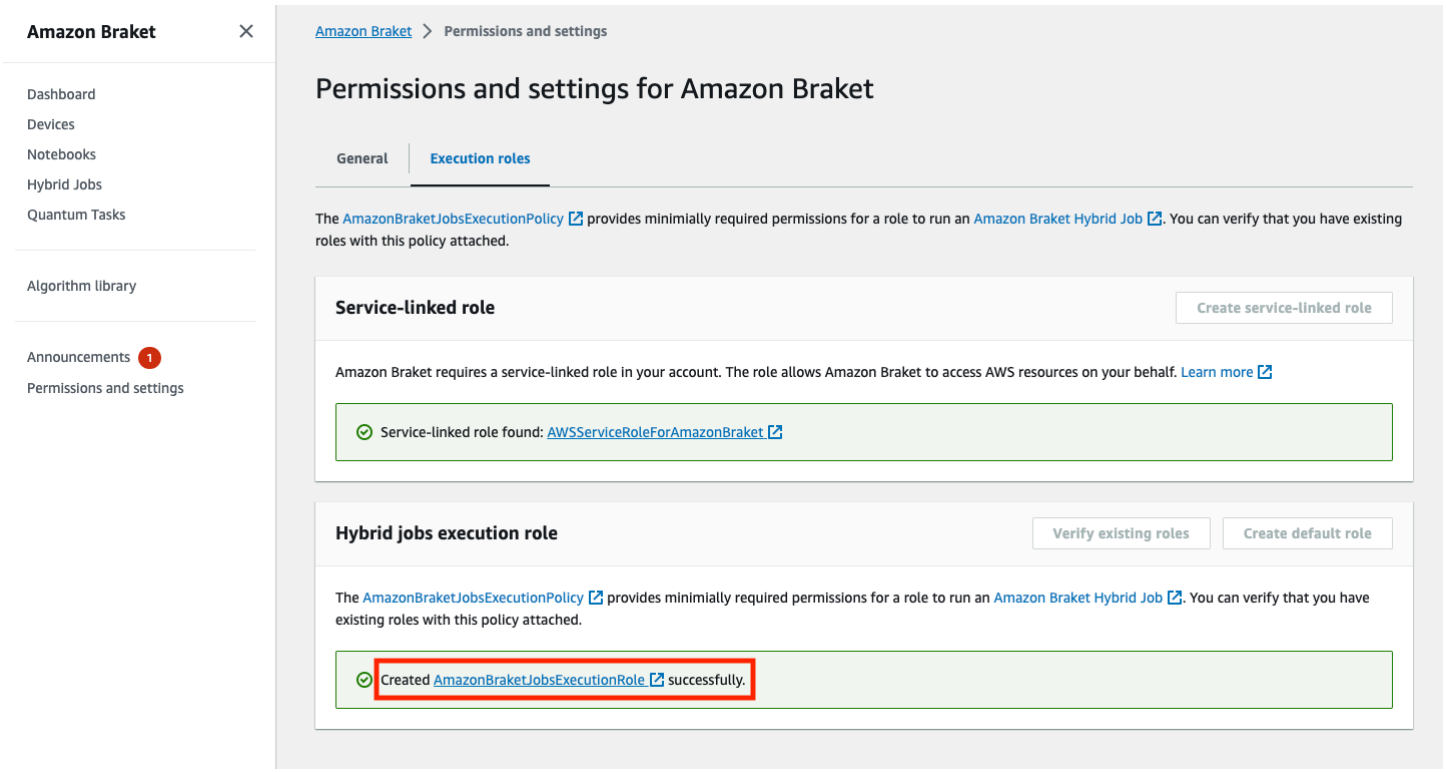
Para comprobar que tiene roles con permisos suficientes para ejecutar un trabajo híbrido, seleccione el botón Verificar el rol existente. Al hacerlo, obtendrá un mensaje que indica que se encontraron los roles. Para ver los nombres de las funciones y su función ARNs, seleccione el botón Mostrar funciones.



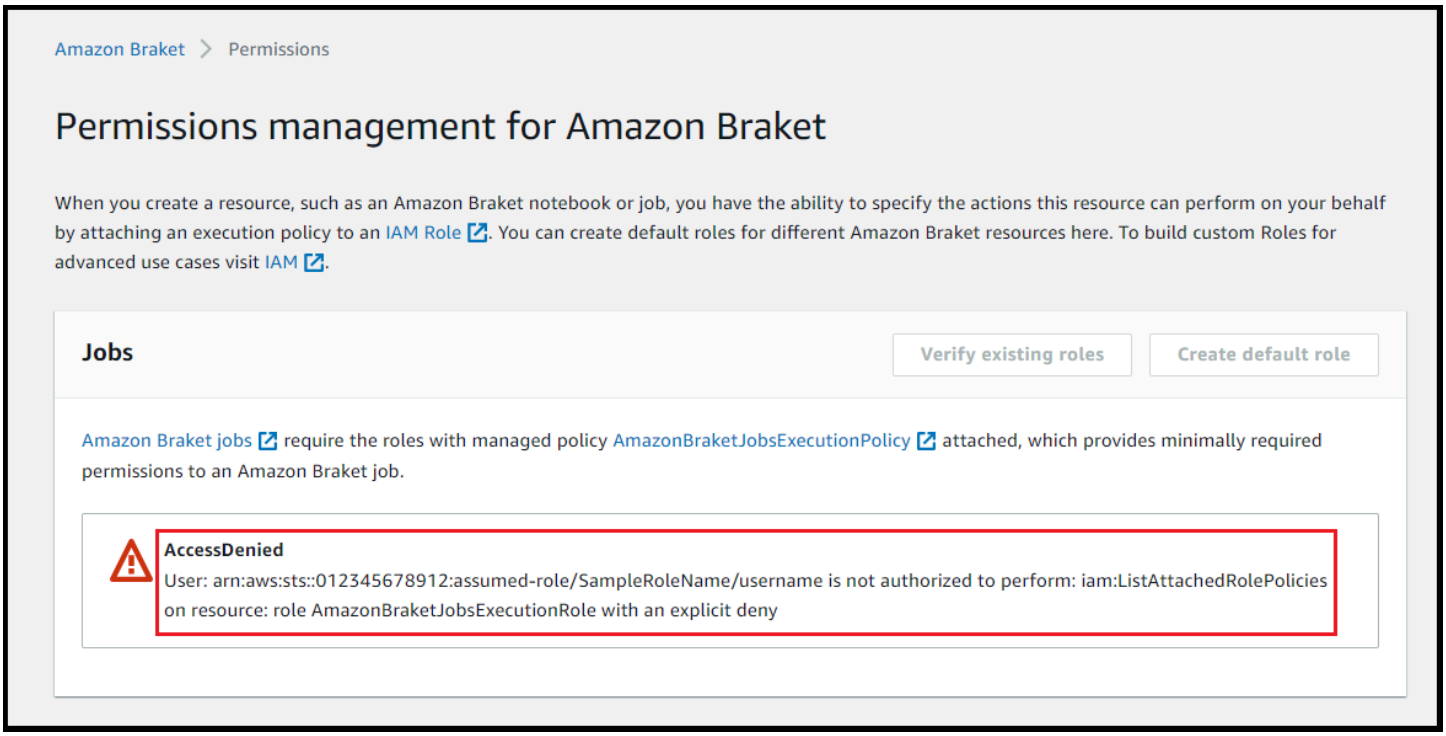
Si no tiene un rol con los permisos suficientes para ejecutar un trabajo híbrido, recibirá un mensaje en el que se indica que no se ha encontrado dicho rol. Seleccione el botón Crear rol predeterminado para obtener un rol con permisos suficientes.



Si el rol se creó correctamente, recibirá un mensaje que lo confirma.



Si no tiene permisos para realizar esta consulta, se le denegará el acceso. En este caso, póngase en contacto con su AWS administrador interno.



Creación de un trabajo híbrido

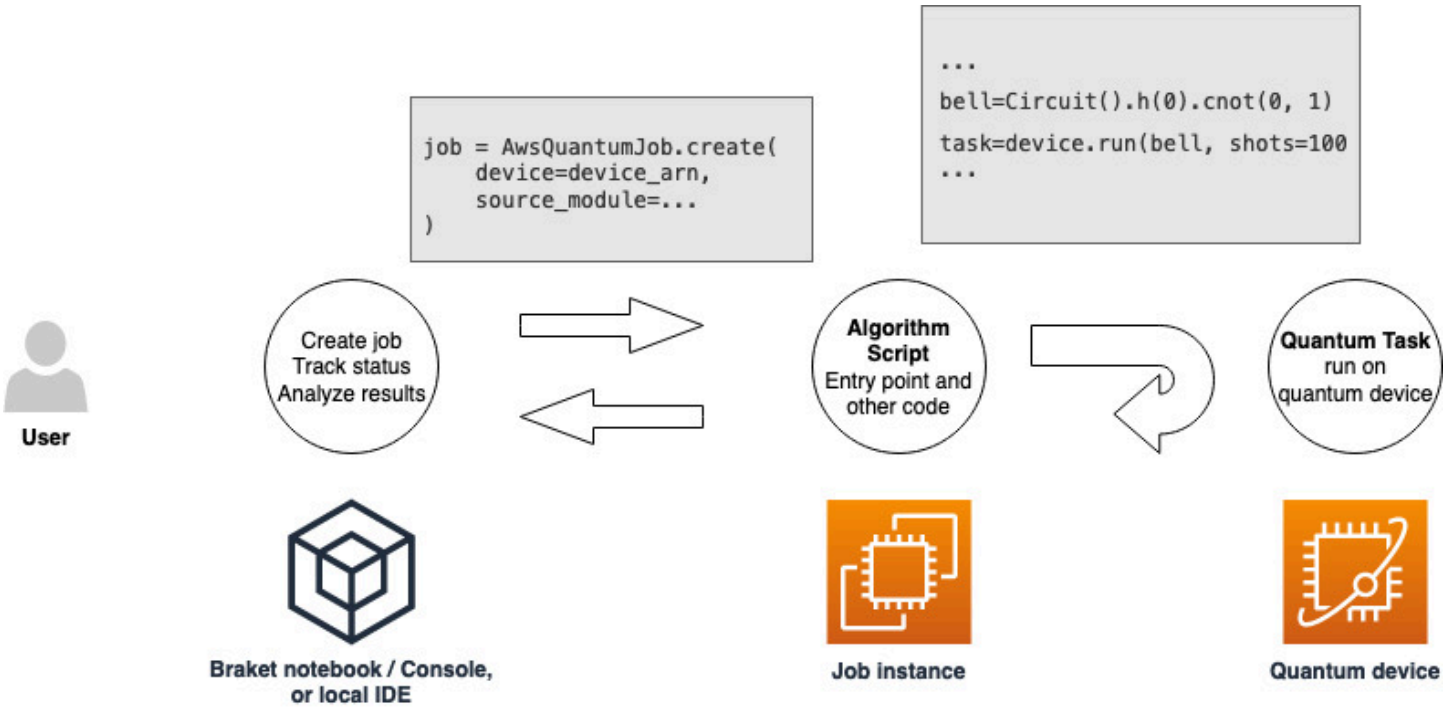
En esta sección se muestra cómo crear un trabajo híbrido usando un script de Python. Como alternativa, para crear un trabajo híbrido a partir de código Python local, como su entorno de desarrollo integrado (IDE) preferido o un cuaderno de Braket, consulte Ejecución del código local como un trabajo híbrido.

En esta sección:

- Creación y ejecución
- Supervisión de los resultados
- Guardar los resultados
- Uso de puntos de comprobación
- Ejecución del código local como un trabajo híbrido
- Uso de la API con trabajos híbridos
- Creación y depuración de un trabajo híbrido con modo local

Creación y ejecución

Una vez que tenga un rol con permisos para ejecutar un trabajo híbrido, estará listo para proceder. La pieza clave de su primer trabajo híbrido de Braket es el script de algoritmo. Define el algoritmo que desea ejecutar y contiene las tareas lógicas clásicas y cuánticas que forman parte de su algoritmo. Además del script de algoritmo, puede proporcionar otros archivos de dependencia. El script de algoritmo, junto con sus dependencias, se denomina módulo fuente. El punto de entrada define el primer archivo o función que se ejecutará en el módulo fuente cuando se inicie el trabajo híbrido.



En primer lugar, consideremos el siguiente ejemplo básico de un script de algoritmo que crea cinco estados Bell e imprime los resultados de medición correspondientes.

```
import os

from braket.aws import AwsDevice
from braket.circuits import Circuit


def start_here():

    print("Test job started!")

    # Use the device declared in the job script
```

```
    device = AwsDevice(os.environ["AMZN_BRAKET_DEVICE_ARN"])

    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)

    print("Test job completed!")
```

Guarde este archivo con el nombre algorithm_script.py en el directorio de trabajo actual de su cuaderno de Braket o en su entorno local. El fichero algorithm_script.py tiene `start_here()` como punto de entrada previsto.

A continuación, cree un archivo de Python o un cuaderno de Python en el mismo directorio que el archivo algorithm_script.py. Este script inicia el trabajo híbrido y gestiona cualquier procesamiento asíncrono, como imprimir el estado o los resultados clave que nos interesen. Como mínimo, este script debe especificar su script de trabajo híbrido y su dispositivo principal.

> **Note**
>
> Para obtener más información sobre cómo crear un cuaderno de Braket o cargar un archivo, como el archivo algorithm_script.py, en el mismo directorio que los cuadernos, consulte Ejecución de su primer circuito utilizando el SDK de Python de Amazon Braket.

Para este primer caso básico, se utiliza un simulador de destino. Independientemente del tipo de dispositivo cuántico de destino, ya sea un simulador o una unidad de procesamiento cuántico (QPU) real, el dispositivo que especifique con `device` en el siguiente script se utilizará para programar el trabajo híbrido y estará disponible para los scripts del algoritmo como variable de entorno `AMZN_BRAKET_DEVICE_ARN`.

> **Note**
>
> Solo puede usar los dispositivos que estén disponibles en Región de AWS su trabajo híbrido. El SDK de Amazon Braket selecciona automáticamente esta Región de AWS. Por ejemplo, un trabajo híbrido en us-east-1 puede utilizar dispositivos IonQ, SV1, DM1 y TN1, pero no dispositivos Rigetti.

Si elige una computadora cuántica en lugar de un simulador, Braket programa sus trabajos híbridos para ejecutar todas sus tareas cuánticas con acceso prioritario.

```
from braket.aws import AwsQuantumJob
from braket.devices import Devices

job = AwsQuantumJob.create(
    Devices.Amazon.SV1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=True
)
```

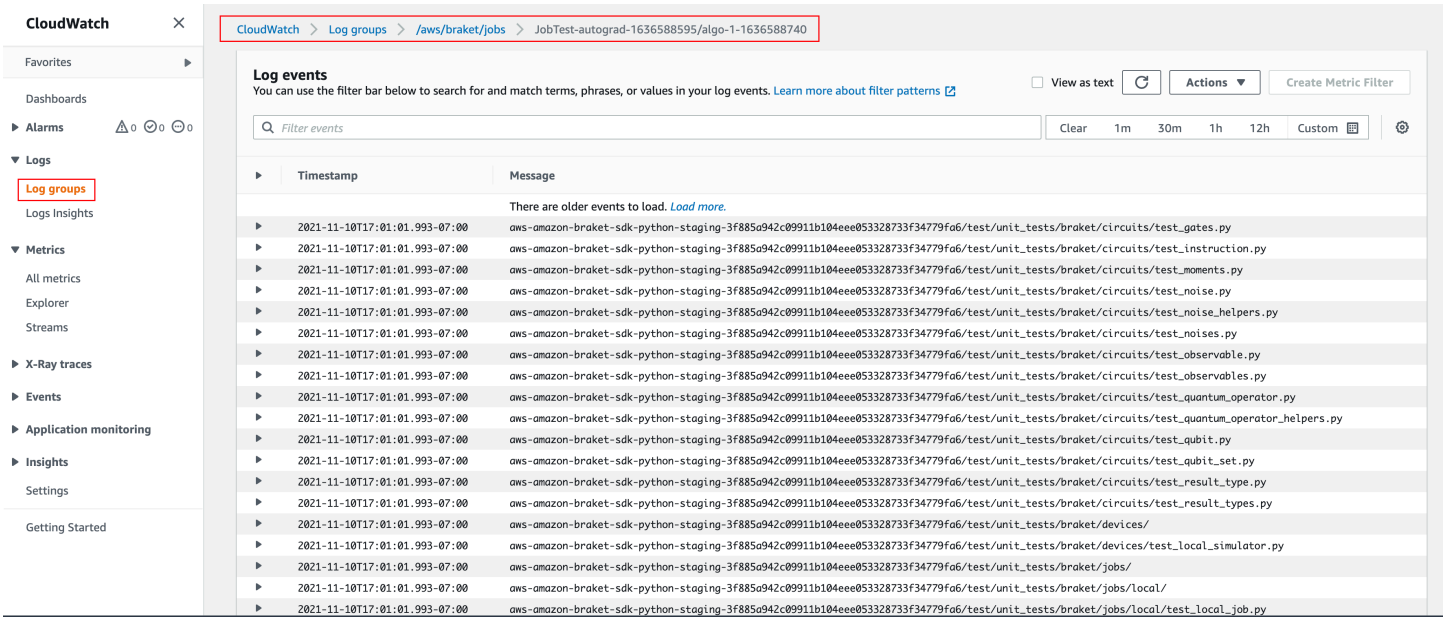
El parámetro `wait_until_complete=True` establece un modo detallado para que su trabajo imprima la salida del trabajo real mientras se está ejecutando. Debería ver un resultado similar al del siguiente ejemplo.

```
Initializing Braket Job: arn:aws:braket:us-west-2:111122223333:job/braket-job-
default-123456789012
Job queue position: 1
Job queue position: 1
Job queue position: 1
..............
.
.
.
Beginning Setup
Checking for Additional Requirements
Additional Requirements Check Finished
Running Code As Process
Test job started!
Counter({'00': 58, '11': 42})
Counter({'00': 55, '11': 45})
Counter({'11': 51, '00': 49})
Counter({'00': 56, '11': 44})
Counter({'11': 56, '00': 44})
Test job completed!
Code Run Finished
2025-09-24 23:13:40,962 sagemaker-training-toolkit INFO     Reporting training SUCCESS
```

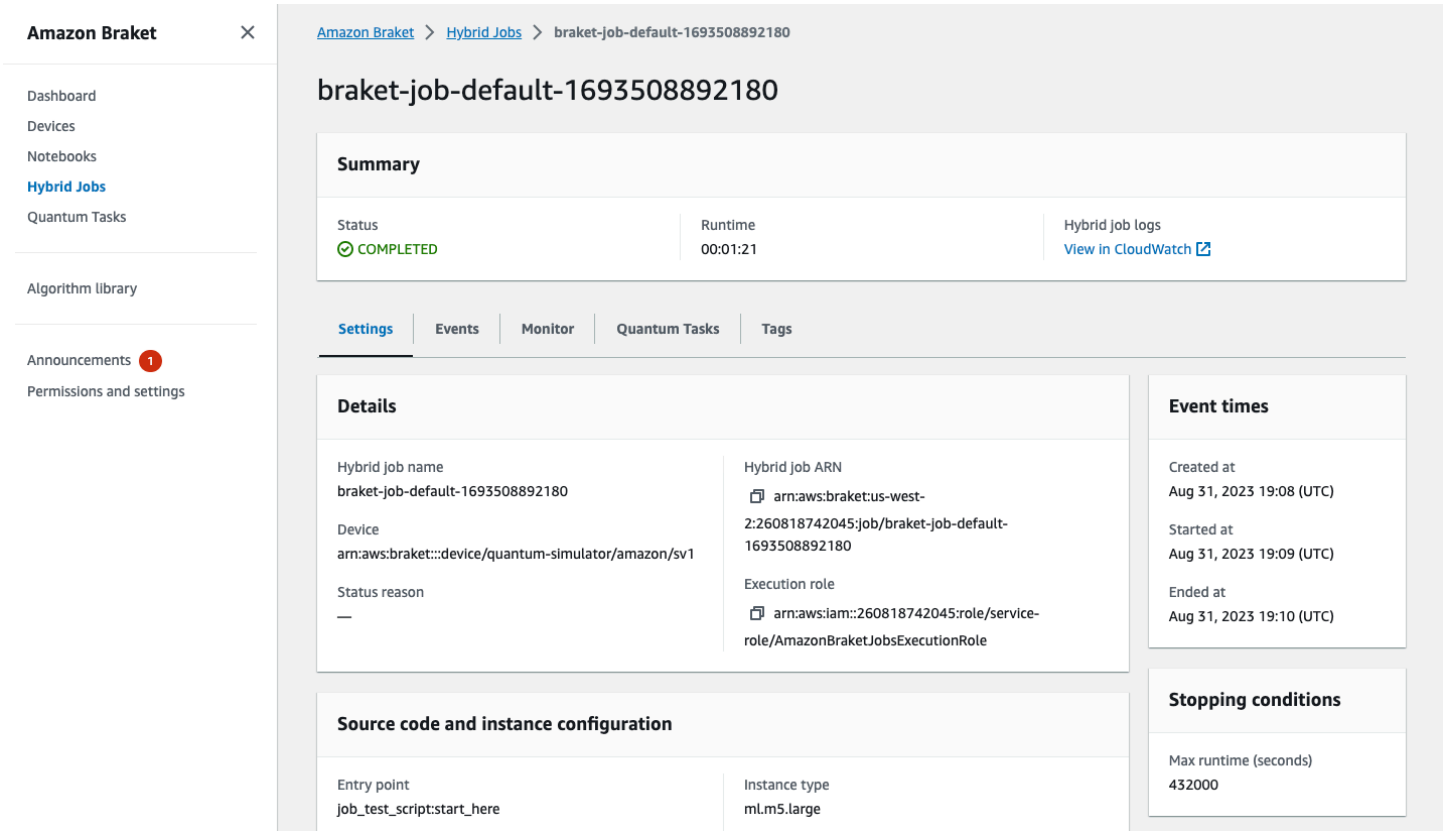
> **Note**
>
> También puedes usar tu módulo personalizado con el método AwsQuantumJob.create pasando su ubicación (ya sea la ruta a un directorio o archivo local, o el URI de S3 de un archivo tar.gz). Para ver un ejemplo de trabajo, consulte el archivo Parallelize_training_for_QML.ipynb en la carpeta de trabajos híbridos en el repositorio de GitHub de ejemplos de Amazon Braket.

Supervisión de los resultados

Como alternativa, puedes acceder a la salida del registro desde Amazon CloudWatch. Para ello, vaya a la pestaña Grupos de registro en el menú izquierdo de la página de detalles del trabajo, seleccione el grupo de registro `aws/braket/jobs` y, a continuación, elija el flujo de registro que contiene el nombre del trabajo. En el ejemplo anterior, es `braket-job-default-1631915042705/algo-1-1631915190`.



También puede ver el estado del trabajo híbrido en la consola seleccionando la página Trabajos híbridos y, a continuación, Configuración.



Su trabajo híbrido produce algunos artefactos en Amazon S3 mientras se ejecuta. El nombre predeterminado del bucket de S3 es `amazon-braket-<region>-<accountid>` y el contenido está en el `jobs/<jobname>/<timestamp>` directorio. Puede configurar las ubicaciones de S3 en las que se almacenan estos artefactos especificando una `code_location` diferente al crear el trabajo híbrido con el SDK de Python de Braket.

> **Note**
>
> Este depósito de S3 debe estar ubicado en el mismo lugar que su script de Región de AWS trabajo.

El directorio `jobs/<jobname>/<timestamp>` contiene una subcarpeta con la salida del script del punto de entrada en un archivo `model.tar.gz`. También hay un directorio denominado `script` que contiene los artefactos del script de algoritmo en un archivo `source.tar.gz`. Los resultados de sus tareas cuánticas reales se encuentran en el directorio denominado `jobs/<jobname>/tasks`.

Guardar los resultados

Puede guardar los resultados generados por el script de algoritmo para que estén disponibles en el objeto de trabajo híbrido del script de trabajo híbrido, así como en la carpeta de salida de Amazon S3 (en un archivo comprimido con tar denominado model.tar.gz).

El resultado debe guardarse en un archivo con un formato de notación de JavaScript objetos (JSON). Si los datos no se pueden serializar fácilmente en texto, como en el caso de una matriz numpy, puede pasar una opción para serializar utilizando un formato de datos pickled. Consulte el módulo braket.jobs.data_persistence para obtener más información.

Para guardar los resultados de los trabajos híbridos, añada las siguientes líneas comentadas con #ADD al archivo algorithm_script.py.

```
import os

from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.jobs import save_job_result  # ADD


def start_here():

    print("Test job started!")

    device = AwsDevice(os.environ['AMZN_BRAKET_DEVICE_ARN'])

    results = []  # ADD

    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)
        results.append(task.result().measurement_counts)  # ADD

        save_job_result({"measurement_counts": results})  # ADD

    print("Test job completed!")
```

A continuación, puede mostrar los resultados del trabajo desde su script de trabajo añadiendo la línea **`print(job.result())`** comentada con #ADD.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
)

print(job.arn)
while job.state() not in AwsQuantumJob.TERMINAL_STATES:
    print(job.state())
    time.sleep(10)

print(job.state())
print(job.result())   # ADD
```

En este ejemplo, hemos eliminado `wait_until_complete=True` para suprimir la salida detallada. Puedes volver a añadirlo para su depuración. Cuando ejecuta este trabajo híbrido, se muestra el identificador y el `job-arn`, seguido del estado del trabajo híbrido cada 10 segundos hasta que el trabajo híbrido esté `COMPLETED`, después de lo cual le muestra los resultados del circuito Bell. Consulte el siguiente ejemplo.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-123456789012
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Uso de puntos de comprobación

Puede guardar las iteraciones intermedias de sus trabajos híbridos mediante puntos de control. En el ejemplo de script de algoritmo de la sección anterior, añadiría las siguientes líneas comentadas con #ADD para crear archivos de punto de control.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.jobs import save_job_checkpoint  # ADD
import os


def start_here():

    print("Test job starts!")

    device = AwsDevice(os.environ["AMZN_BRAKET_DEVICE_ARN"])

    # ADD the following code
    job_name = os.environ["AMZN_BRAKET_JOB_NAME"]
    save_job_checkpoint(checkpoint_data={"data": f"data for checkpoint from
 {job_name}"}, checkpoint_file_suffix="checkpoint-1")  # End of ADD

    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)

    print("Test hybrid job completed!")
```

Cuando ejecuta el trabajo híbrido, se crea el archivo <jobname>-checkpoint-1.json en sus artefactos de trabajo híbrido en el directorio de puntos de control con una ruta de `/opt/jobs/checkpoints` predeterminada. El script del trabajo híbrido permanece inalterado a menos que desee cambiar esta ruta predeterminada.

Si desea cargar un trabajo híbrido desde un punto de control generado por un trabajo híbrido anterior, el script de algoritmo utiliza `from braket.jobs import load_job_checkpoint`. La lógica que se debe cargar en el script de algoritmo es la siguiente.

```
from braket.jobs import load_job_checkpoint
```

```
checkpoint_1 = load_job_checkpoint(
    "previous_job_name",
    checkpoint_file_suffix="checkpoint-1",
)
```

Después de cargar este punto de control, puede continuar con su lógica basándose en el contenido cargado en el `checkpoint-1`.

> **Note**
>
> El checkpoint_file_suffix debe coincidir con el sufijo especificado previamente al crear el punto de control.

Su script de orquestación debe especificar el `job-arn` del trabajo híbrido anterior con la línea comentada con #ADD.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    copy_checkpoints_from_job="<previous-job-ARN>", #ADD
    )
```

Ejecución del código local como un trabajo híbrido

Los trabajos híbridos de Amazon Braket proporcionan una orquestación totalmente administrada de algoritmos híbridos cuánticos-clásicos, combinando los recursos de computación de Amazon EC2 con el acceso a la unidad de procesamiento cuántico (QPU) de Amazon Braket. Las tareas cuánticas creadas en un trabajo híbrido tienen prioridad en la cola sobre las tareas cuánticas individuales, de modo que sus algoritmos no se verán interrumpidos por las fluctuaciones en la cola de tareas cuánticas. Cada QPU mantiene una cola de trabajos híbridos independiente, lo que garantiza que solo se pueda ejecutar un trabajo híbrido en un momento dado.

En esta sección:

- Creación de un trabajo híbrido a partir de código de Python local
- Instalación de paquetes y código fuente de Python adicionales

- Guardar y cargar datos en una instancia de trabajo híbrido
- Prácticas recomendadas para decoradores de trabajos híbridos

Creación de un trabajo híbrido a partir de código de Python local

Puede ejecutar su código de Python local como un trabajo híbrido de Amazon Braket. Puede hacerlo anotando su código con un decorador `@hybrid_job`, como se muestra en el siguiente ejemplo de código. Para los entornos personalizados, puede optar por utilizar un contenedor personalizado de Amazon Elastic Container Registry (ECR).

> **Note**
>
> De forma predeterminada, solo se admite Python 3.12.

Puede usar el decorador `@hybrid_job` para anotar una función. Braket transforma el código dentro del decorador en un script de algoritmo de trabajo híbrido de Braket. A continuación, el trabajo híbrido invoca la función dentro del decorador en una instancia de Amazon EC2. Puede supervisar el progreso del trabajo con `job.state()` o con la consola de Braket. El siguiente ejemplo de código muestra cómo ejecutar una secuencia de cinco estados en el State Vector Simulator (SV1) device.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit, FreeParameter, Observable
from braket.devices import Devices
from braket.jobs.hybrid_job import hybrid_job
from braket.jobs.metrics import log_metric

device_arn = Devices.Amazon.SV1


@hybrid_job(device=device_arn)  # Choose priority device
def run_hybrid_job(num_tasks=1):
    device = AwsDevice(device_arn)  # Declare AwsDevice within the hybrid job

    # Create a parametric circuit
    circ = Circuit()
    circ.rx(0, FreeParameter("theta"))
    circ.cnot(0, 1)
    circ.expectation(observable=Observable.X(), target=0)
```

```
    theta = 0.0  # Initial parameter

    for i in range(num_tasks):
        task = device.run(circ, shots=100, inputs={"theta": theta})  # Input parameters
        exp_val = task.result().values[0]

        theta += exp_val  # Modify the parameter (possibly gradient descent)

        log_metric(metric_name="exp_val", value=exp_val, iteration_number=i)

    return {"final_theta": theta, "final_exp_val": exp_val}
```

El trabajo híbrido se crea invocando la función como lo haría con las funciones normales de Python. Sin embargo, la función de decorador devuelve el identificador del trabajo híbrido en lugar del resultado de la función. Para recuperar los resultados una vez que se haya completado, utilice `job.result()`.

```
job = run_hybrid_job(num_tasks=1)
result = job.result()
```

El argumento del dispositivo en el decorador `@hybrid_job` especifica el dispositivo al que el trabajo híbrido tiene acceso prioritario, en este caso, el simulador SV1. Para obtener la prioridad de la QPU, debe asegurarse de que el ARN del dispositivo utilizado en la función coincida con el especificado en el decorador. Para mayor comodidad, puede utilizar la función auxiliar `get_job_device_arn()` para capturar el ARN del dispositivo declarado en `@hybrid_job`.

> **Note**
>
> Cada trabajo híbrido tiene un tiempo de inicio de al menos un minuto, ya que crea un entorno en contenedor en Amazon EC2. Por lo tanto, para cargas de trabajo muy cortas, como un circuito único o un lote de circuitos, puede bastar con utilizar tareas cuánticas.

Hiperparámetros

La función `run_hybrid_job()` utiliza el argumento `num_tasks` para controlar el número de tareas cuánticas creadas. El trabajo híbrido captura esto automáticamente como un hiperparámetro.

> **Note**
>
> Los hiperparámetros se muestran en la consola de Braket como cadenas, que tienen un límite de 2500 caracteres.

Métricas y registro

En la función `run_hybrid_job()`, las métricas de los algoritmos iterativos se registran con `log_metrics`. Las métricas se representan automáticamente en la página de la consola de Braket, en la pestaña de trabajos híbridos. Puede utilizar las métricas para realizar un seguimiento de los costos de las tareas cuánticas prácticamente en tiempo real durante la ejecución de un trabajo híbrido con el Rastreador de costos de Braket. En el ejemplo anterior, se utiliza el nombre de métrica «probabilidad», que registra la primera probabilidad del tipo de resultado.

Recuperación de resultados

Una vez finalizado el trabajo híbrido, se utiliza `job.result()` para recuperar los resultados del trabajo híbrido. Braket captura automáticamente todos los objetos de la declaración de devolución. Tenga en cuenta que los objetos devueltos por la función deben ser una tupla y cada elemento debe ser serializable. Por ejemplo, el código siguiente muestra un ejemplo de funcionamiento correcto y otro incorrecto.

```
import numpy as np


# Working example
@hybrid_job(device=Devices.Amazon.SV1)
def passing():
    np_array = np.random.rand(5)
    return np_array  # Serializable

# # Failing example
# @hybrid_job(device=Devices.Amazon.SV1)
# def failing():
#     return MyObject() # Not serializable
```

Nombre del trabajo

De forma predeterminada, el nombre de este trabajo híbrido se deduce del nombre de la función. Puede especificar un nombre personalizado que tenga un máximo de 50 caracteres. Por ejemplo, en el código siguiente, el nombre del trabajo es "my-job-name».

```
@hybrid_job(device=Devices.Amazon.SV1, job_name="my-job-name")
def function():
    pass
```

Modo local

Los trabajos locales se crean añadiendo el argumento `local=True` al decorador. Esto ejecuta el trabajo híbrido en un entorno en contenedor en su entorno de computación local, como un portátil. Los trabajos locales no tienen prioridad en las colas para las tareas cuánticas. En casos avanzados, como los de varios nodos o MPI, los trabajos locales pueden tener acceso a las variables de entorno de Braket requeridas. El código siguiente crea un trabajo híbrido local con el dispositivo como SV1 simulador.

```
@hybrid_job(device=Devices.Amazon.SV1, local=True)
def run_hybrid_job(num_tasks=1):
    return ...
```

Todas las demás opciones de trabajo híbrido son compatibles. Para ver una lista de opciones, consulte el módulo braket.jobs.quantum_job_creation.

Instalación de paquetes y código fuente de Python adicionales

Puede personalizar su entorno de tiempo de ejecución para usar sus paquetes de Python preferidos. Puede usar un archivo `requirements.txt`, una lista de nombres de paquetes o utilizar su propio contenedor (BYOC). Por ejemplo, el archivo `requirements.txt` puede incluir otros paquetes para instalar.

```
qiskit
pennylane >= 0.31
mitiq == 0.29
```

Para personalizar un entorno de tiempo de ejecución utilizando un archivo `requirements.txt`, consulte el siguiente ejemplo de código.

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies="requirements.txt")
```

```
def run_hybrid_job(num_tasks=1):
    return ...
```

Como alternativa, puede proporcionar los nombres de los paquetes como una lista de Python de la siguiente manera.

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies=["qiskit", "pennylane>=0.31",
 "mitiq==0.29"])
def run_hybrid_job(num_tasks=1):
    return ...
```

El código fuente adicional se puede especificar como una lista de módulos o como un solo módulo, como en el siguiente ejemplo de código.

```
@hybrid_job(device=Devices.Amazon.SV1, include_modules=["my_module1", "my_module2"])
def run_hybrid_job(num_tasks=1):
    return ...
```

Guardar y cargar datos en una instancia de trabajo híbrido

Especificación de los datos de entrenamiento de entrada

Al crear un trabajo híbrido, puede proporcionar conjuntos de datos de entrenamiento de entrada especificando un bucket de Amazon Simple Storage Service (Amazon S3). También puede especificar una ruta local y, a continuación, Braket carga automáticamente los datos en Amazon S3 en `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>`. Si especifica una ruta local, el nombre del canal predeterminado será «entrada». El siguiente código muestra un archivo numpy de la ruta local `data/file.npy`.

```
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1, input_data="data/file.npy")
def run_hybrid_job(num_tasks=1):
    data = np.load("data/file.npy")
    return ...
```

Para S3, debe usar la función auxiliar `get_input_data_dir()`.

```
import numpy as np
```

```
from braket.jobs import get_input_data_dir

s3_path = "s3://amazon-braket-us-east-1-123456789012/job-data/file.npy"


@hybrid_job(device=None, input_data=s3_path)
def job_s3_input():
    np.load(get_input_data_dir() + "/file.npy")


@hybrid_job(device=None, input_data={"channel": s3_path})
def job_s3_input_channel():
    np.load(get_input_data_dir("channel") + "/file.npy")
```

Puede especificar varias fuentes de datos de entrada proporcionando un diccionario de valores de canal y rutas S3 URIs o locales.

```
import numpy as np
from braket.jobs import get_input_data_dir

input_data = {
    "input": "data/file.npy",
    "input_2": "s3://amzn-s3-demo-bucket/data.json"
}


@hybrid_job(device=None, input_data=input_data)
def multiple_input_job():
    np.load(get_input_data_dir("input") + "/file.npy")
    np.load(get_input_data_dir("input_2") + "/data.json")
```

> **Note**
>
> Si los datos de entrada son grandes (>1 GB), habrá un largo tiempo de espera antes de que se cree el trabajo. Esto se debe a que los datos de entrada locales se cargan por primera vez en un bucket de S3 y, a continuación, se añade la ruta de S3 a la solicitud de trabajo. Por último, la solicitud de trabajo se envía al servicio de Braket.

Guardar los resultados en S3

Para guardar los resultados no incluidos en la declaración de devolución de la función decorada, debe añadir el directorio correcto a todas las operaciones de escritura de archivos. En el siguiente ejemplo, se muestra cómo guardar una matriz numpy y una figura de matplotlib.

```
import matplotlib.pyplot as plt
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1)
def run_hybrid_job(num_tasks=1):
    result = np.random.rand(5)

    # Save a numpy array
    np.save("result.npy", result)

    # Save a matplotlib figure
    plt.plot(result)
    plt.savefig("fig.png")
    return ...
```

Todos los resultados se comprimen en un archivo denominado `model.tar.gz`. Puede descargar los resultados con la función `job.result()` de Python o yendo a la carpeta de resultados desde la página de trabajos híbridos de la consola de administración de Braket.

Guardar y reanudar desde los puntos de control

Para trabajos híbridos de larga duración, se recomienda guardar periódicamente el estado intermedio del algoritmo. Puede utilizar la función auxiliar `save_job_checkpoint()` integrada o guardar los archivos en la ruta `AMZN_BRAKET_JOB_RESULTS_DIR`. Esta última opción está disponible con la función auxiliar `get_job_results_dir()`.

El siguiente es un ejemplo práctico mínimo para guardar y cargar puntos de control con un decorador de trabajo híbrido:

```
from braket.jobs import save_job_checkpoint, load_job_checkpoint, hybrid_job


@hybrid_job(device=None, wait_until_complete=True)
def function():
    save_job_checkpoint({"a": 1})
```

```
job = function()
job_name = job.name
job_arn = job.arn


@hybrid_job(device=None, wait_until_complete=True, copy_checkpoints_from_job=job_arn)
def continued_function():
    load_job_checkpoint(job_name)


continued_job = continued_function()
```

En el primer trabajo híbrido, `save_job_checkpoint()` se llama con un diccionario que contiene los datos que queremos guardar. De forma predeterminada, todos los valores deben poder serializarse como texto. Para comprobar objetos de Python más complejos, como matrices numpy, puede configurar `data_format = PersistedJobDataFormat.PICKLED_V4`. Este código crea y sobrescribe un archivo de puntos de control con el nombre predeterminado `<jobname>.json` en sus artefactos de trabajo híbrido, en una subcarpeta denominada «puntos de control».

Para crear un nuevo trabajo híbrido para continuar desde el punto de control, debemos pasar `copy_checkpoints_from_job=job_arn` dónde `job_arn` es el ARN del trabajo híbrido del trabajo anterior. A continuación, usamos `load_job_checkpoint(job_name)` para realizar la carga desde el punto de control.

Prácticas recomendadas para decoradores de trabajos híbridos

Uso de la asincronicidad

Los trabajos híbridos creados con la anotación de decorador son asíncronos: se ejecutan una vez que los recursos clásicos y cuánticos están disponibles. Usted monitorea el progreso del algoritmo utilizando Amazon Braket Management Console o Amazon CloudWatch. Cuando envía el algoritmo para su ejecución, Braket lo ejecuta en un entorno en contenedor escalable y los resultados se recuperan cuando se completa el algoritmo.

Ejecución de algoritmos variacionales iterativos

Los trabajos híbridos le proporcionan las herramientas necesarias para ejecutar algoritmos cuánticos-clásicos iterativos. Para problemas puramente cuánticos, utilice tareas cuánticas o un lote de tareas cuánticas. El acceso prioritario a ciertos QPUs es más beneficioso para los algoritmos variacionales de larga duración que requieren múltiples llamadas iterativas QPUs con el procesamiento clásico en el medio.

Depuración utilizando el modo local

Antes de ejecutar un trabajo híbrido en una QPU, se recomienda ejecutarlo primero en el simulador para confirmar SV1 que se ejecuta según lo esperado. En el caso de pruebas a pequeña escala, puede ejecutarlas con el modo local para una iteración y depuración rápidas.

Mejora de la reproducibilidad utilizando su propio contenedor (BYOC)

Cree un experimento reproducible encapsulando su software y sus dependencias en un entorno en contenedor. Al empaquetar todo su código, dependencias y configuración en un contenedor, evitará posibles conflictos y problemas de control de versiones.

Simuladores distribuidos multiinstancia

Para ejecutar una gran cantidad de circuitos, considere la posibilidad de utilizar el soporte de MPI integrado para ejecutar simuladores locales en varias instancias en un solo trabajo híbrido. Para obtener más información, consulte simuladores integrados.

Uso de circuitos paramétricos

Los circuitos paramétricos que se envían a partir de un trabajo híbrido se compilan automáticamente en algunos casos QPUs mediante la compilación paramétrica para mejorar los tiempos de ejecución de los algoritmos.

Punto de control periódico

Para trabajos híbridos de larga duración, se recomienda guardar periódicamente el estado intermedio del algoritmo.

Para ver más ejemplos, casos de uso y mejores prácticas, consulta los ejemplos de Amazon GitHub Braket.

Uso de la API con trabajos híbridos

Puede acceder e interactuar con los trabajos híbridos de Amazon Braket directamente utilizando la API. Sin embargo, los métodos predeterminados y prácticos no están disponibles cuando se utiliza la API directamente.

> **Note**
>
> Le recomendamos encarecidamente que interactúe con los trabajos híbridos de Amazon Braket mediante el SDK de Python de Amazon Braket. Ofrece prácticos valores

predeterminados y protecciones que ayudan a que sus trabajos híbridos se ejecuten correctamente.

En este tema se describen los aspectos básicos del uso de la API. Si decide utilizar la API, tenga en cuenta que este enfoque puede ser más complejo, por lo que debe prepararse para varias iteraciones hasta que su trabajo híbrido se ejecute.

Para usar la API, tu cuenta debe tener un rol con la política administrada `AmazonBraketFullAccess`.

> **Note**
>
> Para obtener más información sobre cómo obtener un rol con la política administrada `AmazonBraketFullAccess`, consulte la página Activación de Amazon Braket.

Además, necesita un rol de ejecución. Esta función se transferirá al servicio. Puede crear el rol utilizando la consola de Amazon Braket. Utilice la pestaña Roles de ejecución en la página Permisos y configuración para crear un rol predeterminado para los trabajos híbridos.

La API de `CreateJob` requiere que especifique todos los parámetros necesarios para el trabajo híbrido. Para utilizar Python, comprima los archivos del script de algoritmo en un paquete tar, como un archivo input.tar.gz, y ejecute el siguiente script. Actualice las partes del código entre corchetes angulares (<>) para que coincidan con la información de su cuenta y el punto de entrada que especifican la ruta, el archivo y el método en los que comienza su trabajo híbrido.

```
from braket.aws import AwsDevice, AwsSession
import boto3
from datetime import datetime

s3_client = boto3.client("s3")
client = boto3.client("braket")

project_name = "job-test"
job_name = project_name + "-" + datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")
bucket = "amazon-braket-<your_bucket>"
s3_prefix = job_name

job_script = "input.tar.gz"
```

```
job_object = f"{s3_prefix}/script/{job_script}"
s3_client.upload_file(job_script, bucket, job_object)

input_data = "inputdata.csv"
input_object = f"{s3_prefix}/input/{input_data}"
s3_client.upload_file(input_data, bucket, input_object)

job = client.create_job(
    jobName=job_name,
    roleArn="arn:aws:iam::<your_account>:role/service-role/
AmazonBraketJobsExecutionRole",  # https://docs.aws.amazon.com/braket/latest/
developerguide/braket-manage-access.html#about-amazonbraketjobsexecution
    algorithmSpecification={
        "scriptModeConfig": {
            "entryPoint": "<your_execution_module>:<your_execution_method>",
            "containerImage": {"uri": "292282985366.dkr.ecr.us-west-1.amazonaws.com/
amazon-braket-base-jobs:1.0-cpu-py37-ubuntu18.04"},   # Change to the specific region
 you are using
            "s3Uri": f"s3://{bucket}/{job_object}",
            "compressionType": "GZIP"
        }
    },
    inputDataConfig=[
        {
            "channelName": "hellothere",
            "compressionType": "NONE",
            "dataSource": {
                "s3DataSource": {
                    "s3Uri": f"s3://{bucket}/{s3_prefix}/input",
                    "s3DataType": "S3_PREFIX"
                }
            }
        }
    ],
    outputDataConfig={
        "s3Path": f"s3://{bucket}/{s3_prefix}/output"
    },
    instanceConfig={
        "instanceType": "ml.m5.large",
        "instanceCount": 1,
        "volumeSizeInGb": 1
    },
    checkpointConfig={
        "s3Uri":  f"s3://{bucket}/{s3_prefix}/checkpoints",
```

```
        "localPath": "/opt/omega/checkpoints"
    },
    deviceConfig={
        "priorityAccess": {
            "devices": [
                "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3"
            ]
        }
    },
    hyperParameters={
        "hyperparameter key you wish to pass": "<hyperparameter value you wish to
 pass>",
    },
    stoppingCondition={
        "maxRuntimeInSeconds": 1200,
        "maximumTaskLimit": 10
    },
)
```

Una vez que haya creado su trabajo híbrido, podrá acceder a los detalles de este a través de la API de `GetJob` o de la consola. Para obtener los detalles del trabajo híbrido de la sesión de Python en la que ejecutó el código `createJob`, como en el ejemplo anterior, utilice el siguiente comando de Python.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

Para cancelar un trabajo híbrido, llame a la API de `CancelJob` con el Amazon Resource Name del trabajo ('JobArn').

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

Puede especificar puntos de control como parte de la API de `createJob` usando el parámetro `checkpointConfig`.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> La localPath de `checkpointConfig` no puede comenzar con ninguna de las siguientes rutas reservadas: `/opt/ml`, `/opt/braket`, `/tmp` ni `/usr/local/nvidia`.

Creación y depuración de un trabajo híbrido con modo local

Cuando crea un nuevo algoritmo híbrido, el modo local le ayuda a depurar y probar el script de algoritmo. El modo local es una característica que le permite ejecutar código que tiene previsto utilizar en trabajos híbridos de Amazon Braket, pero sin necesidad de que Braket gestione la infraestructura para ejecutar el trabajo híbrido. En su lugar, ejecute trabajos híbridos localmente en su instancia de cuaderno de Amazon Braket o en un cliente preferido, como una computadora portátil o de sobremesa.

En el modo local, aún puede enviar tareas cuánticas a dispositivos reales, pero no obtiene las ventajas de rendimiento que ofrece el uso de una unidad de procesamiento cuántico (QPU) real mientras se encuentra en el modo local.

Para utilizar el modo local, cambie `AwsQuantumJob` a `LocalQuantumJob` donde sea que aparezca en su programa. Por ejemplo, para ejecutar el ejemplo de Creación de su primer trabajo híbrido, edite el script del trabajo híbrido en el código de la siguiente manera.

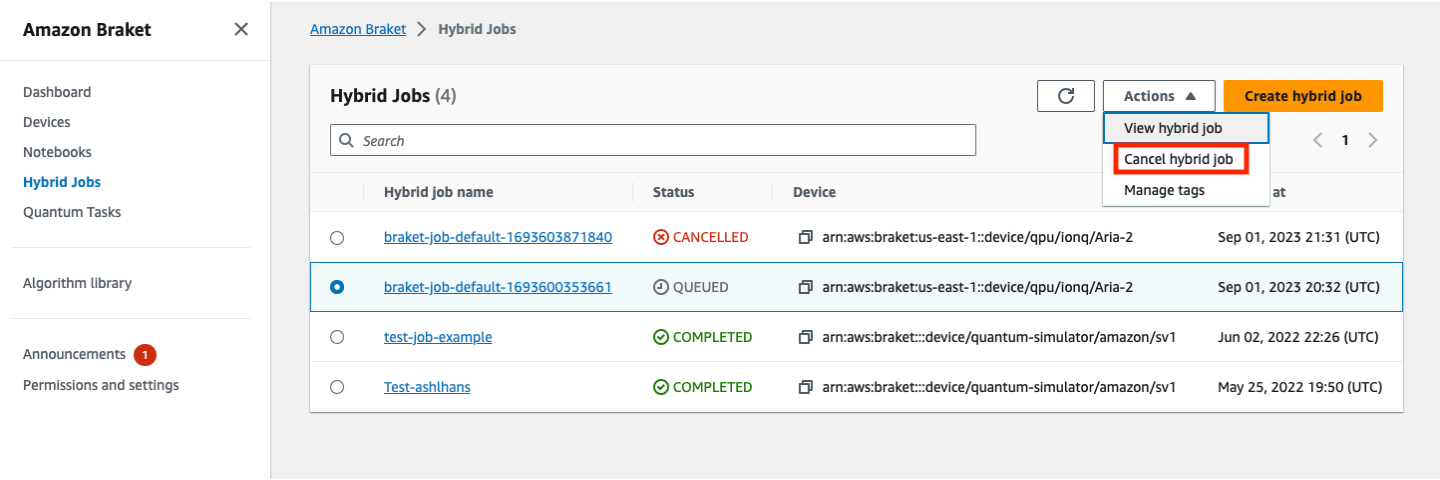
```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

> **Note**
>
> Docker, que ya viene preinstalado en los cuadernos de Amazon Braket, debe instalarse en su entorno local para poder utilizar esta característica. Las instrucciones para instalar Docker se pueden encontrar en la página Get Docker. Además, no todos los parámetros son compatibles en el modo local.

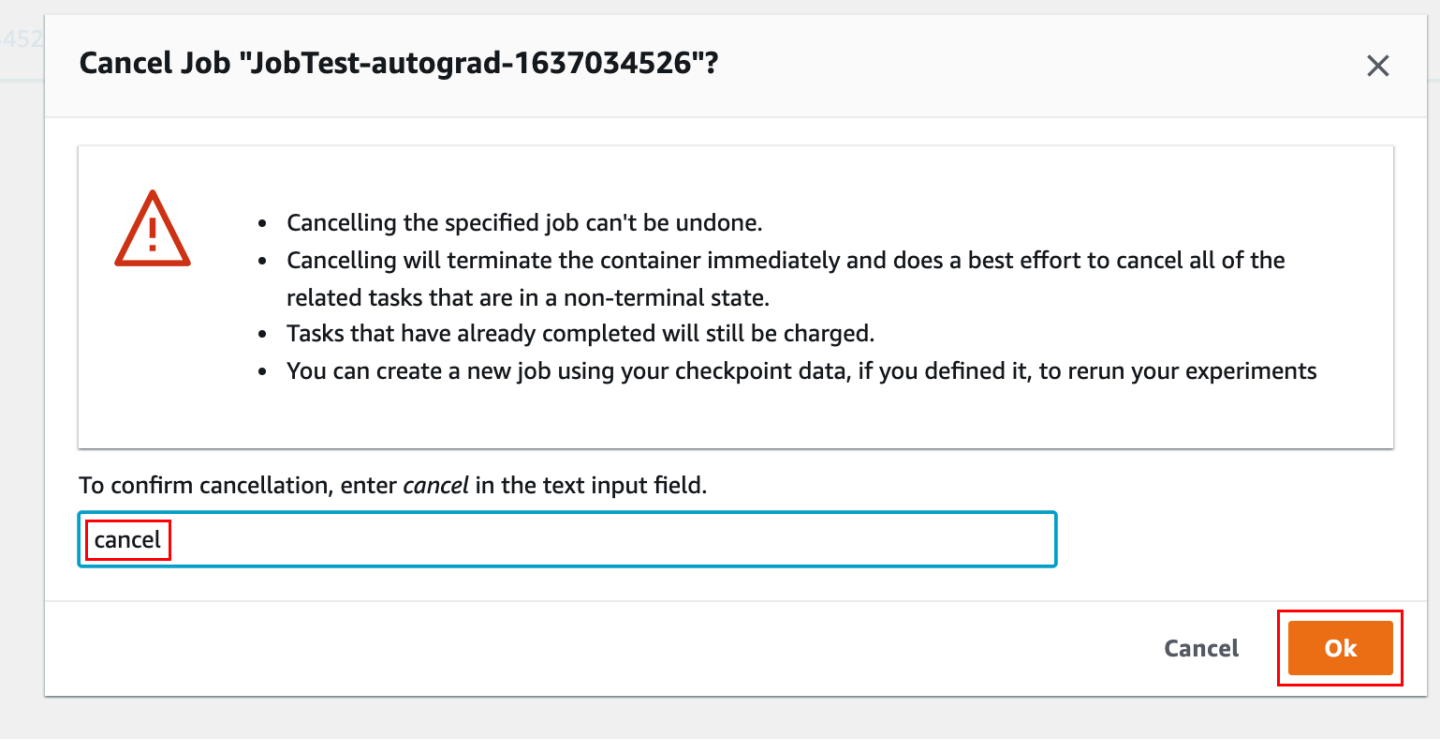
Cancelación de un trabajo híbrido

Es posible que tenga que cancelar un trabajo híbrido en un estado no terminal. Esto se puede hacer en la consola o mediante código.

Para cancelar el trabajo híbrido en la consola, seleccione el trabajo híbrido que desee cancelar en la página Trabajos híbridos y, a continuación, seleccione Cancelar trabajo híbrido en el menú desplegable Acciones.



Para confirmar la cancelación, introduzca cancelar en el campo de entrada cuando se le solicite y, a continuación, seleccione Aceptar.



Para cancelar su trabajo híbrido mediante código desde el SDK de Python de Braket, utilice `job_arn` para identificar el trabajo híbrido y, a continuación, llame al comando `cancel` que contiene, como se muestra en el código siguiente.

```
job = AwsQuantumJob(arn=job_arn)
job.cancel()
```

El comando `cancel` cierra inmediatamente el contenedor de trabajo híbrido clásico y hace todo lo posible por cancelar todas las tareas cuánticas relacionadas que aún se encuentran en un estado no terminal.

Personalización del trabajo híbrido

Amazon Braket ofrece varias formas de personalizar la forma en que se ejecutan sus trabajos híbridos, lo que le permite adaptar el entorno a sus necesidades específicas. En esta sección se exploran las opciones para personalizar los trabajos híbridos, desde definir el entorno del script de algoritmo hasta utilizar su propio contenedor. Aprenderá a optimizar su flujo de trabajo utilizando hiperparámetros, configurar instancias de trabajo y utilizar la compilación paramétrica para mejorar el rendimiento. Estas técnicas de personalización le ayudan a maximizar el potencial de sus cómputos cuánticos híbridos en Amazon Braket.

En esta sección:

- Definición del entorno para el script de algoritmo
- Uso de hiperparámetros
- Configuración de su instancia de trabajo híbrido
- Uso de la compilación paramétrica para acelerar los trabajos híbridos

Definición del entorno para el script de algoritmo

Amazon Braket admite entornos definidos por contenedores para el script de algoritmo:

- Un contenedor base (el predeterminado, si no se especifica un `image_uri`)
- Un contenedor con CUDA-Q
- Un contenedor con Tensorflow y PennyLane
- Un contenedor con PyTorch, y PennyLane CUDA-Q

En la tabla siguiente, se proporcionan detalles sobre los contenedores y las bibliotecas que incluyen.

Contenedores de Amazon Braket

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
URI de imagen	292282985 366.dkr. ecr.us-west-2.amaz onaws.com /:latest amazon-braket-base-jobs	292282985 366.dkr. ecr.us-west-2.amaz onaws.com /:últimc amazon-braket-cudaq-jobs	292282985 366.dkr. ecr.us-east-1.amaz onaws.com /:últimc amazon-braket-tensorflow-jobs	292282985 366.dkr. ecr.us-west-2.amaz onaws.com /:último amazon-braket-pytorch-jobs
Bibliotecas heredadas		• amazon-br aket-default-simulator • amazon-br aket-penn ylane-plugin • amazon-br aket-schemas • amazon-br aket-sdk • awscli • botocore • boto3 • dask • matplotlib • numpy • pandas • PennyLane • PennyLane-Relámpago • qiskit-braket-provider	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
		• solicitudes • sagemaker-training • scikit-learn • scipy		
Bibliotecas adicionales	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • boto3 • ipykernel • matplotlib • networkx • numpy • openbabel • pandas • PennyLane • protobuf • psi4 • rsa • scipy	• cudaq • cudaq-qec • cudaq-solvers	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras • matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-Lightning-GPU • cuQuantum	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras • matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU Lightning • cuQuantum • cudaq • cudaq-qec • cudaq-solvers

Puede ver y acceder a las definiciones de contenedores de código abierto en aws/. amazon-braket-containers Elija el contenedor que mejor se adapte a su caso de uso. Puedes usar cualquiera de las AWS regiones disponibles en Braket (us-east-1, us-west-1, us-west-2, eu-north-1, eu-west-2), pero la región contenedora debe coincidir con la región de tu trabajo híbrido. Especifique la imagen del contenedor al crear un trabajo híbrido añadiendo uno de los tres argumentos siguientes a su llamada a `create(…)` en el script del trabajo híbrido. Puede instalar dependencias adicionales en el contenedor que elija en el tiempo de ejecución (al costo del inicio o el tiempo de ejecución), ya que los contenedores de Amazon Braket tienen conectividad a Internet. En el siguiente ejemplo es para la región us-west-2.

- Imagen base: amazon-braket-base-jobs image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest»
- Imagen CUDA-Q: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest» amazon-braket-cudaq-jobs
- Imagen de Tensorflow: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest» amazon-braket-tensorflow-jobs
- PyTorch imagen: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest» amazon-braket-pytorch-jobs

Los `image-uris` también se pueden recuperar usando la función `retrieve_image()` en el SDK de Amazon Braket. El siguiente ejemplo muestra cómo recuperarlos del us-west-2 Región de AWS.

```
from braket.jobs.image_uris import retrieve_image, Framework

image_uri_base = retrieve_image(Framework.BASE, "us-west-2")
image_uri_cudaq = retrieve_image(Framework.CUDAQ, "us-west-2")
image_uri_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Utilice su propio contenedor (BYOC)

Los trabajos híbridos de Amazon Braket proporcionan tres contenedores preconstruidos para ejecutar código en diferentes entornos. Si uno de estos contenedores es compatible con su caso de uso, solo tiene que proporcionar el script de algoritmo cuando cree un trabajo híbrido. Las pequeñas dependencias que falten se pueden añadir desde el script de algoritmo o desde un archivo `requirements.txt` utilizando `pip`.

Si ninguno de estos contenedores se adapta a su caso de uso, o si desea ampliarlos, los trabajos híbridos de Braket admiten la ejecución de trabajos híbridos con su propia imagen de contenedor de Docker personalizada, o que utilice su propio contenedor (BYOC). Asegúrese de que sea la característica adecuada para su caso de uso.

En esta sección:

¿Cuándo es conveniente utilizar su propio contenedor?

Utilizar su propio contenedor (BYOC) en los trabajos híbridos de Braket le ofrece la flexibilidad de utilizar su propio software instalándolo en un entorno empaquetado. Dependiendo de sus necesidades específicas, puede haber formas de lograr la misma flexibilidad sin tener que pasar por todo el ciclo completo de construcción de BYOC Docker, carga en Amazon ECR y URI de imagen personalizada.

> **Note**
>
> BYOC puede no ser la opción adecuada si desea añadir un pequeño número de paquetes de Python adicionales (generalmente menos de 10) que están disponibles públicamente. Por ejemplo, si estás usando. PyPi

En este caso, puede utilizar una de las imágenes de Braket preconstruidas y, a continuación, incluir un archivo `requirements.txt` en su directorio de origen al enviar el trabajo. El archivo se lee automáticamente y `pip` instalará los paquetes con las versiones especificadas de forma normal. Si va a instalar una gran cantidad de paquetes, es posible que el tiempo de ejecución de sus trabajos aumente considerablemente. Compruebe la versión de Python y, si corresponde, la de CUDA del contenedor preconstruido que desee utilizar para comprobar si el software funciona.

BYOC es necesario cuando desea utilizar un lenguaje que no sea Python (como C++ o Rust) para su script de trabajo, o si desea utilizar una versión de Python que no está disponible a través de los contenedores preconstruidos de Braket. También es una buena opción en los siguientes casos:

- Está utilizando un software con una clave de licencia y necesita autenticar esa clave en un servidor de licencias para ejecutar el software. Con la opción de BYOC, puede incrustar la clave de licencia en su imagen de Docker e incluir un código para autenticarla.
- Está utilizando un software que no es de acceso público. Por ejemplo, el software está alojado en un GitHub repositorio privado GitLab o al que se necesita una clave SSH específica para acceder.
- Necesita instalar un gran conjunto de software que no está incluido en los contenedores proporcionados por Braket. BYOC le permitirá eliminar los largos tiempos de inicio de sus contenedores de trabajos híbridos debido a la instalación de software.

BYOC también le permite poner su SDK o algoritmo personalizado a disposición de los clientes al crear un contenedor de Docker con su software y ponerlo a disposición de sus usuarios. Puede hacerlo configurando los permisos adecuados en Amazon ECR.

> **Note**
>
> Debe cumplir con todas las licencias de software aplicables.

Fórmula para utilizar su propio contenedor

En esta sección, le ofrecemos una step-by-step guía sobre lo que necesita bring your own container (BYOC) para hacer Braket Hybrid Jobs: los scripts, los archivos y los pasos para combinarlos y ponerlos en marcha con sus imágenes personalizadasDocker. Las fórmulas para dos casos comunes:

1. Instale software adicional en una imagen de Docker y utilice únicamente scripts de algoritmo de Python en sus trabajos.
2. Utilice scripts de algoritmo escritos en un lenguaje que no sea Python con los trabajos híbridos o una arquitectura de CPU distinta de x86.

Definir el script de entrada de contenedor es más complejo en el caso 2.

Cuando Braket ejecuta su trabajo híbrido, inicia el número y tipo de instancias de Amazon EC2 solicitados y, a continuación, ejecuta la imagen de Docker especificada por la entrada de URI de imagen en la creación del trabajo. Al utilizar la característica BYOC, debe especificar un URI de imagen alojado en un repositorio privado de Amazon ECR al que tenga acceso de lectura. Los trabajos híbridos de Braket utilizan esa imagen personalizada para ejecutar el trabajo.

Los componentes específicos que necesita para crear una imagen de Docker que se pueda utilizar con trabajos híbridos. Si no está familiarizado con la escritura y la creación de `Dockerfiles`, consulte la documentación de Dockerfile y la documentación de Amazon ECR CLI.

Requisitos:

- Una imagen base para su Dockerfile
- (Opcional) Un script de punto de entrada de contenedor modificado
- Instalación del software y el script de contenedor necesarios con Dockerfile

Una imagen base para su Dockerfile

Si está utilizando Python y desea instalar software sobre lo que se proporciona en los contenedores proporcionados por Braket, una opción para una imagen base es una de las imágenes del contenedor de Braket, alojada en nuestro GitHub repositorio y en Amazon ECR. Tendrá que autenticarse en Amazon ECR para extraer la imagen y construir sobre ella. Por ejemplo, la primera línea del archivo de Docker de BYOC podría ser: `FROM [IMAGE_URI_HERE]`

A continuación, rellene el resto del Dockerfile para instalar y configurar el software que desea añadir al contenedor. Las imágenes de Braket preconstruidas ya contienen el script de punto de entrada de contenedor adecuado, por lo que no tiene que preocuparse por incluirlo.

Si desea utilizar un lenguaje que no sea Python, como C++, Rust o Julia, o si desea crear una imagen para una arquitectura de CPU que no sea x86, como ARM, es posible que tenga que crear una imagen pública básica. Puede encontrar muchas de estas imágenes en la galería pública de Amazon Elastic Container Registry. Asegúrese de elegir una que sea adecuada para la arquitectura de la CPU y, si es necesario, para la GPU que desee utilizar.

(Opcional) Un script de punto de entrada de contenedor modificado

> **Note**
>
> Si solo va a añadir software adicional a una imagen de Braket preconstruida, puede omitir esta sección.

Para ejecutar código que no sea Python como parte de su trabajo híbrido, modifique el script de Python que define el punto de entrada del contenedor. Por ejemplo, el script de Python `braket_container.py` en el GitHub de Amazon Braket. Este es el script que utilizan las imágenes

preconstruidas por Braket para ejecutar el script de algoritmo y establecer las variables de entorno adecuadas. El propio script del punto de entrada del contenedor debe estar en Python, pero puede lanzar scripts que no sean de Python. En el ejemplo preconstruido, puede ver que los scripts de algoritmos de Python se lanzan como un subproceso de Python o como un proceso completamente nuevo. Al modificar esta lógica, puede habilitar el script de punto de entrada para lanzar scripts de algoritmos que no sean de Python. Por ejemplo, puede modificar la función `thekick_off_customer_script()` para lanzar procesos de Rust dependientes del final de la extensión del archivo.

También puede elegir escribir un `braket_container.py` completamente nuevo. Debería copiar los datos de entrada, los archivos fuente y otros archivos necesarios de Amazon S3 en el contenedor y definir las variables de entorno adecuadas.

Instalación del software y el script de contenedor necesarios con **`Dockerfile`**

> **Note**
>
> Si utiliza una imagen de Braket preconstruida como imagen base de Docker, el script de contenedor ya está presente.

Si ha creado un script de contenedor modificado en el paso anterior, deberá copiarlo en el contenedor y definir la variable de entorno `SAGEMAKER_PROGRAM` como `braket_container.py`, o como haya llamado al script del nuevo punto de entrada del contenedor.

El siguiente es un ejemplo de un `Dockerfile` que le permite usar Julia en instancias de trabajos aceleradas por GPU:

```
FROM nvidia/cuda:12.2.0-devel-ubuntu22.04


 ARG DEBIAN_FRONTEND=noninteractive
 ARG JULIA_RELEASE=1.8
 ARG JULIA_VERSION=1.8.3


 ARG PYTHON=python3.11
 ARG PYTHON_PIP=python3-pip
 ARG PIP=pip
```

```
ARG JULIA_URL = https://julialang-s3.julialang.org/bin/linux/x64/${JULIA_RELEASE}/
ARG TAR_NAME = julia-${JULIA_VERSION}-linux-x86_64.tar.gz


ARG PYTHON_PKGS = # list your Python packages and versions here


RUN curl -s -L ${JULIA_URL}/${TAR_NAME} | tar -C /usr/local -x -z --strip-components=1
-f -


RUN apt-get update \

   && apt-get install -y --no-install-recommends \

   build-essential \

   tzdata \

   openssh-client \

   openssh-server \

   ca-certificates \

   curl \

   git \

   libtemplate-perl \

   libssl1.1 \

   openssl \

   unzip \

   wget \

   zlib1g-dev \

   ${PYTHON_PIP} \

   ${PYTHON}-dev \
```

```
 RUN ${PIP} install --no-cache --upgrade ${PYTHON_PKGS}


 RUN ${PIP} install --no-cache --upgrade sagemaker-training==4.1.3


 # Add EFA and SMDDP to LD library path
 ENV LD_LIBRARY_PATH="/opt/conda/lib/python${PYTHON_SHORT_VERSION}/site-packages/
smdistributed/dataparallel/lib:$LD_LIBRARY_PATH"
 ENV LD_LIBRARY_PATH=/opt/amazon/efa/lib/:$LD_LIBRARY_PATH


 # Julia specific installation instructions
 COPY Project.toml /usr/local/share/julia/environments/v${JULIA_RELEASE}/
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using Pkg; Pkg.instantiate(); Pkg.API.precompile()'
 # generate the device runtime library for all known and supported devices
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using CUDA; CUDA.precompile_runtime()'


 # Open source compliance scripts
 RUN HOME_DIR=/root \

 && curl -o ${HOME_DIR}/oss_compliance.zip https://aws-dlinfra-
utilities.s3.amazonaws.com/oss_compliance.zip \

 && unzip ${HOME_DIR}/oss_compliance.zip -d ${HOME_DIR}/ \

 && cp ${HOME_DIR}/oss_compliance/test/testOSSCompliance /usr/local/bin/
testOSSCompliance \

 && chmod +x /usr/local/bin/testOSSCompliance \

 && chmod +x ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh \

 && ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh ${HOME_DIR} ${PYTHON} \
```

```
 && rm -rf ${HOME_DIR}/oss_compliance*


 # Copying the container entry point script
 COPY braket_container.py /opt/ml/code/braket_container.py
 ENV SAGEMAKER_PROGRAM braket_container.py
```

En este ejemplo, se descargan y ejecutan los scripts proporcionados por AWS para garantizar el cumplimiento de todas las licencias de código abierto pertinentes. Por ejemplo, atribuyendo correctamente cualquier código instalado que se rija por una MIT license.

Si necesitas incluir código no público, por ejemplo, código alojado en un repositorio privado GitHub o en un GitLab repositorio, no insertes claves SSH en la Docker imagen para acceder a él. En su lugar, utilice Docker Compose en la creación para permitir el acceso de Docker a SSH en la máquina host en la que está integrado. Para obtener más información, consulte la guía Uso seguro de claves SSH en Docker para acceder a repositorios privados de GitHub.

Cómo crear y subir su imagen de Docker

Una vez definido`Dockerfile` correctamente, ya puede seguir los pasos para crear un repositorio privado de Amazon ECR, si aún no existe. También puede crear, etiquetar y cargar su imagen de contenedor en el repositorio.

Ya puede crear, etiquetar y enviar la imagen. Consulte la documentación de compilación de Docker para obtener una explicación completa de las opciones para `docker build` y algunos ejemplos.

Para el archivo de muestra definido anteriormente, puede ejecutar lo siguiente:

```
aws ecr get-login-password --region ${your_region} | docker login --username AWS --
password-stdin ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com
 docker build -t braket-julia .
 docker tag braket-julia:latest ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com/
braket-julia:latest
 docker push ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com/braket-julia:latest
```

Asignación de los permisos de Amazon ECR adecuados

Las imágenes de Braket Hybrid Jobs Docker deben estar alojadas en repositorios privados de Amazon ECR. De forma predeterminada, un repositorio privado de Amazon ECR no proporciona acceso de lectura al Braket Hybrid Jobs IAM role ni a ningún otro usuario que quiera usar su imagen,

como un colaborador o un estudiante. Debe establecer una política de repositorio para conceder los permisos adecuados. En general, conceda permiso únicamente a los usuarios y roles de IAM específicos a los que quiera que accedan a sus imágenes, en lugar de permitir que cualquier persona con el image URI acceda.

Ejecución de trabajos híbridos de Braket en su propio contenedor

Para crear un trabajo híbrido con su propio contenedor, llame a `AwsQuantumJob.create()` con el argumento de `image_uri` especificado. Puede utilizar una QPU, un simulador bajo demanda o ejecutar su código localmente en el procesador clásico disponible con los trabajos híbridos de Braket. Te recomendamos que pruebes tu código en un simulador o TN1 antes de ejecutarlo en una QPU real. SV1 DM1

Para ejecutar el código en el procesador clásico, especifique el `instanceType` y el `instanceCount` que utiliza actualizando la `InstanceConfig`. Tenga en cuenta que si especifica un `instance_count` > 1, debe asegurarse de que el código puede ejecutarse en varios hosts. El límite máximo de instancias que puede elegir es de 5. Por ejemplo:

```
job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    image_uri="111122223333.dkr.ecr.us-west-2.amazonaws.com/my-byoc-container:latest",
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

> **Note**
>
> Utilice el ARN del dispositivo para realizar un seguimiento del simulador que utilizó como metadatos del trabajo híbrido. Los valores aceptables deben seguir el formato `device = "local:<provider>/<simulator_name>"`. Recuerde que `<provider>` y `<simulator_name>` debe constar únicamente de letras, números, `_`, `-` y `.` El tamaño de la cadena se limita a 256 caracteres.
>
> Si planea BYOC y no usa el SDK de Braket para crear tareas cuánticas, debe pasar el valor de la variable de entorno `AMZN_BRAKET_JOB_TOKEN` al parámetro `jobToken` en la solicitud `CreateQuantumTask`. De lo contrario, las tareas cuánticas no tienen prioridad y se consideran tareas cuánticas normales e independientes.

Uso de hiperparámetros

Puede definir los hiperparámetros que necesita su algoritmo, como la tasa de aprendizaje o el tamaño del paso, al crear un trabajo híbrido. Los valores de los hiperparámetros se utilizan normalmente para controlar diversos aspectos del algoritmo y, a menudo, pueden ajustarse para optimizar el rendimiento del algoritmo. Para utilizar los hiperparámetros en un trabajo híbrido de Braket, es necesario especificar sus nombres y valores de forma explícita en un diccionario. Especifique los valores de los hiperparámetros que se van a probar al buscar el conjunto de valores óptimo. El primer paso para utilizar los hiperparámetros es configurar y definir los hiperparámetros como un diccionario, como se puede ver en el siguiente código.

```
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

hyperparameters = {"shots": 1_000}
```

A continuación, pase los hiperparámetros definidos en el fragmento de código indicado anteriormente para utilizarlos en el algoritmo que elija. Para ejecutar el siguiente ejemplo de código, cree un directorio denominado «src» en la misma ruta que el archivo de hiperparámetros. Dentro del directorio «src», añada los archivos de código 0_Getting_started_papermill.ipynb, notebook_runner.py y requirements.txt.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="src",
    entry_point="src.notebook_runner:run_notebook",
    input_data="src/0_Getting_started_papermill.ipynb",
    hyperparameters=hyperparameters,
    job_name=f"papermill-job-demo-{int(time.time())}",
)

# Print job to record the ARN
print(job)
```

Para acceder a sus hiperparámetros desde dentro de su script de trabajo híbrido, consulte la función `load_jobs_hyperparams()` en el archivo de python notebook_runner.py. Para acceder a los hiperparámetros desde fuera de su script de trabajo híbrido, ejecute el siguiente código.

```
from braket.aws import AwsQuantumJob

# Get the job using the ARN
job_arn = "arn:aws:braket:us-east-1:111122223333:job/5eabb790-d3ff-47cc-98ed-
b4025e9e296f"  # Replace with your job ARN
job = AwsQuantumJob(arn=job_arn)

# Access the hyperparameters
job_metadata = job.metadata()
hyperparameters = job_metadata.get("hyperParameters", {})
print(hyperparameters)
```

Para obtener más información sobre cómo usar los hiperparámetros, consulte los tutoriales de QAOA con Amazon Braket Hybrid Jobs PennyLane y Quantum Machine Learning en Amazon Braket Hybrid Jobs.

Configuración de su instancia de trabajo híbrido

Dependiendo de tu algoritmo, es posible que tengas diferente requisitos. De forma predeterminada, Amazon Braket ejecuta el script de algoritmo en una instancia de `ml.m5.large`. Sin embargo, puede personalizar este tipo de instancia al crear un trabajo híbrido mediante el siguiente argumento de importación y configuración.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"), # Use NVIDIA T4
 instance with 4 GPUs.
    ...
    ),
```

Si está ejecutando una simulación integrada y ha especificado un dispositivo local en la configuración del dispositivo, también puede solicitar más de una instancia en la `InstanceConfig` especificando el `instanceCount` y configurándolo para que sea mayor que uno. El límite superior es 5. Por ejemplo, puede elegir 3 instancias de la siguiente manera.

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3), #
 Use 3 NVIDIA T4 instances
    ...
    ),
```

Cuando utilice varias instancias, considere la posibilidad de distribuir su trabajo híbrido mediante la característica de paralelismo de datos. Consulte el siguiente cuaderno de ejemplo para obtener más información sobre cómo ver este ejemplo de paralelizar el entrenamiento para QML.

Las tres tablas siguientes enumeran los tipos de instancias disponibles y las especificaciones para instancias estándar, de alto rendimiento y aceleradas por GPU.

> **Note**
>
> Para ver las cuotas de instancias de cómputo clásicas predeterminadas para trabajos híbridos, consulte la página Cuotas de Amazon Braket.

Instancia estándar	vCPU	Memoria (GiB)
ml.m5.large (predeterminado)	4	16
ml.m5.xlarge	4	16
ml.m5.2xlarge	8	32
ml.m5.4xlarge	16	64
ml.m5.12xlarge	48	192
ml.m5.24xlarge	96	384

Instancias de alto rendimiento	vCPU	Memoria (GiB)
ml.c5.xlarge	4	8

Instancias de alto rendimiento	vCPU	Memoria (GiB)
ml.c5.2xlarge	8	16
ml.c5.4xlarge	16	32
ml.c5.9xlarge	36	72
ml.c5.18xlarge	72	144
ml.c5n.xlarge	4	10.5
ml.c5n.2xlarge	8	21
ml.c5n.4xlarge	16	32
ml.c5n.9xlarge	36	72
ml.c5n.18xlarge	72	192

Instancias aceleradas por GPU	GPUs	vCPU	Memoria (GiB)	Memoria de GPU (GiB)
ml.p4d.24xlarge	8	96	1152	320
ml.g4dn.xlarge	1	4	16	16
ml.g4dn.2xlarge	1	8	32	16
ml.g4dn.4xlarge	1	16	64	16
ml.g4dn.8xlarge	1	32	128	16
ml.g4dn.12xlarge	4	48	192	64
ml.g4dn.16xlarge	1	64	256	16

Cada instancia usa una configuración predeterminada de almacenamiento de datos (SSD) de 30 GB. Sin embargo, puede ajustar el almacenamiento de la misma manera que configure el `instanceType`. El siguiente ejemplo muestra cómo aumentar el almacenamiento total a 50 GB.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(
        instanceType="ml.g4dn.xlarge",
        volumeSizeInGb=50,
    ),
    ...
    ),
```

Configuración de bucket predeterminado en **`AwsSession`**

El uso de su propia instancia de `AwsSession` le proporciona una mayor flexibilidad, como la posibilidad de especificar una ubicación personalizada para su bucket de Amazon S3 predeterminado. De forma predeterminada, un `AwsSession` tiene una ubicación de bucket de Amazon S3 preconfigurada de `"amazon-braket-{id}-{region}"`. Sin embargo, tiene la opción de anular la ubicación de bucket de Amazon S3 predeterminada al crear un `AwsSession`. Los usuarios pueden pasar opcionalmente un objeto de `AwsSession` al método `AwsQuantumJob.create()` proporcionando el parámetro `aws_session` como se muestra en el siguiente ejemplo de código.

```
aws_session = AwsSession(default_bucket="amazon-braket-s3-demo-bucket")

# Then you can use that AwsSession when creating a hybrid job
job = AwsQuantumJob.create(
    ...
    aws_session=aws_session
)
```

Uso de la compilación paramétrica para acelerar los trabajos híbridos

Amazon Braket admite la compilación paramétrica en algunos casos. QPUs Esto le permite reducir la sobrecarga asociada con el costoso paso de compilación computacional, ya que solo es necesario compilar un circuito una vez y no para cada iteración en su algoritmo híbrido. Esto puede mejorar considerablemente los tiempos de ejecución de los trabajos híbridos, ya que evita la necesidad

de recompilar el circuito en cada paso. Simplemente envíe los circuitos parametrizados a uno de nuestros trabajos compatibles QPUs como Braket Hybrid. Para trabajos híbridos de larga duración, Braket utiliza automáticamente los datos de calibración actualizados del proveedor de hardware al compilar su circuito para garantizar resultados de la máxima calidad.

Para crear un circuito paramétrico, primero debe proporcionar los parámetros como entradas en el script de algoritmo. En este ejemplo, utilizamos un circuito paramétrico pequeño e ignoramos cualquier procesamiento clásico entre cada iteración. Para las cargas de trabajo típicas, debe enviar muchos circuitos por lotes y realizar el procesamiento clásico, como la actualización de los parámetros en cada iteración.

```
import os

from braket.aws import AwsDevice
from braket.circuits import Circuit, FreeParameter

def start_here():

    print("Test job started.")

    # Use the device declared in the job script
    device = AwsDevice(os.environ["AMZN_BRAKET_DEVICE_ARN"])

    circuit = Circuit().rx(0, FreeParameter("theta"))
    parameter_list = [0.1, 0.2, 0.3]

    for parameter in parameter_list:
        result = device.run(circuit, shots=1000, inputs={"theta": parameter})

    print("Test job completed.")
```

Puede enviar el script de algoritmo para que se ejecute como un trabajo híbrido con el siguiente script de trabajo. Al ejecutar el trabajo híbrido en una QPU que admite la compilación paramétrica, el circuito se compila solo en la primera ejecución. En las siguientes ejecuciones, el circuito compilado se reutiliza, lo que aumenta el rendimiento en el tiempo de ejecución del trabajo híbrido sin líneas de código adicionales.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
```

```
    source_module="algorithm_script.py",
)
```

> **Note**
>
> La compilación paramétrica es compatible con todos los sistemas superconductores basados en compuertas, Rigetti Computing con la excepción de los programas QPUs de nivel de pulso.

Uso PennyLane con Amazon Braket

Los algoritmos híbridos son algoritmos que contienen instrucciones clásicas y cuánticas. Las instrucciones clásicas se ejecutan en un hardware clásico (una instancia de EC2 o su portátil) y las instrucciones cuánticas se ejecutan en un simulador o en una computadora cuántica. Se recomienda ejecutar algoritmos híbridos mediante la característica de trabajos híbridos. Para obtener más información, consulte Cuándo utilizar los trabajos de Amazon Braket.

Amazon Braket le permite configurar y ejecutar algoritmos cuánticos híbridos con la ayuda del complemento Amazon Braket o con el SDK de Python de Amazon PennyLane Braket y repositorios de cuadernos de ejemplo. Los cuadernos de ejemplo de Amazon Braket, basados en el SDK, permiten configurar y ejecutar determinados algoritmos híbridos sin el complemento. PennyLane Sin embargo, lo recomendamos PennyLane porque proporciona una experiencia más rica.

Acerca de los algoritmos cuánticos híbridos

Los algoritmos cuánticos híbridos son importantes para la industria actual porque los dispositivos de computación cuántica contemporáneos generalmente producen ruido y, por lo tanto, errores. Cada puerta cuántica que se añade a un cómputo aumenta la probabilidad de añadir ruido; por lo tanto, los algoritmos de larga duración pueden verse abrumados por el ruido, lo que da lugar a cómputos erróneos.

Los algoritmos cuánticos puros como el de Shor (ejemplo de estimación de fase cuántica) o el de Grover (ejemplo de Grover) requieren miles o millones de operaciones. Por este motivo, pueden resultar poco prácticos para los dispositivos cuánticos existentes, que generalmente se denominan dispositivos cuánticos ruidosos de escala intermedia (NISQ).

En los algoritmos cuánticos híbridos, las unidades de procesamiento cuántico (QPUs) funcionan como coprocesadores en los algoritmos clásicos CPUs, específicamente para acelerar ciertos

cálculos en un algoritmo clásico. Las ejecuciones de circuitos se vuelven mucho más cortas, al alcance de las capacidades de los dispositivos actuales.

En esta sección:

Amazon Braket con PennyLane

Amazon Braket ofrece soporte para PennyLaneun marco de software de código abierto creado en torno al concepto de programación cuántica diferenciable. Puede utilizar este marco para entrenar circuitos cuánticos de la misma manera que entrenaría una red neuronal para encontrar soluciones a problemas computacionales en química cuántica, machine learning cuántico y optimización.

La PennyLane biblioteca proporciona interfaces a herramientas conocidas de aprendizaje automático, que incluyen PyTorch y TensorFlow permiten que el entrenamiento de los circuitos cuánticos sea rápido e intuitivo.

- La PennyLane biblioteca: PennyLane viene preinstalada en los cuadernos Amazon Braket. Para acceder a los dispositivos Amazon Braket desde PennyLane, abra un cuaderno e importe la PennyLane biblioteca con el siguiente comando.

```
import pennylane as qml
```

Los cuadernos de tutoriales lo ayudan a empezar rápidamente. Como alternativa, puede utilizarla PennyLane en Amazon Braket desde el IDE que prefiera.

- El PennyLane complemento Amazon Braket: para usar su propio IDE, puede instalar el complemento Amazon Braket manualmente PennyLane . El complemento se conecta PennyLane con el SDK de Python de Amazon Braket, por lo que puede ejecutar circuitos PennyLane en dispositivos Amazon Braket. Para instalar el PennyLane complemento, utilice el siguiente comando.

```
pip install amazon-braket-pennylane-plugin
```

El siguiente ejemplo muestra cómo configurar el acceso a los dispositivos Amazon Braket en PennyLane:

```
# to use SV1
import pennylane as qml
sv1 = qml.device("braket.aws.qubit", device_arn="arn:aws:braket:::device/quantum-
simulator/amazon/sv1", wires=2)

# to run a circuit:
@qml.qnode(sv1)
def circuit(x):
    qml.RZ(x, wires=0)
    qml.CNOT(wires=[0,1])
    qml.RY(x, wires=1)
    return qml.expval(qml.PauliZ(1))

result = circuit(0.543)


#To use the local sim:
local = qml.device("braket.local.qubit", wires=2)
```

Para ver ejemplos de tutoriales y más información al respecto PennyLane, consulte el repositorio de ejemplos de Amazon Braket.

El PennyLane complemento Amazon Braket le permite cambiar entre la QPU de Amazon Braket y los dispositivos simuladores integrados PennyLane con una sola línea de código. Ofrece dos dispositivos cuánticos Amazon Braket con los que trabajar: PennyLane

- `braket.aws.qubit`para funcionar con los dispositivos cuánticos del servicio Amazon Braket, incluidos QPUs los simuladores
- `braket.local.qubit` para ejecutarse con el simulador local del SDK de Amazon Braket.

El PennyLane plugin Amazon Braket es de código abierto. Puedes instalarlo desde el GitHub repositorio de PennyLane complementos.

Para obtener más información al respecto PennyLane, consulte la documentación del sitio PennyLane web.

Algoritmos híbridos en cuadernos de ejemplo de Amazon Braket

Amazon Braket proporciona una variedad de cuadernos de ejemplo que no dependen del PennyLane complemento para ejecutar algoritmos híbridos. Puede empezar con cualquiera de estos cuadernos de ejemplos híbridos de Amazon Braket que ilustran métodos variacionales, como el algoritmo de optimización cuántica aproximada (QAOA) o el solucionador de problemas de autovalores cuánticos variacionales (VQE).

Los cuadernos de ejemplo de Amazon Braket se basan en el SDK de Python de Amazon Braket. El SDK proporciona un marco para interactuar con los dispositivos de hardware de computación cuántica a través de Amazon Braket. Se trata de una biblioteca de código abierto diseñada para ayudarle con la parte cuántica de su flujo de trabajo híbrido.

Puede explorar Amazon Braket más a fondo con nuestros cuadernos de ejemplo.

Algoritmos híbridos con simuladores integrados PennyLane

Amazon Braket Hybrid Jobs ahora incluye simuladores integrados de alto rendimiento basados en CPU y GPU de. PennyLane Esta familia de simuladores integrados se puede incorporar directamente en su contenedor de trabajos híbridos e incluye el simulador rápido de vectores de estado `lightning.qubit`, el simulador `lightning.gpu` acelerado mediante la biblioteca cuQuantum de NVIDIA y otros. Estos simuladores integrados son ideales para algoritmos variacionales, como el machine learning cuántico, que pueden beneficiarse de métodos avanzados como el método de diferenciación adjunta. Puede ejecutar estos simuladores integrados en una o varias instancias de CPU o GPU.

Con Hybrid Jobs, ahora puede ejecutar el código de su algoritmo variacional utilizando una combinación de un coprocesador clásico y una QPU, un simulador Amazon Braket bajo demanda como, por ejemploSV1, o directamente utilizando el simulador integrado desde. PennyLane

El simulador integrado ya está disponible con el contenedor de trabajos híbridos y usted debe decorar su función principal de Python con el decorador `@hybrid_job`. Para usar el PennyLane `lightning.gpu` simulador, también debe especificar una instancia de GPU, `InstanceConfig` tal como se muestra en el siguiente fragmento de código:

```
import pennylane as qml
from braket.jobs import hybrid_job
from braket.jobs.config import InstanceConfig
```

```
@hybrid_job(device="local:pennylane/lightning.gpu",
 instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"))
def function(wires):
    dev = qml.device("lightning.gpu", wires=wires)
    ...
```

Consulta el cuaderno de ejemplo para empezar a usar un simulador PennyLane integrado con Hybrid Jobs.

Activa el gradiente adjunto PennyLane con los simuladores Amazon Braket

Con el PennyLane complemento para Amazon Braket, puede calcular gradientes mediante el método de diferenciación adjunto cuando se ejecuta en el simulador vectorial de estado local o. SV1

Note: Para utilizar el método de diferenciación adjunta, debe especificar `diff_method='device'` en su `qnode`, y no `diff_method='adjoint'`. Consulte el siguiente ejemplo.

```
device_arn = "arn:aws:braket:::device/quantum-simulator/amazon/sv1"
dev = qml.device("braket.aws.qubit", wires=wires, shots=0, device_arn=device_arn)

@qml.qnode(dev, diff_method="device")
def cost_function(params):
    circuit(params)
    return qml.expval(cost_h)

gradient = qml.grad(circuit)
initial_gradient = gradient(params0)
```

> **Note**
>
> Actualmente, PennyLane computará los índices de agrupación para los hamiltonianos de QAOA y los utilizará para dividir el hamiltoniano en diversos valores esperados. Si quiere utilizar SV1 la función de diferenciación adjunta al ejecutar la QAOAPennyLane, necesitará reconstruir el coste hamiltoniano eliminando los índices de agrupamiento, de la siguiente manera: `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False)` `cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Utilizar Hybrid Jobs y ejecutar un algoritmo de QAOA PennyLane

En esta sección, utilizará lo que ha aprendido para escribir un programa híbrido real utilizando la compilación PennyLane paramétrica. Utilizará el script del algoritmo para abordar un problema de algoritmo de optimización cuántica aproximada (QAOA). El programa crea una función de costo correspondiente a un problema clásico de optimización Max Cut, especifica un circuito cuántico parametrizado y utiliza un método de descenso de gradiente para optimizar los parámetros de manera que se minimice la función de costo. En este ejemplo, generamos el gráfico del problema en el script del algoritmo para simplificar, pero para casos de uso más habituales, la mejor práctica es proporcionar la especificación del problema a través de un canal dedicado en la configuración de los datos de entrada. El indicador se establece de `parametrize_differentiable` forma predeterminada para `True` que pueda disfrutar automáticamente de las ventajas de un mejor rendimiento en tiempo de ejecución gracias a la compilación paramétrica si es compatible. QPUs

```
import os
import json
import time

from braket.jobs import save_job_result
from braket.jobs.metrics import log_metric

import networkx as nx
import pennylane as qml
from pennylane import numpy as np
from matplotlib import pyplot as plt

def init_pl_device(device_arn, num_nodes, shots, max_parallel):
    return qml.device(
        "braket.aws.qubit",
        device_arn=device_arn,
        wires=num_nodes,
        shots=shots,
        # Set s3_destination_folder=None to output task results to a default folder
        s3_destination_folder=None,
        parallel=True,
        max_parallel=max_parallel,
        parametrize_differentiable=True, # This flag is True by default.
    )

def start_here():
    input_dir = os.environ["AMZN_BRAKET_INPUT_DIR"]
```

```
    output_dir = os.environ["AMZN_BRAKET_JOB_RESULTS_DIR"]
    job_name = os.environ["AMZN_BRAKET_JOB_NAME"]
    checkpoint_dir = os.environ["AMZN_BRAKET_CHECKPOINT_DIR"]
    hp_file = os.environ["AMZN_BRAKET_HP_FILE"]
    device_arn = os.environ["AMZN_BRAKET_DEVICE_ARN"]

    # Read the hyperparameters
    with open(hp_file, "r") as f:
        hyperparams = json.load(f)

    p = int(hyperparams["p"])
    seed = int(hyperparams["seed"])
    max_parallel = int(hyperparams["max_parallel"])
    num_iterations = int(hyperparams["num_iterations"])
    stepsize = float(hyperparams["stepsize"])
    shots = int(hyperparams["shots"])

    # Generate random graph
    num_nodes = 6
    num_edges = 8
    graph_seed = 1967
    g = nx.gnm_random_graph(num_nodes, num_edges, seed=graph_seed)

    # Output figure to file
    positions = nx.spring_layout(g, seed=seed)
    nx.draw(g, with_labels=True, pos=positions, node_size=600)
    plt.savefig(f"{output_dir}/graph.png")

    # Set up the QAOA problem
    cost_h, mixer_h = qml.qaoa.maxcut(g)

    def qaoa_layer(gamma, alpha):
        qml.qaoa.cost_layer(gamma, cost_h)
        qml.qaoa.mixer_layer(alpha, mixer_h)

    def circuit(params, **kwargs):
        for i in range(num_nodes):
            qml.Hadamard(wires=i)
        qml.layer(qaoa_layer, p, params[0], params[1])

    dev = init_pl_device(device_arn, num_nodes, shots, max_parallel)

    np.random.seed(seed)
    cost_function = qml.ExpvalCost(circuit, cost_h, dev, optimize=True)
```

```
    params = 0.01 * np.random.uniform(size=[2, p])

    optimizer = qml.GradientDescentOptimizer(stepsize=stepsize)
    print("Optimization start")

    for iteration in range(num_iterations):
        t0 = time.time()

        # Evaluates the cost, then does a gradient step to new params
        params, cost_before = optimizer.step_and_cost(cost_function, params)
        # Convert cost_before to a float so it's easier to handle
        cost_before = float(cost_before)

        t1 = time.time()

        if iteration == 0:
            print("Initial cost:", cost_before)
        else:
            print(f"Cost at step {iteration}:", cost_before)

        # Log the current loss as a metric
        log_metric(
            metric_name="Cost",
            value=cost_before,
            iteration_number=iteration,
        )

        print(f"Completed iteration {iteration + 1}")
        print(f"Time to complete iteration: {t1 - t0} seconds")

    final_cost = float(cost_function(params))
    log_metric(
        metric_name="Cost",
        value=final_cost,
        iteration_number=num_iterations,
    )

    # We're done with the hybrid job, so save the result.
    # This will be returned in job.result()
    save_job_result({"params": params.numpy().tolist(), "cost": final_cost})
```

> **Note**
>
> La compilación paramétrica es compatible con todos los programas superconductores basados en compuertas, Rigetti Computing con la excepción de los QPUs programas de nivel de pulso.

Ejecute cargas de trabajo híbridas con simuladores integrados PennyLane

Veamos cómo puede utilizar los simuladores integrados de PennyLane Amazon Braket Hybrid Jobs para ejecutar cargas de trabajo híbridas. El simulador integrado basado en GPU de PennyLane, `lightning.gpu`, usa la biblioteca de Nvidia cuQuantum para acelerar las simulaciones de circuitos. El simulador de GPU integrado viene preconfigurado en todos los contenedores de trabajos de Braket, que los usuarios pueden utilizar de forma inmediata. En esta página, le mostramos cómo usar `lightning.gpu` para acelerar las cargas de trabajo híbridas.

Uso de **`lightning.gpu`** para cargas de trabajo de QAOA

Considere los ejemplos del algoritmo de optimización cuántica aproximada (QAOA) de este cuaderno. Para seleccionar un simulador integrado, debe especificar que el argumento `device` sea una cadena con la forma: `"local:<provider>/<simulator_name>"`. Por ejemplo, configuraría `"local:pennylane/lightning.gpu"` para `lightning.gpu`. La cadena de dispositivo que proporciona al trabajo híbrido al iniciarlo se pasa al trabajo como el `"AMZN_BRAKET_DEVICE_ARN"` de la variable de entorno.

```
device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
prefix, device_name = device_string.split("/")
device = qml.device(simulator_name, wires=n_wires)
```

En esta página, compare los dos simuladores vectoriales de PennyLane estado integrados (que están basados en la CPU) y `lightning.qubit` `lightning.gpu` (que están basados en la GPU). Proporcione a los simuladores descomposiciones de puertas personalizadas para calcular varios gradientes.

Ahora ya puede preparar el script de lanzamiento del trabajo híbrido. Ejecute el algoritmo QAOA mediante dos tipos de instancias: `ml.m5.2xlarge` y `ml.g4dn.xlarge`. El tipo de instancia `ml.m5.2xlarge` es equiparable al de un portátil de desarrollador estándar. `ml.g4dn.xlarge`Se

trata de una instancia de computación acelerada que tiene una sola GPU NVIDIA T4 con 16 GB de memoria.

Para ejecutar la GPU, primero debemos especificar una imagen compatible y la instancia correcta (que por defecto es una `ml.m5.2xlarge` instancia).

```
from braket.aws import AwsSession
from braket.jobs.image_uris import Framework, retrieve_image

image_uri = retrieve_image(Framework.PL_PYTORCH, AwsSession().region)
instance_config = InstanceConfig(instanceType="ml.g4dn.xlarge")
```

A continuación, debemos introducirlos en el decorador de tareas híbridas, junto con los parámetros actualizados del dispositivo, tanto en el sistema como en los argumentos de los trabajos híbridos.

```
@hybrid_job(
        device="local:pennylane/lightning.gpu",
        input_data=input_file_path,
        image_uri=image_uri,
        instance_config=instance_config)
def run_qaoa_hybrid_job_gpu(p=1, steps=10):
    params = np.random.rand(2, p)

    braket_task_tracker = Tracker()

    graph = nx.read_adjlist(input_file_path, nodetype=int)
    wires = list(graph.nodes)
    cost_h, _mixer_h = qaoa.maxcut(graph)

    device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
    prefix, device_name = device_string.split("/")
    dev= qml.device(simulator_name, wires=len(wires))
    ...
```

> **Note**
>
> Si especificas el `instance_config` como mediante una instancia basada en la GPU, pero eliges `device` que sea el simulador integrado basado en la CPU (`lightning.qubit`), no se utilizará la GPU. Asegúrese de utilizar el simulador integrado basado en GPU si desea utilizar la GPU como destino.

El tiempo medio de iteración de la `m5.2xlarge` instancia es de unos 73 segundos, mientras que el de la `ml.g4dn.xlarge` instancia es de unos 0,6 segundos. Para este flujo de trabajo de 21 qubits, la instancia de GPU nos proporciona una aceleración de 100 veces. Si consulta la página de precios de Amazon Braket Hybrid Jobs, verá que el coste por minuto de una `m5.2xlarge` instancia es de 0,00768$, mientras que para la `ml.g4dn.xlarge` instancia es de 0,01227$. En este caso, es más rápido y económico ejecutarlo en la instancia de GPU.

Machine learning cuántico y paralelismo de datos

Si su tipo de carga de trabajo es el machine learning cuántico (QML) que se entrena con conjuntos de datos, puede acelerar aún más su carga de trabajo utilizando el paralelismo de datos. En QML, el modelo contiene uno o más circuitos cuánticos. El modelo puede o no contener también redes neuronales clásicas. Al entrenar el modelo con el conjunto de datos, los parámetros del modelo se actualizan para minimizar la función de pérdida. Por lo general, se define una función de pérdida para un único punto de datos y la pérdida total para la pérdida media de todo el conjunto de datos. En QML, las pérdidas generalmente se calculan en serie antes de promediar la pérdida total para los cómputos de gradientes. Este procedimiento lleva mucho tiempo, especialmente cuando hay cientos de puntos de datos.

Como la pérdida de un punto de datos no depende de otros puntos de datos, las pérdidas se pueden evaluar en paralelo. Las pérdidas y los gradientes asociados a diferentes puntos de datos se pueden evaluar al mismo tiempo. Esto se conoce como paralelismo de datos. Con SageMaker la biblioteca paralela de datos distribuida, Amazon Braket Hybrid Jobs le facilita el uso del paralelismo de datos para acelerar su entrenamiento.

Considere la siguiente carga de trabajo de QML para el paralelismo de datos, que utiliza el conjunto de datos Sonar del conocido repositorio de UCI como ejemplo para la clasificación binaria. El conjunto de datos de Sonar tiene 208 puntos de datos, cada uno con 60 características que se recopilan a partir de señales de Sonar que rebotan en los materiales. Cada punto de datos se etiqueta con una «M» para las minas o con una «R» para las rocas. Nuestro modelo QML consta de una capa de entrada, un circuito cuántico como capa oculta y una capa de salida. Las capas de entrada y salida son redes neuronales clásicas implementadas en. PyTorch El circuito cuántico se integra con las redes PyTorch neuronales mediante PennyLane el módulo qml.qnn. Consulte nuestros cuadernos de ejemplo para obtener más detalles sobre la carga de trabajo. Al igual que en el ejemplo anterior de QAOA, puede aprovechar la potencia de la GPU mediante el uso de simuladores integrados basados en la GPU, como PennyLane los que se utilizan para mejorar el rendimiento en comparación con los simuladores basados en la CPU integrada. `lightning.gpu`

Para crear un trabajo híbrido, puede llamar a `AwsQuantumJob.create` y especificar el script del algoritmo, el dispositivo y otras configuraciones a través de sus argumentos de palabra clave.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.xlarge')

hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_single",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    ...
)
```

Para utilizar el paralelismo de datos, es necesario modificar algunas líneas de código en el script del algoritmo para que la biblioteca distribuida paralelice correctamente el entrenamiento. SageMaker En primer lugar, importa el `smdistributed` paquete que se encarga de la mayor parte del trabajo pesado de distribuir las cargas de trabajo en varias y múltiples instancias. GPUs Este paquete viene preconfigurado en Braket y en los contenedores. PyTorch TensorFlow El `dist` módulo indica a nuestro script de algoritmo cuál es el número total de núcleos GPUs para el entrenamiento (`world_size`) `rank` y el núcleo `local_rank` de la GPU. `rank`es el índice absoluto de una GPU en todas las instancias, mientras que `local_rank` es el índice de una GPU dentro de una instancia. Por ejemplo, si hay cuatro instancias, cada una con ocho GPUs asignadas para el entrenamiento, los `rank` rangos van de 0 a 31 y los `local_rank` rangos de 0 a 7.

```
import smdistributed.dataparallel.torch.distributed as dist

dp_info = {
    "world_size": dist.get_world_size(),
    "rank": dist.get_rank(),
    "local_rank": dist.get_local_rank(),
}
batch_size //= dp_info["world_size"] // 8
batch_size = max(batch_size, 1)
```

A continuación, se define un `DistributedSampler` según el `world_size` y `rank` y, después, se pasa al cargador de datos. Este muestreador evita GPUs acceder a la misma porción de un conjunto de datos.

```
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset,
    num_replicas=dp_info["world_size"],
    rank=dp_info["rank"]
)
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=0,
    pin_memory=True,
    sampler=train_sampler,
)
```

A continuación, use la clase `DistributedDataParallel` para habilitar el paralelismo de datos.

```
from smdistributed.dataparallel.torch.parallel.distributed import
 DistributedDataParallel as DDP

model = DressedQNN(qc_dev).to(device)
model = DDP(model)
torch.cuda.set_device(dp_info["local_rank"])
model.cuda(dp_info["local_rank"])
```

Los cambios anteriores son los que necesita para usar el paralelismo de datos. Generalmente, en QML es conveniente guardar los resultados e imprimir el progreso del entrenamiento. Si cada GPU ejecuta el comando de guardar e imprimir, el registro se saturará con información repetida y los resultados se sobrescribirán entre sí. Para evitarlo, solo puede guardar e imprimir desde la GPU que tiene `rank` 0.

```
if dp_info["rank"]==0:
    print('elapsed time: ', elapsed)
    torch.save(model.state_dict(), f"{output_dir}/test_local.pt")
    save_job_result({"last loss": loss_before})
```

Amazon Braket Hybrid Jobs admite tipos de `ml.g4dn.12xlarge` instancias para la biblioteca paralela de datos SageMaker distribuidos. El tipo de instancia se configura mediante el argumento

`InstanceConfig` en los trabajos híbridos. Para que la biblioteca paralela de datos SageMaker distribuidos sepa que el paralelismo de datos está habilitado, debe agregar dos hiperparámetros adicionales: configurar `"true"` y `"sagemaker_distributed_dataparallel_enabled"` `"sagemaker_instance_type"` configurar el tipo de instancia que está utilizando. El paquete de `smdistributed` utiliza estos dos hiperparámetros. El script de algoritmo no necesita usarlos de forma explícita. En el SDK de Amazon Braket, proporciona un práctico argumento de palabra clave `distribution`. Con el `distribution="data_parallel"` en la creación de trabajos híbridos, el SDK de Amazon Braket inserta automáticamente los dos hiperparámetros por usted. Si utiliza la API de Amazon Braket, debe incluir estos dos hiperparámetros.

Una vez configurados el paralelismo de datos y la instancia, ya puede enviar su trabajo híbrido. Hay 4 GPUs en una instancia. `ml.g4dn.12xlarge` Cuando lo configura`instanceCount=1`, la carga de trabajo se distribuye entre las 8 GPUs de la instancia. Si configuras `instanceCount` más de uno, la carga de trabajo se distribuye entre las GPUs disponibles en todas las instancias. Al usar varias instancias, cada instancia conlleva un cargo en función del tiempo que la utilice. Por ejemplo, cuando utiliza cuatro instancias, el tiempo facturable es cuatro veces el tiempo de ejecución por instancia, ya que hay cuatro instancias ejecutando sus cargas de trabajo al mismo tiempo.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.12xlarge',
                                 instanceCount=1,
)

hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...,
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_dp",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    distribution="data_parallel",
    ...
)
```

> **Note**
>
> En la creación del trabajo híbrido anterior, `train_dp.py` es el script del algoritmo modificado para utilizar el paralelismo de datos. Tenga en cuenta que el paralelismo de datos solo funciona correctamente cuando modifica el script de algoritmo según lo indicado en la sección anterior. Si se habilita la opción de paralelismo de datos sin un script de algoritmo modificado correctamente, el trabajo híbrido puede generar errores, o cada GPU puede procesar repetidamente la misma porción de datos, lo cual es ineficiente.

Si se utiliza correctamente, el uso de varias instancias puede suponer una reducción de varios órdenes de magnitud tanto en tiempo como en costes. Consulte el cuaderno de ejemplo para obtener más información.

Uso de CUDA-Q con Amazon Braket

NVIDIA's CUDA-Qes una biblioteca de software diseñada para programar algoritmos cuánticos híbridos que combinan CPUs GPUs, y unidades de procesamiento cuántico (QPUs). Proporciona un modelo de programación unificado que permite a los desarrolladores expresar instrucciones clásicas y cuánticas en un solo programa, lo que agiliza los flujos de trabajo. CUDA-Qacelera la simulación cuántica y el tiempo de ejecución de los programas con sus simuladores de CPU y GPU integrados. CUDA-Qestá disponible con instancias de notebook Braket nativas (NBIs) y Amazon Braket Hybrid Jobs.

En esta sección:

- CUDA-Q en NBIs
- CUDA-Q en trabajos híbridos

CUDA-Q en NBIs

CUDA-Q está instalado de forma predeterminada en el entorno de NBI de Braket. Para abrir un cuaderno de CUDA-Q ejemplo, vaya a la página del lanzador de Jupyter y seleccione el mosaico de CUDA-Q y Braket. Esto abre el cuaderno de ejemplo `0_Getting_started_with_CUDA-Q.ipynb` en la ventana principal. Para ver más ejemplos de CUDA-Q, consulte el panel izquierdo en el directorio de `nvidia_cuda_q/`.

También puede comprobar la versión de CUDA-Q o cualquier otro paquete de terceros instalado en su NBI. Por ejemplo, puede ejecutar el siguiente comando en una celda de código de un bloc de notas para comprobar las versiones de CUDA-Q los paquetes Qiskit y Braket que están instaladas en el entorno. PennyLane

```
%pip freeze | grep -i -e cudaq -e qiskit -e pennylane -e braket
```

CUDA-Q en trabajos híbridos

El uso de CUDA-Q en trabajos híbridos de Amazon Braket ofrece un entorno de computación flexible y bajo demanda. Las instancias computacionales solo se ejecutan durante el tiempo que dura su carga de trabajo, lo que garantiza que solo pague por lo que utiliza. Los trabajos híbridos de Amazon Braket también proporcionan una experiencia escalable. Los usuarios pueden comenzar con instancias más pequeñas para crear prototipos y realizar pruebas, y luego escalar verticalmente a instancias más grandes capaces de manejar mayores cargas de trabajo para realizar experimentos completos.

Amazon Braket Hybrid Jobs apoya, GPUs que es esencial para maximizar su potencial. CUDA-Q GPUs acelera considerablemente las simulaciones de programas cuánticos en comparación con los simuladores basados en CPU, especialmente cuando se trabaja con circuitos con un alto número de cúbits. La paralelización se simplifica cuando se utiliza CUDA-Q en los trabajos híbridos de Amazon Braket. Los trabajos híbridos simplifican la distribución del muestreo de circuitos y las evaluaciones observables entre varios nodos computacionales. Esta paralelización fluida de las cargas de trabajo de CUDA-Q permite a los usuarios centrarse más en desarrollar sus cargas de trabajo en lugar de configurar la infraestructura para experimentos a gran escala.

Para empezar, consulte el ejemplo para empezar con CUDA-Q en el GitHub de ejemplos de Amazon Braket para utilizar un contenedor de trabajos híbridos de CUDA-Q proporcionado por Braket.

El siguiente fragmento de código es un ejemplo de `hello-world` para ejecutar un programa de CUDA-Q con los trabajos híbridos de Amazon Braket.

```
image_uri = retrieve_image(Framework.CUDAQ, AwsSession().region)

@hybrid_job(device='local:nvidia/qpp-cpu', image_uri=image_uri)
def hello_quantum():
    import cudaq

    # define the backend
```

```
    device=get_job_device_arn()
    cudaq.set_target(device.split('/')[-1])

    # define the Bell circuit
    kernel = cudaq.make_kernel()
    qubits = kernel.qalloc(2)
    kernel.h(qubits[0])
    kernel.cx(qubits[0], qubits[1])

    # sample the Bell circuit
    result = cudaq.sample(kernel, shots_count=1000)
    measurement_probabilities = dict(result.items())

    return measurement_probabilities
```

El ejemplo anterior simula un circuito Bell en un simulador de CPU. Este ejemplo se ejecuta localmente en su portátil o en el cuaderno de Jupyter de Braket. Debido a esta configuración de `local=True`, cuando ejecute este script, se iniciará un contenedor en su entorno local para ejecutar el programa CUDA-Q con fines de prueba y depuración. Cuando termine de realizar las pruebas, puede quitar el indicador `local=True` y continuar con su trabajo en AWS. Para obtener más información, consulte Trabajar con trabajos híbridos de Amazon Braket.

Si sus cargas de trabajo tienen un recuento elevado de qubits, un gran número de circuitos o un gran número de iteraciones, puede utilizar recursos de computación de CPU más potentes especificando la configuración `instance_config`. El siguiente fragmento de código muestra cómo configurar el ajuste `instance_config` en el decorador de `hybrid_job`. Para obtener más información sobre los tipos de instancias compatibles, consulte Configuración de su instancia de trabajo híbrido. Para obtener la lista de los tipos de instancia, consulte Tipos de instancias de Amazon EC2.

```
@hybrid_job(
    device="local:nvidia/qpp-cpu",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.c5.2xlarge"),
)
def my_job_script():
    ...
```

Para cargas de trabajo más exigentes, puede ejecutar sus cargas de trabajo en un simulador de GPU de CUDA-Q. Para habilitar un simulador de GPU, use el nombre de backend `nvidia`. El backend `nvidia` funciona como un simulador GPU de CUDA-Q. A continuación, seleccione un tipo

de instancia de Amazon EC2 que sea compatible con una GPU de NVIDIA. El siguiente fragmento de código muestra el decorador de `hybrid_job` configurado para la GPU.

```
@hybrid_job(
    device="local:nvidia/nvidia",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"),
)
def my_job_script():
    ...
```

Amazon Braket Hybrid Jobs y admite simulaciones de GPU NBIs paralelas con. CUDA-Q Puede paralelizar la evaluación de varias observables o varios circuitos para mejorar el rendimiento de su carga de trabajo. Para paralelizar varios observables, realice los siguientes cambios en el script de algoritmo.

Configura la opción `mgpu` del backend `nvidia`. Esto es necesario para paralelizar los observables. La paralelización utiliza el MPI para la comunicación entre ellas GPUs, por lo que el MPI debe inicializarse antes de la ejecución y finalizarse después de la ejecución.

A continuación, especifique el modo de ejecución mediante la configuración de `execution=cudaq.parallel.mpi`. El siguiente fragmento de código muestra estos cambios.

```
cudaq.set_target("nvidia", option="mqpu")
cudaq.mpi.initialize()
result = cudaq.observe(
    kernel, hamiltonian, shots_count=n_shots, execution=cudaq.parallel.mpi
)
cudaq.mpi.finalize()
```

En el `hybrid_job` decorador, especifique un tipo de instancia que aloje múltiples GPUs , como se muestra en el siguiente fragmento de código.

```
@hybrid_job(
    device="local:nvidia/nvidia-mqpu",
    instance_config=InstanceConfig(instanceType="ml.g4dn.12xlarge", instanceCount=1),
    image_uri=image_uri,
)
def parallel_observables_gpu_job(sagemaker_mpi_enabled=True):
    ...
```

El cuaderno de simulaciones paralelas de Amazon Braket, Github, proporciona end-to-end ejemplos que demuestran cómo ejecutar simulaciones de programas cuánticos en backends de GPU y realizar simulaciones paralelas de observables y lotes de circuitos.

Ejecución de sus cargas de trabajo en computadoras cuánticas

Tras completar las pruebas con el simulador, puede pasar a la ejecución de experimentos en él QPUs. Simplemente cambie el destino a una QPU de Amazon Braket, como los dispositivos IQM, IonQ o Rigetti. El siguiente fragmento de código ilustra cómo establecer el objetivo en el dispositivo IQM Garnet. Para ver una lista de las disponibles QPUs, consulta la consola Amazon Braket.

```
device_arn = "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet"
cudaq.set_target("braket", machine=device_arn)
```

Para obtener más información sobre trabajos híbridos, consulte Trabajar con trabajos híbridos de Amazon Braket en la guía para desarrolladores. Para obtener más información sobre CUDA-Q, consulte la documentación de NVIDIA CUDA-Q.

Solución de problemas con Amazon Braket

Use la información de solución de problemas y las soluciones de esta sección para resolver problemas que pueden presentarse con Amazon Braket.

En esta sección:

AccessDeniedException

Si recibes una AccessDeniedExceptional activar o utilizar Braket, es probable que estés intentando activar o utilizar Braket en una región a la que tu función restringida no tiene acceso.

En esos casos, ponte en contacto con tu AWS administrador interno para saber cuáles de las siguientes condiciones se aplican:

- Si hay restricciones de rol que impidan el acceso a una región.
- Si el rol que intenta usar tiene permiso para usar Braket.

Si su rol no tiene acceso a una región determinada cuando usa Braket, no podrá usar los dispositivos de esa región en particular.

Se ha producido un error (ValidationException) al llamar a la CreateQuantumTask operación

Si recibe un error similar a: `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-`

`braket-…` , compruebe que está haciendo referencia a una carpeta s3_existente. Braket no crea automáticamente nuevos buckets y prefijos de Amazon S3.

Si está accediendo a la API directamente y recibe un error similar a: `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET`, compruebe que no está incluyendo `s3://` en la ruta del bucket de Amazon S3.

Una característica del SDK no funciona

La versión de Python debe ser 3.10 o superior. Para Amazon Braket Hybrid Jobs, recomendamos Python 3.12.

Compruebe que su SDK y sus esquemas lo sean. up-to-date Para actualizar el SDK desde el cuaderno o desde el editor de Python, ejecute el siguiente comando:

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Para actualizar los esquemas, ejecute el siguiente comando:

```
pip install amazon-braket-schemas --upgrade
```

Si accede a Amazon Braket desde su propio cliente, verifique que su región de AWS esté establecida como una región compatible con Amazon Braket.

El trabajo híbrido falla debido a ServiceQuotaExceededException

Es posible que no se cree un trabajo híbrido que ejecute tareas cuánticas en los simuladores de Amazon Braket si supera el límite de tareas cuánticas simultáneas del dispositivo simulador al que se dirige. Para obtener más información sobre los límites, consulte el tema Cuotas.

Si ejecuta tareas simultáneas en un dispositivo simulador en varios trabajos híbridos desde su cuenta, podría producirse este error.

Para ver el número de tareas cuánticas simultáneas en un dispositivo simulador específico, use la API de `search-quantum-tasks`, como se muestra en el siguiente ejemplo de código.

```
DEVICE_ARN=arn:aws:braket:::device/quantum-simulator/amazon/sv1
task_list=""
```

```
for status_value in "CREATED" "QUEUED" "RUNNING" "CANCELLING"; do
    tasks=$(aws braket search-quantum-tasks --filters
 name=status,operator=EQUAL,values=${status_value}
 name=deviceArn,operator=EQUAL,values=$DEVICE_ARN --max-results 100 --query
 'quantumTasks[*].quantumTaskArn' --output text)
    task_list="$task_list $tasks"
done;
echo "$task_list" | tr -s ' \t' '[\n*]' | sort | uniq
```

También puedes ver las tareas cuánticas creadas comparándolas con un dispositivo mediante CloudWatch las métricas de Amazon: Braket > Por dispositivo.

Para evitar que se produzcan estos errores:

1. Solicite un aumento de cuota de servicio para el número de tareas cuánticas simultáneas del dispositivo simulador. Esto solo se aplica al dispositivo SV1.
2. Gestione las excepciones de `ServiceQuotaExceeded` en su código y vuelva a intentarlo.

Los componentes dejaron de funcionar en una instancia de cuaderno

Si algunos componentes de su cuaderno dejan de funcionar, pruebe lo siguiente:

1. Descargue todos los cuadernos que haya creado o modificado en una unidad local.
2. Detenga la instancia de cuaderno.
3. Elimine la instancia de cuaderno.
4. Cree una nueva instancia de cuaderno con un nombre diferente.
5. Suba los cuadernos a la nueva instancia.

Solución de problemas de la actualización a Python 3.12

Fecha de entrada en vigor: 21 de enero de 2026

Descripción general de

A partir del 21 de enero de 2026, Amazon Braket actualizará el entorno de ejecución de Python de la versión 3.10 a la 3.12 para todas las instancias de Notebook y las imágenes de contenedores

gestionadas (Base, CUDA-Q y). TensorFlow PyTorch Esta guía proporciona soluciones para los problemas de compatibilidad más comunes.

En esta sección:

Mensajes de error comunes

Error de discordancia de la versión de Python del SDK

Error:

```
RuntimeError: Python version must match between local environment and container. Client
 is running Python 3.10 locally, but container uses Python 3.12.
```

Causa: el SDK de Braket detectó que tu portátil ejecuta Python 3.10, pero el contenedor Hybrid Job ejecuta Python 3.12.

Solución: actualice su bloc de notas a Python 3.12 o fíjelo a contenedores de Python 3.10.

Error de serialización de Cloudpickle

Error:

```
TypeError: code() argument 13 must be str, not int
```

Causa: si se omite la validación del SDK, cloudpickle no puede serializar el código entre Python 3.10 y 3.12 debido a un cambio de constructor CodeType en Python 3.12.

Solución: asegúrese de que el bloc de notas y el contenedor utilicen la misma versión de Python.

Cuadernos gestionados por Braket

Si ejecuta una instancia de Braket Notebook en Python 3.10 y envía trabajos híbridos, se producirán errores de desajuste de versiones porque los contenedores de trabajos ahora usan Python 3.12 de forma predeterminada.

Tiene dos opciones:

1. [Recomendado] Cree una nueva instancia de Notebook con Python 3.12; consulte Actualización de instancias de Braket Notebook
2. Fije a los contenedores de Python 3.10; consulte Hybrid Job Decorator

Decorador Hybrid Job

Para usar el `@hybrid_job` decorador, la versión de Python de su entorno debe coincidir con la versión de Python del contenedor.

Opción 1: usar contenedores de Python 3.12 (recomendado)

Si ha actualizado su entorno a Python 3.12, utilizará la última etiqueta (comportamiento predeterminado).

Opción 2: usar contenedores de Python 3.10

Si debe permanecer en Python 3.10, especifique explícitamente el `image_uri` parámetro en el `@hybrid_job` decorador.

Etiquetas de contenedor de Python 3.10:

Nombre de la imagen	Tag
Base	1.0-cpu-py310-ubuntu22.04
CUDA-Q	0.12.0-cpu-py310-0.12.0
PyTorch	2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow	2.14.1-gpu-py310-cu118-ubuntu20.04

En el siguiente ejemplo es para la región us-west-2.

Imagen completa: URIs

```
Base:       292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-
cpu-py310-ubuntu22.04
CUDA-Q:     292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-
jobs:0.12.0-cpu-py310-0.12.0
PyTorch:    292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-
jobs:2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow: 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-
jobs:2.14.1-gpu-py310-cu118-ubuntu20.04
```

Ejemplo:

```
from braket.jobs.hybrid_job import hybrid_job
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

@hybrid_job(
    device=device_arn,
    image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-
jobs:1.0-cpu-py310-ubuntu22.04"
)
def my_job():
    pass
```

> **Note**
>
> - Los contenedores de Python 3.10 seguirán disponibles pero no recibirán actualizaciones.
> - Consulte Definir el entorno para el script de su algoritmo.

Bring-Your-Own-Container (BYOC)

Si tu Dockerfile usa una imagen gestionada por Braket con la etiqueta más reciente, si se reconstruye después del 21 de enero de 2026, se obtendrán imágenes compatibles con Python 3.12.

Para seguir utilizando las imágenes gestionadas por Braket compatibles con Python 3.10, actualiza tu Dockerfile:

Antes (obtiene Python 3.12 después de la actualización):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:latest
```

Después (permanece en Python 3.10):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-
py310-ubuntu22.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-
py310-0.12.0
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:2.2.0-gpu-
py310-cu121-ubuntu20.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:2.14.1-
gpu-py310-cu118-ubuntu20.04
```

Actualización de la instancia de Braket Notebook

Siga estos pasos para actualizar a Python 3.12:

> **Important**
>
> Antes de eliminar su instancia de bloc de notas, asegúrese de haber descargado todos los blocs de notas y archivos que desee conservar. Estos datos no se pueden recuperar después de eliminarlos.

1. Descargue todos los cuadernos que haya creado o modificado en una unidad local.
2. Detenga la instancia de cuaderno.
3. Elimine la instancia de cuaderno.
4. Cree una nueva instancia de bloc de notas con un nombre diferente.
5. Sube tus libretas a la nueva instancia.

Solución de problemas de OpenQASM

En esta sección se proporcionan consejos de solución de problemas que pueden resultar útiles cuando se producen errores al utilizar OpenQASM 3.0.

En esta sección:

Incluir un error en la declaración

Actualmente, Braket no tiene un archivo de biblioteca de puerta estándar para incluirlo en los programas de OpenQASM. Por ejemplo, en el siguiente ejemplo se genera un error de analizador.

```
OPENQASM 3;
include "standardlib.inc";
```

Este código genera el mensaje de error: `No terminal matches '"' in the current parser context, at line 2 col 17.`

Error de qubits no contiguo

El uso de qubits no contiguo en dispositivos no contiguos en dispositivos que `requiresContiguousQubitIndices` establecido en `true` en la capacidad del dispositivo provoca un error.

Al ejecutar tareas cuánticas en simuladores y IonQ, el siguiente programa desencadena el error.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

Este código genera el mensaje de error: `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

Error de combinación de qubits físico con qubits virtual

No se permite la combinación de qubits físico con qubits virtual en el mismo programa y se produce un error. El código siguiente genera el error.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

Este código genera el mensaje de error: `[line 4] mixes physical qubits and qubits registers.`

Error de solicitud de tipos de resultados y medición de qubits en el mismo programa

La solicitud de tipos de resultados y la medición explícita de qubits en el mismo programa, da lugar a un error. El código siguiente genera el error.

```
OPENQASM 3;

qubit[2] q;

h q[0];
cnot q[0], q[1];
measure q;
```

```
#pragma braket result expectation x(q[0]) @ z(q[1])
```

Este código genera el mensaje de error: `Qubits should not be explicitly measured when result types are requested.`

Error de superación de límites en registro clásico y registro de qubit

Solo se permite un registro clásico y un registro de qubit. El código siguiente genera el error.

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

Este código genera el mensaje de error: `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

Error de cuadro no precedido de un pragma verbatim

Todos los cuadros deben ir precedidos de un pragma verbatim. El código siguiente genera el error.

```
box{
rx(0.5) $0;
}
```

Este código genera el mensaje de error: `In verbatim boxes, native gates are required. x is not a device native gate.`

Error de falta de puertas nativas en los cuadros verbatim

Los cuadros verbatim deben tener puertas nativas y qubits físicos. El siguiente código genera el error de puertas nativas.

```
#pragma braket verbatim
box{
x $0;
}
```

Este código genera el mensaje de error: `In verbatim boxes, native gates are required. x is not a device native gate.`

Error de falta de qubits físicos en cuadros verbatim

Los cuadros verbatim deben tener qubits físicos. El código siguiente genera el error de falta de qubits físicos.

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

Este código genera el mensaje de error: `Physical qubits are required in verbatim box.`

Error de falta de «braket» en el pragma verbatim

Debe incluir «braket» en el pragma verbatim. El código siguiente genera el error.

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

Este código genera el mensaje de error: `You must include “braket” in the verbatim pragma`

Error de que no es posible indexar un solo qubits

No se puede indexar un solo qubits. El código siguiente genera el error.

```
OPENQASM 3;

qubit q;
h q[0];
```

Este código genera el error: `[line 4] single qubit cannot be indexed.`

Sin embargo, las matrices de qubit individuales se pueden indexar como sigue:

```
OPENQASM 3;
```

```
qubit[1] q;
h q[0];   // This is valid
```

Error de qubits físicos en una puerta de dos qubit no conectados

Para utilizar qubits físicos, confirme primero que el dispositivo utiliza qubits físicos marcando `device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits` y, a continuación, compruebe el gráfico de conectividad marcando `device.properties.paradigm.connectivity.connectivityGraph` o `device.properties.paradigm.connectivity.fullyConnected`.

```
OPENQASM 3;

cnot $0, $14;
```

Este código genera el mensaje de error: `[line 3] has disconnected qubits 0 and 14`

Advertencia de compatibilidad del simulador local

`LocalSimulator`Es compatible con las funciones avanzadas de OpenQASM que pueden no estar disponibles en los simuladores o bajo demanda. QPUs Si su programa contiene características de lenguaje específicas únicamente para el `LocalSimulator`, como se ve en el siguiente ejemplo, recibirá una advertencia.

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

Este código genera la siguiente advertencia: `Este programa utiliza funciones del lenguaje OpenQASM que solo son compatibles con. LocalSimulator Es posible que algunas de estas funciones no sean compatibles con los simuladores o bajo demanda. QPUs

Para obtener más información sobre las características compatibles con OpenQASM, consulte la página Compatibilidad con características avanzadas para OpenQASM en el simulador local.

Seguridad en Amazon Braket

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de los centros de datos y las arquitecturas de red diseñados para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre AWS usted y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta AWS los servicios en la Nube de AWS. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener información sobre los programas de conformidad que se aplican a Amazon Braket, consulte AWS Servicios incluidos en el ámbito de aplicación por programa de conformidad AWS Servicios en el ámbito de aplicación por programa .
- Seguridad en la nube: su responsabilidad viene determinada por el AWS servicio que utilice. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos vigentes.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida a la hora de utilizar Braket. Los siguientes temas le mostrarán cómo configurar Braket para satisfacer sus objetivos de seguridad y de conformidad. También aprenderá a utilizar otros AWS servicios que le ayudan a supervisar y proteger sus recursos de Braket.

En esta sección:

Responsabilidad compartida en materia de seguridad

La seguridad es una responsabilidad compartida entre usted AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que se ejecuta Servicios de AWS en la Nube de AWS. AWS también le proporciona servicios que puede utilizar de forma segura. Auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad en el marco de los programas de conformidad de AWS. Para obtener información acerca de los programas de conformidad que se aplican a Amazon Braket, consulte Servicios de AWS en el ámbito del programa de conformidad.
- Seguridad en la nube: usted es responsable de mantener el control sobre el contenido alojado en esta AWS infraestructura. Este contenido incluye las tareas de configuración y administración de la seguridad Servicios de AWS que utilices.

Protección de datos

El modelo de responsabilidad AWS compartida modelo se aplica a la protección de datos en Amazon Braket. Como se describe en este modelo, AWS es responsable de proteger la infraestructura global en la que se ejecutan todos los Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y GDPR en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Exigimos TLS 1.2 y recomendamos TLS 1.3.

- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte Cómo trabajar con CloudTrail senderos en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados que contienen Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger la información confidencial almacenada en Amazon S3.
- Si necesita módulos criptográficos validados por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-3.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye cuando trabajas con Amazon Braket u otro dispositivo Servicios de AWS mediante la consola, la API o. AWS CLI AWS SDKs Cualquier dato que introduzca en etiquetas o campos de formato libre utilizados para los nombres se pueden emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Retención de datos

Transcurridos 90 días, Amazon Braket elimina automáticamente todas las tareas cuánticas IDs y otros metadatos asociados a sus tareas cuánticas. Como resultado de esta política de retención de datos, estas tareas y resultados ya no se pueden recuperar mediante una búsqueda en la consola de Amazon Braket, aunque permanecen almacenados en su bucket de S3.

Si necesita acceder a tareas cuánticas históricas y resultados que se almacenan en su bucket de S3 durante más de 90 días, debe mantener un registro independiente de su ID de tarea y otros metadatos asociados a esos datos. Asegúrese de guardar la información anterior a los 90 días. Puede utilizar la información guardada para recuperar los datos históricos.

Administración del acceso a Amazon Braket

En este capítulo se describen los permisos necesarios para ejecutar Amazon Braket o para restringir el acceso de usuarios y roles específicos. Puede conceder (o denegar) los permisos necesarios a cualquier usuario o rol de su cuenta. Para ello, asocie la política de Amazon Braket adecuada a ese usuario o rol en su cuenta, tal y como se describe en las siguientes secciones.

Como requisito previo, debe activar Amazon Braket. Para activar Braket, asegúrese de iniciar sesión como un usuario o rol que tenga (1) permisos de administrador o (2) que tenga asignada la AmazonBraketFullAccesspolítica y permisos para crear buckets de Amazon Simple Storage Service (Amazon S3).

En esta sección:

Recursos de Amazon Braket

Braket crea un tipo de recurso: el recurso de tareas cuánticas. El nombre del AWS recurso (ARN) de este tipo de recurso es el siguiente:

- Nombre del recurso: AWS::Service::Braket
- Régex de ARN: ARN: $ {Partition} :braket: $ {Region} :$ {Account} :quantum-task/$ {} RandomId

Cuadernos y roles

Puede utilizar el tipo de recurso cuaderno en Braket. Una libreta es un recurso de Amazon SageMaker AI que Braket puede compartir. Para usar un cuaderno con Braket, debe especificar un rol de IAM con un nombre que comience por `AmazonBraketServiceSageMakerNotebook`.

Para crear un cuaderno, debe utilizar un rol con permisos de administrador o que tenga asociada la siguiente política correspondiente.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CreateTheRole",
      "Effect": "Allow",
      "Action": "iam:CreateRole",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
    },
    {
      "Sid": "CreateThePolicy",
      "Effect": "Allow",
      "Action": "iam:CreatePolicy",
      "Resource": [
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
      ]
    },
    {
      "Sid": "AttachTheRolePolicy",
      "Effect": "Allow",
      "Action": "iam:AttachRolePolicy",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
      "Condition": {
        "ArnLike": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/AmazonBraketFullAccess",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
          ]
        }
      }
    }
  ]
```

```
}
```

Para crear el rol, siga los pasos que se indican en la página Create a notebook o pida a su administrador que lo cree por usted. Asegúrese de que la AmazonBraketFullAccesspolítica esté adjunta.

Una vez que haya creado el rol, podrá reutilizarlo en todos los cuadernos que lance en el futuro.

AWS políticas gestionadas para Amazon Braket

Una política AWS gestionada es una política independiente creada y administrada por AWS. AWS Las políticas administradas están diseñadas para proporcionar permisos para muchos casos de uso comunes, de modo que pueda empezar a asignar permisos a usuarios, grupos y funciones.

Ten en cuenta que es posible que las políticas AWS administradas no otorguen permisos con privilegios mínimos para tus casos de uso específicos, ya que están disponibles para que los usen todos los AWS clientes. Se recomienda definir políticas administradas por el cliente específicas para sus casos de uso a fin de reducir aún más los permisos.

No puedes cambiar los permisos definidos en AWS las políticas administradas. Si AWS actualiza los permisos definidos en una política AWS administrada, la actualización afecta a todas las identidades principales (usuarios, grupos y roles) a las que está asociada la política. AWS es más probable que actualice una política AWS administrada cuando Servicio de AWS se lance una nueva o cuando estén disponibles nuevas operaciones de API para los servicios existentes.

Para obtener más información, consulte Políticas administradas por AWS en la Guía del usuario de IAM.

Temas

- AWS política gestionada: AmazonBraketFullAccess
- AWS política gestionada: AmazonBraketJobsExecutionPolicy
- AWS política gestionada: AmazonBraketServiceRolePolicy
- Amazon Braket actualiza las políticas gestionadas AWS

AWS política gestionada: AmazonBraketFullAccess

La AmazonBraketFullAccesspolítica concede permisos para las operaciones de Amazon Braket, incluidos los permisos para las siguientes tareas:

- Descargar contenedores de Amazon Elastic Container Registry: para leer y descargar imágenes de contenedor que se utilizan para la característica de trabajos híbridos de Amazon Braket. Los contenedores deben cumplir el formato «arn:aws:ecr:::repository/amazon-braket».
- Mantenga AWS CloudTrail registros: para todas las acciones de descripción, obtención y lista, además de iniciar y detener consultas, probar los filtros de métricas y filtrar los eventos de registro. El archivo de AWS CloudTrail registro contiene un registro de toda la API actividad de Amazon Braket que se produce en tu cuenta.
- Utilizar roles de control de recursos: para crear un rol vinculado a servicios en su cuenta. El rol vinculado al servicio tiene acceso a AWS los recursos en su nombre. Solo puede ser utilizado por el servicio de Amazon Braket. Además, transferir las funciones de IAM a Amazon `CreateJob` API Braket y crear una función y adjuntar una política relacionada con la AmazonBraketFullAccess función.
- Crear grupos de registro, eventos de registro y consultar grupos de registro para mantener los archivos de registro de uso de su cuenta: para crear, almacenar y ver información de registro sobre el uso de Amazon Braket en su cuenta. Consulte las métricas de los grupos de registro de trabajos híbridos. Incluya la ruta correcta de Braket y permita la introducción de datos de registro. Introduce los datos de las métricas. CloudWatch
- Crear y almacenar datos en buckets de Amazon S3, y listar todos los buckets: para crear buckets de S3, listar los buckets de S3 de su cuenta y colocar objetos en cualquier bucket de su cuenta cuyo nombre comience por amazon-braket-, así como obtener objetos de dichos buckets. Estos permisos son necesarios para que Braket pueda colocar los archivos que contienen los resultados de las tareas cuánticas procesadas en el bucket y recuperarlos de él.
- Transferir funciones de IAM: para transferir los roles de IAM a la API de `CreateJob`.
- Amazon SageMaker AI Notebook: para crear y gestionar instancias de SageMaker bloc de notas relacionadas con el recurso de «arn:aws:sagemaker: ::notebook-instance/amazon-braket-».
- Valide las cuotas de servicio: para crear cuadernos de SageMaker IA y trabajos híbridos de Amazon Braket, sus recuentos de recursos no pueden superar las cuotas de su cuenta.
- Consultar los precios de los productos: revise y planifique los costos del hardware cuántico antes de enviar sus cargas de trabajo.

Para ver los permisos de esta política, consulte la Referencia AmazonBraketFullAccessde políticas administradas de AWS.

AWS política gestionada: AmazonBraketJobsExecutionPolicy

La AmazonBraketJobsExecutionPolicypolítica concede permisos para las funciones de ejecución utilizadas en Amazon Braket Hybrid Jobs de la siguiente manera:

- Descargar contenedores de Amazon Elastic Container Registry: permisos para leer y descargar imágenes de contenedor que se utilizan para la característica de trabajos híbridos de Amazon Braket. Los contenedores deben cumplir el formato «arn:aws:ecr:*:*:repository/amazon-braket*».
- Crear grupos de registro, eventos de registro y consultar grupos de registro para mantener los archivos de registro de uso de su cuenta: cree, almacene y vea información de registro sobre el uso de Amazon Braket en su cuenta. Consulte las métricas de los grupos de registro de trabajos híbridos. Incluya la ruta correcta de Braket y permita la introducción de datos de registro. Introduzca los datos de las métricas. CloudWatch
- Almacenar datos en buckets de Amazon S3: liste los buckets de S3 de su cuenta, coloque objetos en cualquier bucket de su cuenta cuyo nombre comience por amazon-braket- y obtenga objetos de dichos buckets. Estos permisos son necesarios para que Braket pueda colocar los archivos que contienen los resultados de las tareas cuánticas procesadas en el bucket y recuperarlos de él.
- Transferir las funciones de IAM: transferir las funciones de IAM a. CreateJob API Las funciones deben cumplir el formato arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*.

Para ver los permisos de esta política, consulte la Referencia AmazonBraketJobsExecutionPolicyde políticas administradas de AWS.

AWS política gestionada: AmazonBraketServiceRolePolicy

La AmazonBraketServiceRolePolicypolítica concede permisos para las operaciones de Amazon Braket, incluidos los permisos para las siguientes tareas:

- Amazon S3: permisos para listar los buckets de su cuenta y colocar objetos en cualquier bucket de su cuenta cuyo nombre comience por `amazon-braket-`, así como obtener objetos de dichos buckets.
- Amazon CloudWatch Logs: permisos para enumerar y crear grupos de registros, crear los flujos de registro asociados y colocar eventos en el grupo de registros creado para Amazon Braket.

Para obtener más información sobre los roles vinculados a servicios, consulte Rol vinculado a servicios de Amazon Braket.

Para ver los permisos de esta política, consulte la Referencia AmazonBraketServiceRolePolicyde políticas administradas de AWS.

Amazon Braket actualiza las políticas gestionadas AWS

En la siguiente tabla se proporcionan detalles sobre las actualizaciones de las políticas AWS gestionadas de Amazon Braket desde el momento en que este servicio comenzó a realizar el seguimiento de estos cambios.

Cambio	Descripción	Fecha
AmazonBraketServiceRolePolicy- Política de administración de recursos	Se agregó el ámbito de condición «aws:ResourceAccount": «$ {aws:PrincipalAccount}» a Amazon S3 y CloudWatch registra las acciones.	11 de julio de 2025
AmazonBraketFullAccess- Política de acceso completo a Braket	Se agregó la acción «fijación de precios:GetProducts».	14 de abril de 2025
AmazonBraketFullAccess- Política de acceso completo a Braket	Se agregó el ámbito de condición ResourceAccount«aws: «: «$ {aws:PrincipalAccount}» a las acciones de S3.	7 de marzo de 2025
AmazonBraketFullAccess- Política de acceso completo a Braket	Se agregaron las acciones servicequota GetServiceQuota y cloudwatch. GetMetricData	24 de marzo de 2023
AmazonBraketFullAccess- Política de acceso completo a Braket	Se agregaron los ListAllMy Buckets permisos s3: para ver e inspeccionar los buckets de Amazon S3 usados.	31 de marzo de 2022
AmazonBraketFullAccess- Política de acceso completo para Braket	Objetivo ajustado por corchetes: PassRole permisos AmazonBra	29 de noviembre de 2021

Cambio	Descripción	Fecha
	ketFullAccess para incluir la ruta. `service-role/`	
AmazonBraketJobsExecutionPolicy- Política de ejecución de trabajos híbridos para Amazon Braket Hybrid Jobs	Braket actualizó el ARN del rol de ejecución de trabajos híbridos para incluir la ruta de `service-role/` .	29 de noviembre de 2021
Braket comenzó el seguimiento de los cambios	Braket comenzó a realizar un seguimiento de los cambios en sus políticas AWS gestionadas.	29 de noviembre de 2021

Restringir el acceso de los usuarios a determinados dispositivos

Para restringir el acceso de los usuarios a determinados dispositivos de Braket, puede añadir una política de denegación de permisos a un rol de IAM específico.

Se pueden restringir las siguientes acciones:

- `CreateQuantumTask`: denegar la creación de tareas cuánticas en dispositivos específicos.
- `CreateJob`: denegar la creación de trabajos híbridos en dispositivos específicos.
- `GetDevice`: denegar la obtención de detalles de dispositivos específicos.

El siguiente ejemplo restringe el acceso a todos QPUs los. Cuenta de AWS `123456789012`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
```

```
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ],
    "Condition": {
            "StringEquals": {
                "aws:PrincipalAccount": "123456789012"
            }
      }
    }
  ]
}
```

> **Note**
>
> Excluya la acción `braket:GetDevice` de la política para permitir el acceso de lectura de un usuario a las propiedades del dispositivo, como la disponibilidad del dispositivo, los datos de calibración y los precios a través de la consola de Braket.

Para adaptar este código, sustituya el número de recurso de Amazon (ARN) del dispositivo restringido por la cadena que se muestra en el ejemplo anterior. Esta cadena proporciona el valor Recurso. En Braket, un dispositivo representa una QPU o un simulador al que puede llamar para ejecutar tareas cuánticas. Los dispositivos disponibles aparecen en la página de dispositivos. Se utilizan dos esquemas para especificar el acceso a estos dispositivos:

- `arn:aws:braket:<region>:*:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:*:device/quantum-simulator/<provider>/<device_id>`

A continuación se ofrecen ejemplos de varios tipos de acceso a dispositivos.

- Para seleccionarlos todos QPUs en todas las regiones: `arn:aws:braket:*:*:device/qpu/*`
- Para seleccionar ÚNICAMENTE todo QPUs en la región us-west-2: `arn:aws:braket:us-west-2:*:device/qpu/*`
- Del mismo modo, para seleccionar ÚNICAMENTE todo QPUs en la región us-west-2 (ya que los dispositivos son un recurso de servicio, no un recurso de cliente): `arn:aws:braket:us-west-2:*:device/qpu/*`

- Para restringir el acceso a todos los dispositivos de simulador bajo demanda: `arn:aws:braket:*:*:device/quantum-simulator/*`
- Para restringir el acceso a los dispositivos de un proveedor determinado (por ejemplo, a los dispositivos QPU de Rigetti): `arn:aws:braket:*:*:device/qpu/rigetti/*`
- Para restringir el acceso al dispositivo TN1: `arn:aws:braket:*:*:device/quantum-simulator/amazon/tn1`
- Para restringir el acceso a todas las acciones de `Create`: `braket:Create*`

Restringir el acceso de los usuarios a determinadas instancias de cuaderno

Para restringir el acceso de determinados usuarios a instancias específicas de cuaderno de Braket, puede añadir una política de denegación de permisos a un rol, usuario o grupo específicos.

En el siguiente ejemplo, se utilizan variables de política para restringir de forma eficaz los permisos para iniciar, detener y acceder a instancias específicas del bloc de notas en el Cuenta de AWS `123456789012`, que se denomina según el usuario que debería tener acceso (por ejemplo, el usuario `Alice` tendría acceso a una instancia de bloc de notas denominada). `amazon-braket-Alice`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCreateDeleteUpdateNotebookInstances",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyDescribeStartStopNotebookInstances",
```

```
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Sid": "DenyNotebookInstanceUrl",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

Restringir el acceso de los usuarios a determinados buckets de S3

Para restringir el acceso de determinados usuarios a buckets específicos de Amazon S3, puede añadir una política de denegación a un rol, usuario o grupo específicos.

El siguiente ejemplo restringe los permisos para recuperar objetos y colocarlos en un bucket de S3 (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) específico, y también restringe el listado de esos objetos.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
```

```
      "Action": [
        "s3:ListBucket"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "s3:GetObject"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice/*"
      ]
    }
  ]
}
```

Para restringir el acceso al bucket de una instancia de cuaderno específica, puede añadir la política anterior al rol de ejecución del cuaderno.

Rol vinculado a servicios de Amazon Braket

Cuando activa Amazon Braket, se crea un rol vinculado a servicios en su cuenta.

Un rol vinculado a servicios es un tipo único de rol de IAM que, en este caso, se encuentra vinculado directamente a Amazon Braket. El rol vinculado a servicios de Amazon Braket está predefinido para incluir todos los permisos que Braket necesita cuando realiza llamadas en su nombre.

Un rol vinculado a servicios facilita la configuración de Amazon Braket, ya que no es necesario añadir los permisos necesarios manualmente. Amazon Braket define los permisos de sus roles vinculados a servicios. A menos que cambie estas definiciones, solo Amazon Braket puede asumir sus roles. Los permisos definidos incluyen la política de confianza y la política de permisos. La política de permisos no se puede asociar a ninguna otra entidad de IAM.

El rol vinculado a servicios que configura Amazon Braket forma parte de la capacidad de los roles vinculados a servicios de AWS Identity and Access Management (IAM). Para obtener información sobre otros Servicios de AWS que admiten roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM y busque los servicios que indiquen Sí en la columna Roles vinculados

a servicios. Seleccione una opción Sí con un enlace para ver la documentación acerca del rol vinculado al servicio en cuestión.

Para obtener más información sobre la política administrada de AWS para los roles vinculados a servicios, consulte AmazonBraketServiceRolePolicy.

Validación de conformidad para Amazon Braket

> **Note**
>
> Los informes de conformidad de AWS no incluyen las QPU de proveedores externos de hardware que pueden optar por someterse a sus propias auditorías independientes.

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte Servicios de AWS incluidos por programa de conformidad y escoja el programa de conformidad que le interese. Para obtener información general, consulte Programas de conformidad de AWS.

Puedes descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la confidencialidad de los datos, los objetivos de conformidad de su empresa, así como de la legislación y los reglamentos aplicables. Para obtener más información sobre la responsabilidad de conformidad al usar Servicios de AWS, consulte la Documentación de seguridad de AWS.

Seguridad de la infraestructura en Amazon Braket

Como servicio administrado, Amazon Braket se encuentra protegido por la seguridad de red global de AWS. Para obtener información sobre los servicios de seguridad de AWS y sobre cómo AWS protege la infraestructura, consulte Seguridad en la nube de AWS. Para diseñar su entorno de AWS siguiendo las prácticas recomendadas de seguridad de infraestructura, consulte Protección de la infraestructura en Portal de seguridad de AWS Well-Architected Framework.

Puede utilizar llamadas a la API publicadas en AWS para acceder a Amazon Braket a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Puede llamar a estas operaciones de la API desde cualquier ubicación de red, pero Braket admite políticas de acceso basadas en recursos, que pueden incluir restricciones en función de la dirección IP de origen. También puede utilizar políticas de Braket para controlar el acceso desde puntos de conexión específicos de Amazon Virtual Private Cloud (Amazon VPC) o VPC específicas. Este proceso aísla con eficacia el acceso de red a un recurso de Braket determinado únicamente desde la VPC específica de la red de AWS.

Seguridad de los proveedores de hardware de Amazon Braket

Las QPU de Amazon Braket están alojadas por proveedores externos de hardware. Cuando ejecuta una tarea cuántica en una QPU, Amazon Braket utiliza el DeviceARN como identificador al enviar el circuito a la QPU especificada para su procesamiento.

Si utiliza Amazon Braket para acceder al hardware de computación cuántica operado por uno de los proveedores externos de hardware, tu circuito y sus datos asociados son procesados por proveedores de hardware ajenos a las instalaciones operadas por AWS. La información sobre la ubicación física y la región de AWS en la que está disponible cada QPU se encuentra disponible en la sección Detalles del dispositivo de la consola de Amazon Braket.

Su contenido está anonimizado. Solo el contenido necesario para procesar el circuito se envía a terceros. La información de Cuenta de AWS no se transmite a terceros.

Todos los datos se cifran en reposo y en tránsito. Los datos solo se descifran para el procesamiento. Los proveedores externos de Amazon Braket no están autorizados a almacenar ni utilizar su contenido para otros fines que no sean el procesamiento de su circuito. Una vez que se completa el circuito, los resultados se devuelven a Amazon Braket y se almacenan en el bucket de S3.

La seguridad de los proveedores externos de hardware cuántico de Amazon Braket se audita periódicamente para garantizar que se cumplen los estándares de seguridad de la red, el control de acceso, la protección de datos y la seguridad física.

Puntos de conexión de Amazon VPC para Amazon Braket

Puede establecer una conexión privada entre su VPC y Amazon Braket mediante la creación de un punto de conexión de VPC de interfaz. Los puntos de conexión de la interfaz funcionan con una tecnología que permite el acceso a Braket APIs sin necesidad de una pasarela de Internet, un dispositivo NAT, una conexión VPN o una conexión. AWS PrivateLink Direct Connect Las instancias de su VPC no necesitan direcciones IP públicas para comunicarse con Braket. APIs

Cada punto de conexión de la interfaz está representado por una o más interfaces de red elásticas en las subredes.

De este AWS PrivateLink modo, el tráfico entre su VPC y Braket no sale de la Amazon red, lo que aumenta la seguridad de los datos que comparte con las aplicaciones basadas en la nube, ya que reduce la exposición de sus datos a la Internet pública. Para obtener más información, consulte Acceder a un AWS servicio mediante un punto de enlace de VPC de interfaz en la Guía del usuario de Amazon VPC.

En esta sección:

- Consideraciones sobre los puntos de conexión de VPC de Amazon Braket
- Configure Braket y PrivateLink
- Información adicional sobre la creación de un punto de conexión
- Control del acceso con las políticas de puntos de conexión de Amazon VPC

Consideraciones sobre los puntos de conexión de VPC de Amazon Braket

Antes de configurar un punto de conexión de VPC de interfaz para Braket, asegúrese de revisar los requisitos previos del punto de conexión de interfaz en la guía del usuario de Amazon VPC.

Braket admite realizar llamadas a todas sus acciones de la API desde su VPC.

De forma predeterminada, el acceso completo a Braket se permite a través del punto de conexión de VPC. Puede controlar el acceso si especifica políticas de punto de conexión de VPC. Para obtener más información, consulte Controlar el acceso a los puntos de conexión de la VPC mediante políticas de puntos de conexión en la Guía de usuario de Amazon VPC.

Configure Braket y PrivateLink

Para usarlo AWS PrivateLink con Amazon Braket, debe crear un punto final de Amazon Virtual Private Cloud (Amazon VPC) como interfaz y, a continuación, conectarse al punto de enlace a través del servicio Amazon Braket. API

Estos son los pasos generales de este proceso, que se explican en detalle en secciones posteriores.

- Configure y lance una Amazon VPC para alojar sus AWS recursos. Si ya tiene una VPC, puede omitir este paso.
- Creación de un punto de conexión de VPC para Braket
- Conexión y ejecución de tareas cuánticas de Braket a través de su punto de conexión

Paso 1: Lanzar una Amazon VPC si es necesario

Recuerde que puede omitir este paso si su cuenta ya tiene una VPC en funcionamiento.

Una VPC controla la configuración de red, como el rango de direcciones IP, las subredes, la tabla de enrutamiento y las puertas de enlace de red. Básicamente, está lanzando sus AWS recursos en una red virtual personalizada. Para obtener más información VPCs, consulte la Guía del usuario de Amazon VPC.

Abra la consola Amazon VPC y cree una nueva VPC con subredes, grupos de seguridad y puertas de enlace de red.

Paso 2: Crear un punto de conexión de VPC para Braket

Puede crear un punto final de VPC para el servicio Braket mediante la consola Amazon VPC o el (). AWS Command Line Interface AWS CLI Para obtener más información, consulte Puntos de conexión de VPC en la Guía del usuario de Amazon VPC.

Para crear un punto de conexión de VPC en la consola, abra la consola de Amazon VPC, abra la página Puntos de conexión y proceda a crear el nuevo punto de conexión. Anote el ID del punto de conexión para consultarlo más adelante. Es obligatorio como parte del indicador `–endpoint-url` cuando se realizan determinadas llamadas a la API de Braket.

Cree un punto de conexión de VPC para Braket usando el siguiente nombre de servicio:

- `com.amazonaws.substitute_your_region.braket`

Para obtener más información, consulte Acceder a un AWS servicio mediante un punto de enlace de VPC de interfaz en la Guía del usuario de Amazon VPC.

Paso 3: Conectar y ejecutar tareas cuánticas de Braket a través de su punto de conexión

Después de crear un punto de conexión de VPC, puede ejecutar comandos de CLI que incluyan el parámetro `endpoint-url` para especificar puntos de conexión de interfaz a la API o el tiempo de ejecución, como en el siguiente ejemplo:

```
aws braket search-quantum-tasks --endpoint-url
 VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

Si habilita los nombres de host de DNS privados para su punto de conexión de VPC, no es necesario que especifique el punto de conexión como una URL en sus comandos de CLI. En su lugar, el nombre de host DNS de la API de Amazon Braket, que la CLI y el SDK de Braket utilizan de forma predeterminada, se resuelve en el punto de conexión de VPC. Tiene el formato que se muestra en el siguiente ejemplo:

```
https://braket.substituteYourRegionHere.amazonaws.com
```

La entrada del blog titulada Acceso directo a las libretas Amazon SageMaker AI desde Amazon VPC mediante AWS PrivateLink un punto de conexión proporciona un ejemplo de cómo configurar un punto de conexión para establecer conexiones seguras con libretas, que son similares SageMaker Amazon a las libretas Braket.

Si sigues los pasos de la entrada del blog, recuerda sustituir el nombre AmazonBraket por Amazon SageMaker AI. Para el nombre del servicio, introduce `com.amazonaws.us-east-1.braket` o sustituye tu Región de AWS nombre correcto en esa cadena si tu región no es us-east-1.

Información adicional sobre la creación de un punto de conexión

- Para obtener información sobre la creación de una VPC con subredes privadas, consulte Creación de una VPC con subredes privadas.
- Para obtener información sobre cómo crear y configurar un punto de conexión mediante la consola de Amazon VPC o la AWS CLI, consulte Crear un punto de enlace de VPC en la Guía del usuario de Amazon VPC.

- Para obtener información sobre cómo crear y configurar un punto final mediante CloudFormation, consulte el VPCEndpoint recurso AWS: :EC2:: de la Guía del usuario.CloudFormation

Control del acceso con las políticas de puntos de conexión de Amazon VPC

Para controlar el acceso de la conectividad a Amazon Braket puede asociar una política de punto de conexión de AWS Identity and Access Management (IAM) a su punto de conexión de Amazon VPC. La política especifica la siguiente información:

- La entidad principal que puede llevar a cabo acciones (usuario y rol).
- Las acciones que se pueden realizar.
- Los recursos en los que se pueden llevar a cabo las acciones.

Para obtener más información, consulte Controlar el acceso a los puntos de conexión de la VPC mediante políticas de puntos de conexión en la Guía de usuario de Amazon VPC.

Ejemplo: Política de punto de conexión de VPC para acciones de Braket

A continuación se muestra un ejemplo de una política de punto de conexión para Braket. Cuando se asocia con un punto de conexión, esta política concede acceso a las acciones de Braket mostradas para todas las entidades principales en todos los recursos.

```
{
 "Statement":[
 {
   "Principal":"*",
   "Effect":"Allow",
   "Action":[
     “braket:action-1",
     “braket:action-2",
     “braket:action-3”
     ],
   "Resource":"*"
   }
  ]
}
```

Puede crear reglas de IAM complejas al asociar varias políticas de punto de conexión. Para obtener más información y ejemplos, consulte:

- Políticas de puntos de conexión de Amazon Virtual Private Cloud para Step Functions
- Creación de permisos de IAM granulares para usuarios no administradores
- Controle el acceso a los puntos de conexión de la VPC mediante políticas de puntos de conexión

Registro y supervisión

Después de enviar una tarea cuántica a través del servicio de Amazon Braket, puede supervisar de cerca el estado y el progreso de esa tarea a través del SDK y la consola de Amazon Braket. Esto le proporciona una interfaz centralizada para realizar un seguimiento de la implementación de sus cargas de trabajo, identificar posibles problemas o cuellos de botella y tomar las medidas adecuadas para optimizar el rendimiento y la fiabilidad de sus aplicaciones cuánticas. Cuando se completa la tarea cuántica, Braket guarda los resultados en la ubicación de Amazon S3 que haya especificado. El tiempo de finalización de las tareas cuánticas puede variar, especialmente en el caso de las que se ejecutan en dispositivos con unidades de procesamiento cuántico (QPU). Esto se debe en gran medida a la longitud de la cola de ejecución, ya que los recursos de hardware cuántico se comparten entre varios usuarios.

Lista de tipos de estado:

- `CREATED`: Amazon Braket ha recibido su tarea cuántica.
- `QUEUED`: Amazon Braket ha procesado su tarea cuántica y ahora está esperando para ejecutarse en el dispositivo.
- `RUNNING`: su tarea cuántica se está ejecutando en una QPU o en un simulador bajo demanda.
- `COMPLETED`: su tarea cuántica ha terminado de ejecutarse en la QPU o en el simulador bajo demanda.
- `FAILED`: su tarea cuántica intentó ejecutarse, pero se ha producido un error. Dependiendo de la razón del error producido en la tarea cuántica, intente volver a enviarla.
- `CANCELLED`: ha cancelado la tarea cuántica. La tarea cuántica no se ha ejecutado.

En esta sección:

Seguimiento de tareas cuánticas desde el SDK de Amazon Braket

El comando `device.run(…)` define una tarea cuántica con un identificador de tarea cuántica único. Puede consultar y realizar el seguimiento del estado con `task.state()`, como se muestra en el siguiente ejemplo.

Nota: `task = device.run()` es una operación asíncrona, lo que significa que puede seguir trabajando mientras el sistema procesa su tarea cuántica en segundo plano.

Recuperación de un resultado

Cuando llama a `task.result()`, el SDK comienza a sondear a Amazon Braket para ver si la tarea cuántica se ha completado. El SDK usa los parámetros de sondeo que haya definido en `.run()`. Una vez completada la tarea cuántica, el SDK recupera el resultado del bucket de S3 y lo devuelve como un objeto `QuantumTaskResult`.

```
# create a circuit, specify the device and run the circuit
circ = Circuit().rx(0, 0.15).ry(1, 0.2).cnot(0,2)
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
task = device.run(circ, s3_location, shots=1000)

# get ID and status of submitted task
task_id = task.id
status = task.state()
print('ID of task:', task_id)
print('Status of task:', status)
# wait for job to complete
while status != 'COMPLETED':
    status = task.state()
    print('Status:', status)
```

```
ID of task:
arn:aws:braket:us-west-2:123412341234:quantum-task/b68ae94b-1547-4d1d-aa92-1500b82c300d
Status of task: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: RUNNING
```

```
Status: RUNNING
Status: COMPLETED
```

Cancelación de una tarea cuántica

Para cancelar una tarea cuántica, llame al método `cancel()`, como se muestra en el siguiente ejemplo.

```
# cancel quantum task
task.cancel()
status = task.state()
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

Comprobación de los metadatos

Puede comprobar los metadatos de la tarea cuántica finalizada, como se muestra en el siguiente ejemplo.

```
# get the metadata of the quantum task
metadata = task.metadata()
# example of metadata
shots = metadata['shots']
date = metadata['ResponseMetadata']['HTTPHeaders']['date']
# print example metadata
print("{} shots taken on {}.".format(shots, date))

# print name of the s3 bucket where the result is saved
results_bucket = metadata['outputS3Bucket']
print('Bucket where results are stored:', results_bucket)
# print the s3 object key (folder name)
results_object_key = metadata['outputS3Directory']
print('S3 object key:', results_object_key)

# the entire look-up string of the saved result data
look_up = 's3://'+results_bucket+'/'+results_object_key
print('S3 URI:', look_up)
```

```
1000 shots taken on Wed, 05 Aug 2020 14:44:22 GMT.
Bucket where results are stored: amazon-braket-123412341234
S3 object key: simulation-output/b68ae94b-1547-4d1d-aa92-1500b82c300d
```

```
S3 URI: s3://amazon-braket-123412341234/simulation-output/b68ae94b-1547-4d1d-
aa92-1500b82c300d
```

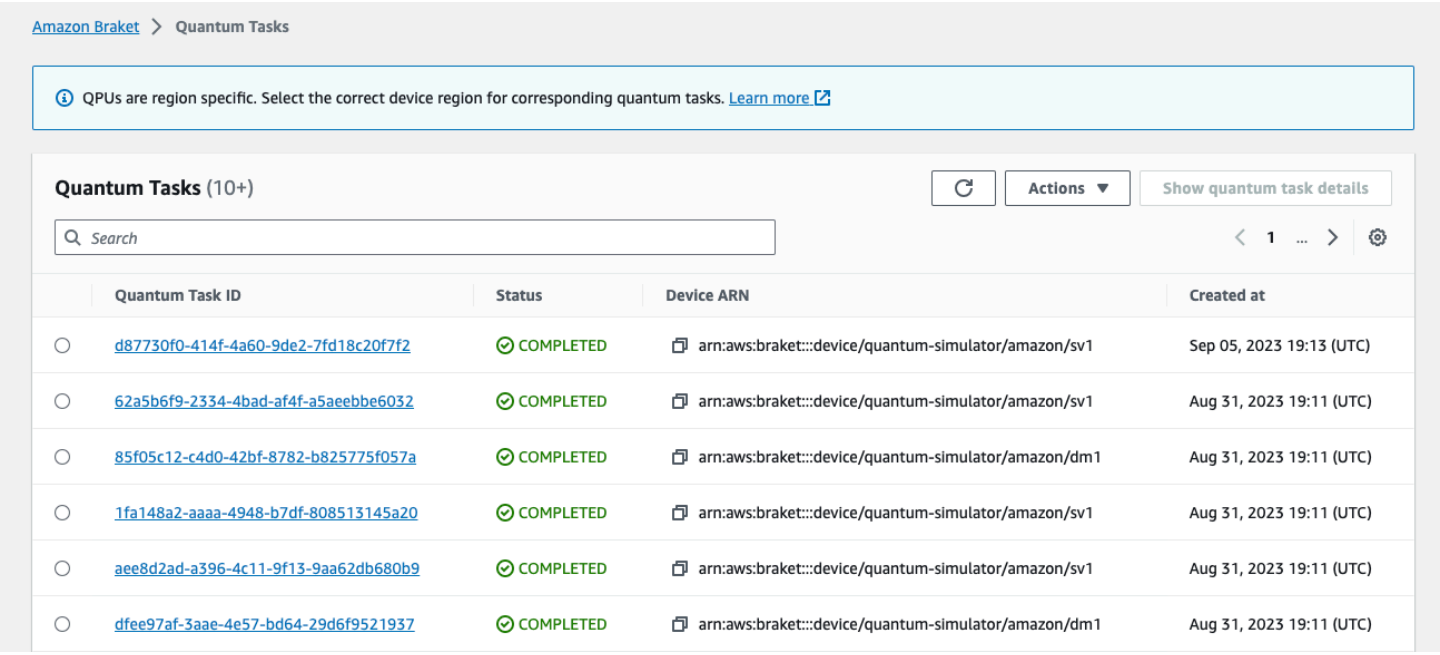
Recuperación de una tarea cuántica o un resultado

Si su kernel se bloquea después de enviar la tarea cuántica o si cierra su cuaderno o su computadora, puede reconstruir el objeto `task` con su ARN único (ID de tarea cuántica). A continuación, puede llamar a `task.result()` para obtener el resultado del bucket de S3 en el que está almacenado.

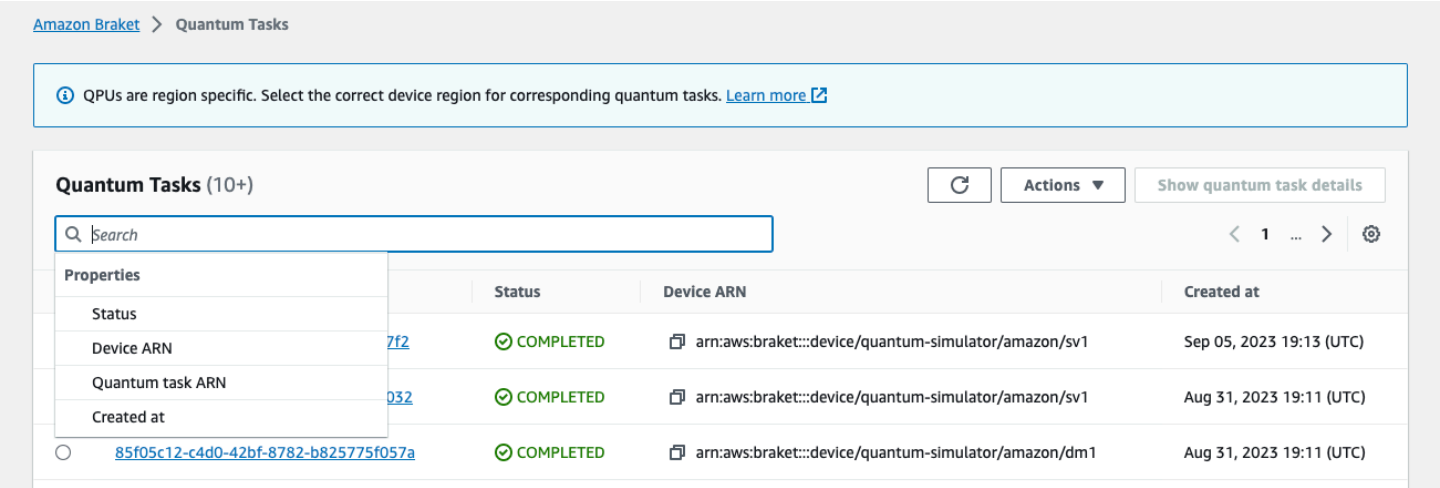
```
from braket.aws import AwsSession, AwsQuantumTask

# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Supervisión de tareas cuánticas a través de la consola de Amazon Braket

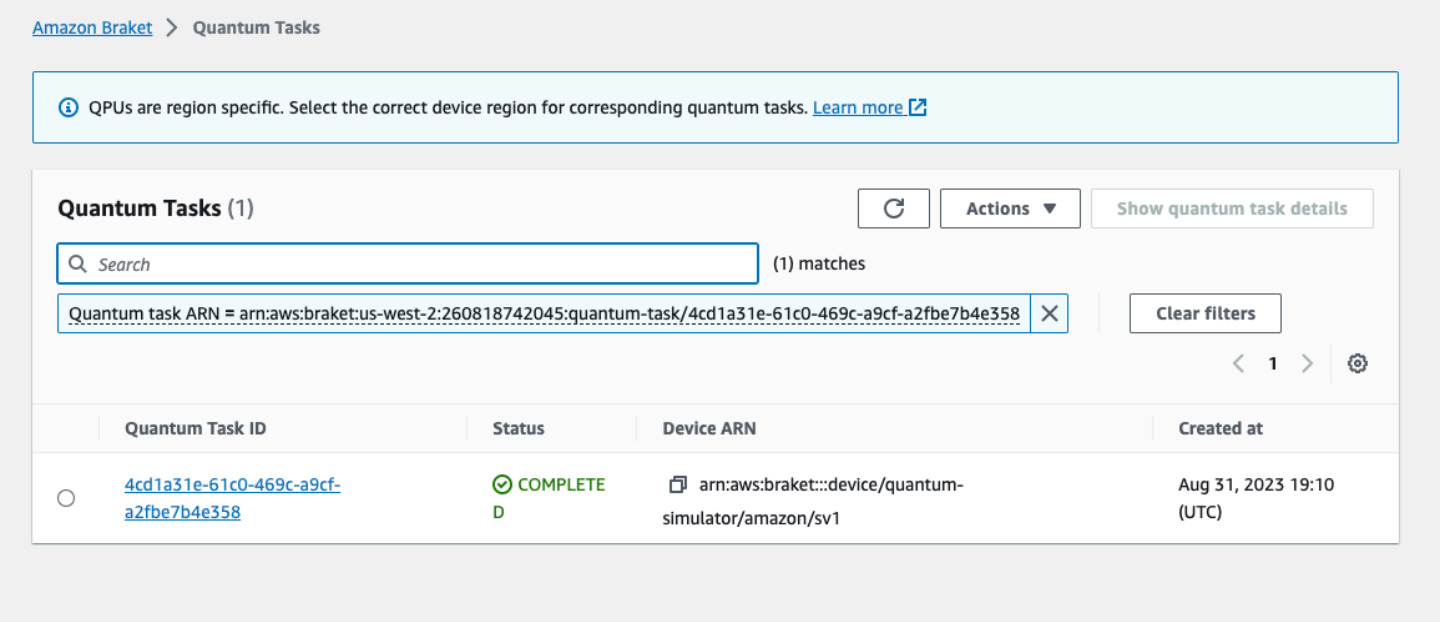
Amazon Braket ofrece una forma cómoda de supervisar la tarea cuántica a través de la consola de Amazon Braket. Todas las tareas cuánticas enviadas se muestran en el campo Tareas cuánticas, como se muestra en la siguiente figura. Este servicio es específico de una región, lo que significa que solo puede ver las tareas cuánticas creadas en la región específica. Región de AWS



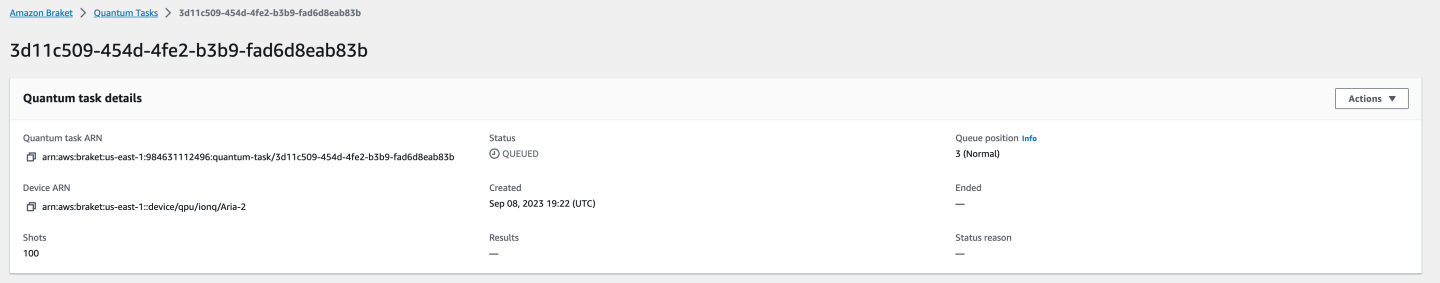
Puede buscar tareas cuánticas específicas a través de la barra de navegación. La búsqueda se puede basar en el ARN (ID), el estado, el dispositivo y la hora de creación de la tarea cuántica. Las opciones aparecen automáticamente al seleccionar la barra de navegación, como se muestra en el siguiente ejemplo.



La siguiente imagen muestra un ejemplo de búsqueda de una tarea cuántica basada en su ID de tarea cuántica única, que se puede obtener llamando a `task.id`.



Además, como se puede ver en la siguiente figura, el estado de una tarea cuántica se puede supervisar mientras se encuentra en un estado `QUEUED`. Al hacer clic en el ID de la tarea cuántica, se muestra la página de detalles. Esta página muestra la posición dinámica de la cola de la tarea cuántica en relación con el dispositivo en el que se procesará.



Las tareas cuánticas enviadas como parte de un trabajo híbrido tendrán prioridad en la cola. Las tareas cuánticas enviadas fuera de un trabajo híbrido tendrán una prioridad de cola normal.

Los clientes que deseen consultar el SDK de Braket pueden obtener sus posiciones en la cola de tareas cuánticas y trabajos híbridos mediante programación. Para obtener más información, consulte la página When will my task run.

Etiquetado de los recursos de Amazon Braket

Una etiqueta es una etiqueta de atributo personalizada que se asigna o que se AWS asigna a un AWS recurso. Una etiqueta es un metadato que proporciona más información sobre su recurso. Cada etiqueta consta de una clave y un valor. En conjunto, se conocen como pares clave-valor. En el caso de etiquetas que usted asigna, debe definir la clave y el valor.

En la consola de Amazon Braket, puede ir a una tarea cuántica o un cuaderno y ver la lista de etiquetas asociadas. Puede añadir una etiqueta, eliminarla o modificarla. Puede etiquetar una tarea cuántica o un cuaderno al crearla y, a continuación, administrar las etiquetas asociadas a través de la consola AWS CLI, oAPI.

Más información sobre AWS y etiquetas

- Para obtener información general sobre el etiquetado, incluidas las convenciones de nomenclatura y uso, consulta ¿Qué es el editor de etiquetas? en la Guía del usuario de AWS los recursos de etiquetado y del editor de etiquetas.
- Para obtener información sobre las restricciones del etiquetado, consulte los límites y requisitos de denominación de las etiquetas en la Guía del usuario de Tagging AWS Resources y Tag Editor.
- Para obtener información sobre las prácticas recomendadas y las estrategias de etiquetado, consulte Prácticas recomendadas para el etiquetado de los recursos de AWS.
- Para obtener una lista de los servicios que admiten el uso de etiquetas, consulte Referencia de la API de etiquetado de Resource Groups Tagging API Reference.

En las siguientes secciones, se ofrece más información sobre las etiquetas de Amazon Braket.

En esta sección:

Uso de etiquetas

Las etiquetas pueden organizar sus recursos en categorías que le resulten útiles. Por ejemplo, puede asignar una etiqueta «Departamento» para especificar el departamento al que pertenece este recurso.

Cada etiqueta de tiene dos partes:

- Una clave de etiqueta (por ejemplo CostCenter, entorno o proyecto). Las claves de etiquetas distinguen entre mayúsculas y minúsculas.
- Un campo opcional conocido como valor de etiqueta (por ejemplo, 111122223333 o Production). Omitir el valor de etiqueta es lo mismo que utilizar una cadena vacía. Al igual que las claves de etiquetas, los valores de las etiquetas distinguen mayúsculas de minúsculas.

Las etiquetas le ayudan a hacer lo siguiente:

- Identifique y organice sus AWS recursos. Muchos Servicios de AWS admiten el etiquetado, por lo que puede asignar la misma etiqueta a los recursos de diferentes servicios para indicar que los recursos están relacionados.
- Realice un seguimiento de sus AWS costes. Estas etiquetas se activan en el Administración de facturación y costos de AWS panel de control. AWS utiliza las etiquetas para clasificar los costes y entregarle un informe mensual de asignación de costes. Para obtener más información, consulte Uso de etiquetas de asignación en la guía del usuario de Administración de facturación y costos de AWS .
- Controle el acceso a sus AWS recursos. Para obtener más información, consulte Control de acceso mediante etiquetas.

Recursos compatibles para el etiquetado en Amazon Braket

El siguiente tipo de recurso en Amazon Braket admite el etiquetado:

- Recurso de `quantum-task`
- Nombre del recurso: `AWS::Service::Braket`
- Regex de ARN: `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

Nota: Puedes aplicar y gestionar etiquetas para tus libretas Amazon Braket en la consola Amazon Braket, utilizando la consola para navegar hasta el recurso de libreta, aunque las libretas en realidad son recursos de Amazon AI. SageMaker Para obtener más información, consulte los metadatos de las instancias de Notebook en la documentación. SageMaker

Etiquetado con la Amazon Braket API

- Su utiliza la API de Amazon Braket para configurar etiquetas en un recurso, llame a la API de `TagResource`.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tags {\"city\":
\"Seattle\"}
```

- Para eliminar etiquetas de un recurso, llame a la API de `UntagResource`.

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- Para enumerar todas las etiquetas asociadas a un recurso concreto, llame a la API de `ListTagsForResource`.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city
\",\"state\"]"
```

Restricciones de etiquetado

Las siguientes restricciones básicas se aplican a las etiquetas en los recursos de Amazon Braket:

- Número máximo de etiquetas que puede asignar a un recurso: 50

- Longitud máxima de la clave: 128 caracteres Unicode
- Longitud máxima del valor: 256 caracteres Unicode
- Caracteres válidos para la clave y el valor: `a-z, A-Z, 0-9, space`, y estos caracteres: `_ . : / = + -` y `@`
- Las claves y los valores distinguen entre mayúsculas y minúsculas.
- No lo utilices `aws` como prefijo para las claves; está reservado para AWS su uso.

Administración de etiquetas en Amazon Braket

Puede establecer etiquetas como propiedades en un recurso. Puede ver, añadir, modificar, enumerar y eliminar etiquetas a través de la consola de Amazon Braket, la API de Amazon Braket o la AWS CLI. Para obtener más información, consulte la referencia de la API de Amazon Braket.

En esta sección:

Adición de etiquetas

Puede añadir etiquetas a los recursos etiquetables en los siguientes momentos:

- Cuando cree el recurso: use la consola o incluya el parámetro `Tags` con la operación `Create` en la API de AWS.
- Después de crear el recurso: use la consola para ir a la tarea cuántica o el recurso del cuaderno, o llame a la operación `TagResource` en la API de AWS.

Para añadir etiquetas a un recurso en el momento en el que lo cree, también debe tener permiso para crear un recurso del tipo especificado.

Visualización de etiquetas

Puede ver las etiquetas de cualquiera de los recursos etiquetables de Amazon Braket utilizando la consola para navegar hasta la tarea o el recurso del bloc de notas, o llamando AWS `ListTagsForResource` API a la operación.

Puede usar el siguiente AWS API comando para ver las etiquetas de un recurso:

- AWS API: `ListTagsForResource`

Edición de etiquetas

Puede editar etiquetas utilizando la consola para ir a la tarea cuántica o al recurso del cuaderno, o puede utilizar el siguiente comando para modificar el valor de una etiqueta asociada a un recurso etiquetable. Al especificar una clave de etiqueta que ya existe, se sobrescribe el valor de esa clave:

- AWS API: `TagResource`

Eliminación de etiquetas

Puede eliminar etiquetas de un recurso especificando las claves que desea eliminar, utilizando la consola para ir al recurso de tarea cuántica o cuaderno, o al llamar a la operación `UntagResource`.

- AWS API: `UntagResource`

Ejemplo de AWS CLI etiquetado en Amazon Braket

Cuando trabaje con AWS Command Line Interface (AWS CLI) para interactuar con Amazon Braket, el siguiente código es un comando de ejemplo que muestra cómo crear una etiqueta que se aplique a una tarea cuántica que cree. En este ejemplo, la tarea se ejecuta en el simulador cuántico SV1 con la configuración de parámetros especificados para la unidad de procesamiento cuántico (QPU) Rigetti. Es importante que, dentro del comando de ejemplo, la etiqueta se especifique al final, después de todos los demás parámetros requeridos. En este caso, la etiqueta tiene una clave de `state` y un valor de `Washington`. Estas etiquetas podrían usarse para ayudar a categorizar o identificar esta tarea cuántica en particular.

```
aws braket create-quantum-task --action /
"{\"braketSchemaHeader\": {\"name\": \"braket.ir.jaqcd.program\", /
```

```
    \"version\": \"1\"}, /
    \"instructions\": [{\"angle\": 0.15, \"target\": 0, \"type\": \"rz\"}], /
    \"results\": null, /
    \"basis_rotation_instructions\": null}" /
  --device-arn "arn:aws:braket:::device/quantum-simulator/amazon/sv1" /
  --output-s3-bucket  "my-example-braket-bucket-name" /
  --output-s3-key-prefix "my-example-username"  /
  --shots 100   /
  --device-parameters /
  "{\"braketSchemaHeader\": /
     {\"name\": \"braket.device_schema.rigetti.rigetti_device_parameters\", /
      \"version\": \"1\"}, \"paradigmParameters\": /
       {\"braketSchemaHeader\": /
         {\"name\": \"braket.device_schema.gate_model_parameters\", /
          \"version\": \"1\"}, /
          \"qubitCount\": 2}}" /
          --tags {\"state\":\"Washington\"}
```

En este ejemplo, se muestra cómo aplicar etiquetas a las tareas cuánticas al ejecutarlas a través de la AWS CLI, lo que resulta útil para organizar y hacer un seguimiento de sus recursos de Braket.

Supervise sus tareas cuánticas con EventBridge

Amazon EventBridge monitorea los eventos de cambio de estado en las tareas cuánticas de Amazon Braket. Los eventos de Amazon Braket se envían casi en tiempo real. EventBridge Puede crear reglas para indicar qué eventos le resultan de interés, incluidas las acciones automatizadas que se van a realizar cuando un evento cumple una de las reglas. Las acciones automáticas que se pueden activar incluyen las siguientes:

- Invocar una función AWS Lambda
- Activar una máquina de AWS Step Functions estados
- Notificar un tema sobre Amazon SNS

EventBridge supervisa estos eventos de cambio de estado de Amazon Braket:

- El estado de las tareas cuánticas cambia.

Amazon Braket garantiza la entrega de eventos de cambio de estado de las tareas cuánticas. Estos eventos se producen al menos una vez, pero posiblemente fuera de orden.

Para obtener más información, consulta los eventos en Amazon EventBridge.

En esta sección:

- Supervise el estado de las tareas cuánticas con EventBridge
- Ejemplo de evento Amazon Braket EventBridge

Supervise el estado de las tareas cuánticas con EventBridge

Con él EventBridge, puedes crear reglas que definan las acciones que se deben realizar cuando Amazon Braket envía una notificación de un cambio de estado relacionado con una tarea cuántica de Braket. Por ejemplo, puede crear una regla que le envíe un mensaje de correo electrónico cada vez que cambie el estado de una tarea cuántica.

1. Inicie sesión AWS con una cuenta que tenga permisos para usar Braket EventBridge . Amazon
2. Abre la EventBridge consola de Amazon.
3. Con los siguientes valores, cree una EventBridge regla:
   - En Tipo de regla, elija Regla con un patrón de evento.
   - En Origen del evento, elija Otro.
   - En la sección Patrón de eventos, elija Patrones personalizados (editor JSON) y pegue el siguiente patrón de eventos en el área de texto:

```
{
  "source": [
    "aws.braket"
  ],
  "detail-type": [
    "Braket Task State Change"
  ]
}
```

Para capturar todos los eventos de Amazon Braket, excluya la sección `detail-type`, tal y como se muestra en el siguiente código:

```
{
  "source": [
    "aws.braket"
  ]
```

```
}
```

- Para los tipos de objetivos Servicio de AWS, elija y, para Seleccione un objetivo, elija un objetivo, como un tema o AWS Lambda una función de Amazon SNS. El destino se activa cuando se recibe un evento de cambio de estado de tarea cuántica de Amazon Braket.

  Por ejemplo, utilice un tema de Amazon Simple Notification Service (SNS) para enviar un correo electrónico o un mensaje de texto cuando se produce un evento. Para ello, primero cree un tema de Amazon SNS utilizando la consola de Amazon SNS. Para obtener más información, consulte Uso de Amazon SNS para notificaciones de usuario.

Para obtener más información sobre la creación de reglas, consulta Cómo crear EventBridge reglas de Amazon que reaccionen a los eventos.

Ejemplo de evento Amazon Braket EventBridge

Para obtener información sobre los campos de un evento de cambio de estado de Amazon Braket Quantum Task, consulte Eventos en Amazon. EventBridge

Los siguientes atributos aparecen en el campo «detalle» de JSON.

- **`quantumTaskArn`** (str): la tarea cuántica para la que se generó este evento.
- **`status`** (Optional[str]): el estado al que pasó la tarea cuántica.
- **`deviceArn`** (str): el dispositivo especificado por el usuario para el que se creó esta tarea cuántica.
- shots (int): El número de shots solicitados por el usuario.
- **`outputS3Bucket`** (str): el bucket de salida especificado por el usuario.
- **`outputS3Directory`** (str): el prefijo de clave de salida especificado por el usuario.
- **`createdAt`** (str): el tiempo de creación de la tarea cuántica como una cadena ISO-8601.
- **`endedAt`** (Optional[str]): el momento en el que la tarea cuántica alcanzó un estado terminal. Este campo solo está presente cuando la tarea cuántica ha pasado a un estado terminal.

El siguiente código JSON muestra un ejemplo de evento de cambio de estado de una tarea cuántica de Amazon Braket.

```
{
    "version":"0",
```

```
    "id":"6101452d-8caf-062b-6dbc-ceb5421334c5",
    "detail-type":"Braket Task State Change",
    "source":"aws.braket",
    "account":"012345678901",
    "time":"2021-10-28T01:17:45Z",
    "region":"us-east-1",
    "resources":[
        "arn:aws:braket:us-east-1:012345678901:quantum-task/834b21ed-77a7-4b36-a90c-
c776afc9a71e"
    ],
    "detail":{
        "quantumTaskArn":"arn:aws:braket:us-east-1:012345678901:quantum-
task/834b21ed-77a7-4b36-a90c-c776afc9a71e",
        "status":"COMPLETED",
        "deviceArn":"arn:aws:braket:::device/quantum-simulator/amazon/sv1",
        "shots":"100",
        "outputS3Bucket":"amazon-braket-0260a8bc871e",
        "outputS3Directory":"sns-testing/834b21ed-77a7-4b36-a90c-c776afc9a71e",
        "createdAt":"2021-10-28T01:17:42.898Z",
        "eventName":"MODIFY",
        "endedAt":"2021-10-28T01:17:44.735Z"
      }
}
```


Supervise sus métricas con CloudWatch

Puedes monitorizar Amazon Braket con Amazon CloudWatch, que recopila datos sin procesar y los procesa para convertirlos en métricas legibles prácticamente en tiempo real. Puedes ver la información histórica generada hasta hace 15 meses o las métricas de búsqueda que se han actualizado en las últimas 2 semanas en la CloudWatch consola de Amazon para tener una mejor perspectiva del rendimiento de Amazon Braket. Para obtener más información, consulta Uso de CloudWatch métricas.

> **Note**
>
> Para ver las secuencias de CloudWatch registro de las libretas Amazon Braket, diríjase a la página de detalles de las libretas de la consola Amazon AI. SageMaker Los ajustes adicionales del portátil Amazon Braket están disponibles a través de la SageMaker consola.

En esta sección:

Métricas y dimensiones de Amazon Braket

Las métricas son el concepto fundamental en CloudWatch. Una métrica representa un conjunto de puntos de datos ordenados en función del tiempo que se publican en. CloudWatch Cada métrica se caracteriza por un conjunto de dimensiones. Para obtener más información sobre las dimensiones de las métricas en CloudWatch, consulte CloudWatch las dimensiones.

Amazon Braket envía los siguientes datos de métricas, específicos de Amazon Braket, a las métricas de Amazon: CloudWatch

Métricas de tareas cuánticas

Las métricas están disponibles si existen tareas cuánticas. Se muestran en AWS/Braket/By Device en la consola. CloudWatch

Métrica	Description (Descripción)
Recuento	Número de tareas cuánticas.
Latencia	Esta métrica se emite cuando se ha completad o una tarea cuántica. Representa el tiempo total desde la inicialización de la tarea cuántica hasta su finalización.

Dimensiones de las métricas de tareas cuánticas

Las métricas de las tareas cuánticas se publican con una dimensión basada en el parámetro `deviceArn`, que tiene el formato arn:aws:braket:::device/xxx.

Registra tus tareas cuánticas con CloudTrail

Amazon Braket está integrado con AWS CloudTrail un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o una persona Servicio de AWS en Amazon Braket. CloudTrail captura todas las API convocatorias de Amazon Braket como eventos. Las llamadas capturadas incluyen llamadas desde la consola de Amazon Braket y llamadas de código a las

operaciones de Amazon Braket. Si crea una ruta, puede habilitar la entrega continua de CloudTrail eventos a un bucket de Amazon S3, incluidos los eventos de Amazon Braket. Si no configura una ruta, podrá ver los eventos más recientes en la CloudTrail consola, en el historial de eventos. Con la información recopilada por CloudTrail, puedes determinar la solicitud que se realizó a Amazon Braket, la dirección IP desde la que se realizó la solicitud, quién la hizo, cuándo se realizó y detalles adicionales.

Para obtener más información CloudTrail, consulte la Guía del AWS CloudTrail usuario.

En esta sección:

- Información sobre Amazon Braket en CloudTrail
- Explicación de las entradas del archivo de registro de Amazon Braket

Información sobre Amazon Braket en CloudTrail

CloudTrail está activado en tu cuenta Cuenta de AWS al crear la cuenta. Cuando se produce una actividad en Amazon Braket, esa actividad se registra en un CloudTrail evento junto con otros Servicio de AWS eventos del historial de eventos. Puede ver, buscar y descargar eventos recientes en su Cuenta de AWS. Para obtener más información, consulte Visualización de eventos con el historial de CloudTrail eventos.

Para tener un registro continuo de tus eventos Cuenta de AWS, incluidos los eventos de Amazon Braket, crea una ruta. Un rastro permite CloudTrail entregar archivos de registro a un bucket de Amazon S3. De forma predeterminada, cuando se crea un registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las Regiones de AWS. La ruta registra los eventos de todas las regiones de la AWS partición y envía los archivos de registro al bucket de Amazon S3 que especifique. Además, puede configurar otros Servicios de AWS para que analicen más a fondo los datos de eventos recopilados en los CloudTrail registros y actúen en función de ellos. Para más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- CloudTrail Integraciones y servicios compatibles
- Configuración de las notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de CloudTrail registro de varias regiones y recibir archivos de CloudTrail registro de varias cuentas

Todas las acciones de Amazon Braket se registran en. CloudTrail Por ejemplo, las llamadas a las `GetDevice` acciones `GetQuantumTask` o acciones generan entradas en los archivos de CloudTrail registro.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro Servicio de AWS.

Para obtener más información, consulte el Elemento userIdentity de CloudTrail .

Explicación de las entradas del archivo de registro de Amazon Braket

Un rastro es una configuración que permite la entrega de eventos como archivos de registro a un bucket de Amazon S3 que usted especifique. CloudTrail Los archivos de registro contienen una o más entradas de registro. Un evento representa una solicitud única de cualquier fuente e incluye información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. CloudTrail Los archivos de registro no son un registro ordenado de las API llamadas públicas, por lo que no aparecen en ningún orden específico.

El siguiente ejemplo es una entrada de registro de la acción `GetQuantumTask`, que obtiene los detalles de una tarea cuántica.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
        "userName": "foobar"
```

```
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:56:57Z"
      }
    }
  },
  "eventTime": "2020-08-07T01:00:08Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetQuantumTask",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "aws-cli/1.18.110 Python/3.6.10
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.17.33",
  "requestParameters": {
    "quantumTaskArn": "foobar"
  },
  "responseElements": null,
  "requestID": "20e8000c-29b8-4137-9cbc-af77d1dd12f7",
  "eventID": "4a2fdb22-a73d-414a-b30f-c0797c088f7c",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```

A continuación se muestra una entrada de registro de la acción `GetDevice`, que devuelve los detalles de un evento del dispositivo.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
```

```
        "userName": "foobar"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:46:29Z"
      }
    }
  },
  "eventTime": "2020-08-07T00:46:32Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetDevice",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "Boto3/1.14.33 Python/3.7.6 Linux/4.14.158-129.185.amzn2.x86_64 exec-
env/AWS_ECS_FARGATE Botocore/1.17.33",
  "errorCode": "404",
  "requestParameters": {
    "deviceArn": "foobar"
  },
  "responseElements": null,
  "requestID": "c614858b-4dcf-43bd-83c9-bcf9f17f522e",
  "eventID": "9642512a-478b-4e7b-9f34-75ba5a3408eb",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```


Registro avanzado con Amazon Braket

Puede registrar todo el proceso de procesamiento de tareas utilizando un registrador. Estas técnicas de registro avanzadas permiten ver el sondeo en segundo plano y crear un registro para su posterior depuración.

Para utilizar el registrador, recomendamos cambiar los parámetros `poll_timeout_seconds` y `poll_interval_seconds`, de modo que una tarea cuántica pueda ejecutarse durante mucho tiempo y el estado de la tarea cuántica se registre de forma continua de forma que los resultados se guarden en un archivo. Puede transferir este código a un script de Python en lugar de a un cuaderno de Jupyter, de modo que el script se pueda ejecutar como un proceso en segundo plano.

Configuración del registrador

En primer lugar, configure el registrador para que todos los registros se escriban automáticamente en un archivo de texto, tal y como se muestra en las siguientes líneas de ejemplo.

```
# import the module
import logging
from datetime import datetime

# set filename for logs
log_file = 'device_logs-'+datetime.strftime(datetime.now(), '%Y%m%d%H%M%S')+'.txt'
print('Task info will be logged in:', log_file)

# create new logger object
logger = logging.getLogger("newLogger")

# configure to log to file device_logs.txt in the appending mode
logger.addHandler(logging.FileHandler(filename=log_file, mode='a'))

# add to file all log messages with level DEBUG or above
logger.setLevel(logging.DEBUG)
```

```
Task info will be logged in: device_logs-20200803203309.txt
```

Creación y ejecución del circuito

Ahora puede crear un circuito, enviarlo a un dispositivo para que lo ejecute y ver qué sucede, tal y como se muestra en este ejemplo.

```
# define circuit
circ_log = Circuit().rx(0, 0.15).ry(1, 0.2).rz(2, 0.25).h(3).cnot(control=0,
 target=2).zz(1, 3, 0.15).x(4)
print(circ_log)
# define backend
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
# define what info to log
logger.info(
    device.run(circ_log, s3_location,
               poll_timeout_seconds=1200, poll_interval_seconds=0.25, logger=logger,
 shots=1000)
    .result().measurement_counts
)
```

Comprobación del archivo de registro

Puede comprobar lo que está escrito en el archivo mediante el siguiente comando.

```
# print logs
! cat {log_file}
```

```
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: start polling for completion
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status CREATED
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status CREATED
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status QUEUED
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status RUNNING
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status RUNNING
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status COMPLETED
Counter({'00001': 493, '00011': 493, '01001': 5, '10111': 4, '01011': 3, '10101': 2})
```

Obtención del ARN a partir del archivo de registro

A partir de la salida del archivo de registro que se devuelve, como se muestra en el ejemplo anterior, puede obtener la información del ARN. Con el identificador del ARN, puede recuperar el resultado de la tarea cuántica completada.

```
# parse log file for arn
with open(log_file) as openfile:
    for line in openfile:
        for part in line.split():
            if "arn:" in part:
                arn = part
                break
# remove final semicolon in logs
arn = arn[:-1]

# with this arn you can restore again task from unique arn
task_load = AwsQuantumTask(arn=arn, aws_session=AwsSession())

# get results of task
result = task_load.result()
```

Cuotas de Amazon Braket

En la tabla siguiente se enumeran las cuotas de servicio de Amazon Braket. Service Quotas, también denominadas límites, establecen el número máximo de recursos u operaciones de servicio para su cuenta de Cuenta de AWS.

Algunas cuotas se pueden aumentar. Para obtener más información, consulte Cuotas de Servicio de AWS.

- Las cuotas de tasa de ráfaga no se pueden aumentar.
- El aumento máximo de la tasa para las cuotas ajustables (excepto la tasa de ráfaga, que no se puede ajustar) es el doble del límite de tasa predeterminado que se ha especificado. Por ejemplo, una cuota predeterminada de 60 se puede ajustar a un máximo de 120.
- La cuota ajustable para las tareas cuánticas simultáneas de SV1 (DM1) permite un máximo de 60 por Región de AWS.
- El número máximo de instancias de cómputo permitidas para un trabajo híbrido es 1 y las cuotas se pueden ajustar.

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de solicitudes API	El número máximo de solicitudes por segundo que puede enviar desde esta cuenta en la región actual.	140	Sí
Tasa de ráfaga de solicitudes a la API	El número máximo de solicitudes de adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	600	No

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de solicitudes `CreateQuantumTask`	El número máximo de solicitudes de `CreateQuantumTask` que puede enviar por segundo en esta cuenta por región.	20 por segundo	Sí
Tasa de ráfaga de solicitudes a la `CreateQuantumTask`	El número máximo de solicitudes de `CreateQuantumTask` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	40	No
Tasa de solicitudes `SearchQuantumTasks`	El número máximo de solicitudes de `SearchQuantumTasks` que puede enviar por segundo en esta cuenta por región.	5 por segundo	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de ráfaga de solicitudes a la `SearchQuantumTasks`	El número máximo de solicitudes de `SearchQuantumTasks` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	50	No
Tasa de solicitudes `GetQuantumTask`	El número máximo de solicitudes de `GetQuantumTask` que puede enviar por segundo en esta cuenta por región.	100 por segundo	Sí
Tasa de ráfaga de solicitudes a la `GetQuantumTask`	El número máximo de solicitudes de `GetQuantumTask` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	500	No
Tasa de solicitudes `CancelQuantumTask`	El número máximo de solicitudes de `CancelQuantumTask` que puede enviar por segundo en esta cuenta por región.	2 por segundo	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de ráfaga de solicitudes a la `CancelQuantumTask`	El número máximo de solicitudes de `CancelQuantumTask` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	20	No
Tasa de solicitudes `GetDevice`	El número máximo de solicitudes de `GetDevice` que puede enviar por segundo en esta cuenta por región.	5 por segundo	Sí
Tasa de ráfaga de solicitudes a la `GetDevice`	El número máximo de solicitudes de `GetDevice` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	50	No
Tasa de solicitudes `SearchDevices`	El número máximo de solicitudes de `SearchDevices` que puede enviar por segundo en esta cuenta por región.	5 por segundo	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de ráfaga de solicitudes a la `SearchDevices`	El número máximo de solicitudes de `SearchDevices` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	50	No
Tasa de solicitudes `CreateJob`	El número máximo de solicitudes de `CreateJob` que puede enviar por segundo en esta cuenta por región.	1 por segundo	Sí
Tasa de ráfaga de solicitudes a la `CreateJob`	El número máximo de solicitudes de `CreateJob` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	5	No
Tasa de solicitudes `SearchJobs`	El número máximo de solicitudes de `SearchJob` que puede enviar por segundo en esta cuenta por región.	5 por segundo	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de ráfaga de solicitudes a la `SearchJobs`	El número máximo de solicitudes de `SearchJob` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	50	No
Tasa de solicitudes `GetJob`	El número máximo de solicitudes de `GetJob` que puede enviar por segundo en esta cuenta por región.	5 por segundo	Sí
Tasa de ráfaga de solicitudes a la `GetJob`	El número máximo de solicitudes de `GetJob` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	25	No
Tasa de solicitudes `CancelJob`	El número máximo de solicitudes de `CancelJob` que puede enviar por segundo en esta cuenta por región.	2 por segundo	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Tasa de ráfaga de solicitudes a la `CancelJob`	El número máximo de solicitudes de `CancelJob` adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	5	No
Número de tareas cuánticas de SV1 simultáneas	El número máximo de tareas cuánticas simultáneas que se ejecutan en el simulador de vector de estado (SV1) en la región actual.	100 us-east-1, 50 us-west-1, 100 us-west-2, 50 eu-west-2	No
Número de tareas cuánticas de DM1 simultáneas	El número máximo de tareas cuánticas simultáneas que se ejecutan en el simulador de matriz de densidad (DM1) en la región actual.	100 us-east-1, 50 us-west-1, 100 us-west-2, 50 eu-west-2	No
Número de tareas cuánticas de TN1 simultáneas	El número máximo de tareas cuánticas simultáneas que se ejecutan en el simulador de red tensorial (TN1) en la región actual.	10 us-east-1, 10 us-west-2, 5 eu-west-2,	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable
Número de trabajos híbridos simultáneos	El número máximo de trabajos híbridos simultáneos en la región actual.	3	Sí
Límite de tiempo de ejecución de trabajos híbridos	El tiempo máximo en días que puede ejecutarse un trabajo híbrido.	5	No

A continuación se indican las cuotas predeterminadas de instancias de cómputo clásicas para trabajos híbridos. Para aumentar estas cuotas, póngase en contacto con Soporte. Además, se especifican las regiones disponibles para cada instancia.

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c4.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c4.xlarge permitido para todos los trabajos	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
	híbridos de Amazon Braket en esta cuenta y región.							
Número máximo de instancias de ml.c4.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c4.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c4.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c4.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c4.8xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c4.8xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	1	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5.9xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5.9xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	1	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5.18xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5.18xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5n.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5n.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5n.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5n.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5n.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5n.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5n.9xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5n.9xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c5n.18xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.c5n.18xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.8xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.8xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.12xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.12xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.g4dn.16xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.g4dn.16xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m4.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m4.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m4.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m4.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m4.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m4.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	2	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m4.10› large para los trabajos híbridos	El número máximo de instancias del tipo ml.m4.10› large permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m4.16› large para los trabajos híbridos	El número máximo de instancias del tipo ml.m4.16› large permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.large para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.large permitido para todos los Trabajos híbridos de Amazon Braket de esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.2xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.4xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.4xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	5	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.12› large para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.12› large permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.m5.24› large para los trabajos híbridos	El número máximo de instancias del tipo ml.m5.24› large permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	Sí	Sí	Sí	Sí

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p2.xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.p2.xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p2.8xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.p2.8xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p2.16xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.p2.16xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Recurso	Description (Descripción)	Límites de las s	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p4d.24xlarge para los trabajos híbridos	El número máximo de instancias del tipo ml.p4d.24xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Solicitud de actualizaciones de límites

Si recibe una ServiceQuotaExceeded excepción para un tipo de instancia y no tiene suficientes instancias disponibles para ella, puede solicitar un aumento del límite en la página Service Quotas de la AWS consola y buscar Amazon Braket en Servicios.AWS

> **Note**
>
> Si su trabajo híbrido no puede aprovisionar la capacidad de computación de ML solicitada, utilice otra región. Además, si no ve una instancia en la tabla, significa que no está disponible para trabajos híbridos.

Cuotas y límites adicionales

- La acción de la tarea cuántica de Amazon Braket tiene un tamaño limitado de 3 MB.
- SV1En efecto, la duración máxima de funcionamiento es de 3 horas para circuitos de hasta 31 qubits y de 11 horas para circuitos de más de 31 qubits.
- El número máximo de shots por tarea permitido para los dispositivos SV1, DM1 y Rigetti es 50 000.
- El número máximo de shots por tarea permitido para TN1 es 1000.
- Para AQT el IBEX-Q1 dispositivo, el máximo es de 2000 disparos por tarea.
- Para todos IonQ los dispositivos: el número mínimo de disparos por tarea es de 100. Cuando se utiliza un modelo bajo demanda, hay un límite de 1 millón de disparos y un mínimo de 2500 disparos para las tareas de mitigación de errores. Para una reserva directa, no hay límite de gateshot y hay un mínimo de 500 shots para tareas de mitigación de errores.
- Para el dispositivo Aquila de QuEra, el máximo es de 1000 shots por tarea.
- Para IQM Emerald dispositivos Garnet y dispositivos, el máximo es de 20 000 disparos por tarea.
- En el TN1 caso de los dispositivos QPU, los disparos por tarea deben ser > 0.

Historial de documentación de la guía para desarrolladores de Amazon Braket

En la siguiente tabla se describen las publicaciones de la documentación de Amazon Braket.

- Última actualización API de referencia: 20 de noviembre de 2025
- Última actualización de la documentación: 2 de marzo de 2026

Cambio	Descripción	Fecha
Retirada del dispositivo IonQ Aria-1	Se ha eliminado el soporte para el dispositivo IonQ Aria-1.	2 de marzo de 2026
Actualice las páginas «Cómo trabajar con reservas»	Se ha mejorado la claridad de las páginas «Cómo trabajar con reservas»	3 de febrero de 2026
Support Python versión 3.12 para cuadernos Braket y contenedores gestionados	Se agregó compatibilidad con la versión 3.12 de Python para las libretas Amazon Braket y los contenedores gestionados (Base, CUDA-Q PennyLane y Tensorflow). Incluye una guía de solución de problemas para la actualización a Python 3.12.	21 de enero de 2026
Elimine los ejemplos de P3	SageMaker está retirando su familia de `ml.p3` instancias. Se reemplazaron los ejemplos por la familia de `ml.g4dn` instancias recomendada.	19 de diciembre de 2025
Nueva función de límite de gastos	Se agregó compatibilidad con la función de límite de gasto Amazon Braket, que permite establecer límites	20 de noviembre de 2025

	presupuestarios opcionales para las personas que validan y rechazan automáticamente las tareas QPUs que superen el umbral de gasto configura do.	
Nuevo dispositivo Braket AQT IBEX-Q1	Se agregó soporte para el AQT IBEX-Q1dispositivo. Este dispositivo se basa en un cristal de ^{40} iones Ca^{+} en una trampa macroscópica de radiofrecuencia situada en una cámara de vacío ultra alto.	18 de noviembre de 2025
Nuevo soporte nativo para CUDA-Q Amazon Braket NBIs	Se añadió compatibilidad nativa con CUDA-Q en las instancias de cuaderno de Amazon Braket. Para obtener más información, consulte CUDA-Q en. NBIs	10 de noviembre de 2025
Retirada de dispositivo IonQ Aria-2	Se eliminó la compatibilidad con el dispositivo IonQ Aria-2.	27 de octubre de 2025
Documentación consolidada sobre trabajos híbridos	Se consolidaron las secciones de trabajos híbridos para que aparezcan en Trabajar con trabajos híbridos de Amazon Braket.	21 de octubre de 2025
Nuevo Braket suministrado por el contenedor de CUDA-Q	Se añadió compatibilidad con un contenedor de trabajos híbridos de CUDA-Q. Para obtener más información, consulte Definición del entorno para el script de algoritmo.	2 de septiembre de 2025

Nueva característica de emulador de dispositivo local	Se añadió compatibilidad con una herramienta de emulador de dispositivo cuántico local para emular sus programas verbatim antes de enviarlos a dispositivos cuánticos.	25 de agosto de 2025
Se movieron las páginas de PennyLane y CUDA-Q a la sección «Desarrollo».	Se movieron las páginas PennyLane y CUDA-Q para que aparezcan en la sección «Desarrollo» del índice.	15 de agosto de 2025
Nueva característica de `ProgramSet` .	Se añadió compatibilidad con conjuntos de programas, una operación para ejecutar varios circuitos cuánticos en una sola tarea cuántica.	14 de agosto de 2025
Nuevo dispositivo IQM Emerald	Se añadió compatibilidad con el dispositivo IQM Emerald. Un dispositivo de 54 bits con una topología de retícula cuadrada (cristal).	21 de julio de 2025
Se actualizó la AmazonBraketServiceRolePolicypolítica	AmazonBraketServiceRolePolicyahora solo proporciona las acciones s3: * y logs: * a aws: PrincipalAccount. Esto restringe el acceso únicamente a los buckets y grupos de registro del solicitante.	11 de julio de 2025

Nueva característica de capacidad experimental: circuitos dinámicos	Las operaciones de medición y prealimentación del circuito medio están disponibles como capacidades experimentales; consulte Acceso a circuitos dinámicos en dispositivos IQM.	26 de junio de 2025
Se actualizó la AmazonBraketFullAccesspolítica	AmazonBraketFullAccessahora incluye los precios: GetProducts para mostrar los costos de hardware en la consola.	14 de abril de 2025
Nuevo dispositivo IonQ Forte-Enterprise-1	Se añadió compatibilidad con el dispositivo IonQ Forte-Enterprise-1. Un dispositivo de 36 bits que utiliza la tecnología de iones atrapados.	17 de marzo de 2025
Permisos mejorados de las condiciones de S3	Para mejorar la seguridad, `AmazonBraketFullAccess` ahora solo proporciona acciones de `s3:*` a la `aws:PrincipalAccount`. Esto restringe el acceso solo a los buckets del solicitante.	7 de marzo de 2025
Nuevo dispositivo Rigetti Ankaa-3	Se añadió compatibilidad con el dispositivo Rigetti Ankaa-3. Un dispositivo de 84 bits que utiliza tecnología escalable de varios chips.	14 de enero de 2025
Retirada de dispositivo Rigetti Ankaa-2	Se eliminó la compatibilidad con el dispositivo Rigetti Ankaa-2.	14 de enero de 2025

Support for IPv6 traffic	Amazon Braket ahora admite el IPv6 tráfico mediante el punto de conexión de doble pila. `braket.{region}.api.aws`	12 de diciembre de 2024
Compatibilidad con NVIDIA's CUDA-Q en Amazon Braket	Los clientes ahora pueden ejecutar programas cuánticos utilizando el marco de desarrollador de NVIDIA's CUDA-Q en Amazon Braket.	6 de diciembre de 2024
El dispositivo IonQ Forte-1 está disponible de forma inmediata.	El dispositivo IonQ Forte-1 ya no es exclusivo para reservas y ahora está disponible para nuestros clientes.	22 de noviembre de 2024
Retirada de dispositivo Rigetti Aspen-M-3	Se eliminó la compatibilidad con el dispositivo Rigetti Aspen-M-3.	27 de septiembre de 2024
Retirada de dispositivo IonQ Harmony	Se eliminó la compatibi lidad con el dispositivo IonQ Harmony.	29 de agosto de 2024
Nuevo dispositivo Rigetti Ankaa-2	Se añadió compatibilidad con el dispositivo Rigetti Ankaa-2. Un dispositivo de 84 bits que utiliza tecnología escalable de varios chips.	26 de agosto de 2024
Reorganización de la guía para desarrolladores	La nueva guía para desarroll adores emplea el recorrido existente del cliente (Desarrol lo, Prueba, Ejecución) y guía a los usuarios a lo largo del recorrido con Amazon Braket.	23 de agosto de 2024

Retirada de dispositivo OQC Lucy	Se eliminó la compatibilidad con el dispositivo OQC Lucy.	28 de junio de 2024
Nuevo dispositivo IQM Garnet y región Europe North 1	Se añadió compatibilidad con el dispositivo IQM Garnet. Un dispositivo de 20 qubits con una topología de retícula cuadrada. Se ampliaron las regiones aceptadas de Braket a Europe North 1 (Estocolmo).	22 de mayo de 2024
Lanzamiento de desintoni zación local	Las capacidades experimen tales ahora incluyen la función de desafinación local de la QPU Aquila. QuEra	11 de abril de 2024
Lanzamiento del administrador de inactividad del cuaderno	Al crear una instancia de cuaderno, habilite el administr ador de inactividad y establezc a un tiempo de inactividad para restablecer automátic amente la instancia del cuaderno de Braket.	27 de marzo de 2024
Reelaboración del índice	Se reorganizó el índice de Amazon Braket para cumplir con los requisitos de AWS la guía de estilo y mejorar el flujo de contenido para la experienc ia del cliente.	12 de diciembre de 2023

Lanzamiento de Braket Direct	Se añadió compatibilidad con las características de Braket Direct, que incluyen: • Trabajar con reservas • Asesoramiento de expertos • Exploración de las capacidades experimentales	27 de noviembre de 2023
Se actualizó Creación de una instancia de cuaderno de Amazon Braket	Se actualizó la documentación para añadir información sobre cómo crear una instancia de cuaderno para clientes nuevos y existentes de Amazon Braket.	27 de noviembre de 2023
Se actualizó Utilice su propio contenedor (BYOC)	Se actualizó la documentación para añadir información sobre cuándo utilizar BYOC, la fórmula para BYOC y la ejecución de trabajos híbridos de Braket en el contenedor.	18 de octubre de 2023

Lanzamiento del decorador de trabajos híbridos	Se añadió la página Ejecución del código local como un trabajo híbrido. Contiene ejemplos: • Creación de un trabajo híbrido a partir de código de Python local • Instalación de paquetes y código fuente de Python adicionales • Guardar y cargar datos en una instancia de trabajo híbrido • Prácticas recomendadas para decoradores de trabajos híbridos	16 de octubre de 2023
Se añadió Visibilidad de las colas	Se actualizó la documentación de la guía para desarrolladores para incluir queue depth y queue position. Se actualizó la documentación de la API para reflejar los nuevos cambios en la API en lo que respecta a la visibilidad de las colas.	25 de septiembre de 2023
Normalización de la nomenclatura en la documentación	Se actualizó la documentación para cambiar todas las instancias de «trabajo» a «trabajo híbrido» y «tarea» a «tarea cuántica».	11 de septiembre de 2023
Nuevo dispositivo IonQ Aria 2	Se añadió compatibilidad con el dispositivo IonQ Aria 2.	8 de septiembre de 2023

Se actualizó Puertas nativas	Se actualizó la documentación para añadir información sobre el acceso programático a las puertas nativas desde Rigetti.	16 de agosto de 2023
Salida de Xanadu	Se actualizó la documenta ción para eliminar todos los dispositivos Xanadu.	2 de junio de 2023
Nuevo dispositivo IonQ Aria	Se añadió compatibilidad con el dispositivo IonQ Aria.	16 de mayo de 2023
Retirada de dispositivo Rigetti	Se interrumpió la compatibi lidad con Rigetti Aspen-M-2.	2 de mayo de 2023
Información de política actualizada AmazonBra ketFullAccess	Se actualizó el script que define el contenido de la AmazonBraketFullAc cesspolítica para incluir las GetMetricData acciones de servicequota GetServic eQuota y cloudwatch, así como información sobre las limitaciones con respecto a las cuotas.	19 de abril de 2023
Lanzamiento de recorridos guiados	Se modificó la documentación para reflejar el método más actualizado y simplificado de incorporación de Braket.	5 de abril de 2023
Nuevo dispositivo Rigetti Aspen-M-3	Se añadió compatibilidad con el dispositivo Rigetti Aspen-M-3.	17 de enero de 2023
Nueva característica de gradiente adjunto	Se añadió información sobre la característica de gradiente adjunto que ofrece SV1.	7 de diciembre de 2022

Nueva característica de biblioteca de algoritmos	Se añadió información sobre la biblioteca de algoritmos de Braket, que proporciona un catálogo de algoritmos cuánticos prediseñados.	28 de noviembre de 2022
Salida de D-Wave	Se actualizó la documentación para adaptarse a la eliminación de todos los dispositivos D-Wave.	17 de noviembre de 2022
Nuevo dispositivo QuEra Aquila	Se añadió compatibilidad con el dispositivo QuEra Aquila.	31 de octubre de 2022
Compatibilidad con Braket Pulse	Se añadió compatibilidad con Braket Pulse, lo que permite utilizar el control de pulso en los dispositivos Rigetti y OQC.	20 de octubre de 2022
Compatibilidad con puertas nativas de IonQ	Se añadió compatibilidad con el conjunto de puertas nativas que ofrece el dispositivo IonQ.	13 de septiembre de 2022
Nuevas cuotas de instancias	Se actualizaron las cuotas predeterminadas de instancias de cómputo clásicas asociadas con los trabajos híbridos.	22 de agosto de 2022
Nuevo panel de servicio	Se actualizaron las capturas de pantalla de la consola para incluir el panel de servicio.	17 de agosto de 2022
Nuevo dispositivo Rigetti Aspen-M-2	Se añadió compatibilidad con el dispositivo Rigetti Aspen-M-2.	12 de agosto de 2022

Nuevas características de OpenQASM	Se añadió compatibilidad de características de OpenQASM con los simuladores locales (braket_sv y braket_dm).	4 de agosto de 2022
Nuevos procedimientos de seguimiento de costos	Se añadió cómo obtener estimaciones de costos máximos casi en tiempo real para simuladores y cargas de trabajo de hardware.	18 de julio de 2022
Nuevo dispositivo Xanadu Borealis	Se añadió compatibilidad con el dispositivo Xanadu Borealis.	2 de junio de 2022
Nuevos procedimientos de simplificación de incorporación	Se añadió información sobre cómo funcionan los procedimientos de incorporación nuevos y simplificados.	16 de mayo de 2022
Nuevo dispositivo D-Wave Advantage_system6.1	Se añadió compatibilidad con el dispositivo D-Wave Advantage_system6.1.	12 de mayo de 2022
Compatibilidad con simuladores integrados	Se agregó cómo ejecutar simulaciones integradas con trabajos híbridos y cómo usar el PennyLane simulador Lightning	4 de mayo de 2022
AmazonBraketFullAccess - Política de acceso completo a Amazon Braket	Se agregaron ListAllMyBuckets permisos s3: para permitir a los usuarios ver e inspeccionar los depósitos creados y utilizados para Amazon Braket	31 de marzo de 2022

Compatibilidad con OpenQASM	Se añadió compatibilidad con OpenQASM 3.0 para simuladores y dispositivos cuánticos basados en puertas.	7 de marzo de 2022
Nuevo proveedor de hardware cuántico, Oxford Quantum Circuits y nueva región, eu-west-2	Se añadió compatibilidad con OQC y eu-west-2.	28 de febrero de 2022
Nuevo dispositivo Rigetti	Se ha agregado compatibilidad para Rigetti Aspen M-1	15 de febrero de 2022
Nuevos límites de recursos	Se aumentó el número máximo de tareas simultáneas de DM1 y SV1 de 55 a 100.	5 de enero de 2022
Nuevo dispositivo Rigetti	Se ha agregado compatibilidad para Rigetti Aspen-11	20 de diciembre de 2021
Retirada de dispositivo Rigetti	Se interrumpió la compatibilidad con el dispositivo Rigetti Aspen-10.	20 de diciembre de 2021
Nuevo tipo de resultado	Tipo de resultado de matriz de densidad reducida compatible con simulador de matriz de densidad local y dispositivos DM1	20 de diciembre de 2021

Actualización de descripción de política	Amazon Braket actualizó el ARN de rol para incluir la ruta `servicerole/` . Para obtener información sobre las actualizaciones de políticas , consulte la tabla Actualizaciones de Amazon Braket a políticas administradas de AWS.	29 de noviembre de 2021
Trabajos de Amazon Braket	Se añadió la guía del usuario para trabajos híbridos de Amazon Braket y la API.	29 de noviembre de 2021
Nuevo dispositivo Rigetti	Se ha agregado compatibilidad para Rigetti Aspen-10	20 de noviembre de 2021
Retirada de dispositivo D-Wave	Se interrumpió la compatibilidad con la QPU de D-Wave, Advantage_system1.	4 de noviembre de 2021
Nuevo dispositivo D-Wave	Se añadió compatibilidad con una QPU de D-Wave adicional , Advantage_system4.	5 de octubre de 2021
Nuevos simuladores de ruido	Se añadió compatibilidad con un simulador de matriz de densidad (DM1), que puede simular circuitos de hasta 17 qubits y un simulador de ruido local braket_dm.	25 de mayo de 2021
PennyLane soporte	Se agregó soporte para PennyLane Amazon Braket	8 de diciembre de 2020

Nuevo simulador	Se añadió compatibilidad con un Tensor Network Simulator (TN1), que permite circuitos más grandes.	8 de diciembre de 2020
Procesamiento por lotes de tareas	Braket admite el procesamiento por lotes de tareas de clientes.	24 de noviembre de 2020
Asignación manual de qubit	Braket admite la asignación manual de qubit en el dispositivo Rigetti.	24 de noviembre de 2020
Cuotas ajustables	Braket admite cuotas ajustables de autoservicio para los recursos de sus tareas.	30 de octubre de 2020
Support para PrivateLink	Puede configurar puntos de conexión de VPC privados para sus trabajos de Braket.	30 de octubre de 2020
Admite etiquetas	Braket admite etiquetas basadas en API para el recurso de tareas cuánticas.	30 de octubre de 2020
Nuevo dispositivo D-Wave	Se añadió compatibilidad con una QPU de D-Wave adicional, Advantage_system1.	29 de septiembre de 2020
Versión inicial	Versión inicial de la documentación de Amazon Braket	12 de agosto de 2020

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.