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? 1
Términos y conceptos de Amazon Braket 3
AWS terminología y consejos para Amazon Braket 7
Precios 8
Seguimiento de costes prácticamente en tiempo real 9
Mejores prácticas para ahorrar costos 11
Funcionamiento 13
Flujo de tareas cuánticas de Amazon Braket 14
Tratamiento de datos por parte de terceros 15
Repositorios y complementos principales de Braket 15
Repositorios principales 15
Complementos 15
Dispositivos compatibles 16
IonQ 21
IQM 21
Rigetti 22
Oxford Quantum Circuits (OQC) 22
QuEra 23
Simulador vectorial de estado local () `braket_sv` 23
Simulador de matriz de densidad local () `braket_dm` 24
Simulador AHS local () `braket_ahs` 25
SV1Simulador vectorial de estados () 25
Simulador de matrices de densidad (DM1) 26
Simulador de redes tensoras () TN1 27
Simuladores integrados 28
Compare los simuladores 29
Regiones y puntos de conexión 33
¿Cuándo se ejecutará mi tarea cuántica? 34
Notificaciones de cambio de estado por correo electrónico o SMS 35
Ventanas de disponibilidad y estado de la QPU 35
Visibilidad de las colas 35
Introducción 38
Activar Amazon Braket 38
Requisitos previos 38

Pasos para activar Amazon Braket 39
Crear una instancia de bloc de notas Amazon Braket 40
Ejecuta tu primer circuito con el Amazon Braket Python SDK 42
Ejecuta tus primeros algoritmos cuánticos 47
Trabaja con Amazon Braket 49
Hola AHS: Ejecute su primera simulación hamiltoniana analógica 50
HA 50
Cadena de espines interactiva 51
Arreglo 52
Interacción 54
Campo de conducción 55
Programa AHS 57
Se ejecuta en un simulador local 58
Analizando los resultados del simulador 58
Se ejecuta en la QPU QuEra Aquila 61
Analizando los resultados de la QPU 63
Next 64
Construya circuitos en el SDK 65
Puertas y circuitos 65
Medición parcial 72
Asignación manual qubit 72
Compilación literal 73
Simulación de ruido 74
Inspeccionar el circuito 76
Tipos de resultados 78
Enviar tareas cuánticas a QPUs y simuladores 82
Ejemplo de tareas cuánticas en Amazon Braket 84
Enviar tareas cuánticas a una QPU 89
Ejecutar una tarea cuántica con el simulador local 91
Agrupación cuántica de tareas 93
Configura las notificaciones de SNS (opcional) 95
Inspeccionar circuitos compilados 96
Ejecute sus circuitos con OpenQASM 3.0 96
¿Qué es OpenQASM 3.0? 97
¿Cuándo usar OpenQASM 3.0 98
¿Cómo funciona OpenQASM 3.0 98

Requisitos previos 98
¿Qué funciones de OpenQASM admite Braket? 98
Cree y envíe un ejemplo de tarea cuántica de OpenQASM 3.0 104
Support para OpenQASM en diferentes dispositivos Braket 107
Simule el ruido con OpenQASM 3.0 119
Qubitrecableado con OpenQASM 3.0 121
Compilación textual con OpenQASM 3.0 121
La consola Braket 122
Más recursos 122
Calcular gradientes con OpenQASM 3.0 122
Medición de qubits específicos con OpenQASM 3.0 123
Envíe un programa analógico utilizando QuEra Aquila 124
Hamiltoniano 124
Esquema del programa AHS de Braket 126
Esquema de resultados de la tarea AHS de Braket 131
QuEra esquema de propiedades del dispositivo 138
Trabajando con Boto3 144
Encienda el cliente Amazon Braket Boto3 144
Configurar AWS CLI perfiles para Boto3 y el SDK de Amazon Braket 148
Control de pulsos en Amazon Braket 151
Braket Pulse 151
Fotogramas 151
Puertos 152
Formas de onda 152
Funciones de los marcos y los puertos 153
Rigetti 153
OQC 155
Hola, Pulse 156
Hello Pulse usando OpenPulse 161
Acceder a las puertas nativas mediante pulsos 168
Empleos híbridos en Amazon Braket 170
¿Qué es un Hybrid Job? 171
Cuándo usar Amazon Braket Hybrid Jobs 171
Ejecute su código local como un trabajo híbrido 172
Cree un trabajo híbrido a partir del código Python local 172
Instalar paquetes y código fuente de Python adicionales 176

Guarde y cargue datos en una instancia de trabajo híbrida 177
Mejores prácticas para decoradores de trabajos híbridos 11
Realice un trabajo híbrido con Amazon Braket Hybrid Jobs 180
Cree su primer trabajo híbrido 182
Configure los permisos 183
Crear y ejecutar 186
Supervisar resultados 190
Entradas, salidas, variables de entorno y funciones auxiliares 192
Entradas 192
Salidas 193
Variables de entorno 194
Funciones auxiliares 195
Guarde los resultados del trabajo 195
Guarde y reinicie los trabajos híbridos mediante puntos de control 197
Defina el entorno para el script de su algoritmo 199
Usar hiperparámetros 201
Configure la instancia de trabajo híbrida para ejecutar el script de su algoritmo 203
Cancelar un trabajo híbrido 206
Uso de la compilación paramétrica para acelerar las tareas híbridas 208
PennyLane Utilízalo con Amazon Braket 209
Amazon Braket con PennyLane 210
Algoritmos híbridos en cuadernos de ejemplo de Amazon Braket 212
Algoritmos híbridos con simuladores integrados PennyLane 212
Activa el gradiente adjunto PennyLane con los simuladores Amazon Braket 213
Utilice Amazon Braket Hybrid Jobs y ejecute un PennyLane algoritmo QAOA 214
Acelere sus cargas de trabajo híbridas con simuladores integrados de PennyLane 217
Utilización `lightning.gpu` para cargas de trabajo de algoritmos de optimización aproximada cuántica 217
Aprendizaje automático cuántico y paralelismo de datos 220
Cree y depure un trabajo híbrido con el modo local 225
Traiga su propio contenedor (BYOC) 225
¿Cuándo es la decisión correcta traer mi propio contenedor? 226
Receta para traer tu propio contenedor 227
Ejecuta las tareas híbridas de Braket en tu propio contenedor 233
Configure el depósito predeterminado en `AwsSession` 233
Interactúe directamente con los trabajos híbridos mediante el API 234

Mitigación de errores 238
Mitigación de errores en IonQ Aria 238
Definición 239
Braket Direct 240
Reservas 240
Crea una reserva 241
Gestiona tu carga de trabajo con una reserva 242
Cancela o reprograma una reserva existente 246
Asesoramiento de expertos 246
Capacidades experimentales 248
Acceso a IonQ Forte solo con reserva 248
Acceso a la sintonización local en Aquila QuEra 249
Acceso a geometrías altas en Aquila QuEra 249
Acceso a geometrías reducidas en Aquila QuEra 250
Registro y supervisión 251
Seguimiento de tareas cuánticas desde el SDK de Amazon Braket 251
Supervisión de tareas cuánticas a través de la consola Amazon Braket 254
Etiquetado de recursos 256
Uso de etiquetas 256
Más información sobre AWS las etiquetas y 257
Recursos compatibles en Amazon Braket 257
Restricciones de las etiquetas 258
Gestión de etiquetas en Amazon Braket 258
Ejemplo de etiquetado CLI en Amazon Braket 259
Etiquetado con la Amazon Braket API 260
Amazon Braket Events con EventBridge 260
Supervise el estado de las tareas cuánticas con EventBridge 261
Ejemplo de evento Amazon Braket EventBridge 262
Monitorea con CloudWatch 264
Métricas y dimensiones de Amazon Braket 264
Dispositivos admitidos 265
Iniciar sesión con CloudTrail 265
Información sobre Amazon Braket en CloudTrail 265
Descripción de las entradas de los archivos de registro de Amazon Braket 266
Cree un cuaderno Braket usando CloudFormation 269
Paso 1: Crear un script de configuración SageMaker del ciclo de vida de Amazon 269

Paso 2: Crear la función de IAM que asume Amazon SageMaker 270
Paso 3: Crea una instancia de Amazon SageMaker Notebook con el prefijo `amazon-braket-` 271
Registro avanzado 272
Seguridad 275
Responsabilidad compartida en materia de seguridad 275
Protección de datos 275
Retención de datos 276
Administrar el acceso a Amazon Braket 277
Recursos de Amazon Braket 277
Cuadernos y funciones 278
Acerca de la política AmazonBraketFullAccess 279
Acerca de la AmazonBraketJobsExecutionPolicy política 284
Restrinja el acceso de los usuarios a determinados dispositivos 287
Amazon Braket actualiza las políticas gestionadas AWS 288
Restrinja el acceso de los usuarios a determinadas instancias de notebook 289
Restrinja el acceso de los usuarios a determinados buckets de S3 290
Rol vinculado a servicio 291
Permisos de rol vinculados a servicios para Amazon Braket 292
Resiliencia 293
Validación de conformidad 294
Seguridad de la infraestructura 294
Seguridad de terceros 295
Puntos de conexión de VPC (PrivateLink) 295
Consideraciones sobre los puntos de conexión de Amazon Braket VPC 296
Configure Braket y PrivateLink 296
Más información sobre la creación de un punto final 298
Controle el acceso con las políticas de puntos finales de Amazon VPC 298
Resolución de problemas 300
AccessDeniedException 300
Se ha producido un error (ValidationException) al llamar a la operación CreateQuantumTask .. 300
Una función del SDK no funciona 301
El trabajo híbrido falla debido a ServiceQuotaExceededException 301
Los componentes dejaron de funcionar en una instancia de bloc de notas 302
Cuotas 302
Cuotas y límites adicionales 349

Solucione los problemas de OpenQASM 349
Incluya un error en la sentencia 350
Error no contiguo qubits 350
Mezclando un error físico qubits con qubits un error virtual 350
Al solicitar los tipos de resultados y medirlos qubits en el mismo programa, se produjo un error 351
Error: se superaron los límites clásicos y de qubit registro 351
El cuadro no va precedido de un error pragmático literal 352
Error en los cuadros literales sin puertas nativas 352
Falta un error físico en las cajas textuales qubits 352
Falta el error «braket» del pragma literal 352
Un solo qubits error no se puede indexar 353
El error físico qubits en una qubit puerta con dos puertas no está conectada 353
GetDevice no devuelve el error de resultados de OpenQASM 354
Advertencia de compatibilidad con simuladores locales 355
Referencia de API y SDK 356
Historial de documentos 357
Glosario de AWS 366
..... ccclxvii

¿Qué es Amazon Braket?

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

AmazonBraket es un sistema totalmente gestionado Servicio de AWS que ayuda a los investigadores, científicos y desarrolladores a iniciarse en la computación cuántica. La computación cuántica tiene el potencial de resolver problemas computacionales que están fuera del alcance de las computadoras clásicas porque aprovecha las leyes de la mecánica cuántica para procesar la información de nuevas maneras.

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

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

- Explore y diseñe algoritmos cuánticos e híbridos.
- Pruebe algoritmos en diferentes simuladores de circuitos cuánticos.
- Ejecute algoritmos en diferentes tipos de ordenadores cuánticos.
- Cree aplicaciones de prueba de concepto.

Definir problemas cuánticos y programar ordenadores cuánticos para resolverlos requiere un nuevo conjunto de habilidades. Para ayudarte a adquirir estas habilidades, Braket ofrece diferentes entornos para simular y ejecutar tus 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: creación, prueba y ejecución:

Compilación: Braket proporciona entornos de portátiles Jupyter totalmente gestionados que facilitan la puesta en marcha. Los cuadernos Braket vienen preinstalados con ejemplos de algoritmos, recursos y herramientas de desarrollo, incluido el SDK de Braket. Amazon Con el SDK de Amazon Braket, puedes crear algoritmos cuánticos y luego probarlos y ejecutarlos en diferentes ordenadores y simuladores cuánticos cambiando una sola línea de código.

Prueba: Braket proporciona acceso a simuladores de circuitos cuánticos de alto rendimiento y totalmente gestionados. Puede probar y validar sus circuitos. Braket gestiona 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).

Run: Braket proporciona un acceso seguro y bajo demanda a diferentes tipos de ordenadores cuánticos. Tiene acceso a ordenadores cuánticos basados en puertas desdeIonQ, y OQCRigetti, así como a un simulador hamiltoniano analógico desde. QuEra Tampoco 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 ningún ordenador cuántico universal y tolerante a fallos. 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 informático. Braket ofrece una variedad de hardware a través de proveedores externos.

El hardware cuántico existente está limitado debido al ruido, que introduce errores. La industria se encuentra en la era de la ruidosa cuántica de escala intermedia (NISQ). En la era 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 informáticos clásicos (tradicionales) con ordenadores cuánticos para crear algoritmos híbridos. Braket le ayuda a trabajar con algoritmos cuánticos híbridos.

En los algoritmos cuánticos híbridos, las unidades de procesamiento cuántico (QPU) se utilizan como coprocesadores para las CPU, lo que acelera los cálculos específicos de un algoritmo clásico. Estos algoritmos utilizan el procesamiento iterativo, en el que la computación se mueve entre ordenadores clásicos y cuánticos. Por ejemplo, las aplicaciones actuales de la computación cuántica en química, optimización y aprendizaje automático 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 aprendizaje automático. 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 cálculos de la teoría de números)
- Optimización: incluye la satisfacción de las restricciones, la resolución de sistemas lineales y el aprendizaje automático
- Computación oracular: incluye la búsqueda, los subgrupos ocultos y la búsqueda de órdenes (por ejemplo, el algoritmo de Grover es el principal método cuántico para los cálculos oráculos)
- Simulación: incluye aplicaciones de simulación directa, invariantes de nudos y algoritmos de optimización cuántica aproximada (QAOA)

Las aplicaciones de estas categorías de cálculos se encuentran en los servicios financieros, la biotecnología, la industria manufacturera y los productos farmacéuticos, por nombrar algunos. 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.

Términos y conceptos de Amazon Braket

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

En Braket se utilizan los siguientes términos y conceptos:

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 el ordenador cuántico está ajustado de tal manera que emula directamente la evolución temporal continua con este hamiltoniano. Los dispositivos AHS suelen ser dispositivos de uso especial y no ordenadores cuánticos universales, como los dispositivos basados en puertas. Están limitados a una clase de hamiltonianos que puedan simular. Sin embargo, dado que estos hamiltonianos se incorporan de forma natural en el dispositivo, el AHS no sufre la sobrecarga necesaria para formular algoritmos como circuitos e implementar operaciones de compuerta.

Soporte

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

Braket: trabajo híbrido

AmazonBraket tiene una función llamada Amazon Braket Hybrid Jobs que proporciona ejecuciones totalmente gestionadas de algoritmos híbridos. Un trabajo híbrido de Braket consta de tres componentes:

1. La definición del algoritmo, que se puede proporcionar como un script, un módulo de Python o un contenedor de Docker.
2. La instancia de trabajo híbrida, basada en Amazon EC2, en la que se ejecutará el algoritmo. La instancia predeterminada es una instancia ml.m5.xlarge.
3. El dispositivo cuántico en el que ejecutar las tareas cuánticas que forman parte del 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, consulta Dispositivos compatibles con Amazon Braket.

Computación cuántica basada en Gate

En la computación cuántica basada en puertas (QC), también denominada QC basada en circuitos, los cálculos se dividen en operaciones elementales (puertas). Ciertos conjuntos de puertas son universales, lo que significa que cada cálculo 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.

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. En las máquinas clásicas, el hamiltoniano de un sistema de cúbits de N se suele representar como una matriz de números complejos de 2^{N} por 2 N. 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 cúbits. Se describe como una forma de onda reproducida en un marco que sirve de soporte a la señal portadora y está vinculada al canal o puerto del hardware. Los clientes pueden diseñar sus propios pulsos proporcionando la envolvente analógica 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 de |0# y |1# del cúbit. Por lo tanto, las compuertas se representan como pulsos con una forma predeterminada y parámetros calibrados, como su amplitud, frecuencia y duración. Los casos de uso que no estén cubiertos por las formas de onda de la plantilla se habilitarán mediante formas de onda personalizadas que se especificarán con la resolución de una sola muestra, 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 cálculo en un ordenador cuántico basado en puertas. Un circuito cuántico es una secuencia de puertas cuánticas, que son transformaciones reversibles en un qubit registro, junto con instrucciones de medición.

Simulador de circuitos cuánticos

Un simulador de circuitos cuánticos es un programa de computadora que se ejecuta en computadoras clásicas 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 circuitos qubits a simular. Braket proporciona acceso a simuladores de circuitos cuánticos gestionados (se accede a través de BraketAPI) 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 cálculos. Existen diferentes paradigmas en la computación cuántica (QC), como el QC basado en puertas.

Unidad de procesamiento cuántico (QPU)

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

Puertas nativas de QPU

El sistema de control QPU puede mapear directamente las puertas nativas de la QPU para controlar los pulsos. Las puertas nativas se pueden ejecutar en el dispositivo QPU sin necesidad de realizar más compilaciones. Subconjunto de puertas compatibles con la QPU. Puedes 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 puedan funcionar directamente en la QPU, lo que significa que es posible que deban descomponerse en puertas nativas. Puedes encontrar las compuertas compatibles de un dispositivo en la página Dispositivos de la consola de Amazon Braket y a través del Amazon SDK de Braket.

Tarea cuántica

En Braket, una tarea cuántica es la solicitud atómica a un dispositivo. En el caso de los dispositivos de control de calidad basados en puertas, esto incluye el circuito cuántico (incluidas las instrucciones de medición y el número de ellosshots) y otros metadatos de solicitud. Puede crear tareas cuánticas mediante el SDK de Amazon Braket o utilizando la CreateQuantumTask API operación directamente. Después de crear una tarea cuántica, se pondrá en cola hasta que el dispositivo solicitado esté disponible. Puedes ver tus tareas cuánticas en la página Quantum Tasks de la consola Amazon Braket o mediante las operaciones GetQuantumTask o SearchQuantumTasksAPI.

Qubit

La unidad básica de información de un ordenador cuántico se denomina bit cuántico, muy parecido a un bit en la computación clásica. qubit A 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 qubit tipos se basan en fotones, espines electrónicos o nucleares o sistemas cuánticos más exóticos.

Queue depth

Queue depthse refiere a la cantidad de tareas cuánticas y trabajos híbridos en espera para un dispositivo en particular. Se puede acceder a las tareas cuánticas y al recuento de colas de trabajos híbridos de un dispositivo a través del Braket Software Development Kit (SDK) quirófano. Amazon Braket Management Console

1. La profundidad de la cola de tareas se refiere al número total de tareas cuánticas que están actualmente en espera de ejecutarse con una prioridad normal.
2. La profundidad de la cola de tareas prioritarias se refiere al número total de tareas cuánticas enviadas en espera de ser ejecutadas. 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 actualmente en cola en un dispositivo. Quantum taskspresentados como parte de un trabajo híbrido tienen prioridad y se agregan en el. Priority Task Queue

Queue position

Queue positionse refiere a la posición actual de su tarea cuántica o trabajo híbrido dentro de la cola de dispositivos correspondiente. Se puede obtener para tareas cuánticas o trabajos híbridos a través del quirófanoBraket Software Development Kit (SDK). 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 disparo. El número de disparos (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 los permisos Servicios de AWS y los recursos. 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 conceder permisos. Si sigue este principio, ayuda 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 segmento específico, especifique el grupo en la política de IAM en lugar de conceder al empleado acceso a todos los grupos de la suya. Cuenta de AWS

Roles de IAM

Un rol de IAM es una identidad que puedes asumir para obtener acceso temporal a los permisos. Antes de que un usuario, una aplicación o un servicio puedan asumir una función de IAM, se le deben conceder permisos para cambiar a esa función. Cuando alguien asume un rol de IAM, abandona todos los permisos anteriores que tenía en el rol anterior y asume los permisos del nuevo rol.

- Práctica recomendada: las funciones 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.

Cubeta 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 un objeto en un bucket de Amazon S3 es de 5 TB. Puede cargar cualquier tipo de datos de archivos a un bucket de Amazon S3, como imágenes, vídeos, archivos de texto, archivos de respaldo, 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 y las políticas de usuario en la documentación de Amazon S3.

Precios de Amazon Braket

> **Tip**
>
> ¡Aprende los fundamentos de la computación cuántica con AWS! Inscríbase en el plan de aprendizaje digital 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 a pedido sin compromiso previo. Solo paga por lo que utiliza. Para obtener más información sobre los precios, visite nuestra página de precios.

Seguimiento de costes prácticamente en tiempo real

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

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

Seguimiento de los costes del SV1

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

Cuando utilice su portátil, podrá esperar el siguiente resultado para su simulación de Bell State. La función de seguimiento le mostrará el número de capturas enviadas, las tareas cuantitativas

completadas, la duración de la ejecución, la duración de la ejecución facturada y su coste máximo en USD. El tiempo de ejecución puede variar para cada simulación.

```
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()
$0.00375
```

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 tengas un límite máximo de cuánto quieres 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. En el siguiente ejemplo, se utiliza el mismo circuito de una Rigetti QPU y se limita el coste a 1 USD. El coste de ejecutar una iteración del circuito en nuestro código es de 0,37 USD. Hemos establecido la lógica para repetir las iteraciones hasta que el coste total supere 1 USD; por lo tanto, el fragmento de código se ejecutará tres veces hasta que la siguiente iteración supere 1 USD. Por lo general, un programa seguiría iterándose hasta alcanzar el coste máximo deseado, en este caso, tres iteraciones.

```
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-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/Aspen-M-3': {'shots': 600, 'tasks':
 {'COMPLETED': 3}}}
1.11 USD
```

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

Mejores prácticas para ahorrar costos

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

Verifica con simuladores

- Verifique sus circuitos con un simulador antes de ejecutarlo en una QPU, de modo que pueda ajustar su circuito sin incurrir en cargos por el uso de la QPU.
- Si bien es posible que los resultados de hacer funcionar el circuito en un simulador no sean idénticos a los de ejecutar el circuito en una QPU, puedes identificar los errores de codificación o los problemas de configuración con un simulador.

Restrinja 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.
- Te recomendamos que no utilices tu cuenta de administrador para conceder o restringir el acceso de los usuarios a los dispositivos Amazon Braket.

Configura alarmas de facturación

- Puedes configurar una alarma de facturación para que te notifique cuando tu factura alcance un límite preestablecido. La forma recomendada de configurar una alarma es a través de AWS Budgets. Puede establecer presupuestos personalizados y recibir alertas cuando los costes o el uso superen el importe presupuestado. La información está disponible en. AWS Budgets

Pruebe tareas TN1 cuánticas con un bajo número de disparos

- Los simuladores cuestan menos que los QHP, pero algunos simuladores pueden resultar caros si las tareas cuánticas se ejecutan con un elevado número de disparos. Le recomendamos que pruebe sus TN1 tareas con un recuento bajo. shot Shotel recuento no afecta al coste ni a SV1 las tareas del simulador local.

Compruebe si hay tareas cuánticas en todas las regiones

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

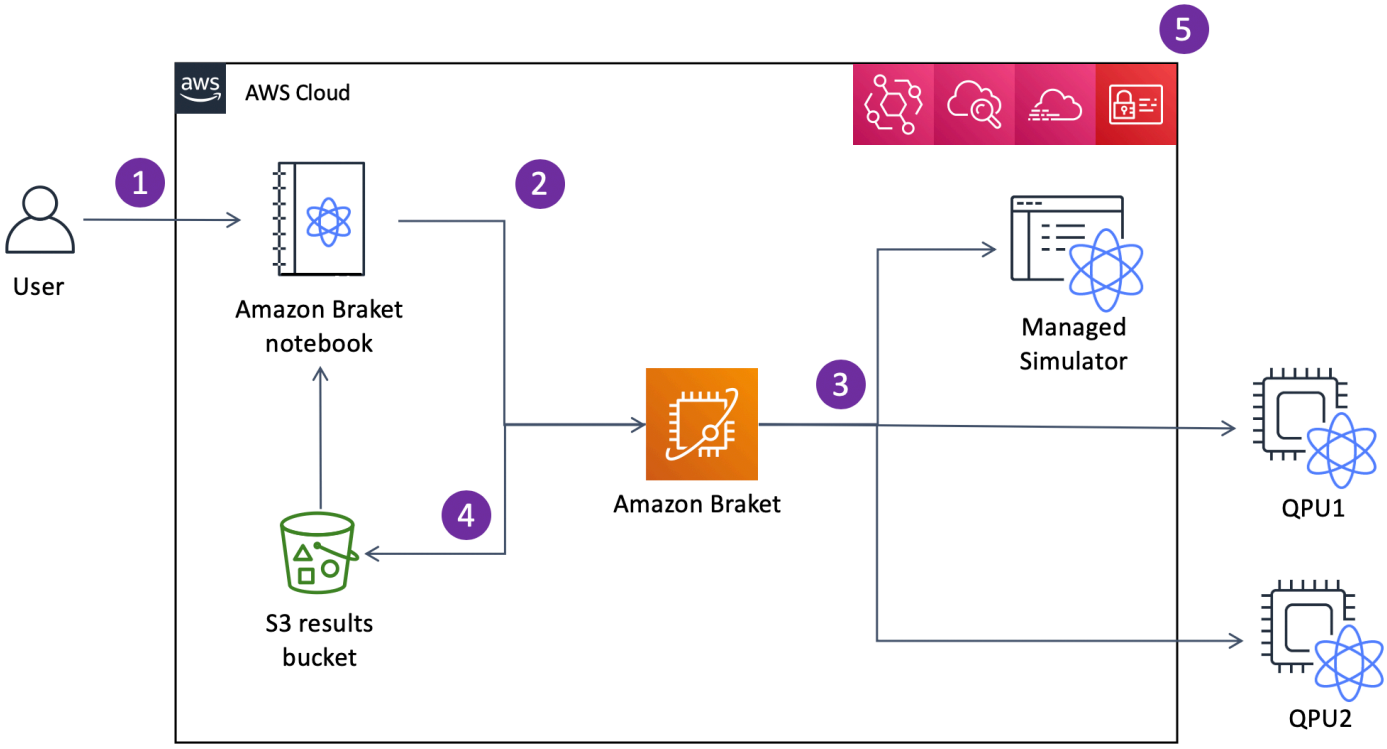
Cómo funciona Amazon Braket

> **Tip**
>
> ¡Aprende los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital 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 QPUs. En Amazon Braket, la solicitud atómica a un dispositivo es una tarea cuántica. En el caso de los dispositivos de control de calidad 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.

En esta sección, vamos a aprender sobre el flujo de alto nivel de ejecución de tareas cuánticas en Braket. Amazon

Flujo de tareas cuánticas de Amazon Braket



Con las Jupyter libretas, puede definir, enviar y supervisar cómodamente sus tareas cuánticas desde la consola Amazon Braket o mediante el Amazon Braket SDK. Puede crear sus circuitos cuánticos directamente en el SDK. Sin embargo, en el caso de los simuladores hamiltonianos analógicos, usted define el diseño de los registros y los campos de control. 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 da acceso a distintos tipos de QPUs (IonQ,,,Rigetti) Oxford Quantum Circuits (OQC)QuEra, tres simuladores bajo demanda (,,TN1) SV1DM1, dos simuladores locales y un simulador integrado. Para obtener más información, consulta Dispositivos compatibles con Amazon Braket.

Tras procesar su 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 busca los resultados en segundo plano y los carga en el cuaderno Jupyter al finalizar la tarea cuántica. También puedes ver y gestionar tus tareas cuánticas en la página Quantum Tasks de la consola Amazon Braket o mediante el `GetQuantumTask` funcionamiento del Braket. Amazon API

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

Tratamiento de datos por parte de terceros

Las tareas cuánticas que se envían a un dispositivo QPU se procesan en ordenadores cuánticos ubicados 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.

Repositorios y complementos principales de Braket

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

Repositorios principales

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

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

Complementos

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.

Amazon Braket compatible:

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

Tercero (el equipo de Braket monitorea y contribuye):

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

Dispositivos compatibles con Amazon Braket

> **Tip**
>
> ¡Aprenda los fundamentos de la computación cuántica con! AWS Inscríbase en el plan de aprendizaje digital 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 QPU o un simulador al que puedes llamar para ejecutar tareas cuánticas. Amazon Braket proporciona acceso a los dispositivos QPU desdeIonQ,, IQM Oxford Quantum CircuitsQuEra, y tres simuladores bajo demandaRigetti, tres simuladores locales y un simulador integrado. Para todos los dispositivos, puedes encontrar más propiedades del dispositivo, como la topología del dispositivo, los datos de calibración y los conjuntos de puertas nativos, en la pestaña Dispositivos de la consola Amazon Braket o mediante la `GetDevice` API. Al construir un circuito con los simuladores, Amazon Braket requiere actualmente que utilices qubits o índices contiguos. Si trabaja con el SDK de Amazon Braket, tiene acceso a las propiedades del dispositivo, tal y como se muestra en el siguiente ejemplo de código.

```
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:us-east-1::device/qpu/ionq/Harmony')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet')
 #IQM Garnet
# device = AwsDevice('arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy')
 #OQC Lucy
# 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/Aspen-M-3')
 #Rigetti Aspen-M-3

# get device properties
device.properties
```

Proveedores de hardware cuántico compatibles

- IonQ
- IQM
- Oxford Quantum Circuits (OQC)
- QuEra Computing

- Rigetti

Simuladores compatibles

- Simulador vectorial de estado local (braket_sv) («Simulador predeterminado»)
- Simulador `braket_dm` de matriz de densidad local ()
- Simulador AHS local
- Simulador vectorial de estados (SV1)
- Simulador de matrices de densidad (DM1)
- Simulador de redes tensoras () TN1
- PennyLanees Lightning Simulators

Elige el mejor simulador para tu tarea cuántica

- Compara simuladores

> **Note**
>
> Para ver los disponibles Regiones de AWS para cada dispositivo, desplázate hacia la derecha por la siguiente tabla.

Dispositivos Amazon Braket

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Región
IonQ	Aria 1	basado en puertas	QPU	arn:aws:braket:us-east-1:: dispositivo/QPU/IONQ/ARIA-1	us-east-1
IonQ	Aria 2	basado en puertas	QPU	arn:aws:braket:us-east-1:: dispositivo/QPU/IONQ/ARIA-2	us-east-1

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Región
IonQ	Forte 1	basado en puertas	QPU (solo con reserva)	arn:aws:braket:us-east-1:: device/QPU/IONQ/FORTE-1	us-east-1
IonQ	Harmony	basado en puertas	QPU	arn:aws:braket:us-east-1:: dispositivo/QPU/IONQ/Harmony	us-east-1
IQM	Garnet	basado en puertas	QPU	arn:aws:braket:eu-north-1:: device/QPU/iQM/Garnet	eu-north-1
Oxford Quantum Circuits	Lucy	basado en puertas	QPU	arn:aws:braket:eu-west-2:: dispositivo/QPU/OQC/Lucy	eu-west-2
QuEra	Aquila	Simulación hamiltoniana analógica	QPU	arn:aws:braket:us-east-1:: device/QPU/Quera/Aquila	us-east-1
Rigetti	Aspen M-3	basado en puertas	QPU	arn:aws:braket:us-west-1:: dispositivo/QPU/Rigetti/Aspen-M-3	us-west-1
AWS	braket_sv	basado en puertas	Simulador local	N/A (simulador local en Braket SDK)	N/A
AWS	braket_dm	basado en puertas	Simulador local	N/A (simulador local en Braket SDK)	N/A

Proveedor	Nombre de dispositivo	Paradigma	Tipo	ARN del dispositivo	Región
AWS	SV1	basado en puertas	Simulador bajo demanda	arn:aws:braket: ::device/quantum-simulator/amazon/sv1	Todas las regiones en las que Braket está disponible. Amazon
AWS	DM1	basado en puertas	Simulador bajo demanda	arn:aws:braket: ::device/quantum-simulator/amazon/dm1	Todas las regiones en las que Braket está disponible. Amazon
AWS	TN1	basado en puertas	Simulador bajo demanda	arn:aws:braket: ::device/quantum-simulator/amazon/tn1	us-west-2, us-east-1 y eu-west-2

> **Note**
>
> Solo se puede acceder a ciertas QPU mediante reservas a través de Braket Direct (consulte Reservas).

Para ver detalles adicionales sobre las QPU que puede usar con Amazon Braket, consulte Amazon Braket Hardware Providers.

IonQ

IonQofrece QPUs basadas en puertas basadas en la tecnología de captura de iones. IonQ'sLas QPU de iones atrapados se construyen 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.

IonQlos dispositivos 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, las IonQ QPU admiten las siguientes puertas nativas.

```
'gpi', 'gpi2', 'ms'
```

Si solo especificas dos parámetros de fase al utilizar la puerta MS nativa, se ejecutará una puerta MS totalmente entrelazada. Una compuerta MS totalmente entrelazada siempre realiza una rotación π /2. Para especificar un ángulo diferente y hacer funcionar una puerta MS que se enrede parcialmente, 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 literal. Para obtener más información sobre la compilación literal, consulte Compilación literal.

IQM

IQMLos procesadores cuánticos son dispositivos universales y tipo puerta basados en qubits transmónicos superconductores. El IQM Garnet dispositivo es un dispositivo de 20 qubits con una topología reticular cuadrada.

Los IQM dispositivos 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 literal, los IQM dispositivos admiten las siguientes puertas nativas.

```
'cz', 'prx'
```

Rigetti

RigettiLos procesadores cuánticos son máquinas universales tipo compuerta basadas en superconductores totalmente ajustables. qubits El Aspen-M-3 dispositivo de 79 qubits aprovecha su tecnología patentada de múltiples chips y se ensambla a partir de 2 procesadores de 40 qubits.

El Rigetti dispositivo 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 literal, los dispositivos Rigetti admiten las siguientes puertas nativas.

```
'rx', 'rz', 'cz', 'cphaseshift', 'xy'
```

RigettiLos procesadores cuánticos superconductores pueden ejecutar la puerta «rx» solo con los ángulos de ±π /2 o ±π.

El control a nivel de pulsos está disponible en los Rigetti dispositivos, que admiten un conjunto de marcos predefinidos de los siguientes tipos:

```
'rf', 'rf_f12', 'ro_rx', 'ro_rx', 'cz', 'cphase', 'xy'
```

Para obtener más información sobre estas tramas, consulte Funciones de las tramas y los puertos.

Oxford Quantum Circuits (OQC)

OQCLos procesadores cuánticos son máquinas universales tipo puerta, construidas con la tecnología escalable de Coaxmon. El OQC Lucy sistema es un 8-qubit dispositivo con la topología de un anillo en el que cada uno qubit está conectado a sus dos vecinos más cercanos.

El Lucy dispositivo admite las siguientes puertas cuánticas.

```
'ccnot', 'cnot', 'cphaseshift', 'cswap', 'cy', 'cz', 'h', 'i', 'phaseshift', 'rx',
 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'v', 'vi', 'x', 'y', 'z', 'ecr'
```

Con la compilación literal, el dispositivo OQC admite las siguientes puertas nativas.

```
'i', 'rz', 'v', 'x', 'ecr'
```

El control a nivel de pulso está disponible en los dispositivos. OQC Los OQC dispositivos admiten un conjunto de marcos predefinidos de los siguientes tipos:

```
'drive', 'second_state', 'measure', 'acquire', 'cross_resonance',
 'cross_resonance_cancellation'
```

OQClos dispositivos admiten la declaración dinámica de tramas siempre que se proporcione un identificador de puerto válido. Para obtener más información sobre estas tramas y puertos, consulte Funciones de las tramas y los puertos.

> **Note**
>
> Al utilizar el control de impulsos conOQC, la duración de los programas no puede superar un máximo de 90 microsegundos. La duración máxima es de aproximadamente 50 nanosegundos para las puertas de un solo qubit y 1 microsegundo para las puertas de dos qubits. Estos números pueden variar según los qubits utilizados, la calibración actual del dispositivo y la compilación del circuito.

QuEra

QuEraofrece 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.

Se pueden programar estos dispositivos según el paradigma de la simulación hamiltoniana analógica, determinando el diseño del registro de qubits y la dependencia temporal y espacial de los campos de manipulación. Amazon Braket proporciona utilidades para crear dichos programas a través del módulo AHS del SDK de Python,. `braket.ahs`

Para obtener más información, consulte los ejemplos de cuadernos de simulación hamiltoniana analógica o la página Enviar un programa analógico utilizando Aquila. QuEra

Simulador vectorial de estado local () **`braket_sv`**

El simulador vectorial de estado local (`braket_sv`) forma parte del SDK de Amazon Braket y se ejecuta localmente en su entorno. Es ideal para la creación rápida de prototipos en circuitos

pequeños (hasta 25qubits), en función de las especificaciones de hardware de su ordenador portátil Braket o de su entorno local.

El simulador local es compatible con todas las compuertas del SDK de Amazon Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puedes encontrar las compuertas compatibles de un dispositivo en las propiedades del dispositivo.

> **Note**
>
> El simulador local admite funciones avanzadas de OpenQASM que pueden no ser compatibles con los dispositivos QPU u otros simuladores. Para obtener más información sobre las funciones compatibles, consulte los ejemplos proporcionados en el cuaderno OpenQASM Local Simulator.

Para obtener más información sobre cómo trabajar con simuladores, consulta 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 y se ejecuta localmente en su entorno. Es ideal para la creación rápida de prototipos en circuitos pequeños con ruido (hasta 12qubits), según las especificaciones de hardware de la instancia de su portátil Braket o de su entorno local.

Puede construir circuitos ruidosos habituales desde cero mediante operaciones de ruido de compuerta, como el cambio de bits y el error de despolarización. También puede aplicar operaciones de ruido a circuitos específicos qubits y a compuertas de circuitos existentes que estén diseñados para funcionar tanto con ruido como sin él.

El simulador `braket_dm` local puede proporcionar los siguientes resultados, dado el número especificado deshots:

- Matriz de densidad reducida: Shots = 0

> **Note**
>
> El simulador local es compatible con las funciones avanzadas de OpenQASM, que pueden no ser compatibles con los dispositivos QPU u otros simuladores. Para obtener más

información sobre las funciones compatibles, consulte los ejemplos proporcionados en el cuaderno OpenQASM Local Simulator.

Para obtener más información sobre el simulador de matriz de densidad local, consulte el ejemplo introductorio del simulador de ruido Braket.

Simulador AHS local () `braket_ahs`

El simulador AHS (simulación hamiltoniana analógica) local (`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 la creación de prototipos en registros pequeños (de hasta 10 a 12 átomos), en función de las especificaciones de hardware de su ordenador portátil Braket o de su entorno local.

El simulador local es compatible con los programas AHS con un campo de conducción uniforme, un campo de cambio (no uniforme) y una disposición de átomos arbitraria. Para obtener más información, consulte la clase Braket AHS y el esquema del programa Braket AHS.

Para obtener más información sobre el simulador AHS local, consulte la página Hello AHS: ejecute su primera simulación hamiltoniana analógica y los cuadernos de ejemplos de simulación hamiltoniana analógica.

SV1Simulador vectorial de estados ()

SV1es un simulador vectorial de estado universal, de alto rendimiento y bajo demanda. Puede simular circuitos de hasta 34qubits. Es de esperar que un 34-qubit circuito denso y cuadrado (profundidad del circuito = 34) tarde aproximadamente entre 1 y 2 horas en completarse, según el tipo de compuertas utilizadas y otros factores. Los circuitos con all-to-all compuertas son muy adecuados paraSV1. Devuelve los resultados en formas tales como un vector de estado completo o una matriz de amplitudes.

SV1tiene 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.

SV1resultados

SV1puede proporcionar los siguientes resultados, dado el número especificado deshots:

- Muestra: Shots > 0
- Expectativa: Shots >= 0
- Varianza: >= 0 Shots
- Probabilidad: > 0 Shots
- Amplitud: Shots = 0
- Gradiente adjunto: Shots = 0

Para obtener más información sobre los resultados, consulte Tipos de resultados.

SV1está siempre disponible, ejecuta sus circuitos a pedido 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, visita Comparar simuladores.

Los simuladores son compatibles con todos los componentes del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puedes encontrar las compuertas compatibles de un dispositivo en las propiedades del dispositivo.

Simulador de matrices de densidad (DM1)

DM1es un simulador de matrices de densidad de alto rendimiento y bajo demanda. Puede simular circuitos de hasta 17qubits.

DM1tiene un tiempo de ejecución máximo de 6 horas, un valor predeterminado de 35 tareas cuánticas simultáneas y un máximo de 50 tareas cuánticas simultáneas.

DM1resultados

DM1puede proporcionar los siguientes resultados, dado el número especificado deshots:

- Muestra: Shots > 0
- Expectativa: Shots >= 0
- Varianza: >= 0 Shots
- Probabilidad: > 0 Shots
- Matriz de densidad reducida: Shots = 0, hasta un máximo de 8 qubits

Para obtener más información sobre los resultados, consulte Tipos de resultados.

DM1está siempre disponible, ejecuta sus circuitos a pedido 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 Comparar simuladores.

Limitaciones y barreras de ruido

```
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 redes tensoras () TN1

TN1es un simulador de redes tensoriales de alto rendimiento y bajo demanda. TN1puede simular ciertos tipos de circuitos con hasta 50 qubits y una profundidad de circuito igual o inferior a 1000. 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 TN1 simulación, TN1 no intente contraerla.

TN1tiene un límite de tiempo de ejecución de 6 horas. Está limitado a un máximo de 10 tareas cuánticas simultáneas (5 en eu-west-2).

TN1resultados

La fase de contracción consiste en 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 Tipos de resultados.

TN1está siempre disponible, ejecuta sus circuitos a pedido y puede ejecutar varios circuitos en paralelo. Para obtener más información, consulte Comparar simuladores.

Los simuladores son compatibles con todos los componentes del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puedes encontrar las compuertas compatibles de un dispositivo en las propiedades del dispositivo.

Visita el GitHub repositorio de Amazon Braket para ver un ejemplo de portátil TN1 que te ayude a empezar. TN1

Mejores prácticas para trabajar con TN1

- Evite all-to-all los circuitos.
- Pruebe un circuito o clase de circuitos nuevos con un número pequeño deshots, para conocer la «dureza» del circuitoTN1.
- Divida shot simulaciones de gran tamaño en varias tareas cuánticas.

Simuladores integrados

Los simuladores integrados funcionan integrando la simulación con el código del algoritmo en el mismo contenedor y ejecutando la simulación directamente en la instancia de trabajo híbrida. Esto

puede resultar útil para eliminar los cuellos de botella asociados a la comunicación de la simulación con un dispositivo remoto. Esto puede reducir considerablemente el uso de memoria, reducir el número de ejecuciones de circuitos para lograr el resultado deseado y mejorar el rendimiento diez veces o más. Para obtener más información sobre los simuladores integrados, consulte la página Ejecute un trabajo híbrido con Amazon Braket Hybrid Jobs.

PennyLanees simuladores de relámpagos

Puedes usar los simuladores PennyLane de relámpagos como simuladores integrados en Braket. Con PennyLane los simuladores de relámpagos, puedes aprovechar los métodos avanzados de cálculo de gradientes, como la diferenciación contigua, para evaluar los gradientes más rápido. El simulador lightning.qubit está disponible como dispositivo mediante Braket NBIs y como simulador integrado, mientras que el simulador lightning.gpu debe ejecutarse como un simulador integrado con una instancia de GPU. Consulte los simuladores integrados del cuaderno Braket Hybrid Jobs para ver un ejemplo del uso de lightning.gpu.

Compare los simuladores

Esta sección le ayuda a seleccionar el simulador Amazon Braket que mejor se adapte a su tarea cuántica, describiendo algunos conceptos, limitaciones y casos de uso.

Elegir 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 Braket Notebook, 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 sobrecarga de latencia por tarea cuántica o lote de tareas cuánticas. Esto puede implicar una compensación si se trata de muchas tareas cuánticas. Dadas estas características generales de rendimiento, la siguiente guía puede ayudarle a elegir cómo ejecutar las simulaciones, incluidas las que tienen ruido.

Para simulaciones:

- Si emplea a menos de 18 personasqubits, utilice un simulador local.
- Si emplea entre 18 y 24 añosqubits, elija un simulador en función de la carga de trabajo.
- Si emplea a más de 24 personasqubits, utilice un simulador a pedido.

Para simulaciones de ruido:

- Si emplea menos de 9qubits, utilice un simulador local.
- Cuando emplee de 9 a 12 añosqubits, elija un simulador en función de la carga de trabajo.
- Cuando emplee a más de 12qubits, utilice. DM1

¿Qué es un simulador de vectores de estado?

SV1es un simulador de vectores de estado universal. Almacena la función de onda completa del estado cuántico y aplica secuencialmente las operaciones de compuerta al estado. Almacena todas las posibilidades, incluso las más improbables. El tiempo de ejecución del SV1 simulador para una tarea cuántica aumenta linealmente con el número de puertas en el circuito.

¿Qué es un simulador de matriz de densidad?

DM1simula circuitos cuánticos con ruido. Almacena la matriz de densidad completa del sistema y aplica secuencialmente las compuertas y las operaciones de ruido del circuito. La matriz de densidad final contiene información completa sobre el estado cuántico una vez que se ejecuta el circuito. El tiempo de ejecución generalmente se escala linealmente con el número de operaciones y exponencialmente con el número de. qubits

¿Qué es un simulador de redes tensoriales?

TN1codifica los circuitos cuánticos en un gráfico estructurado.

- Los nodos del grafo consisten en puertas cuánticas, oqubits.
- Los bordes del gráfico representan las conexiones entre las compuertas.

Como resultado de esta estructura, TN1 se pueden encontrar soluciones simuladas para circuitos cuánticos relativamente grandes y complejos.

TN1requiere dos fases

Por lo general, TN1 opera en un enfoque de dos fases para simular la computación cuántica.

- La fase de ensayo: En esta fase, TN1 se busca una forma de recorrer el gráfico de manera eficiente, lo que implica visitar todos los nodos para que pueda obtener la medida que desee. Como cliente, no ve esta fase porque TN1 realiza ambas fases juntas para usted. Completa la primera fase y determina si se debe realizar la segunda fase por sí sola en función de las limitaciones prácticas. Una vez comenzada la simulación, no tiene nada que ver con esa decisión.

- La fase de contracción: esta fase es análoga a la fase de ejecución de un cálculo en un ordenador clásico. La fase consiste en una serie de multiplicaciones de matrices. El orden de estas multiplicaciones tiene un gran efecto en la dificultad del cálculo. Por lo tanto, la fase de ensayo se lleva a cabo primero para encontrar las rutas de cálculo más eficaces a lo largo del gráfico. Cuando encuentre la trayectoria de contracción durante la fase de ensayo, TN1 contrae las compuertas del circuito para obtener los resultados de la simulación.

TN1los gráficos son análogos a un mapa

Metafóricamente, se puede comparar el TN1 gráfico subyacente con las calles de una ciudad. En una ciudad con una cuadrícula planificada, es fácil encontrar una ruta a tu destino mediante un mapa. En una ciudad con calles no planificadas, nombres de calles duplicados, etc., puede resultar difícil encontrar una ruta a tu destino consultando un mapa.

Si TN1 no realizas la fase de ensayo, sería como caminar por las calles de la ciudad para encontrar tu destino, en lugar de mirar primero un mapa. Pasar más tiempo mirando el mapa puede dar sus frutos en términos de tiempo de caminata. Del mismo modo, la fase de ensayo proporciona información valiosa.

Se podría decir que TN1 tiene cierta «conciencia» de la estructura del circuito subyacente por el que atraviesa. Adquiere esta conciencia durante la fase de ensayo.

Tipos de problemas más adecuados para cada uno de estos tipos de simuladores

SV1es ideal para cualquier clase de problemas que se basen principalmente en tener un número determinado de puertas qubits y compuertas. Por lo general, el tiempo necesario crece de forma lineal con el número de puertas, aunque no depende del número de ellas. shots SV1es generalmente más rápido que TN1 para los circuitos de menos de 28qubits.

SV1puede ser más lento para qubit números más altos porque en realidad simula todas las posibilidades, incluso las más improbables. No tiene forma de determinar qué resultados son probables. Por lo tanto, para una 30-qubit evaluación, SV1 debe calcular 2^30 configuraciones. El límite de 34 qubits para el SV1 simulador Amazon Braket es una restricción práctica debido a las limitaciones de memoria y almacenamiento. Puedes pensarlo así: cada vez que añades un qubit aSV1, el problema se vuelve el doble de difícil.

Para muchas clases de problemas, TN1 puede evaluar circuitos mucho más grandes en un tiempo realista que SV1 porque TN1 aprovecha la estructura de la gráfica. Básicamente, hace un seguimiento de la evolución de las soluciones desde su punto de partida y conserva solo

las configuraciones que contribuyen a un recorrido eficiente. Dicho de otro modo, guarda las configuraciones para crear un orden de multiplicación de matrices que resulta en un proceso de evaluación más simple.

TN1Pues el número de compuertas qubits y es importante, pero la estructura de la gráfica importa mucho más. Por ejemplo, TN1 es muy bueno para evaluar circuitos (gráficos) en los que las compuertas son de corto alcance (es decir, cada una qubit está conectada por compuertas únicamente a su vecina más cercanaqubits) y circuitos (gráficos) en los que las conexiones (o compuertas) tienen un alcance similar. Un rango típico TN1 es hacer que cada una se qubit comunique solo con otras qubits que estén a 5 qubits minutos de distancia. Si la mayor parte de la estructura se puede descomponer en relaciones más simples como estas, que se pueden representar en matrices más pequeñas o más uniformes, TN1 realiza la evaluación fácilmente.

Limitaciones de TN1

TN1puede ser más lento que en SV1 función de la complejidad estructural del gráfico. En algunos gráficos, TN1 finaliza la simulación después de la fase de ensayo y muestra un estado de`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 trayectoria transversal y el simulador abandona el cálculo. TN1no puede realizar la contracción. Es posible que veas 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 coste sería prohibitivo y, en TN1 cambio, se produce 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 o, TN1 aunque no se realice la contracción, y verá un estado. `FAILED`

El tiempo de ejecución previsto también depende del recuento. shot En el peor de los casos, el tiempo de TN1 contracción depende linealmente del recuento. shot El circuito puede contraerse con menos. shots Por ejemplo, puedes enviar una tarea cuántica con 100shots, lo que TN1 se considera

incontratable, pero si vuelves a presentarla con solo 10, la contracción continúa. En este caso, para obtener 100 muestras, podrías enviar 10 tareas cuánticas de 10 shots para el mismo circuito y, al final, combinar los resultados.

Como práctica recomendada, te recomendamos que pruebes siempre tu circuito o clase de circuito con unas cuantas shots (por ejemplo, 10) para saber qué tan duro es el circuitoTN1, antes de proceder con un número mayor deshots.

> **Note**
>
> La serie de multiplicaciones que forma la fase de contracción comienza con matrices pequeñas de NxN. Por ejemplo, una 2-qubit puerta requiere una matriz de 4 x 4. Las matrices intermedias necesarias durante una contracción que se considera demasiado difícil son gigantescas. Un cálculo de este tipo tardaría días en completarse. Por eso Amazon Braket no intenta realizar contracciones extremadamente complejas.

Concurrency (Simultaneidad)

Todos los simuladores Braket permiten ejecutar varios circuitos simultáneamente. Los límites de simultaneidad varían según el simulador y la región. Para obtener más información sobre los límites de simultaneidad, consulta la página de cuotas.

Regiones y puntos de conexión de Amazon Braket

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

Disponibilidad regional de Amazon Braket

Nombre de la región	Región	Braket Endpoint	QPU
Este de EE. UU. (Norte de Virginia)	us-east-1	braket.us-east-1.amazonaws.com	iONQ
Este de EE. UU. (Norte de Virginia)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
Oeste de EE. UU. (Norte de California)	us-west-1	braket.us-west-1.amazonaws.com	Rigetti

Nombre de la región	Región	Braket Endpoint	QPU
EU North 1 (Estocolmo)	eu-north-1	braket.eu-north-1.amazonaws.com	IQM
EU West 2 (Londres)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Puede ejecutar Amazon Braket desde cualquier región en la que esté disponible, pero cada QPU solo está disponible en una región. Las tareas de Quantum que se ejecutan en un dispositivo QPU se pueden ver en la consola Amazon Braket de la región de ese dispositivo. Si utilizas el SDK de Amazon Braket, puedes enviar tareas cuánticas a cualquier dispositivo QPU, independientemente de la región en la que trabajes. El SDK crea automáticamente una sesión en la región para la QPU especificada.

Para obtener información general sobre cómo AWS funciona con las regiones y los puntos finales, consulte los Servicio de AWS puntos finales en la AWS Referencia general.

¿Cuándo se ejecutará mi tarea cuántica?

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

Cuando envías un circuito, Amazon Braket lo envía al dispositivo que especifiques. 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.

Notificaciones de cambio de estado 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 cambio de estado del dispositivo y de las tareas cuánticas por correo electrónico o mensaje SMS:

1. Cree un tema de Amazon SNS y una suscripción a correo electrónico o SMS. La disponibilidad del correo electrónico o los 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

alertas de finalización de tareas de Quantum

Puedes configurar las notificaciones a través del Amazon Simple Notification Service (SNS) para recibir una alerta cuando se complete tu tarea cuántica de Amazon Braket. Las notificaciones activas son útiles si espera un tiempo de espera prolongado, por ejemplo, cuando envía una tarea grande o cuando envía una tarea fuera del período de disponibilidad de un dispositivo. Si no quieres esperar a que se complete la tarea, puedes configurar una notificación de SNS.

Una libreta Amazon Braket te guía por los pasos de configuración. Para obtener más información, consulta el bloc de notas de ejemplo de Amazon Braket para configurar las notificaciones.

Ventanas de disponibilidad y estado de la QPU

La disponibilidad de la QPU varía de un dispositivo a otro.

En la página Dispositivos de la consola Amazon Braket, puedes ver las ventanas de disponibilidad actuales y futuras y el estado del dispositivo. Además, la página de cada dispositivo muestra las profundidades de las colas individuales para 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 fuera de línea 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 de espera de dispositivos.

Profundidad de cola

Queue depthse 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 colas de trabajos híbridos de un dispositivo a través del Braket Software Development Kit (SDK) quirófano. Amazon Braket Management Console

1. La profundidad de la cola de tareas se refiere al número total de tareas cuánticas que están esperando ejecutarse con una prioridad normal.
2. La profundidad de la cola de tareas prioritarias se refiere al número total de tareas cuánticas enviadas en espera de ser ejecutadas. 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 actualmente en cola en un dispositivo. Quantum taskspresentados como parte de un trabajo híbrido tienen prioridad y se agregan en el. Priority Task Queue

Los clientes que deseen ver la profundidad de la cola mediante el 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/Harmony")

# 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 estanque. `QUEUED` Amazon Braket ofrece a los clientes visibilidad de sus tareas cuánticas y de su posición en la cola de trabajos híbridos.

Posición en la cola

Queue positionse refiere a la posición actual de su tarea cuántica o trabajo híbrido dentro de la cola de dispositivos correspondiente. Se puede obtener para tareas cuánticas o trabajos híbridos a través del quirófanoBraket Software Development Kit (SDK). Amazon Braket Management Console

Los clientes que deseen ver la posición de la cola mediante el Braket SDK pueden modificar el siguiente fragmento de código para obtener la posición de cola de su tarea cuántica o trabajo híbrido:

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

#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:us-east-1::device/qpu/ionq/Harmony",
    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
```

Empieza a usar Amazon Braket

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

Una vez que hayas seguido las instrucciones de Enable Amazon Braket, podrás empezar Amazon a utilizar Braket.

Los pasos para empezar incluyen:

- Activar Amazon Braket
- Crear una instancia de bloc de notas Amazon Braket
- Ejecuta tu primer circuito con el Amazon Braket Python SDK
- Ejecuta tus primeros algoritmos cuánticos

Activar Amazon Braket

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

Puedes activar Amazon Braket en tu cuenta a través de la AWS consola.

Requisitos previos

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

> **Note**
>
> Si es 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 usar Amazon Braket, consulta Administrar el acceso a Amazon Braket.

Pasos para activar Amazon Braket

1. Inicia sesión en la consola Amazon Braket con tu. Cuenta de AWS
2. Abre la consola Amazon Braket.
3. En la página de inicio de Braket, haga clic en Comenzar para ir a la página del panel de control del servicio. La alerta que aparece en la parte superior del panel de servicio le guiará por los tres pasos siguientes:
   a. Crear funciones vinculadas a servicios (SLR)
   b. Permitir el acceso a ordenadores cuánticos de terceros
   c. Crear una nueva instancia de Jupyter Notebook

Para poder utilizar dispositivos cuánticos de terceros, debes aceptar ciertas condiciones en relación 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 de permisos y configuración de la consola Amazon Braket.

> **Note**
>
> Los dispositivos Quantum en los que no intervengan terceros, como los simuladores locales Braket o los simuladores bajo demanda, se pueden utilizar sin necesidad de aceptar el acuerdo de activación de dispositivos de terceros.
> La aceptación de estos términos para permitir el uso de dispositivos de terceros solo debe hacerse una vez por cuenta si accedes a hardware de terceros.

Crear una instancia de bloc de notas Amazon Braket

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

Amazon Braket ofrece cuadernos Jupyter totalmente gestionados para que puedas empezar. Las instancias de Amazon Braket notebook se basan en las instancias de Amazon SageMaker Notebook. Las siguientes instrucciones describen los pasos necesarios para crear una nueva instancia de bloc de notas para clientes nuevos y existentes.

Nuevos clientes de Amazon Braket

1. Abre la consola Amazon Braket y dirígete a la página del panel izquierdo del panel izquierdo.
2. Haz clic en Comenzar en el modal Bienvenido a Amazon Braket, ubicado en el centro de la página de tu panel de control, para proporcionar un nombre para la libreta. Esto creará una libreta Jupyter predeterminada.
3. Crear tu bloc de notas puede tardar varios minutos. Su bloc de notas aparecerá en la página de blocs de notas con el estado Pendiente. Cuando la instancia de tu bloc de notas esté lista para usarse, el estado cambiará a InService. Puede que tengas que actualizar la página para mostrar el estado actualizado del bloc de notas.

Clientes actuales de Amazon Braket

1. Abre la consola Amazon Braket, selecciona Notebooks en el panel izquierdo y selecciona Crear instancia de bloc de notas. Si no tiene libretas, seleccione la configuración estándar para crear una libreta Jupyter predeterminada e introduzca el nombre de una instancia de Notebook utilizando únicamente caracteres alfanuméricos y guiones y seleccione el modo visual que prefiera. A continuación, active o desactive el administrador de inactividad de su bloc de notas.
   a. Si está activado, seleccione el tiempo de inactividad deseado antes de restablecer el portátil. Cuando se reinicie un ordenador portátil, los gastos de procesamiento dejarán de generarse, pero los gastos de almacenamiento se mantendrán.

b. Para ver el tiempo de inactividad restante en la instancia de tu notebook, navega hasta la barra de comandos y selecciona la pestaña Braket, seguida de la pestaña Administrador de inactividad.

> **Note**
>
> Para evitar que tu trabajo se pierda, considera la posibilidad de integrar tu instancia de SageMaker notebook con un repositorio de git. Como alternativa, si mueves tu trabajo fuera de las `/Braket Examples` carpetas `/Braket Algorithms` y, evitarás que el archivo se sobrescriba al reiniciar la instancia del bloc de notas.

2. (Opcional) Con la configuración avanzada, puedes crear un bloc de notas con permisos de acceso, configuraciones adicionales y ajustes de acceso a la red:
   a. En la configuración de Notebook, elija el tipo de instancia. 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 SageMaker los precios de Amazon. Si quieres asociar un repositorio público de Github a tu instancia de notebook, haz clic en el menú desplegable del repositorio de Git y selecciona Clonar un repositorio de git público desde la URL en el menú desplegable Repositorio. Introduce la URL del repositorio en la barra de texto URL del repositorio de Git.
   b. En Permisos, configura las funciones de IAM, el acceso raíz y las claves de cifrado opcionales.
   c. En Red, configura los ajustes de acceso y red personalizados para tu Jupyter Notebook instancia.
3. Revisa la configuración, establece cualquier etiqueta para identificar la instancia de tu bloc de notas y haz clic en Iniciar.

> **Note**
>
> Puedes ver y gestionar las instancias de tu bloc de notas Amazon Braket en las consolas Amazon Braket y SageMaker Amazon. Los ajustes adicionales del portátil Amazon Braket están disponibles a través de la SageMaker consola.

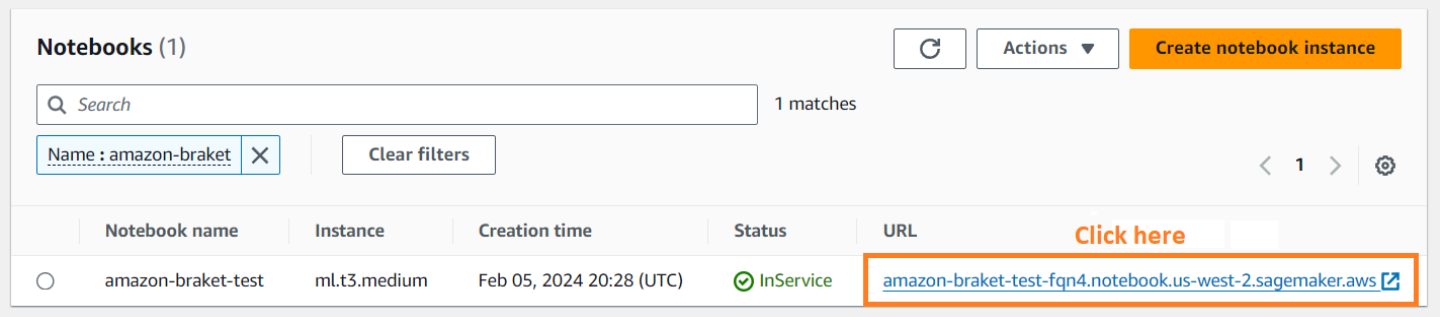
Si estás trabajando en la consola de Amazon Braket, el SDK de Amazon Braket y AWS los complementos vienen precargados en las libretas que has creado. Si desea ejecutarla en su propia máquina, puede instalar el SDK y los complementos cuando ejecute el comando `pip`

`install amazon-braket-sdk` o cuando ejecute el comando `pip install amazon-braket-pennylane-plugin` para usarlo con los complementos. PennyLane

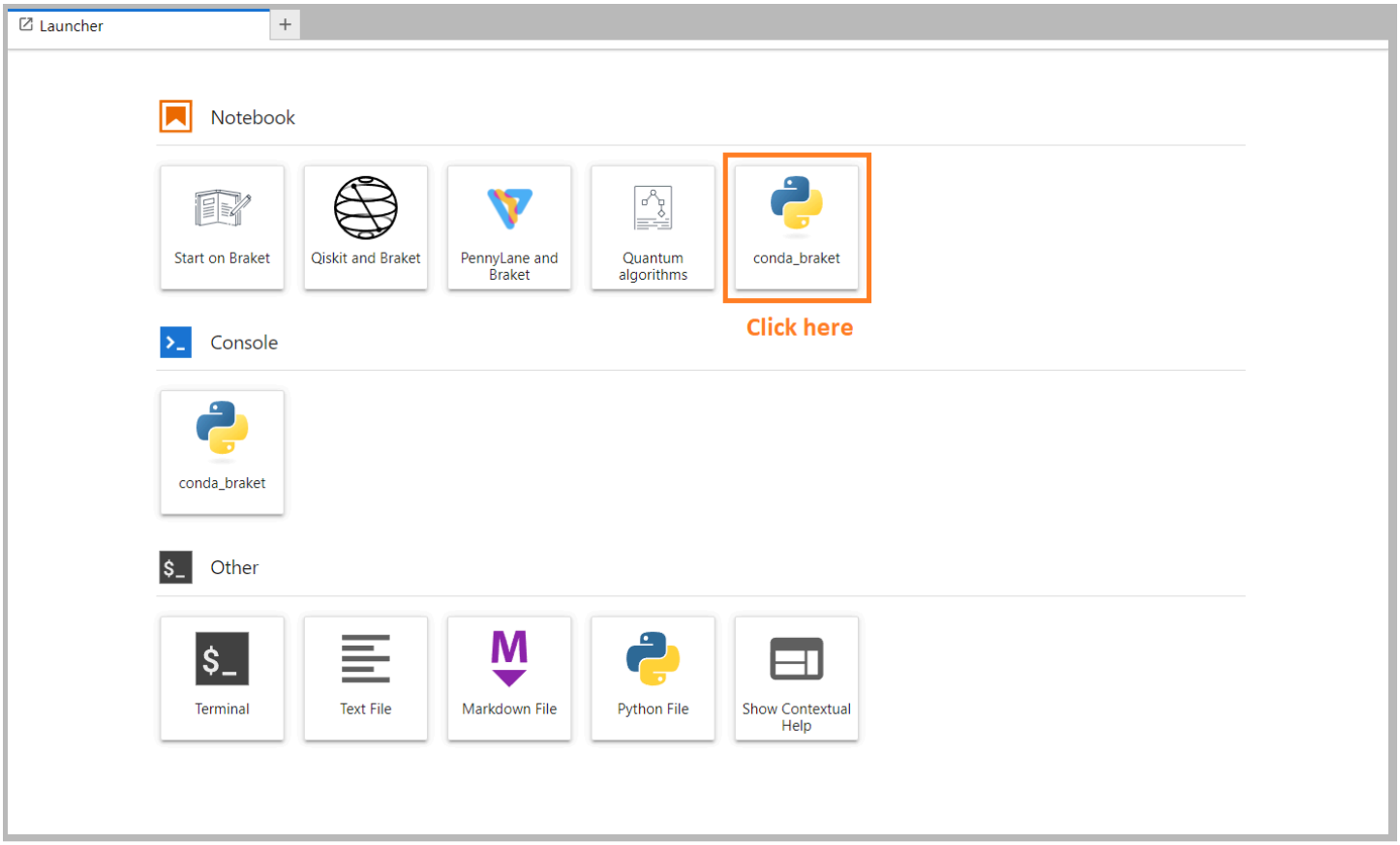
Ejecuta tu primer circuito con el Amazon Braket Python SDK

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

Una vez lanzada la instancia de bloc de notas, ábrela con una interfaz estándar de Jupyter y elige la libreta que acabas de crear.



AmazonLas instancias de Braket Notebook vienen preinstaladas con el SDK de Amazon Braket y todas sus dependencias. Comience por crear un nuevo bloc de notas con núcleo. `conda_braket`



Puedes empezar con un simple «¡Hola, mundo!» ejemplo. Primero, construya un circuito que prepare un estado de Bell y, a continuación, ejecute ese circuito en diferentes dispositivos para obtener los resultados.

Comience importando los módulos del SDK de Amazon Braket y definiendo un circuito Bell State simple.

```
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 visualizar el circuito con este comando:

```
print(bell)
```

Ejecute su 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 usar el simulador local para circuitos más pequeños, que pueden ser de hasta 25 qubits (dependiendo del hardware local).

A continuación, te explicamos cómo crear una instancia del simulador local:

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

y ejecuta el circuito:

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

Deberías ver un resultado parecido a este:

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

El estado de Bell específico que ha preparado es una superposición igual de |00> y |11>, y encontrará una distribución aproximadamente igual (hasta el shot ruido) de 00 y 11 como resultados de medición, como era de esperar.

Ejecute su circuito en un simulador bajo demanda

AmazonBraket también proporciona acceso a un simulador de alto rendimiento bajo demanda para ejecutar circuitos más grandes. SV1 SV1es un simulador de vectores 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 ejecutas tareas cuánticas en SV1 (TN1o en cualquier QPU), los resultados de la tarea cuántica se almacenan en un depósito de S3 en tu cuenta. Si no especificas un bucket, el SDK de Braket crea un bucket predeterminado `amazon-braket-{region}-{accountID}` para ti. Para obtener más información, consulta Administrar el acceso a Amazon Braket.

> **Note**
>
> Escribe el nombre de tu bucket actual, donde aparece el siguiente ejemplo `example-bucket` como nombre de tu bucket. Los nombres de los cubos de Amazon Braket siempre comienzan con una letra `amazon-braket-` seguida de los demás caracteres identificativos que añadas. 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 = "example-bucket"
# the name of the folder in the bucket
my_prefix = "simulation-output"
s3_folder = (my_bucket, my_prefix)
```

Para ejecutar un circuitoSV1, debe proporcionar la ubicación del depósito de S3 que seleccionó previamente como argumento posicional en la `.run()` llamada.

```
# 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 Amazon Braket proporciona más información sobre su tarea cuántica. Ve a la pestaña Quantum Tasks de la consola y tu tarea cuántica debería estar en la parte superior de la lista. Como alternativa, puedes buscar tu tarea cuántica utilizando el identificador único de la tarea cuántica u otros criterios.

> **Note**
>
> Transcurridos 90 días, Amazon Braket elimina automáticamente todos los identificadores de tareas cuánticas y demás metadatos asociados a tus tareas cuánticas. Para obtener más información, consulte Retención de datos.

Se ejecuta en una QPU

Con Amazon Braket, puede ejecutar el ejemplo anterior de circuito cuántico en un ordenador cuántico físico simplemente cambiando una línea de código. Amazon Braket proporciona acceso a QPU dispositivos desdeIonQ, Oxford Quantum CircuitsQuEra, y. Rigetti Puedes 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 una instancia de un Rigetti dispositivo.

```
# choose the Rigetti hardware to run your circuit
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")
```

Elige un IonQ dispositivo con este código:

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

Tras seleccionar un dispositivo y antes de ejecutar la carga de trabajo, puede consultar la profundidad de la cola del dispositivo con el siguiente código para determinar el número de tareas cuánticas o híbridas. Además, los clientes pueden ver las profundidades de las colas específicas de cada dispositivo en la página Dispositivos del. 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)
'2' # returns the number of hybrid jobs queued on the device
```

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 `poll_timeout_seconds` parámetro en el `.run()` comando, tal y como se muestra en el siguiente ejemplo. Tenga en cuenta que 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 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)
```

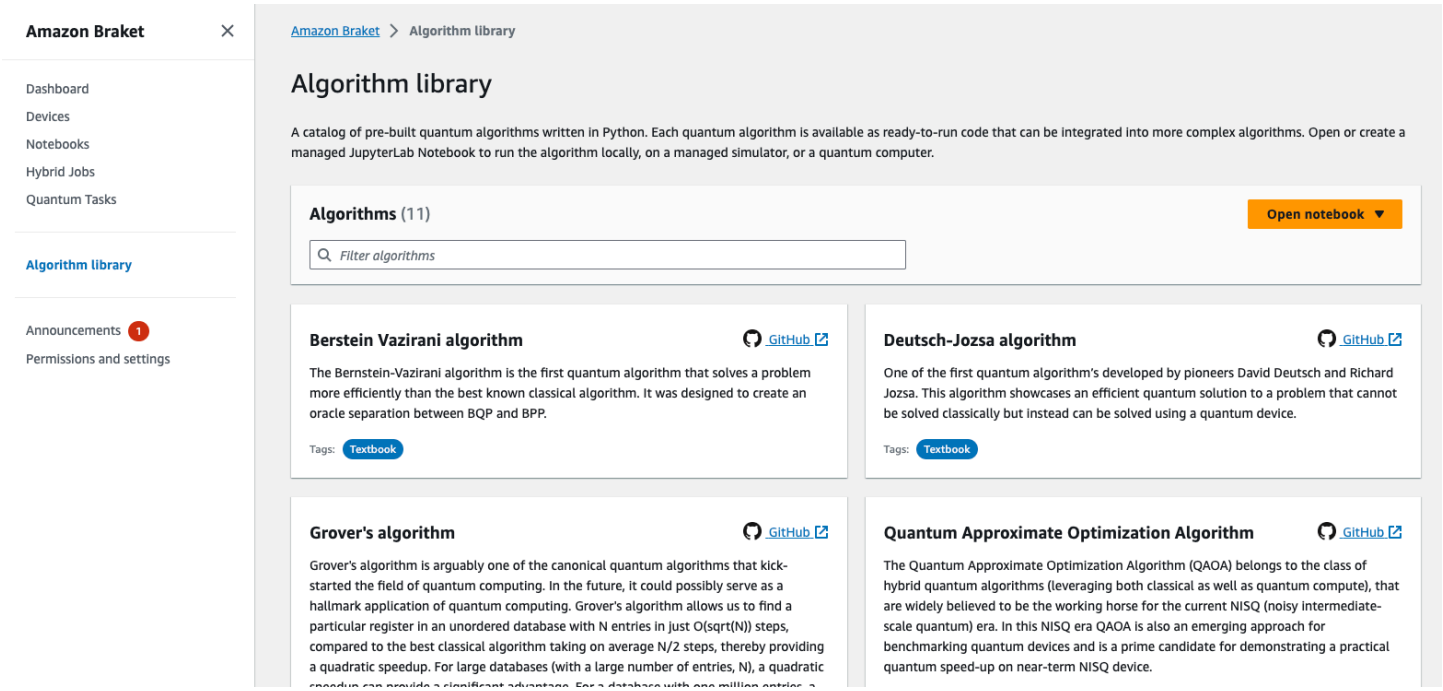
Además, después de enviar su tarea cuántica o híbrida, puede llamar a la `queue_position()` función para comprobar su posición en la cola.

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

Ejecuta tus primeros algoritmos cuánticos

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

La biblioteca de algoritmos Amazon Braket es un catálogo de algoritmos cuánticos prediseñados escritos en Python. Puede ejecutar estos algoritmos tal como están o utilizarlos como punto de partida para crear algoritmos más complejos. Puede acceder a la biblioteca de algoritmos desde la consola de Braket. También puedes acceder a la biblioteca de algoritmos de Braket en Github: https://github.com/aws-samples/amazon-braket-algorithm-library.



La consola 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 elija Abrir bloc de notas para abrir o crear un bloc de notas que contenga todos los algoritmos disponibles. Si elige la opción de bloc de notas, encontrará la biblioteca de algoritmos de Braket en la carpeta raíz del bloc de notas.

Trabaja con Amazon Braket

En esta sección se muestra cómo diseñar circuitos cuánticos, enviar estos problemas como tareas cuánticas a los dispositivos y monitorizar las tareas cuánticas con el Amazon Braket SDK.

Los siguientes son los principales medios de interactuar con los recursos de Amazon Braket.

- La consola Amazon Braket proporciona información y estado del dispositivo para ayudarlo a crear, administrar y monitorear 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 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 al API si está creando aplicaciones que funcionen con la computación cuántica mediante programación.

Los ejemplos de esta sección muestran cómo se puede trabajar con Amazon Braket API directamente mediante el SDK de Amazon Python de Braket junto con el SDK de AWS Python para Braket (Boto3).

Más información sobre el Amazon SDK Braket Python

Para trabajar con el SDK de Python de Amazon Braket, primero instale el SDK de AWS Python para Braket (Boto3) de modo que pueda comunicarse con el. AWS API Puede pensar en el SDK de Python de Amazon Braket como un práctico envoltorio para Boto3 para los clientes de Quantum.

- Boto3 contiene interfaces que necesitas 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 Amazon Braket Python SDK contiene módulos de software para circuitos, puertas, dispositivos, tipos de resultados y otras partes de una tarea cuántica. Cada vez que creas un programa, importas los módulos que necesitas 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 que necesita para ejecutar tareas cuánticas.
- Puede importar módulos del SDK de Python de Amazon Braket a cualquier script de Python si no desea trabajar con cuadernos.

Tras instalar Boto3, se muestra una descripción general de los pasos para crear una tarea cuántica mediante el SDK de Python de Amazon Braket similar a la siguiente:

1. (Opcionalmente) Abre tu bloc de notas.
2. Importe los módulos del SDK que necesite para sus circuitos.
3. Especifique una QPU o un simulador.
4. Cree una instancia del circuito.
5. Ejecute el circuito.
6. Recopila 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:

- Hola AHS: Ejecute su primera simulación hamiltoniana analógica
- Construya circuitos en el SDK
- Enviar tareas cuánticas a QPUs y simuladores
- Ejecute sus circuitos con OpenQASM 3.0
- Envíe un programa analógico utilizando QuEra Aquila
- Trabajando con Boto3

Hola AHS: Ejecute su primera simulación hamiltoniana analógica

HA

La simulación hamiltoniana analógica (AHS) es un paradigma de la computación cuántica diferente de los circuitos cuánticos: en lugar de una secuencia de puertas, cada una de las cuales actúa solo sobre un par de cúbits a la vez, un programa AHS se define mediante los parámetros dependientes del tiempo y el espacio del hamiltoniano en cuestión. El hamiltoniano de un sistema codifica sus niveles de energía y los efectos de las fuerzas externas, que en conjunto rigen la evolución temporal de sus estados.

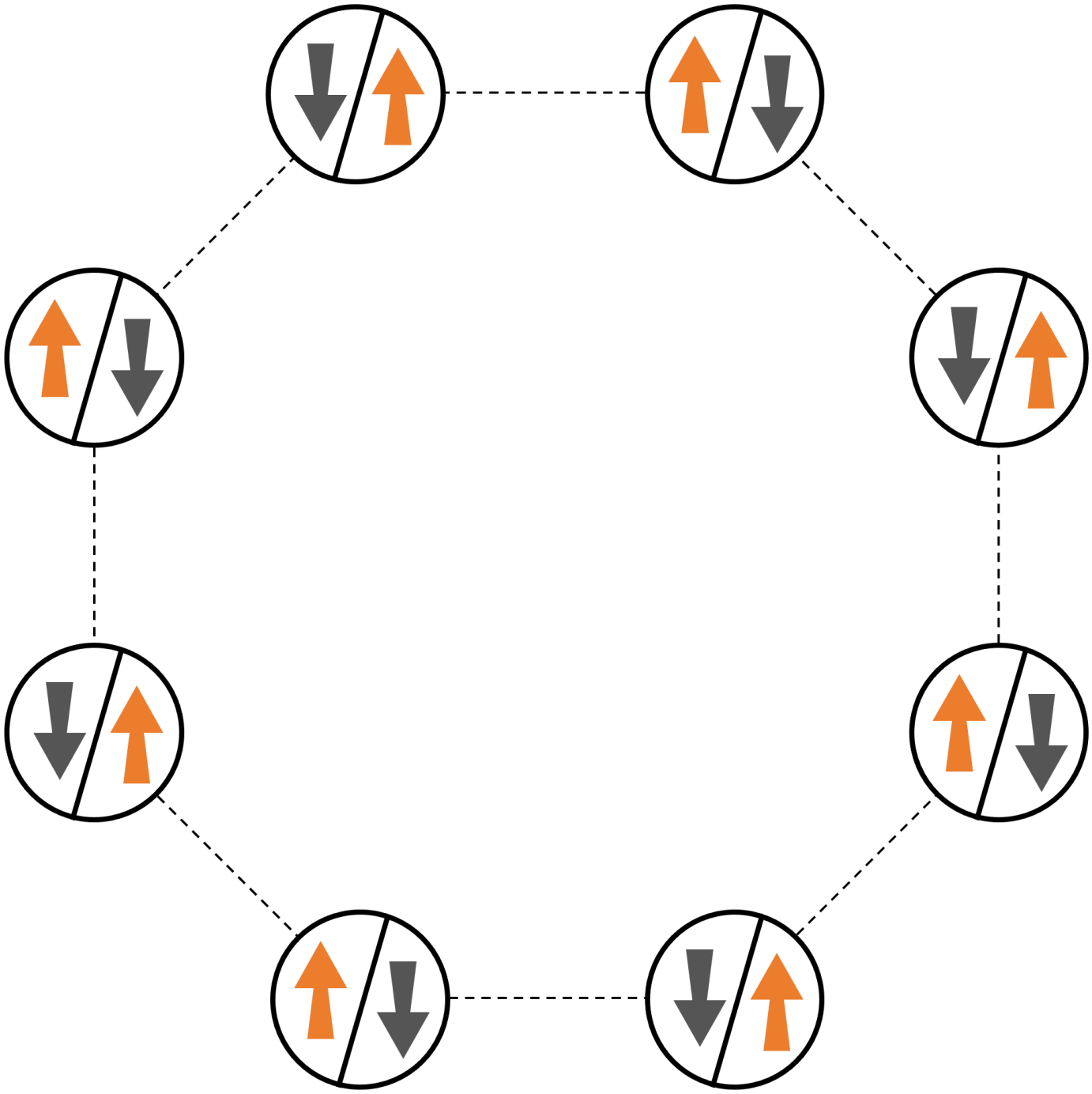
Para un sistema de N cúbits, el hamiltoniano se puede representar mediante una matriz cuadrada de números complejos de 2 N X2 N.

Según el hamiltoniano personalizado, los dispositivos cuánticos capaces de ejecutar el AHS ajustarán sus parámetros (por ejemplo, la amplitud y la desafinación de un campo conductor

coherente) para aproximarse con precisión a la evolución temporal del sistema cuántico. El paradigma AHS es adecuado para simular las propiedades estáticas y dinámicas de los sistemas cuánticos de muchas partículas que interactúan. Las QPUs diseñadas específicamente, como el dispositivo Aquila, QuEra pueden simular la evolución temporal de sistemas con tamaños que de otro modo serían imposibles con un hardware clásico.

Cadena de espines 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»). Si bien es pequeño, este sistema modelo ya presenta un puñado de fenómenos interesantes relacionados con materiales magnéticos de origen natural. En este ejemplo, mostraremos cómo preparar el denominado orden antiferromagnético, en el que los espines consecutivos apuntan en direcciones opuestas.



Arreglo

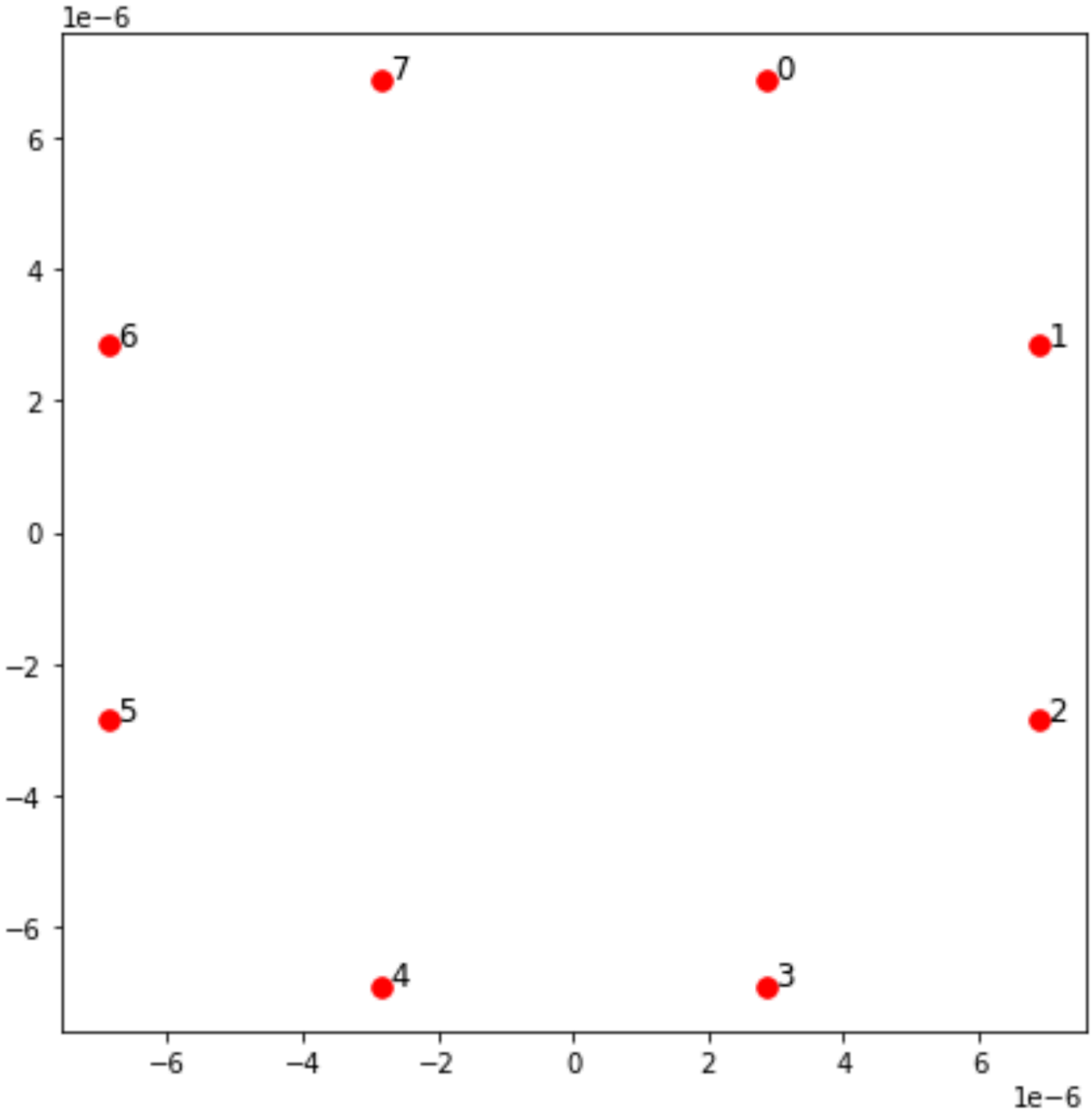
Usaremos un átomo neutro para representar cada giro, y los estados de espín «arriba» y «abajo» se codificarán en el estado de Rydberg excitado y en el estado fundamental de los átomos, respectivamente. En primer lugar, creamos la disposición bidimensional. Podemos programar el anillo de tiradas anterior con el siguiente código.

Requisitos previos: Es necesario instalar el SDK de Braket. (Si utilizas una instancia de notebook alojada en Braket, este SDK viene preinstalado con las notebooks). Para reproducir los gráficos, también es necesario instalar matplotlib por separado con el comando shell. `pip install matplotlib`

```
import numpy as np
import matplotlib.pyplot as plt  # required for plotting

from braket.ahs.atom_arrangement import AtomArrangement

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)
```

con el que también podemos graficar

```
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 Aquila dispositivo de). QuEra Usando la representación del

espín, el término hamiltoniano para esta interacción se puede expresar como la suma de todos los pares de espines (j, k).

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

En este caso, nj=↑ j #↑ j es un operador que toma el valor de 1 solo si el espín j está en el estado «hacia arriba» y 0 en caso contrario. La fuerza es V j,k =C6/(dj,k) 6, donde C 6 es el coeficiente fijo y d es la distancia euclidiana entre los j,k 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 «hacia arriba» tiene una energía elevada (en la cantidad V). j,k Al diseñar cuidadosamente el resto del programa AHS, esta interacción evitará que los giros vecinos estén ambos en estado «positivo», un efecto que se conoce comúnmente como «bloqueo de Rydberg».

Campo de conducción

Al comienzo del programa AHS, todos los giros (por defecto) comienzan en su estado «inactivo», es decir, se encuentran en la denominada fase ferromagnética. Con la vista puesta en nuestro objetivo de preparar la fase antiferromagnética, especificamos un campo impulsor coherente y dependiente del tiempo que hace que los espines pasen sin problemas de este estado a un estado compuesto por varios cuerpos, en el que se prefieren los estados «ascendentes». 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), ω (t) y △ (t) son la amplitud global (también conocida como frecuencia de Rabi), la fase y la desafinación del campo impulsor, que dependen del tiempo y afectan a todos los giros de manera uniforme. En este caso−,k, S k =↓ k ~ ↑ y S +,k =(S−,k) † =↑ k ~ ~ ↓ k son los operadores de subida y bajada del espín k, respectivamente, y n k k =↑ k ↑ es el mismo operador que antes. La parte Ω del campo conductor acopla de forma coherente los estados «hacia abajo» y «hacia arriba» de todos los giros simultáneamente, mientras que la parte λ controla la recompensa de energía en los estados «hacia arriba».

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

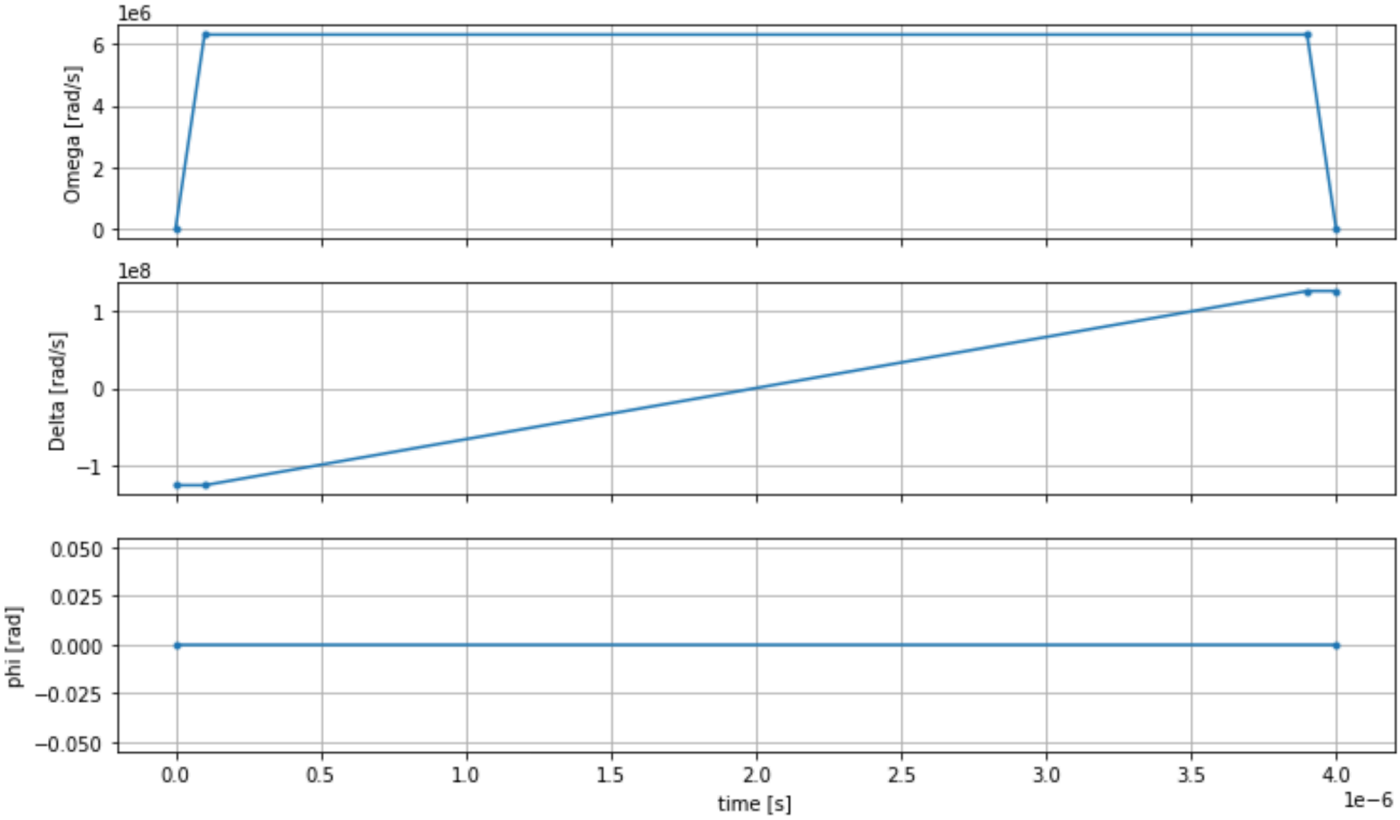
```
from braket.timings.time_series import TimeSeries
```

```
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 conductor 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 AHS

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

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

ahs_program = AnalogHamiltonianSimulation(
    register=register,
```

```
    hamiltonian=drive
)
```

Se ejecuta en un simulador local

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

En este caso, podemos establecer el número de disparos en un valor alto (por ejemplo, 1 millón) porque el simulador local registra la evolución temporal del estado cuántico y extrae muestras del estado final; por lo tanto, el número de disparos aumenta y el tiempo de ejecución total solo se incrementa marginalmente.

```
from braket.devices import LocalSimulator
device = LocalSimulator("braket_ahs")

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

Analizando los resultados del simulador

Podemos agregar los resultados de los tiros con la siguiente función que deduce el estado de cada giro (que puede ser «d» para «abajo», «u» para «arriba» o «e» para un sitio vacío) y cuenta cuántas veces se ha producido cada configuración en los tiros.

```
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
```

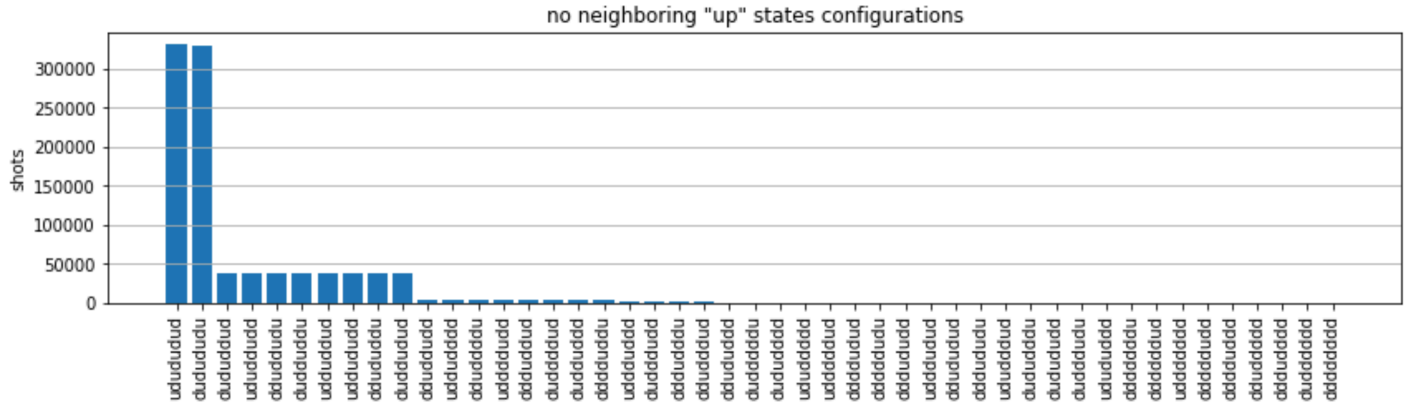
```
    Args:
       result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

    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)
```

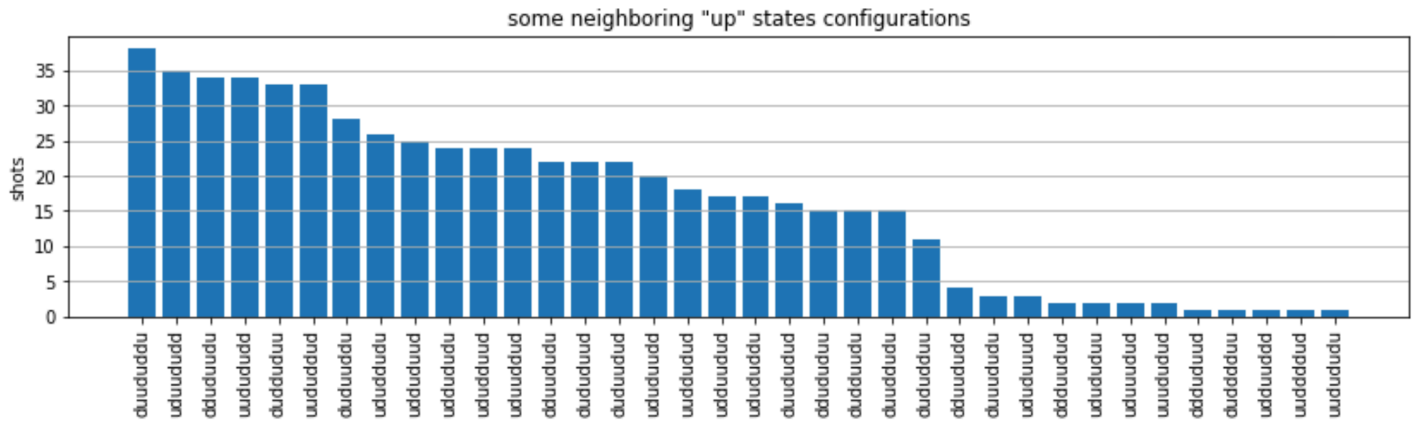
```
{'udududud': 330944, 'dudududu': 329576, 'dududdud': 38033, ...}
```

Este `counts` es un diccionario que cuenta el número de veces que se observa cada configuración de estado en las tomas. 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 leer las siguientes observaciones para comprobar que hemos preparado con éxito la fase antiferromagnética.

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

Se ejecuta en la QPU QuEra Aquila

Requisitos previos: Además de instalar el SDK de Braket, si eres nuevo en Amazon Braket, asegúrate de haber completado los pasos de introducción necesarios.

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

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

```
from braket.aws import AwsDevice

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

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

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

Ahora podemos ejecutar el programa (por ahora solo se ejecutan 100 disparos) en la Aquila QPU.

> **Note**
>
> La ejecución de este programa en el Aquila procesador tendrá un coste. El Amazon Braket SDK incluye un rastreador de costos que permite a los clientes establecer límites de costos y realizar un 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}")
```

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

Debido a la gran variación del tiempo que puede tardar en ejecutarse una tarea cuántica (según las ventanas de disponibilidad y el uso de la QPU), es una buena idea anotar el ARN de la tarea

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

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

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

Analizando los resultados de la QPU

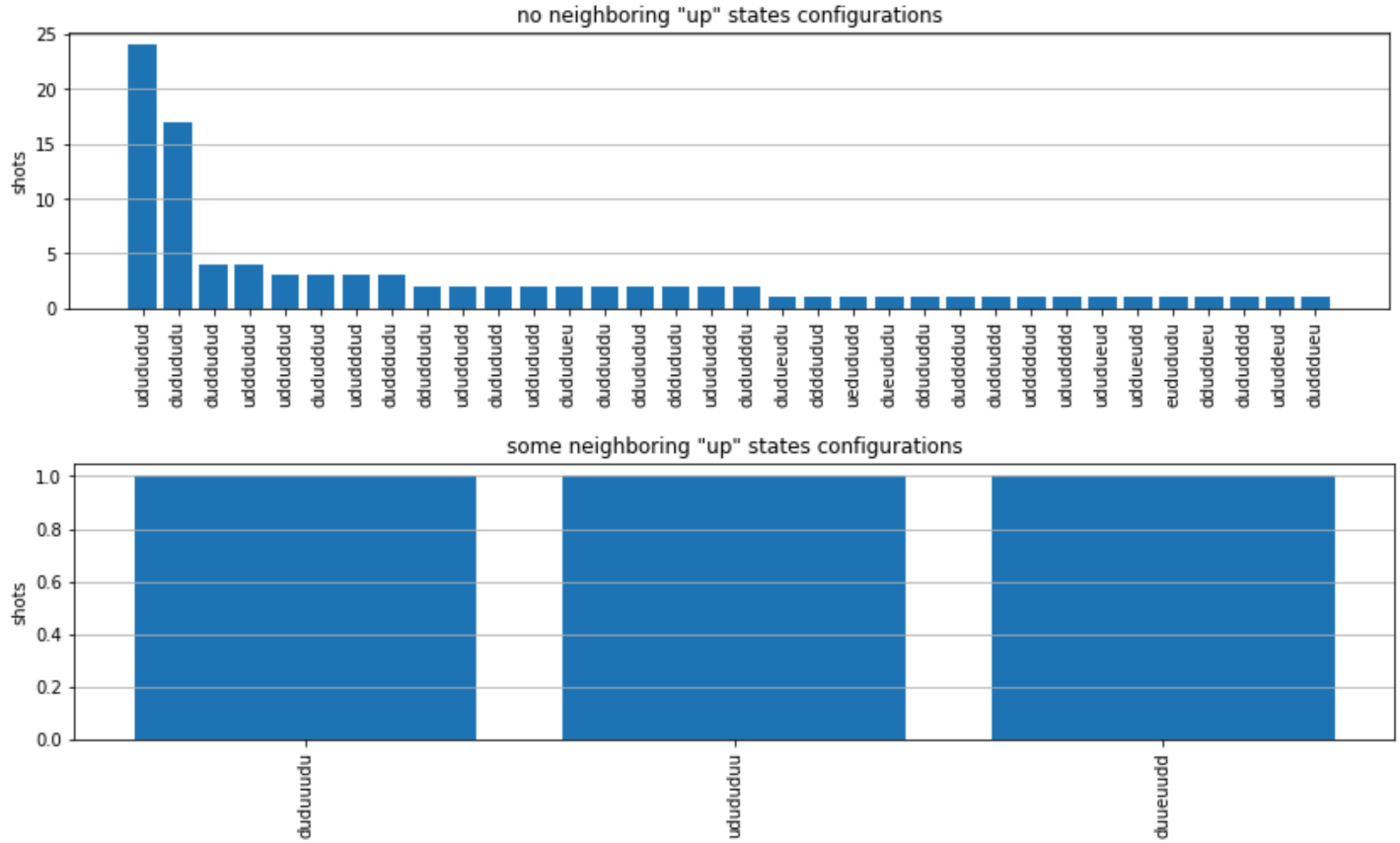
Usando las mismas `get_counts` funciones que antes, podemos calcular los recuentos:

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

```
*[Output]*
{'udududud': 24, 'dudududu': 17, 'dududdud': 3, ...}
```

y graficarlos con`plot_counts`:

```
plot_counts(counts_aquila)
```



Tenga en cuenta que una pequeña fracción de las tomas tiene sitios vacíos (marcados con una «e»). Esto se debe a que la Aquila QPU 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 disparos.

Next

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

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

Construya circuitos en el SDK

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

También puedes trabajar al nivel del pulso en Braket para varias compuertas con determinadas QPUs. Para obtener más información, consulta Pulse Control en Amazon Braket.

En esta sección:

- Puertas y circuitos
- Medición parcial
- Asignación manual qubit
- Compilación literal
- Simulación de ruido
- Inspeccionar el circuito
- Tipos de resultados

Puertas y circuitos

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

Ejemplo: definir un circuito

El ejemplo comienza con la definición de un circuito de muestra de cuatro qubits (denominado `q0` `q1q2`, `yq3`) compuesto por puertas Hadamard estándar de un solo qubit y puertas CNOT de dos cúbits. Puede visualizar este circuito llamando a la función, tal y como se muestra en el siguiente ejemplo. `print`

```
# 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-C---
        |
q1 : -H-|-C-
        | |
q2 : -H-X-|-
          |
q3 : -H---X-

T  : |0| 1 |
```

Ejemplo: Defina 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 nuevo circuito 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 argumento único `float` (que es el valor que tomarán todos los parámetros libres) o un argumento de palabra clave que especifique el valor de cada parámetro para el circuito de la siguiente manera.

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

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

Ejemplo: modificar las compuertas de un circuito

El siguiente ejemplo define un circuito con compuertas que utilizan modificadores de control y potencia. Puede utilizar estas modificaciones para crear nuevas compuertas, como la Ry compuerta 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: consulte todas las puertas disponibles

El siguiente ejemplo 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', 'GPi', 'GPi2', 'H', 'I', 'ISwap', 'MS', 'PSwap',
 'PhaseShift', 'PulseGate', 'Rx', 'Ry', 'Rz', 'S', 'Si', 'Swap', 'T', 'Ti', '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, llamaría `acirc.h(0)`, para añadir una puerta de Hadamard a la primera. qubit

> **Note**
>
> Las compuertas están colocadas en su sitio y en el siguiente ejemplo se agregan todas las compuertas 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 cúbits aplicadas a lo definido por el parámetro. qubits `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. `Instruction`los operadores incluyen `Gate` únicamente objetos de tipo.

```
# 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])
new_instr = instr.copy(target_mapping={0: 5})
```

Ejemplo: vea las puertas que admite cada dispositivo

Los simuladores admiten todas las compuertas del SDK de Braket, pero los dispositivos QPU admiten un subconjunto más pequeño. Puedes encontrar las compuertas 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/Harmony")

# 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 the Harmony device:
 ['x', 'y', 'z', 'rx', 'ry', 'rz', 'h', 'cnot', 's', 'si', 't', 'ti', 'v', 'vi', 'xx',
 'yy', 'zz', 'swap', 'i']
```

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

Ejemplo: recupere 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 Braket. A veces resulta útil acceder a la misma información mediante programación. El siguiente código muestra cómo extraer la fidelidad de las dos qubit puertas entre dos puertas de una QPU.

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

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

#specify the qubits
a=10
```

```
b=113
print(f"Fidelity of the XY gate between qubits {a} and {b}: ",
 device.properties.provider.specs["2Q"][f"{a}-{b}"]["fXY"])
```

Medición parcial

Siguiendo los ejemplos anteriores, hemos medido todos los cúbits del circuito cuántico. Sin embargo, es posible medir qubits individuales o un subconjunto de qubits.

Ejemplo: mida un subconjunto de qubits

En este ejemplo, demostramos una medición parcial añadiendo una `measure` instrucción con los cúbits objetivo al final del circuito.

```
# 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 qubit

Cuando ejecuta un circuito cuántico en ordenadores cuánticosRigetti, puede utilizar opcionalmente la qubit asignación manual para controlar cuáles se qubits utilizan para su algoritmo. La consola Amazon Braket y el Amazon Braket SDK le ayudan a inspeccionar los datos de calibración más recientes del dispositivo de unidad de procesamiento cuántico (QPU) que haya seleccionado, de modo que pueda seleccionar el qubits mejor para su experimento.

La qubit asignación manual le permite ejecutar los circuitos con mayor precisión e investigar las propiedades individuales. qubit Los investigadores y los usuarios avanzados optimizan el diseño

de sus circuitos en función de los datos de calibración más recientes de los dispositivos y pueden obtener resultados más precisos.

El siguiente ejemplo demuestra cómo asignar de qubits forma explícita.

```
circ = Circuit().h(0).cnot(0, 7)  # Indices of actual qubits in the QPU
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.

> **Note**
>
> El OQC compilador no admite la configuración. `disable_qubit_rewiring=True` Si se establece este indicador, se `True` produce el siguiente error:`An error occurred (ValidationException) when calling the CreateQuantumTask operation: Device arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy does not support disabled qubit rewiring`.

Compilación literal

Cuando ejecuta un circuito cuántico en ordenadores cuánticos desde RigettiIonQ, o Oxford Quantum Circuits (OQC), puede indicar al compilador que ejecute los circuitos exactamente como se ha definido sin ninguna modificación. Mediante la compilación literal, puede especificar que todo un circuito se conserve de forma precisa (con el apoyo de RigettiIonQ, yOQC) tal como se especifica o que solo se conserven determinadas partes del mismo (solo con el apoyo deRigetti). Al desarrollar algoritmos para la evaluación comparativa del hardware o los protocolos de mitigación de errores, debe tener la opción de especificar con precisión las compuertas y los diseños de circuitos que está ejecutando en el hardware. La compilación literal le permite controlar directamente el proceso de compilación al desactivar ciertos pasos de optimización, lo que garantiza que sus circuitos funcionen exactamente como se diseñaron.

Actualmente, la compilación literal es compatible con los dispositivos RigettiIonQ, y Oxford Quantum Circuits (OQC) y requiere el uso de puertas nativas. Al utilizar la compilación literal, se recomienda comprobar la topología del dispositivo para garantizar que las compuertas estén conectadas qubits y que el circuito utilice las compuertas nativas compatibles con el hardware. El siguiente ejemplo muestra cómo acceder mediante programación a la lista de puertas nativas compatibles con un dispositivo.

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

Para elloRigetti, el qubit recableado debe desactivarse configurándolo `disableQubitRewiring=True` para su uso con la compilación literal. Si `disableQubitRewiring=False` se establece cuando se utilizan recuadros literales en una compilación, el circuito cuántico no pasa la validación y no se ejecuta.

Si la compilación literal está habilitada 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 error de la tarea. A medida que más hardware cuántico admita de forma nativa las funciones del compilador, esta función se ampliará para incluir estos dispositivos. Los dispositivos que admiten la compilación literal 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/Aspen-M-3")
device.properties.action[DeviceActionType.OPENQASM].supportedPragmas
```

El uso de la compilación literal no conlleva ningún coste adicional. Se te seguirán cobrando las tareas cuánticas ejecutadas en dispositivos QPU de Braket, instancias de portátiles y simuladores bajo demanda en función de las tarifas actuales especificadas en la página de precios de Amazon Braket. Para obtener más información, consulte el cuaderno de ejemplo de compilación de Verbatim.

> **Note**
>
> Si utiliza OpenQASM para escribir los circuitos de los IonQ dispositivos OQC y desea mapear su circuito directamente a los cúbits físicos, debe utilizar el, `#pragma braket verbatim` ya que OpenQasm ignora por completo este `disableQubitRewiring` indicador.

Simulación de ruido

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

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

Puedes construir circuitos ruidosos de dos maneras:

1. Construya el circuito ruidoso de abajo hacia arriba.
2. Tome un circuito existente y libre de ruido e inyecte ruido en todas partes.

El siguiente ejemplo muestra los enfoques que utilizan un circuito simple con ruido despolarizante y un canal Kraus personalizado.

```
# 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)
```

```
# 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)
```

La ejecución de un circuito es la misma experiencia de usuario que antes, como se muestra en los dos ejemplos siguientes.

Ejemplo 1

```
task = device.run(circ, s3_location)
```

Or (Disyunción)

Ejemplo 2

```
task = device.run(circ_noise, s3_location)
```

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

Inspeccionar el circuito

Los circuitos cuánticos en Amazon Braket tienen un concepto de pseudotiempo llamado. `Moments` Cada uno qubit puede experimentar una sola puerta por. `Moment` El objetivo `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 compuertas 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, tal y `circuit.depth` como se muestra en el siguiente ejemplo.

```
# 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)-C----------X-
                |
q1 : -Ry(0.2)--|-ZZ(0.15)---
                | |
q2 : ----------X-|----------
                  |
q3 : ------------ZZ(0.15)---

T  : |   0    |    1     |2|
Total circuit depth: 3
```

La profundidad total del circuito anterior es 3 (se muestran como momentos `0`1, y2). Puede comprobar el funcionamiento de la compuerta en cada momento.

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

- La clave es `MomentsKey()` que contiene pseudotiempo e información. 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)]))
Instruction('operator': Rx('angle': 0.15, 'qubit_count': 1), 'target':
 QubitSet([Qubit(0)]))

MomentsKey(time=0, qubits=QubitSet([Qubit(1)]))
Instruction('operator': Ry('angle': 0.2, 'qubit_count': 1), 'target':
 QubitSet([Qubit(1)]))

MomentsKey(time=1, qubits=QubitSet([Qubit(0), Qubit(2)]))
Instruction('operator': CNot('qubit_count': 2), 'target': QubitSet([Qubit(0),
 Qubit(2)]))

MomentsKey(time=1, qubits=QubitSet([Qubit(1), Qubit(3)]))
Instruction('operator': ZZ('angle': 0.15, 'qubit_count': 2), 'target':
 QubitSet([Qubit(1), Qubit(3)]))

MomentsKey(time=2, qubits=QubitSet([Qubit(0)]))
Instruction('operator': X('qubit_count': 1), 'target': QubitSet([Qubit(0)]))
```

También puede añadir puertas a un circuito que las atraviese`Moments`.

```
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 : ---C-H-
```

```
T  : |0|1|2|
```

Tipos de resultados

AmazonBraket 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 vectorial) 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 puedes usar este método cuando los disparos son iguales a 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 en los simuladores locales.
- `Expectation`devuelve el valor esperado de un observable dado, que se puede especificar con la `Observable` clase que se presenta más adelante en este capítulo. Se debe especificar el objetivo qubits utilizado para medir lo observable y el número de objetivos especificados debe ser igual al número qubits sobre el que actúa el observable. Si no se especifican objetivos, el observable debe operar solo en 1 qubit y se aplica a todos 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 básicos. Si se especifican los objetivos, solo se devuelven las probabilidades marginales de los vectores base de qubits los especificados.
- `Reduced density matrix`devuelve una matriz de densidad para un subsistema del objetivo especificado de un sistema qubits de. qubits Para limitar el tamaño de este tipo de resultado, Braket limita el número de objetivos qubits 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 qubit conjunto objetivo específico y observable. Si no se especifican objetivos, el observable debe operar solo en 1 qubit y se aplica a todos qubits en paralelo. Si se especifican objetivos, el número de objetivos especificados debe ser igual al número qubits sobre el que actúa el observable.
- `Variance`devuelve la varianza (`mean([x-mean(x)]`^{2}`)`) del qubit conjunto de objetivos especificado y 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 qubits en paralelo. De lo contrario, el número de objetivos especificados debe ser igual al número de objetivos qubits a los que se puede aplicar el observable.

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

	SIM local	SV1	DM1	TN1	Rigetti	IonQ	OQC
Gradiente adjunto	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
Expectativa	Y	Y	Y	Y	Y	Y	Y
Probability	Y	Y	Y	N	Y*	Y	Y
Matriz de densidad reducida	Y	N	Y	N	N	N	N
Vector de estado	Y	N	N	N	N	N	N
Muestra	Y	Y	Y	Y	Y	Y	Y
Varianza	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> * Rigetti solo admite tipos de resultados probabilísticos de hasta 40qubits.

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

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

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

```
name='Sample' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=100000
name='Expectation' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=100000
name='Variance' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=100000
name='Probability' observables=None minShots=10 maxShots=100000
```

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

```
from braket.circuits import 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**
>
> Algunos dispositivos proporcionan mediciones (por ejemploRigetti) como resultados y otros proporcionan probabilidades como resultados (por ejemplo, IonQ yOQC). El SDK proporciona una propiedad de medición en los resultados, pero en el caso de los dispositivos que devuelven probabilidades, se calcula posteriormente. Por lo tanto, los dispositivos como los que proporciona IonQ y OQC cuyos resultados de medición están determinados por la probabilidad, ya que no se obtienen mediciones por disparo. Puede comprobar si un resultado se ha poscalculado consultando `measurements_copied_from_device` el objeto resultante, tal y como se muestra en este archivo.

Observables

AmazonBraket incluye una `Observable` clase que se puede utilizar para especificar un observable que se va a medir.

Puede aplicar como máximo un único observable sin identidad a cada uno. qubit Si especificas dos o más observables distintos que no son de identidad en un mismo objetoqubit, aparecerá un error. Para ello, cada factor de un producto tensorial cuenta como un observable individual, por lo que está

permitido tener varios productos tensoriales actuando sobre el mismoqubit, siempre que el factor que actúa sobre ellos sea el mismo. qubit

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

La `Observable` clase incluye los siguientes observables.

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

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

# get the tensor product of 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())

# also 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)))
```

Parámetros

Los circuitos pueden incluir parámetros libres, que se pueden utilizar para «construir una vez y ejecutar varias veces» y para calcular gradientes. Los parámetros libres tienen un nombre codificado en cadenas que se puede utilizar para especificar sus valores o para determinar si hay que diferenciarlos con respecto a ellos.

```
from braket.circuits import Circuit, FreeParameter, Observable
theta = FreeParameter("theta")
phi = FreeParameter("phi")
circ = Circuit().h(0).rx(0, phi).ry(0, phi).cnot(0, 1).xx(0, 1, theta)
circ.adjoint_gradient(observable=Observable.Z() @ Observable.Z(), target=[0, 1],
 parameters = ["phi", theta]
```

Para los parámetros que desee diferenciar, especifíquelos utilizando su nombre (en forma de cadena) o mediante referencia directa. Tenga en cuenta que el cálculo del gradiente utilizando el tipo de `AdjointGradient` resultado se realiza con respecto al valor esperado del observable.

Nota: Si ha fijado los valores de los parámetros libres pasándolos como argumentos al circuito parametrizado, al ejecutar un circuito con `AdjointGradient` el tipo de resultado y los parámetros especificados se producirá un error. Esto se debe a que los parámetros que utilizamos para diferenciar ya no están presentes. Consulte el siguiente ejemplo.

```
device.run(circ(0.2), shots=0) # will error, as no free parameters will be present
device.run(circ, shots=0, inputs={'phi'=0.2, 'theta'=0.2) # will succeed
```

Enviar tareas cuánticas a QPUs y simuladores

AmazonBraket 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.

QPUs

Puede enviar tareas cuánticas a las QPU 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.

- IonQ Aria 1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1`
- IonQ Aria 2 : `arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2`
- IonQ Forte 1(solo con reserva): `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1`
- IonQ Harmony : `arn:aws:braket:us-east-1::device/qpu/ionq/Harmony`
- IQM Garnet : `arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet`
- OQC Lucy : `arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Aspen-M-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3`

Simuladores

- Simulador de matrices de densidad, DM1: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- Simulador vectorial de estados, SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Simulador de redes tensoras, TN1: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- El simulador local: `LocalSimulator()`

> **Note**
>
> Puede cancelar las tareas cuánticas en el `CREATED` estado para las QPU y los simuladores bajo demanda. Puede cancelar las tareas cuánticas en el `QUEUED` estado como sea posible si utiliza simuladores y QPUs bajo demanda. Tenga en cuenta que es poco probable que las tareas `QUEUED` cuánticas de la QPU se cancelen correctamente durante los períodos de disponibilidad de la QPU.

En esta sección:

- Ejemplo de tareas cuánticas en Amazon Braket
- Enviar tareas cuánticas a una QPU
- Ejecutar una tarea cuántica con el simulador local
- Agrupación cuántica de tareas
- Configura las notificaciones de SNS (opcional)
- Inspeccionar circuitos compilados

Ejemplo de tareas cuánticas en Amazon Braket

En esta sección se explican las etapas de la ejecución de una tarea cuántica de ejemplo, desde la selección del dispositivo hasta la visualización del resultado. Como práctica recomendada para Amazon Braket, te recomendamos empezar por ejecutar el circuito en un simulador, comoSV1.

En esta sección:

- Especifique el dispositivo
- Envíe un ejemplo de tarea cuántica
- Envíe una tarea parametrizada
- Especifique shots.
- Encuesta para ver los resultados
- Vea los resultados de ejemplo

Especifique el dispositivo

Primero, seleccione y especifique el dispositivo para su tarea cuántica. En este ejemplo se muestra cómo elegir el simulador,SV1.

```
# choose the on-demand simulator to run the circuit
from braket.aws import AwsDevice
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', <DeviceActionType.JAQCD: 'braket.ir.jaqcd.program'>)
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
 'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', '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)])
```

Envíe un ejemplo de tarea cuántica

Envíe un ejemplo de tarea cuántica para ejecutarla en el simulador bajo demanda.

```
# 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-your-s3-bucket-name" # 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 `device.run()` comando crea una tarea cuántica a través de la `CreateQuantumTask` API. 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 cálculo, Amazon Braket escribe los resultados en la ubicación de Amazon S3 especificada en la llamada. El argumento posicional `s3_location` es obligatorio para todos los dispositivos, excepto para el simulador local.

> **Note**
>
> La acción de la tarea cuántica de Braket tiene un tamaño limitado a 3 MB.

Envíe una tarea parametrizada

Los simuladores locales y bajo demanda y las QPU de Amazon Braket también admiten la especificación de valores de parámetros libres al enviar la tarea. Para ello, utilice el `inputs` argumento to`device.run()`, como se muestra en el siguiente ejemplo. `inputs`Debe ser un diccionario de pares de cadenas flotantes, donde las claves son los nombres de los parámetros.

La compilación paramétrica puede mejorar el rendimiento de la ejecución de circuitos paramétricos en determinadas 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 hay recompilación para las actualizaciones de parámetros posteriores en el 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 del hardware al compilar el circuito para garantizar resultados de la más alta calidad.

> **Note**
>
> La compilación paramétrica es compatible con todas las QPU superconductoras basadas en compuertas Rigetti Computing y Oxford Quantum Circuits con la excepción de los 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})
```

Especifique shots.

El shots argumento se refiere al número de medidas deseadas. shots Este tipo de simuladores 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 enTN1.)
- Para valores distintos de ceroshots, el simulador toma muestras de la distribución de salida para emular el shot ruido de las QPUs reales. Los dispositivos QPU solo permiten valores superiores a 0. shots

Para obtener información sobre el número máximo de disparos por tarea cuántica, consulta las cuotas de braket.

Encuesta para ver los resultados

Al ejecutarlomy_task.result(), el SDK comienza a sondear un resultado con los parámetros que definas al crear la tarea cuántica:

- poll_timeout_secondses el número de segundos que se necesitan para sondear la tarea cuántica antes de que se agote el tiempo de espera cuando se ejecuta la tarea cuántica en el simulador bajo demanda o en los dispositivos QPU. El valor predeterminado es 432.000 segundos, es decir, 5 días.
- Nota: Para dispositivos QPU como Rigetti yIonQ, le recomendamos que espere unos días. Si el tiempo de espera para votar es demasiado corto, es posible que los resultados no se devuelvan dentro del tiempo de votación. Por ejemplo, cuando una QPU no está disponible, se devuelve un error de tiempo de espera local.
- poll_interval_secondses la frecuencia con la que se sondea la tarea cuántica. Especifica la frecuencia con la que se llama al Braket API para obtener el estado cuando la tarea cuántica se ejecuta en el simulador bajo demanda y en los dispositivos QPU. El valor predeterminado es 1 segundo.

Esta ejecución asíncrona facilita la interacción con los 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 comprobar el resultado de la medición con los siguientes comandos:

```
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]
 [1 0 1]
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Counts for collapsed states:
 Counter({'000': 761, '101': 226, '010': 10, '111': 3})
Probabilities for collapsed states:
 {'101': 0.226, '000': 0.761, '111': 0.003, '010': 0.01}
```

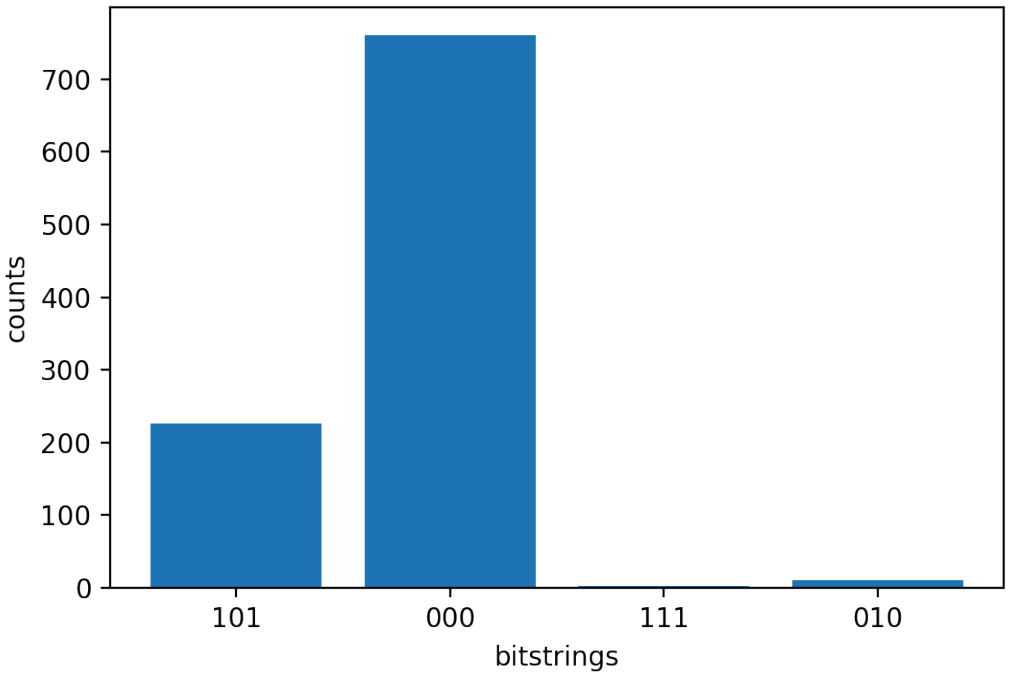
Vea los resultados de ejemplo

Como también especificó el`ResultType`, puede ver los resultados devueltos. Los tipos de resultados aparecen en el orden en que se agregaron al circuito.

```
print('Result types include:\n', result.result_types)
print('Variance=',result.values[0])
print('Probability=',result.values[1])

# you can 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={'observable': ['z'], 'targets': [0], 'type': 'variance'},
 value=0.7062359999999999), ResultTypeValue(type={'targets': [0, 2], 'type':
 'probability'}, value=array([0.771, 0.   , 0.   , 0.229]))]
Variance= 0.7062359999999999
Probability= [0.771 0.    0.    0.229]
```



Enviar tareas cuánticas a una QPU

AmazonBraket permite ejecutar un circuito cuántico en un dispositivo QPU. El siguiente ejemplo muestra cómo enviar una tarea cuántica a nuestros dispositivosRigetti. IonQ

Elige el Rigetti Aspen-M-3 dispositivo y, a continuación, mira 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/Aspen-M-3")

# take a look at the device connectivity graph
device.properties.dict()['paradigm']['connectivity']
```

```
{'fullyConnected': False,
```

```
 'connectivityGraph': {'0': ['1', '7'],
  '1': ['0', '16'],
  '2': ['3', '15'],
  '3': ['2', '4'],
  '4': ['3', '5'],
  '5': ['4', '6'],
  '6': ['5', '7'],
  '7': ['0', '6'],
  '11': ['12', '26'],
  '12': ['13', '11'],
  '13': ['12', '14'],
  '14': ['13', '15'],
  '15': ['2', '14', '16'],
  '16': ['1', '15', '17'],
  '17': ['16'],
  '20': ['21', '27'],
  '21': ['20', '36'],
  '22': ['23', '35'],
  '23': ['22', '24'],
  '24': ['23', '25'],
  '25': ['24', '26'],
  '26': ['11', '25', '27'],
  '27': ['20', '26'],
  '30': ['31', '37'],
  '31': ['30', '32'],
  '32': ['31', '33'],
  '33': ['32', '34'],
  '34': ['33', '35'],
  '35': ['22', '34', '36'],
  '36': ['21', '35', '37'],
  '37': ['30', '36']}}
```

El diccionario anterior `connectivityGraph` contiene información sobre la conectividad del Rigetti dispositivo actual.

Elija el IonQ Harmony dispositivo

En el IonQ Harmony caso del dispositivo, `connectivityGraph` está vacío, como se muestra en el siguiente ejemplo, porque el dispositivo ofrece conectividad total. Por lo tanto, no `connectivityGraph` es necesaria una información detallada.

```
# or choose the IonQ Harmony device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Harmony")
```

```
# 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 la ubicación shots (por defecto = 1000), `poll_timeout_seconds` (por defecto = 432000 = 5 días), `poll_interval_seconds` (por defecto = 1) y la ubicación del depósito de S3 (`s3_location`) en la que se almacenarán los resultados si decide especificar una ubicación distinta del depósito predeterminado.

```
my_task = device.run(circ, s3_location = 'amazon-braket-my-folder', shots=100,
 poll_timeout_seconds = 100, poll_interval_seconds = 10)
```

RigettiLos dispositivos IonQ y compilan automáticamente el circuito proporcionado en sus respectivos conjuntos de puertas nativas y asignan los qubit índices abstractos a físicos qubits en la QPU respectiva.

> **Note**
>
> Los dispositivos QPU tienen una capacidad limitada. Puede esperar tiempos de espera más largos cuando se alcance la capacidad.

AmazonBraket puede ejecutar tareas cuánticas de QPU dentro de ciertos períodos de disponibilidad, pero usted 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 depósito S3 correspondiente. Como se muestra en la siguiente sección, puedes recuperar tu tarea cuántica utilizando `AwsQuantumTask` un identificador único de tarea cuántica.

Ejecutar una tarea cuántica con el simulador local

Puede enviar las tareas cuánticas directamente a un simulador local para crear prototipos y probarlos rápidamente. Este simulador se ejecuta en su entorno local, por lo que no necesita especificar una ubicación de Amazon S3. Los resultados se calculan directamente en la sesión. Para ejecutar una tarea cuántica en el simulador local, solo debe especificar el shots parámetro.

> **Note**
>
> La velocidad de ejecución y la cantidad máxima que qubits el simulador local puede procesar dependen del tipo de instancia del portátil Amazon Braket o de las especificaciones de hardware locales.

Los siguientes comandos son todos idénticos y crean una instancia del simulador local vectorial de estado (libre de ruidos).

```
# 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 crear una instancia del simulador de matriz de densidad local (ruido), los clientes cambian 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 vectorial de estado local y el simulador de matriz de densidad local admiten circuitos en los que se puede medir un subconjunto de los cúbits del circuito, lo que a menudo se denomina medición parcial.

Por ejemplo, en el código siguiente puedes crear un circuito de dos cúbits y medir solo el primer cúbit añadiendo una `measure` instrucción con los cúbits objetivo 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)
```

Agrupación cuántica de tareas

El procesamiento por lotes cuántico de tareas está disponible en todos los dispositivos Amazon Braket, excepto en el simulador local. El procesamiento por lotes es especialmente útil para las tareas cuánticas que se ejecutan en los simuladores bajo demanda (TN1oSV1) porque pueden procesar múltiples tareas cuánticas en paralelo. Para ayudarte a configurar diversas tareas cuánticas, Amazon Braket proporciona ejemplos de cuadernos.

El procesamiento por lotes le permite lanzar tareas cuánticas en paralelo. Por ejemplo, si desea realizar un cálculo que requiera 10 tareas cuánticas y los circuitos de esas tareas cuánticas sean independientes entre sí, es una buena idea utilizar el procesamiento por lotes. De esta forma, no tendrá que esperar a que se complete una tarea cuántica antes de que comience otra.

El siguiente ejemplo muestra cómo ejecutar un lote de tareas cuánticas:

```
circuits = [bell for _ in range(5)]
batch = device.run_batch(circuits, s3_folder, shots=100)
print(batch.results()[0].measurement_counts)  # The result of the first quantum task in
 the batch
```

Para obtener más información, consulta los ejemplos de Amazon Braket sobre GitHub el procesamiento por lotes de tareas de Quantum, que contienen información más específica sobre el procesamiento por lotes.

Acerca del procesamiento por lotes de tareas cuánticas y sus costes

Algunas advertencias a tener en cuenta con respecto a los costos de facturación y 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 falla las tareas cuánticas 3 veces.
- Un lote de tareas cuánticas de larga duración, como 34 qubits tareasSV1, puede generar grandes costes. Asegúrese de comprobar detenidamente los valores de las `run_batch` asignaciones antes de iniciar un lote de tareas cuánticas. No recomendamos su uso TN1 con`run_batch`.
- TN1pueden incurrir en costes si no se realizan tareas de la fase de ensayo (consulta la descripción del TN1 para obtener más información). Los reintentos automáticos pueden aumentar el coste, por lo que recomendamos establecer el número máximo de reintentos en el procesamiento por lotes en 0 cuando se utilice TN1 (consulte Quantum Task Batching, línea 186).

Procesamiento por lotes de tareas cuánticas y PennyLane

Aproveche el procesamiento por lotes cuando utilice PennyLane Amazon Braket configurando `parallel = True` cuándo instanciar un dispositivo Amazon Braket, como se muestra en el siguiente ejemplo.

```
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 `inputs` diccionario, que se utiliza para todas las tareas cuánticas del lote, o un conjunto de diccionarios `list` de entrada, en cuyo caso el diccionario -ésimo se empareja con la `i` `i` -ésima tarea, como se muestra en el siguiente ejemplo.

```
from braket.circuits import Circuit, FreeParameter, Observable
from braket.aws import AwsQuantumTaskBatch

# 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 `i` -ésima tarea cuántica corresponde al circuito ejecutado con el `i` -ésimo diccionario de entrada.

```
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)
```

Configura las notificaciones de SNS (opcional)

Puedes configurar las notificaciones a través del Amazon Simple Notification Service (SNS) para recibir una alerta cuando se complete tu tarea cuántica de Amazon Braket. Las notificaciones activas son útiles si espera un tiempo de espera prolongado; por ejemplo, cuando envía una tarea cuántica de gran tamaño o cuando envía una tarea cuántica fuera del período de disponibilidad de

un dispositivo. Si no quieres esperar a que se complete la tarea cuántica, puedes configurar una notificación de SNS.

Un cuaderno Amazon Braket le guiará por 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.

Inspeccionar circuitos compilados

Cuando un circuito se ejecuta en un dispositivo de hardware, debe compilarse en un formato aceptable, como transpilar el circuito hasta las puertas nativas compatibles con la QPU. Inspeccionar el resultado compilado real puede resultar muy útil para la depuración. Puede ver este circuito tanto para los dispositivos como para Rigetti los OQC dispositivos utilizando el siguiente código.

```
task = AwsQuantumTask(arn=task_id, aws_session=session)
# after task finished
task_result = task.result()
compiled_circuit = task_result.get_compiled_circuit()
```

> **Note**
>
> Hoy en día no puede ver su circuito compilado para IonQ dispositivos.

Ejecute sus circuitos con OpenQASM 3.0

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

En los temas de esta guía se explican varios ejemplos de cómo realizar las siguientes tareas cuánticas.

- Cree y envíe tareas cuánticas de OpenQASM en diferentes dispositivos Braket
- Acceda a las operaciones y los tipos de resultados compatibles
- Simule el ruido con OpenQASM
- Utilice la compilación literal con OpenQASM

- Solucione los problemas de OpenQASM

Esta guía también proporciona una introducción a determinadas funciones 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?
- ¿Cuándo usar OpenQASM 3.0
- ¿Cómo funciona OpenQASM 3.0
- Requisitos previos
- ¿Qué funciones de OpenQASM admite Braket?
- Cree y envíe un ejemplo de tarea cuántica de OpenQASM 3.0
- Support para OpenQASM en diferentes dispositivos Braket
- Simule el ruido con OpenQASM 3.0
- Qubitrecableado con OpenQASM 3.0
- Compilación textual con OpenQASM 3.0
- La consola Braket
- Más recursos
- Calcular gradientes con OpenQASM 3.0
- Medición de qubits específicos con OpenQASM 3.0

¿Qué es OpenQASM 3.0?

El lenguaje de ensamblaje cuántico abierto (OpenQASM) es una representación intermedia de las instrucciones cuánticas. OpenQASM es un marco de código abierto y se usa ampliamente para la especificación de programas cuánticos para dispositivos basados en puertas. Con OpenQASM, los usuarios pueden programar las compuertas cuánticas y las operaciones de medición que forman los componentes básicos de la computación cuántica. Varias bibliotecas de programación cuántica utilizaban la versión anterior de OpenQASM (2.0) para describir programas sencillos.

La nueva versión de OpenQASM (3.0) amplía la versión anterior para incluir más funciones, como el control a nivel de pulso, la temporización de las puertas y el flujo de control clásico, a fin de 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 activa de OpenQASM

3.x. GitHub 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 usar OpenQASM 3.0

OpenQASM proporciona un marco expresivo para especificar programas cuánticos mediante controles de bajo nivel que no son específicos de la arquitectura, lo que lo hace ideal como representación en varios dispositivos basados en puertas. La compatibilidad de Braket con OpenQASM impulsa 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 ya tiene 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

El soporte para OpenQASM 3.0 de Braket proporciona paridad de funciones con la representación intermedia actual. Esto significa que cualquier cosa que pueda hacer hoy en día en dispositivos de hardware y simuladores bajo demanda con Braket, puede hacerlo con OpenQASM utilizando Braket. API Para ejecutar programas de OpenQASM 3.0, basta con suministrar cadenas OpenQASM directamente a todos los dispositivos basados en puertas, de forma similar a como se suministran actualmente los circuitos a los dispositivos de Braket. Los usuarios de Braket también pueden integrar bibliotecas de terceros compatibles con OpenQASM 3.0. El resto de esta guía detalla cómo desarrollar representaciones de OpenQASM para usarlas con Braket.

Requisitos previos

Para usar OpenQASM 3.0 en Amazon Braket, debe tener la versión 1.8.0 de los esquemas de Python de Amazon Braket y la versión 1.17.0 o superior del SDK de Python de Amazon Braket.

Si es la primera vez que utiliza Braket, debe activar Braket. Amazon Amazon Para obtener instrucciones, consulta Cómo activar Amazon Braket.

¿Qué funciones 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 OpenQASM compatibles
- Declaraciones OpenQASM compatibles
- Braket: pragmas de OpenQASM
- Compatibilidad con funciones avanzadas para OpenQASM en el simulador local
- Se admiten operaciones y gramática con OpenPulse

Tipos de datos OpenQASM compatibles

Braket admite los siguientes tipos de datos de OpenQASM. Amazon

- 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 las compuertas:
  - `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 `im` notación 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 OpenQASM compatibles

Braket apoya las siguientes declaraciones de OpenQASM. Amazon

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

> **Note**
>
> Las compuertas compatibles con un dispositivo se encuentran en las propiedades del dispositivo para las acciones de OpenQASM; no se necesitan definiciones de compuertas para utilizarlas.

- Declaraciones textuales en recuadros literales. Actualmente, no admitimos la notación de duración de las casillas. Las puertas nativas y físicas qubits son obligatorias en los recuadros textuales.

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

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

- `b = measure q;`
- `measure q # b;`

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

Braket: pragmas de OpenQASM

Braket admite las siguientes instrucciones pragmáticas de OpenQASM. Amazon

- Pragmas sobre el 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 literales
  - `#pragma braket verbatim`
- Pragmas del tipo de resultado
  - Tipos de resultados invariantes básicos:
    - Vector de estado: `#pragma braket result state_vector`
    - Matriz de densidad: `#pragma braket result density_matrix`
  - Pragmas de cálculo de gradientes:
    - 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 basados en rotación
    - Expectativa: `#pragma braket result expectation x(q[0]) @ y([q1])`
    - Varianza: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`

- Muestra: `#pragma braket result sample h($1)`

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

Compatibilidad con funciones avanzadas para OpenQASM en el simulador local

`LocalSimulator`Es compatible con funciones avanzadas de OpenQASM que no se ofrecen como parte de las QPU de Braket ni de los simuladores bajo demanda. La siguiente lista de funciones solo es compatible con: `LocalSimulator`

- Modificadores de puerta
- Puertas integradas OpenQASM
- Variables clásicas
- Operaciones clásicas
- Puertas personalizadas
- Control clásico
- Ficheros QASM
- Subrutinas

Para ver ejemplos de cada función avanzada, consulte este ejemplo de bloc de notas. Para ver la especificación completa de OpenQASM, consulte el sitio web de OpenQASM.

Se admiten operaciones y gramática con OpenPulse

Tipos OpenPulse de datos compatibles

Bloques de llamadas:

```
cal {
```

```
    ...
}
```

Bloques adhesivos:

```
// 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 compuerta 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);
}
```

Cree y envíe 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 Braket. Amazon

En esta sección:

- Un ejemplo de programa OpenQASM 3.0
- Utilice el SDK de Python para crear tareas cuánticas de OpenQASM 3.0
- Utilice Boto3 para crear tareas cuánticas de OpenQASM 3.0
- Úselo para crear tareas de OpenQASM 3.0 AWS CLI

Un ejemplo de programa OpenQASM 3.0

Para crear una tarea de OpenQASM 3.0, puede empezar con un sencillo programa OpenQASM 3.0 (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;
```

Utilice el 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 Amazon Braket con el siguiente código.

```
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/Aspen-M-3")
from braket.ir.openqasm import Program

program = Program(source=ghz_qasm_string)
my_task = device.run(program)

# You can also specify an optional s3 bucket location and number of shots,
```

```
 # if you so choose, when running the program
s3_location = ("amazon-braket-my-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
)
```

Utilice Boto3 para crear tareas cuánticas de OpenQASM 3.0

También puede usar el SDK de AWS Python para Braket (Boto3) para crear las tareas cuánticas mediante cadenas 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 arriba.

```
import boto3
import json

my_bucket = "amazon-braket-my-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/Aspen-M-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/Aspen-M-3" \
    --shots 100 \
    --output-s3-bucket "amazon-braket-my-bucket" \
    --output-s3-key-prefix "openqasm-tasks" \
    --action '{
        "braketSchemaHeader": {
            "name": "braket.ir.openqasm.program",
            "version": "1"
        },
        "source": $(cat ghz.qasm)
    }'
```

Support para OpenQASM en diferentes dispositivos Braket

En el caso de los dispositivos compatibles con OpenQASM 3.0, el `action` campo admite una nueva acción a través de la `GetDevice` respuesta, como se muestra en el siguiente ejemplo para los dispositivos y. Rigetti 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 impulsos, el `pulse` campo se muestra en la respuesta. `GetDevice` Los siguientes ejemplos muestran este `pulse` campo para los OQC dispositivos Rigetti y.

```
// 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
    }
  }
}

// OQC

{
  "pulse": {
    "braketSchemaHeader": {
      "name": "braket.device_schema.pulse.pulse_device_action_properties",
      "version": "1"
    },
    "supportedQhpTemplateWaveforms": {
      "gaussian": {
        "functionName": "gaussian",
        "arguments": [
          {
            "name": "length",
            "type": "float",
            "optional": false
          },
          {
            "name": "sigma",
            "type": "float",
```

```
            "optional": false
          },
          {
            "name": "amplitude",
            "type": "float",
            "optional": true
          },
          {
            "name": "zero_at_edges",
            "type": "bool",
            "optional": true
          }
        ]
      },
      ...
    },
    "ports": {
      "channel_1": {
        "portId": "channel_1",
        "direction": "tx",
        "portType": "port_type_1",
        "dt": 5e-10,
        "qubitMappings": [
          0
        ]
      },
      ...
    },
    "supportedFunctions": {
      "new_frame": {
        "functionName": "new_frame",
        "arguments": [
          {
            "name": "port",
            "type": "port",
            "optional": false
          },
          {
            "name": "frequency",
            "type": "float",
            "optional": false
          },
          {
            "name": "phase",
```

```
            "type": "float",
            "optional": true
          }
        ]
      },
      ...
    },
    "frames": {
      "q0_drive": {
        "frameId": "q0_drive",
        "portId": "channel_1",
        "frequency": 5500000000,
        "centerFrequency": 5500000000,
        "phase": 0,
        "qubitMappings": [
          0
        ]
      },
     ...
    },
    "supportsLocalPulseElements": false,
    "supportsDynamicFrames": true,
    "supportsNonNativeGatesWithPulses": true,
    "validationParameters": {
      "MAX_SCALE": 1,
      "MAX_AMPLITUDE": 1,
      "PERMITTED_FREQUENCY_DIFFERENCE": 1,
      "MIN_PULSE_LENGTH": 8e-9,
      "MAX_PULSE_LENGTH": 0.00012
    }
  }
}
```

Los campos anteriores detallan lo siguiente:

Puertos:

Describe los puertos de dispositivos externos (`extern`) prefabricados 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 `OpenQASM 3.0` programa 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 (dirección)
  - La dirección del puerto. Los puertos de accionamiento transmiten pulsos (dirección «tx»), mientras que los puertos de medición reciben pulsos (dirección «rx»).
- Tipo de puerto (PortType)
  - El tipo de acción de la que es responsable este puerto (por ejemplo, conducir, capturar o desactivar el flujo rápido).
- Dt (dt)
  - El tiempo en segundos que representa un único paso de tiempo de muestreo en el puerto dado.
- Mapeos de qubits (QubitMappings)
  - Los qubits asociados al puerto dado.
- Frecuencias centrales (CenterFrequencies)
  - Una lista de las frecuencias centrales asociadas a todas las tramas predeclaradas o definidas por el usuario en el puerto. Para obtener más información, consulte Frames.
- Propiedades específicas de QHP (SpecificPropertiesqhp)
  - Un mapa opcional que detalla las propiedades existentes sobre el puerto específico del QHP.

Marcos:

Describe los marcos externos prefabricados declarados en la QPU, así como las propiedades asociadas a los marcos. Todos los marcos enumerados en esta estructura se declaran previamente como identificadores válidos en el `OpenQASM 3.0` programa 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 al marco.
- Frecuencia (frecuencia)
  - La frecuencia inicial predeterminada del fotograma.
- Frecuencia central (CenterFrequency)
  - El centro del ancho de banda de frecuencia del marco. Por lo general, las tramas solo se pueden ajustar a un cierto ancho de banda alrededor de la frecuencia central. Como resultado, los

ajustes de frecuencia deben permanecer dentro de un delta determinado de la frecuencia central. Puede encontrar el valor del ancho de banda en los parámetros de validación.

- Fase (fase)
  - La fase inicial predeterminada del fotograma.
- Puerta asociada (AssociatedGate)
  - Las puertas asociadas al marco dado.
- Mapeos de qubits (QubitMappings)
  - Los qubits asociados al fotograma dado.
- Propiedades específicas de QHP (qhp) SpecificProperties
  - Un mapa opcional que detalla las propiedades existentes sobre el marco específico del QHP.

SupportsDynamicMarcos:

Describe si un marco puede declararse o bloquearse en la OpenPulse `newframe` función `cal` o `defcal` bloquearse a través de ella. Si es falso, en el programa solo se pueden utilizar los marcos que figuran en la estructura de marcos.

SupportedFunctions:

Describe las OpenPulse funciones compatibles con el dispositivo, además de los argumentos asociados, los tipos de argumentos y los tipos de retorno de 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 fotograma según un valor especificado
- set_phase
  - Establece la fase del fotograma en el valor especificado
- shift_frequency
  - Cambia la frecuencia de un fotograma según un valor especificado
- set_frequency
  - Establece la frecuencia del fotograma en el valor especificado
- jugar
  - Programa una forma de onda

- capture_v0
  - Devuelve el valor de un fotograma 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 formas de onda predefinidas en todos los dispositivos, que son:

Constante

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

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

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

gaussiano

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

`τ`es la longitud de la onda, es la anchura de la gaussiana y σ es la amplitud. `A` 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 su valor máximo. `A`

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

ARRASTRE EL GAUSSIANO

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

`τ`es la longitud de la forma de onda, σ es el ancho de la gaussiana, es un parámetro libre y es la amplitud. β `A` Si se establece `ZaE` en`True`, el gaussiano Eliminación de derivadas por puerta adiabática (DRAG) se desplaza y se reescala de manera que sea igual a cero al principio y al final de la forma de onda, y la parte real alcance su valor máximo. `A` Para obtener más información sobre

la forma de onda DRAG, consulte el artículo Pulsos simples para la eliminación de fugas en cúbits débilmente no lineales.

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

SupportsLocalPulseElements:

Describe si los elementos de impulso, como puertos, tramas y formas de onda, pueden definirse localmente en bloques o no. `defcal` Si el valor es`false`, los elementos se deben definir en `cal` bloques.

SupportsNonNativeGatesWithPulses:

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

ValidationParameters:

Describe los límites de validación de los elementos de impulso, incluidos:

- Valores de escala máxima y amplitud máxima para las formas de onda (arbitrarios y prediseñados)
- Ancho de banda de frecuencia máximo a partir de la frecuencia central suministrada en Hz
- Longitud/duración mínima del pulso en segundos
- Longitud/duración máxima del pulso en segundos

Operaciones, resultados y tipos de resultados compatibles con OpenQASM

Para saber qué funciones de OpenQASM 3.0 admite cada dispositivo, consulte la `braket.ir.openqasm.program` clave que aparece en el `action` campo 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 Braket State Vector. 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
        }
      ]
    }
  },
...
```

Simule el ruido con OpenQASM 3.0

Para simular el 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 pragmático compatibles 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 Kraus

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

Cuando utilice los operadores de Kraus, recuerde lo siguiente:

- El número de no qubits 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 2 matrices de $^{2*num\_qubits}$. Esto significa 4 matrices para 1 y 16 para 2qubit. qubits
- Todas las matrices suministradas tienen un sistema de conservación de trazas completamente positivo (CPTP).
- El producto de los operadores de Kraus con sus conjugados de transposición debe sumarse a una matriz de identidad.

Qubitrecableado con OpenQASM 3.0

AmazonBraket admite la qubit notación física de OpenQASM en los Rigetti dispositivos (para obtener más información, consulte esta página). Cuando utilices dispositivos físicos qubits con la ingenua estrategia de recableado, asegúrate de que estén conectados al dispositivo seleccionadoqubits. Como alternativa, si se utilizan qubit registros 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 textual con OpenQASM 3.0

Cuando ejecuta un circuito cuántico en ordenadores cuánticos desdeRigetti, y OQCIonQ, puede indicar al compilador que ejecute sus circuitos exactamente como se ha definido, sin ninguna modificación. Esta función se conoce como compilación literal. 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, necesitarás usar puertas nativas dentro de las regiones preservadas. Actualmente, IonQ OQC solo admite la compilación literal de todo el circuito, por lo que todas las instrucciones del circuito deben estar incluidas en un recuadro textual.

Con OpenQASM, puede especificar un pragma literal en torno a un cuadro de código que no ha sido modificado ni optimizado por la rutina de compilación de bajo nivel del hardware. El siguiente ejemplo de código muestra cómo utilizar el. `#pragma braket verbatim`

```
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 más información sobre la compilación literal, consulte el cuaderno de muestra de compilación literal.

La consola Braket

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

Más recursos

OpenQASM está disponible en todas las regiones de Braket. Amazon

Para ver un ejemplo de cuaderno para empezar a utilizar OpenQASM en Braket, consulte los tutoriales de Braket. Amazon GitHub

Calcular gradientes con OpenQASM 3.0

Amazon Braket admite el cálculo de gradientes en simuladores locales y bajo demanda en modo `shots=0` (exacto) mediante el método de diferenciación adjunto. Puede proporcionar el pragma adecuado para especificar el gradiente que desea calcular, como se muestra en el 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 individualmente, también puede especificarlos `all` en el pragma. Esto calcula el gradiente con respecto a todos los `input` parámetros listados. Esto puede resultar práctico cuando el número de parámetros es muy grande. En este caso, el pragma se verá como el siguiente ejemplo.

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

Se admiten todos los tipos de observables, incluidos los operadores individuales, los productos tensoriales, los observables hermitianos y. `Sum` El operador que se quiere usar para calcular el gradiente debe estar incluido en el `expectation()` encapsulador y se deben especificar los qubits sobre los que actúa cada término.

Medición de qubits específicos con OpenQASM 3.0

El simulador vectorial estatal local y el simulador de matrices de densidad local admiten el envío de OpenQASM programas en los que se pueda medir un subconjunto de los qubits del circuito. Esto suele denominarse medición parcial. Por ejemplo, en el siguiente código puedes crear un circuito de dos cúbits y medir solo el primer cúbit.

```
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];
"""
```

Hay dos qubits, `q[0]` `q[1]` pero aquí solo estamos midiendo el qubit 0:. `b[0] = measure q[0]` Ahora, ejecute lo siguiente en el simulador vectorial de estado local.

```
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 `requiresAllQubitsMeasurement` campo en sus propiedades de acción; si lo es`False`, se admite la medición parcial.

```
AwsDevice(Devices.Rigetti.AspenM3).properties.action['braket.ir.openqasm.program'].requiresAllQ
```

Aquí `requiresAllQubitsMeasurement` está`False`, lo que indica que no se deben medir todos los cúbits.

Envíe un programa analógico utilizando QuEra Aquila

Esta página proporciona una documentación completa sobre las capacidades de la Aquila máquina deQuEra. Los detalles que se tratan aquí son los siguientes: 1) El hamiltoniano parametrizado simulado porAquila, 2) los parámetros del programa AHS, 3) el contenido de los resultados del AHS, 4) el parámetro de capacidades. Aquila Le sugerimos que utilice la búsqueda de texto con las teclas Ctrl+F para encontrar los parámetros relevantes para sus preguntas.

Hamiltoniano

La Aquila máquina 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 desafinación local es una capacidad experimental y está disponible previa solicitud a través de Braket Direct.

where

- $H_{\text{drive},k}(t) = \left(\frac{1}{2} \Omega(t) e^{i\varphi(t)} \sigma_{-,k} + \frac{1}{2} \Omega(t) e^{-i\varphi(t)} \sigma_{+,k} \right) - \frac{1}{2} \Delta(t) n_k$, global k

- Ω (t) es la amplitud impulsora 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 que suben y bajan los espines del átomo k (en la base |↓# =|g#, |↑ p=|r#, son S =|g ~ #r|, S =(S) † =|r⇒ g|) − + −
- △ (t) es la desafinación global que depende del tiempo $_{global}$
- n_k es el operador de proyección en el estado de Rydberg del átomo k (es decir, n=|r ~ r|)

- H (t) =-△ (th) h n $_{local\ detuning,k\ local\ k\ k}$
  - △ $_{local}$ (t) es el factor dependiente del tiempo del cambio de frecuencia local, expresado en unidades de (rad/s)
  - h_k es el factor dependiente del sitio, un número adimensional entre 0.0 y 1.0
- $V_{vdw,k,l} = C_6/(d_{k,l})\ 6\ n\ n,$ $_{k\ l}$
  - C_6 es el coeficiente de van der Waals, expresado en unidades de (rad/s) * (m) ^6
  - $d_{k,l}$ es la distancia euclidiana entre el átomo k y l, medida en metros.

Los usuarios pueden controlar los siguientes parámetros mediante el esquema del programa Braket AHS.

- Disposición de átomos bidimensional ($_k$coordenadas x $_k$ e y de cada átomo k, en unidades de um), que controla las distancias atómicas por pares $d_{k,l}$ con k, l=1,2,... N
- Ω (t), la frecuencia de Rabi global y dependiente del tiempo, en unidades de (rad/s)
- φ (t), la fase global dependiente del tiempo, en unidades de (rad)
- △ $_{global}$ (t), la desafinación global dependiente del tiempo, en unidades de (rad/s)
- $_{local}$Δ(t), el factor (global) dependiente del tiempo de la magnitud de la desafinación local, en unidades de (rad/s)
- h_k, el factor (estático) dependiente del sitio de la magnitud de la desafinación local, un número adimensional entre 0.0 y 1.0

> **Note**
>
> El usuario no puede controlar qué niveles están involucrados (es decir, los operadores S_-, S_+ , n son fijos) ni la intensidad del coeficiente de interacción Rydberg-Rydberg (C). $_6$

Esquema del programa AHS de Braket

Objeto Braket.ir.AHS.Program_v1.PROGRAM (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('0.000001')]
                    ),
                    pattern='uniform'
                ),
                detuning=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('-54000000.0'), Decimal('54000000.0')],
                        times=[Decimal('0'), Decimal('0.000001')]
                    ),
```

```
                    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)

```
{
    "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, 0.000001000]
                    },
                    "pattern": "uniform"
                },
                "detuning": {
                    "time_series": {
                        "values": [-54000000.0, 54000000.0],
                        "times": [0E-9, 0.000001000]
                    },
                    "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	Lista [Lista [Decimal]]	Lista de coordenadas bidimensionales donde las pinzas atrapan los átomos

Campo de programa	type	description
setup.ahs_register.filling	Lista [int]	Marca los átomos que ocupan los sitios de captura con 1 y los sitios vacíos con 0
hamiltonian.DrivingFields [] .amplitude.time_series.times	Lista [Decimal]	puntos temporales de la amplitud impulsora, Omega (t)
hamiltonian.DrivingFields [] .amplitude.time_series.values	Lista [Decimal]	valores de la amplitud de accionamiento, Omega (t)
hamiltoniano. Campos de conducción [] .amplitud .patrón	str	patrón espacial de la amplitud de accionamiento, Omega (t); debe ser «uniforme»
hamiltonian.DrivingFields [] .phase.time_series.times	Lista [Decimal]	puntos temporales de la fase de conducción, phi (t)
hamiltonian.DrivingFields [] .phase.time_series .values	Lista [Decimal]	valores de la fase de conducción, phi (t)
hamiltonian.DrivingFields [] .phase.pattern	str	patrón espacial de la fase de conducción, phi (t); debe ser «uniforme»

Campo de programa	type	description
hamiltonian.DrivingFields [] .detuning.time_series.times	Lista [Decimal]	puntos temporales de desajuste de la conducción, delta_global (t)
hamiltonian.DrivingFields [] .detuning.time_series.values	Lista [Decimal]	valores de desajuste de conducción, delta_global (t)
hamiltonian.DrivingFields [] .dettuning.pattern	str	patrón espacial de desajuste de conducción, delta_global (t); debe ser «uniforme»
Hamiltonian.LocalDetuning [] .magnitude.time_series.times	Lista [Decimal]	puntos temporales del factor dependiente del tiempo de la magnitud de desafinación local, delta_local (t)
Hamiltonian.LocalDetuning [] .magnitude.time_series.values	Lista [Decimal]	valores del factor dependiente del tiempo de la magnitud de desafinación local, delta_local (t)

Campo de programa	type	description
Hamiltonian.LocalDetuning [] .magnitude.pattern	Lista [Decimal]	factor dependiente del sitio de la magnitud de desafinación local, h_k (los valores corresponden a los sitios de setup.ahs_register.sites)

Campos de metadatos

Campo de programa	type	description
corchete .name SchemaHeader	str	nombre del esquema; debe ser 'braket.ir.ahs.program'
SchemaHeadercorchete: .versión	str	versión del esquema

Esquema de resultados de la tarea 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
medidas [] .shotResult.PreSequence	Lista [int]	Bits de medición presecuen ciados (uno para cada sitio atómico) para cada toma: 0 si el sitio está vacío, 1 si el sitio está lleno, medidos antes de las secuencias de pulsos que impulsan la evolución cuántica
medidas [] .shotResult.postSequence	Lista [int]	Bits de medición posteriores a la secuencia para cada toma:

Campo de resultados de la tarea	type	description
		0 si el átomo está en el estado de Rydberg o el sitio está vacío, 1 si el átomo está en el estado fundamental, medidos al final de las secuencias de pulsos que rigen la evolución cuántica

Campos de metadatos

Campo de resultados de la tarea	type	description
corchete .name SchemaHeader	str	nombre del esquema; debe ser 'braket.task_result.analog_hamiltonian_simulation_task_result'
corchete. versión SchemaHeader	str	versión del esquema
TaskMetadata.braket .name SchemaHeader	str	nombre del esquema; debe ser 'braket.task_result.task_metadata'
TaskMetadata.braket .version SchemaHeader	str	versión del esquema

Campo de resultados de la tarea	type	description
taskMetadata.id	str	El ID de la tarea cuántica. Para las tareas AWS cuánticas, esta es la tarea cuántica ARN.
Metadatos de tareas. Fotos	int	El número de disparos de la tarea cuántica
Metadata.shots.DeviceID de la tarea	str	El ID del dispositivo en el que se ejecutó la tarea cuántica. En el AWS caso de los dispositivos, este es el ARN del dispositivo.

Campo de resultados de la tarea	type	description
Metadata.shots.creado en	str	La marca de tiempo de la creación; el formato debe estar en el formato de cadena ISO-8601/ RFC3339 yyyy-MM-DDTHH:mm:ss.sssz. El valor predeterminado es Ninguno.
TaskMetadata.shots.endat	str	La marca de tiempo que indica cuándo finalizó la tarea cuántica; el formato debe estar en el formato de cadena ISO-8601/ RFC3339 yyyy-MM-DDTHH:mm:ss.sssz. El valor predeterminado 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.
Motivo del error de TaskMetadata.Shots.	str	La razón del fracaso de la tarea cuántica. El valor predeterminado es Ninguno.
Metadatos adicionales. Acción	Braket.ir.AHS.Program_v1.Program	(Consulte la sección del esquema del programa Braket AHS)
Metadata.action.Braket .queraMetadata.name SchemaHeader	str	nombre del esquema; debe ser 'braket.task_result.quera_metadata'

Campo de resultados de la tarea	type	description
SchemaHeaderMetadata.action.Braket.QueraMetadata.Version	str	versión del esquema
Metadata.action.num adicionales SuccessfulShots	int	número de disparos completamente exitosos; debe ser igual al número de disparos solicitado
medidas [] .shotMetadata.shotStatus	int	El estado de la toma (éxito, éxito parcial, fracaso); debe ser «Éxito»

QuEra esquema de propiedades del dispositivo

braket.device_schema.quera.quera_device_capabilities_v1. QueraDeviceCapacidades (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 del servicio

Campo de propiedades del servicio	type	description
Service.ExecutionWindows []. ExecutionDay	ExecutionDay	Días de la ventana de ejecución; deben ser «Todos los días», «días laborables», «fin de semana», «lunes», «martes», «miércoles», jueves, «viernes», «sábado» o «domingo»
service.executionWindows [] .window StartHour	fecha y hora	Formato UTC de 24 horas de la hora en que se inicia la ventana de ejecución
Service.ExecutionWindows [] .window EndHour	fecha y hora	Formato UTC de 24 horas de la hora en que finaliza la ventana de ejecución
service.qpu_capabilities.service.shotsRange	Tupla [int, int]	Número mínimo y máximo de disparos para el 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», «tiro», «tarea»

Campos de metadatos

Campo de metadatos	type	description
acción [] .versión	str	versión del esquema del programa AHS
acción [] .actionType	ActionType	Nombre del esquema del programa AHS; debe ser 'braket.ir.ahs.program'
SchemaHeaderservice.braket .name	str	nombre del esquema; debe ser 'braket.device_schema.device_service_properties'
SchemaHeaderservice.braket .version	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. Externo DocumentationUrl	str	URL de documentación externa
Service.Ubicación del dispositivo	str	ubicación geográfica del dispositivo
Servicio. Actualizado en	datetime	hora en que se actualizaron las propiedades del dispositivo por última vez

Trabajando con 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 un acceso a Braket orientado a objetos y de API bajo nivel. Amazon

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

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

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

En esta sección:

- Encienda el cliente Amazon Braket Boto3
- Configurar AWS CLI perfiles para Boto3 y el SDK de Amazon Braket

Encienda el cliente Amazon Braket Boto3

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

> **Note**
>
> Por motivos de compatibilidad con versiones anteriores de BraketSchemas, la información de OpenQASM se omite en las llamadas. `GetDevice` API Para obtener esta información, el agente de usuario debe presentar una versión reciente de la BraketSchemas versión 1.8.0 o posterior. El SDK de Braket le informa de ello automáticamente. Si no ve los resultados de OpenQASM en la `GetDevice` respuesta cuando utiliza un SDK de Braket, puede que necesite configurar la variable de `AWS_EXECUTION_ENV` entorno para configurar el agente de usuario. Consulta los ejemplos de código que aparecen en el tema de error GetDevice No

devuelve resultados de OpenQASM para saber cómo hacerlo con los SDK Boto3 y AWS CLI Go, Java y/. JavaScript TypeScript

```
import boto3
import botocore

braket = boto3.client("braket",
 config=botocore.client.Config(user_agent_extra="BraketSchemas/1.8.0"))
```

Ahora que ya tiene un `braket` cliente establecido, puede realizar solicitudes y procesar las respuestas desde el servicio Braket. Amazon Puedes 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.

- Busca dispositivos
- Recupera un dispositivo
- Crea una tarea cuántica
- Recupera una tarea cuántica
- Busque tareas cuánticas
- Cancela la tarea cuántica

Busca dispositivos

- `search_devices(**kwargs)`

Busque 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'])
```

Recupera un dispositivo

- `get_device(deviceArn)`

Recupera 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']}")
```

Crea una tarea cuántica

- `create_quantum_task(**kwargs)`

Crea 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")
```

Recupera una tarea cuántica

- `get_quantum_task(quantumTaskArn)`

Recupera 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'])
```

Busque tareas cuánticas

- `search_quantum_tasks(**kwargs)`

Busque 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']}")
```

Cancela la tarea cuántica

- `cancel_quantum_task(quantumTaskArn)`

Cancela 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']}")
```

Configurar AWS CLI perfiles para Boto3 y el SDK de Amazon Braket

El SDK de Amazon Braket se basa en las AWS CLI credenciales predeterminadas, a menos que especifique lo contrario de forma explícita. Te recomendamos que mantengas la configuración predeterminada cuando ejecutes un portátil Amazon Braket gestionado, ya que debes proporcionar un rol de IAM que tenga permisos para lanzar la instancia del bloc de notas.

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 periódicamente el perfil predeterminado.

Esta sección proporciona una breve explicación de cómo configurar dicha CLI `profile` y cómo incorporar ese perfil en Amazon Braket para que las API llamadas se realicen con los permisos de ese perfil.

En esta sección:

- Paso 1: Configurar un local AWS CLIprofile
- Paso 2: Establecer un objeto de sesión de Boto3
- Paso 3: Incorpora la sesión de Boto3 al Braket AwsSession

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

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

- Introducción

- Configuración del que se AWS CLI va a utilizar AWS IAM Identity Center

Para usar Amazon Braket, debe proporcionar a este usuario (y a la CLI asociada`profile`) los permisos de Braket necesarios. 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 API llamadas esperadas tienen restricciones basadas en regiones que no están alineadas con tu región `profile` predeterminada, puedes 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`)
```

Para el argumento designado como`region`, sustituya 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

El siguiente ejemplo muestra cómo inicializar una sesión de Boto3 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, puedes enviar tareas cuánticas a ese `AwsDevice` objeto instanciado (por ejemplo, llamando al comando). `device.run(…)` Todas las API llamadas realizadas por ese dispositivo pueden aprovechar las credenciales de IAM asociadas al perfil CLI que designó anteriormente`profile`.

Control de pulsos en Amazon Braket

En esta sección se explica cómo utilizar el control de impulsos en varias QPU de Amazon Braket.

En esta sección:

- Braket Pulse
- Funciones de los marcos y los puertos
- Hola, Pulse
- Acceder a las puertas nativas mediante pulsos

Braket Pulse

Los pulsos son las señales analógicas que controlan los cúbits en un ordenador cuántico. Con algunos dispositivos de Amazon Braket, puedes acceder a la función de control de pulsos para enviar circuitos mediante pulsos. Puede acceder al control de pulsos a través del SDK de Braket, mediante OpenQASM 3.0, o directamente a través de las API de Braket. Primero, presentemos algunos conceptos clave para el control de pulsos en Braket.

Fotogramas

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 cúbit, un marco determina la frecuencia portadora del cúbit, el desfase y el momento en el 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.pulse import Frame
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")
drive_frame = device.frames["q0_rf_frame"]

device = AwsDevice("arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy")
readout_frame = Frame(name="r0_measure", port=port0, frequency=5e9, phase=0)
```

Puertos

Un puerto es una abstracción de software que representa cualquier componente de hardware de entrada/salida 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 cúbits. Los puertos se caracterizan por una sola cadena 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)
```

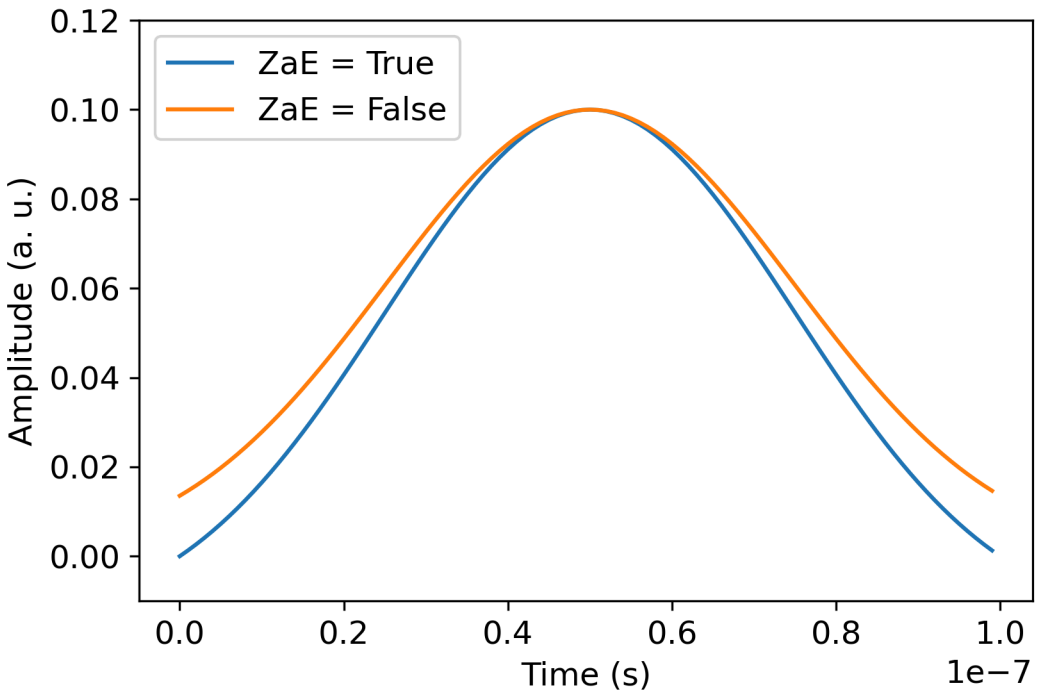
Formas de onda

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 las formas de onda directamente mediante una lista de números complejos o mediante una plantilla de forma de onda para generar una lista del proveedor del hardware.

```
from braket.pulse import ArbitraryWaveform, ConstantWaveform
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 compuerta adiabática (DRAG). Puede recuperar los datos de la forma de onda a través de la `sample` función para dibujar la forma de la onda, como se muestra en el siguiente ejemplo.

```
gaussian_waveform = GaussianWaveform(1e-7, 25e-9, 0.1)
x = np.arange(0, gaussian_waveform.length, drive_frame.port.dt)
plt.plot(x, gaussian_waveform.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 están centradas 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 cambia la escala de su amplitud de manera que el valor máximo corresponda al argumento. `amplitude`

Ahora que hemos explicado los conceptos básicos del acceso a nivel de pulsos, a continuación veremos cómo construir un circuito utilizando compuertas y pulsos.

Funciones de los marcos y los puertos

En esta sección se describen los marcos y puertos predefinidos disponibles para cada dispositivo. También analizaremos brevemente los mecanismos que intervienen cuando se reproducen los pulsos en determinados fotogramas.

Rigetti

Frames (Fotogramas)

Rigettilos dispositivos admiten fotogramas predefinidos cuya frecuencia y fase están calibradas para que estén en resonancia con el cúbit asociado. La nomenclatura es `q{i}[_q{j}]_{role}_frame` aquella en la que `{i}` se hace referencia al primer qubit, `{j}` al segundo qubit en caso de que el fotograma sirva para activar una interacción de dos qubits, y `{role}` se refiere a la función del fotograma. Las funciones son las siguientes:

- `rf`es el marco para controlar la transición 0-1 del qubit. Los pulsos se transmiten como señales transitorias de microondas de frecuencia y fase previamente proporcionadas a través de las funciones y. `set shift` La amplitud de la señal en función del tiempo viene dada por la forma de onda reproducida en el fotograma. El marco permite una interacción de un solo qubit, fuera de la diagonal. Para obtener más información, consulte Krantz et al. y Rahamim et al. .
- `rf_f12`es similar a la transición 1-2 `rf` y sus parámetros apuntan a ella.
- `ro_rx`se utiliza para lograr una lectura dispersiva del cúbit a través de una guía de ondas coplanar acoplada. La frecuencia, la fase y el conjunto completo de parámetros de la forma de onda de lectura están precalibrados. Actualmente se utiliza mediante el`capture_v0`, que no requiere ningún argumento aparte del identificador de fotograma.
- `ro_tx`sirve para transmitir señales desde el resonador. Actualmente no se utiliza.
- `cz`es un marco calibrado para habilitar la puerta de dos cúbits`cz`. Como ocurre con todos los fotogramas asociados a un `ff` puerto, activa una interacción entrelazada a través de la línea de flujo al modular el cúbit ajustable del par en función de la resonancia con su vecino. Para obtener más información sobre el mecanismo de entrelazamiento, consulte Reagor et al. , Caldwell y col. , y Didier y col. .
- `cphase`es un marco calibrado para habilitar la `cphaseshift` puerta de dos cúbits y está conectado a un puerto. `ff` Para obtener más información sobre el mecanismo de enredo, consulte la descripción del marco. `cz`
- `xy`es un marco calibrado para habilitar las compuertas XY (θ) de dos cúbits y que está conectado a un puerto. `ff` Para obtener más información sobre el mecanismo de enmarañamiento y sobre cómo conseguir las compuertas XY, consulte la descripción del `cz` marco y la de Abrams et al. .

A medida que las tramas basadas en el `ff` puerto cambien la frecuencia del cúbit sintonizable, todas las demás tramas impulsoras relacionadas con el cúbit se desfasarán en una cantidad que esté relacionada con la amplitud y la duración del cambio de frecuencia. Por lo tanto, debes compensar este efecto añadiendo el cambio de fase correspondiente a los fotogramas de los cúbits vecinos.

Puertos

Los Rigetti dispositivos proporcionan una lista de puertos que puede inspeccionar a través de las capacidades del dispositivo. Los nombres de los puertos siguen la convención, `q{i}_{type}` donde `{i}` se refieren al número de qubit y `{type}` al tipo de puerto. Tenga en cuenta que no todos los qubits tienen un conjunto completo de puertos. Los tipos de puertos son los siguientes:

- `rf`representa la interfaz principal para impulsar la transición de un solo qubit. Está asociada a los marcos `rf` y. `rf_f12` Está acoplado capacitivamente al cúbit, lo que permite la conducción de microondas en el rango de los gigahercios.
- `ro_tx`sirve para transmitir señales al resonador de lectura acoplado capacitivamente al qubit. La entrega de la señal de lectura se multiplexa ocho veces por octágono.
- `ro_rx`sirve para recibir señales del resonador de lectura acoplado al cúbit.
- `ff`representa la línea de flujo rápido acoplada inductivamente al cúbit. Podemos usar esto para ajustar la frecuencia del transmón. Solo los qubits diseñados para ser altamente ajustables tienen un puerto. `ff` Este puerto sirve para activar la interacción qubit-qubit, ya que hay un acoplamiento capacitivo estático entre cada par de transmones vecinos.

Para obtener más información sobre la arquitectura, consulte Valery et al. .

OQC

Frames (Fotogramas)

OQClos dispositivos admiten fotogramas predefinidos cuya frecuencia y fase están calibradas para que estén en resonancia con el cúbit asociado. La convención de nomenclatura de estos fotogramas es la siguiente:

- marco impulsor: `q{i}[_q{j}]_{role}` donde `{i}` se refiere al primer número de qubits, `{j}` se refiere al segundo número de qubit en caso de que el marco sirva para activar una interacción de dos cúbits y `{role}` se refiere a la función del marco, tal como se describe a continuación.
- marco de lectura de qubits: `r{i}_{role}` donde `{i}` se refiere al número de qubits y a la función del marco, tal como `{role}` se describe a continuación.

Recomendamos usar cada marco para la función que se diseñó de la siguiente manera:

- `drive`se utiliza como marco principal para controlar la transición 0-1 del qubit. Los pulsos se transmiten como señales transitorias de microondas de frecuencia y fase previamente proporcionadas a través de las funciones y. `set shift` La amplitud de la señal en función del

tiempo viene dada por la forma de onda reproducida en el fotograma. El marco permite una interacción de un solo qubit, fuera de la diagonal. Para obtener más información, consulte Krantz et al. y Rahamim et al. .

- `second_state`es equivalente al `drive` fotograma, pero su frecuencia se ajusta en resonancia con la transición 1-2.
- `measure`es para lectura. La frecuencia, la fase y el conjunto completo de parámetros de la forma de onda de lectura están precalibrados. Actualmente se usa a través de`capture_v0`, que no requiere ningún argumento aparte del identificador de fotograma.
- `acquire`sirve para capturar las señales del resonador. Actualmente no se utiliza.
- `cross_resonance`activa la interacción de resonancia cruzada entre los cúbits `i` y `j` acciona el cúbit de control `i` a la frecuencia de transición del cúbit objetivo. `j` En consecuencia, la frecuencia del cuadro se establece utilizando la frecuencia del qubit objetivo. La interacción se produce con una velocidad proporcional a la amplitud de esta unidad de resonancia cruzada. Los diferentes tipos de interferencias inducen efectos no deseados que requieren correcciones. Véase Patterson et al. para obtener más información sobre la interacción de resonancia cruzada con los qubits transmónicos de forma coaxial («coaxmones»).
- `cross_resonance_cancellation`le ayuda a añadir correcciones para suprimir los efectos nocivos inducidos por las interferencias cuando se activa la interacción de resonancia cruzada. La frecuencia de cuadro inicial se establece en la frecuencia de transición del cúbit de control. `i` Para obtener más información sobre el método de cancelación, consulte Patterson et al. .

Puertos

Los OQC dispositivos proporcionan una lista de puertos que puede inspeccionar a través de las capacidades del dispositivo. Las tramas descritas anteriormente están asociadas a puertos que se identifican por su identificador, `channel_{N}` donde `{N}` es un número entero. Los puertos son la interfaz para controlar las líneas (dirección`tx`) y los resonadores de lectura (dirección`rx`) conectados a los coaxmons. Cada qubit está asociado a una línea de control y a un resonador de lectura. El puerto de transmisión es la interfaz para la manipulación de un qubit y dos qubits. El puerto de recepción sirve para la lectura de qubits.

Hola, Pulse

Aquí aprenderá a construir un par de campanas simple directamente con pulsos y a ejecutar este programa de pulsos en el Rigetti dispositivo. Un par Bell es un circuito de dos cúbits que consiste en

una puerta de Hadamard en el primer qubit seguida de una `cnot` puerta entre el primer y el segundo qubit. La creación de estados entrelazados con pulsos requiere mecanismos específicos que dependen del tipo de hardware y de la arquitectura del dispositivo. No utilizaremos un mecanismo nativo para crear la `cnot` puerta. En su lugar, utilizaremos formas de onda y marcos específicos que habiliten la `cz` puerta de forma nativa. En este ejemplo, crearemos una puerta de Hadamard utilizando las puertas nativas de un solo qubit `rx` `rz` y expresaremos la puerta mediante pulsos. `cz`

Primero, importemos las bibliotecas necesarias. Además de la `Circuit` clase, ahora también tendrás que `PulseSequence` importarla.

```
from braket.aws import AwsDevice
from braket.pulse import PulseSequence, ArbitraryWaveform, GaussianWaveform

from braket.circuits import Circuit
import braket.circuits.circuit as circuit
```

A continuación, cree una instancia de un nuevo dispositivo Braket con el nombre de recurso de Amazon (ARN) del dispositivo. Rigetti Aspen-M-3 Consulta la página Dispositivos de la consola Amazon Braket para ver el diseño del Rigetti Aspen-M-3 dispositivo.

```
a=10 #specifies the control qubit
b=113 #specifies the target qubit
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")
```

Como la compuerta Hadamard no es una compuerta nativa del Rigetti dispositivo, no se puede utilizar en combinación con pulsos. Por lo tanto, es necesario descomponerla en una secuencia de puertas y compuertas nativas. `rx` `rz`

```
import numpy as np
import matplotlib.pyplot as plt
@circuit.subroutine(register=True)
def rigetti_native_h(q0):
    return (
        Circuit()
        .rz(q0, np.pi)
        .rx(q0, np.pi/2)
        .rz(q0, np.pi/2)
        .rx(q0, -np.pi/2)
    )
```

Para la `cz` compuerta, utilizaremos una forma de onda arbitraria con parámetros (amplitud, tiempo de subida/bajada y duración) predeterminados por el proveedor del hardware durante una etapa de calibración. Esta forma de onda se aplicará a. `q10_q113_cz_frame` Para obtener una versión más reciente de la forma de onda arbitraria utilizada aquí, consulte QCS, en el sitio web. Rigetti Es posible que deba crear una cuenta de QCS.

```
a_b_cz_wfm = ArbitraryWaveform([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00017888439538396808, 0.00046751103636033026, 0.0011372942989106456,
 0.00257705961192697, 0.005443941944632366, 0.010731922770068104, 0.01976701723583167,
 0.03406712171899736, 0.05503285980691202, 0.0835067075582934, 0.11932853352131022,
 0.16107456696238298, 0.20614055551722368, 0.2512065440720643, 0.292952577513137,
 0.328774403476157, 0.3572482512275353, 0.3782139893154499, 0.3925140937986156,
 0.40154918826437913, 0.4068371690898149, 0.4097040514225177, 0.41114381673553674,
 0.411813599998087, 0.4121022266390633, 0.4122174383870584, 0.41226003881132406,
 0.4122746298554775, 0.4122792591252675, 0.4122806196003006, 0.41228098995582513,
 0.41228108334474756, 0.4122811051578895, 0.4122811098772742, 0.4122811108230642,
 0.4122811109986316, 0.41228111102881937, 0.41228111103362725, 0.4122811110343365,
 0.41228111103443343, 0.4122811110344457, 0.4122811110344471, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.4122811110344471, 0.4122811110344457, 0.41228111103443343, 0.4122811110343365,
 0.41228111103362725, 0.41228111102881937, 0.4122811109986316, 0.4122811108230642,
 0.4122811098772742, 0.4122811051578895, 0.41228108334474756, 0.41228098995582513,
 0.4122806196003006, 0.4122792591252675, 0.4122746298554775, 0.41226003881132406,
 0.4122174383870584, 0.4121022266390633, 0.411813599998087, 0.41114381673553674,
 0.4097040514225176, 0.4068371690898149, 0.40154918826437913, 0.3925140937986155,
 0.37821398931544986, 0.3572482512275351, 0.32877440347615655, 0.2929525775131368,
 0.2512065440720641, 0.20614055551722307, 0.16107456696238268, 0.11932853352131002,
 0.08350670755829034, 0.05503285980691184, 0.03406712171899729, 0.01976701723583167,
 0.010731922770068058, 0.005443941944632366, 0.002577059611929697,
 0.0011372942989106229, 0.00046751103636033026, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0])
a_b_cz_frame = device.frames[f'q{a}_q{b}_cz_frame']

dt = a_b_cz_frame.port.dt
a_b_cz_wfm_duration = len(a_b_cz_wfm.amplitudes)*dt
print('CZ pulse duration:', a_b_cz_wfm_duration*1e9, 'ns')
```

Esto debería devolver:

```
CZ pulse duration: 124 ns
```

Ahora podemos construir la `cz` puerta usando la forma de onda que acabamos de definir. Recuerde que la `cz` puerta consiste en un cambio de fase del cúbit objetivo si el cúbit de control está en ese estado. `|1>`

```
phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

a_rf_frame = device.frames[f'q{a}_rf_frame']
b_rf_frame = device.frames[f'q{b}_rf_frame']

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

cz_pulse_sequence = (
    PulseSequence()
    .barrier(frames)
    .play(a_b_cz_frame, a_b_cz_wfm)
    .delay(a_rf_frame, a_b_cz_wfm_duration)
    .shift_phase(a_rf_frame, phase_shift_a)
    .delay(b_rf_frame, a_b_cz_wfm_duration)
    .shift_phase(b_rf_frame, phase_shift_b)
    .barrier(frames)
)
```

La `a_b_cz_wfm` forma de onda se reproduce en un fotograma que está asociado a un puerto de flujo rápido. Su función es cambiar la frecuencia del cúbit para activar una interacción qubit-qubit. Para obtener más información, consulte Funciones de los marcos y los puertos. A medida que varía la frecuencia, los fotogramas de los qubits giran a velocidades diferentes a las de los `rf` fotogramas de un solo qubit que se mantienen intactos: estos últimos se desfasan. Estos cambios de fase se calibraron previamente mediante Ramsey secuencias y se proporcionan aquí como información codificada de principio a fin (período completo). `phase_shift_a` `phase_shift_b` Corregimos esta separación de fases utilizando `shift_phase` las instrucciones de los marcos. `rf` Tenga en cuenta que esta secuencia solo funcionará en programas en los que no haya ningún XY fotograma relacionado con el qubit a y b se utilice, ya que no compensamos el cambio de fase que se produce en estos fotogramas. Este es el caso de este programa de un solo par de Bell, que utiliza únicamente `cz` fotogramas `rf` y. Para obtener más información, consulte Caldwell et al. .

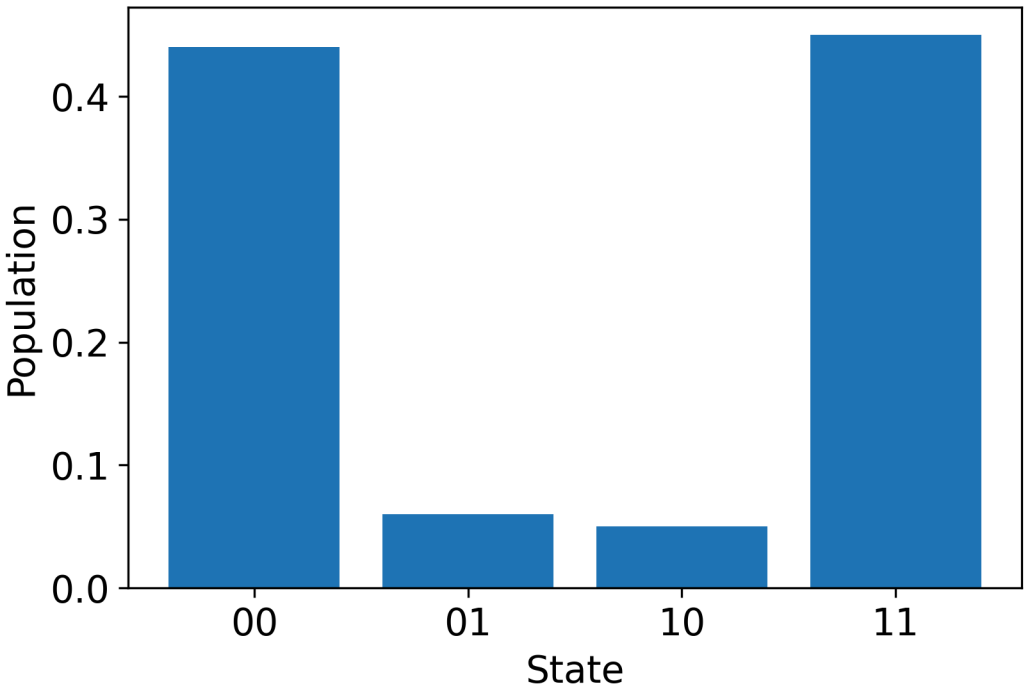
Ahora estamos listos para crear un par de campanas con pulsos.

```
bell_circuit_pulse = (
    Circuit()
    .rigetti_native_h(a)
    .rigetti_native_h(b)
    .pulse_gate([a, b], cz_pulse_sequence)
    .rigetti_native_h(b)
)
print(bell_circuit_pulse)
```

```
T  : |   0    |   1    |   2    |    3    |4 |   5    |   6    |   7    |    8    |

q5 : -Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-PG-------------------------------------
                                            |
q6 : -Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-PG-Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-

T  : |   0    |   1    |   2    |    3    |4 |   5    |   6    |   7    |    8    |
```

Vamos a ejecutar este par de campanas en el Rigetti dispositivo. Ten en cuenta que ejecutar este bloque de código tendrá un coste. Para obtener más información sobre estos costes, consulta la página de precios Braket de Amazon. Te recomendamos que pruebes tus circuitos con una pequeña cantidad de disparos para asegurarte de que funcionan en el dispositivo antes de aumentar el número de disparos.

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

counts = task.result().measurement_counts

plt.bar(sorted(counts), [counts[k] for k in sorted(counts)])
```



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 la programación directa OpenPulse de pulsos mediante la representación OpenQASM 3.0.

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

En este ejemplo, construiremos un circuito Bell con OpenQASM 3.0 y, OpenPulse en un dispositivo, con transmones ajustables en frecuencia. Recuerde que un circuito Bell es un circuito de dos cúbits que consiste en una puerta de Hadamard en el primer cúbit seguida de una puerta entre los dos cúbits. `cnot` Como `cnot` las compuertas se diferencian de `cz` las compuertas solo por una transformación básica, aquí definiremos un par Bell utilizando Hadamard y, en su lugar, `cz` compuertas, ya que el dispositivo proporciona una forma más sencilla de crear compuertas para esta demostración. `cz`

Empecemos por definir la puerta de Hadamard utilizando las puertas nativas del dispositivo.

```
client = boto3.client('braket', region_name='us-west-1')
defcal h $10 {
    rz(pi) $10;
    rx(pi/2) $10;
    rz(pi/2) $10;
    rx(-pi/2) $10;
}
defcal h $113 {
    rz(pi) $113;
    rx(pi/2) $113;
    rz(pi/2) $113;
    rx(-pi/2) $113;
}
```

Para la compuerta CZ, utilizaremos una forma de onda arbitraria con parámetros (amplitud, tiempo de subida/bajada y duración) predeterminados de antemano. Esta forma de onda se aplicará a la. `q10_q113_cz_frame`

```
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.484432196472809,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
0.48443979687791755, 0.4844396697373718, 0.484438680618381, 0.484432196472809 6,
```

```
 0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
 0.4753811409710734, 0.46121311095968553, 0.4331554251320285, 0.38631750509562957,
 0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
 0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
 0.0008644760431374626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
}
defcal cz $10, $113 {
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
    play(q10_q113_cz_frame, q10_q113_cz_wfm);
    delay[124ns] q10_rf_frame;
    shift_phase(q10_rf_frame, 1.1733407221086924);
    delay[124ns] q113_rf_frame;
    shift_phase(q113_rf_frame, 6.269846678712192);
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
}
```

La duración de la `q10_q113_cz_wfm` forma de onda es de 124 muestras, lo que corresponde a 124 ns, ya que el incremento de tiempo mínimo `dt` es de 1 ns.

La `q10_q113_cz_wfm` forma de onda se reproduce en un fotograma enlazado a un puerto de flujo rápido. Su función es cambiar la frecuencia del cúbit para activar una interacción qubit-qubit. Para obtener más información, consulte Funciones de los marcos y los puertos. A medida que varía la frecuencia, los fotogramas del qubit giran a velocidades diferentes en comparación con los `rf` fotogramas de un solo qubit que se mantienen intactos: estos últimos se desfasan. Este desfase puede medirse con Ramsey secuencias durante una fase de calibración y compensarse con instrucciones sobre los fotogramas. `shift_phase` `rf` `xy` Para obtener más información, consulte Caldwell et al. .

Ahora podemos ejecutar el circuito de pares Bell, en el que descomponemos la `cnot` puerta con un par de Hadamard y Gates. `cz`

```
bit[2] c;
h $10;
h $113;
cz $10, $113;
h $113;
c[0] = measure $10;
c[1] = measure $113;
```

La representación completa de OpenQASM 3.0 del circuito Bell construido con una combinación de puertas y pulsos nativos es la siguiente.

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.4835753359644072, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.484438680618381 7, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
 0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
 0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.4844321964728096,
 0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
 0.4753811409710734, 0.46121311095968553, 0.4331554251320285, 0.38631750509562957,
 0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
 0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
 0.0008644760431374626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
}
defcal h $10 {
    rz(pi) $10;
    rx(pi/2) $10;
    rz(pi/2) $10;
    rx(-pi/2) $10;
}
defcal h $113 {
    rz(pi) $113;
    rx(pi/2) $113;
    rz(pi/2) $113;
    rx(-pi/2) $113;
```

```
}
defcal cz $10, $113 {
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
    play(q10_q113_cz_frame, q10_q113_cz_wfm);
    delay[124ns] q10_rf_frame;
    shift_phase(q10_rf_frame, 1.1733407221086924);
    delay[124ns] q113_rf_frame;
    shift_phase(q113_rf_frame, 6.269846678712192);
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
}
bit[2] c;
h $10;
h $113;
cz $10, $113;
h $113;
c[0] = measure $10;
c[1] = measure $113;
```

Ahora puede usar el SDK de Braket para ejecutar este programa OpenQASM 3.0 en el dispositivo mediante el siguiente código. Rigetti

```
# import the device module
from braket.aws import AwsDevice
from braket.ir.openqasm import Program

client = boto3.client('braket', region_name='us-west-1')

with open("pulse.qasm", "r") as pulse:
    pulse_qasm_string = pulse.read()

# choose the Rigetti device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")

program = Program(source=pulse_qasm_string)
my_task = device.run(program)

# You can also specify an optional s3 bucket location and number of shots,
 # if you so choose, when running the program
s3_location = ("amazon-braket-my-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
```

```
)
```

Acceder a las 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. Los proveedores de hardware calibran cuidadosamente las secuencias de pulsos, pero acceder a ellas brinda a los investigadores la oportunidad de diseñar mejores compuertas o explorar protocolos para mitigar errores, como la extrapolación de ruido cero mediante el estiramiento de los pulsos de compuertas 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/Aspen-M-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 compuertas más recientes.

```
device.refresh_gate_calibrations()
```

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

```
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 `filter` función. Pasas 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 compuertas 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)
.cz(0,1)
.rx(1,-math.pi/2)
)
```

Puede ejecutarlo con una calibración de puerta personalizada para la `rx` puerta activada pasando `qubit 0` un diccionario de `PulseSequence` objetos al argumento de la `gate_definitions` palabra clave. Puede crear un diccionario a partir del atributo `pulse_sequences` del `GateCalibrations` objeto. Todas las compuertas 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)
```

Guía del usuario de Amazon Braket Hybrid Jobs

En esta sección se proporcionan instrucciones sobre cómo configurar y gestionar los trabajos híbridos en Amazon Braket.

Puede acceder a los trabajos híbridos en Braket mediante:

- El SDK para Python de Amazon Braket.
- La consola Amazon Braket.
- El Amazon API Braket.

En esta sección:

- ¿Qué es un Hybrid Job?
- Cuándo usar Amazon Braket Hybrid Jobs
- Ejecute su código local como un trabajo híbrido
- Realice un trabajo híbrido con Amazon Braket Hybrid Jobs
- Cree su primer trabajo híbrido
- Entradas, salidas, variables de entorno y funciones auxiliares
- Guarde los resultados del trabajo
- Guarde y reinicie los trabajos híbridos mediante puntos de control
- Defina el entorno para el script de su algoritmo
- Usar hiperparámetros
- Configure la instancia de trabajo híbrida para ejecutar el script de su algoritmo
- Cancelar un trabajo híbrido
- Uso de la compilación paramétrica para acelerar las tareas híbridas
- PennyLane Utilízalo con Amazon Braket
- Utilice Amazon Braket Hybrid Jobs y ejecute un PennyLane algoritmo QAOA
- Acelere sus cargas de trabajo híbridas con simuladores integrados de PennyLane
- Cree y depure un trabajo híbrido con el modo local
- Traiga su propio contenedor (BYOC)

- Configure el depósito predeterminado en AwsSession
- Interactúe directamente con los trabajos híbridos mediante el API

¿Qué es un Hybrid Job?

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). Hybrid Jobs está diseñado para utilizar los recursos clásicos solicitados, ejecutar el algoritmo y liberar las instancias una vez terminadas, de forma que solo pague por lo que utilice.

Hybrid Jobs es ideal para algoritmos iterativos de larga duración que utilizan recursos clásicos y cuánticos. Usted envía su algoritmo para que se ejecute, Braket lo ejecuta en un entorno contenerizado escalable y recupera los resultados cuando el algoritmo está completo.

Además, las tareas cuánticas creadas a partir de un trabajo híbrido se benefician de una mayor prioridad al hacer cola en una QPU de destino. Esto garantiza que sus tareas cuánticas se procesen y se ejecuten antes que las demás tareas de la cola. Esto es particularmente beneficioso para los algoritmos híbridos iterativos, en los que la tarea posterior depende de los resultados de las tareas cuánticas anteriores. Algunos ejemplos de estos algoritmos son el algoritmo de optimización cuántica aproximada (QAOA), el autosolucionador cuántico variacional o el aprendizaje automático cuántico. También puedes monitorizar el progreso de tu algoritmo prácticamente en tiempo real, lo que te permitirá llevar un registro de los costes, el presupuesto o las métricas personalizadas, como las pérdidas por formación o los valores esperados.

Cuándo usar Amazon Braket Hybrid Jobs

Amazon Braket Hybrid Jobs le permite ejecutar algoritmos híbridos cuántico-clásicos, como el solucionador propio cuántico variacional (VQE) y el algoritmo de optimización cuántica aproximada (QAOA), que combinan recursos informáticos clásicos con dispositivos de computación cuántica para optimizar el rendimiento de los sistemas cuánticos actuales. Amazon Braket Hybrid Jobs ofrece tres ventajas principales:

1. Rendimiento: Amazon Braket Hybrid Jobs ofrece un mejor rendimiento que la ejecución de algoritmos híbridos desde su propio entorno. Mientras su trabajo está en ejecución, 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. Amazon Braket Hybrid Jobs también admite

la compilación paramétrica. Puede enviar un circuito utilizando parámetros gratuitos y Braket compila el circuito una vez, sin necesidad de volver a compilarlo para futuras actualizaciones de parámetros en el mismo circuito, lo que se traduce en tiempos de ejecución aún más rápidos.

2. Comodidad: Amazon Braket Hybrid Jobs simplifica la configuración y la administración del entorno informático y lo mantiene en funcionamiento mientras se ejecuta el algoritmo híbrido. Solo tiene que proporcionar el script del 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 prediseñados, devuelve los resultados a Amazon Simple Storage Service (Amazon S3) y libera los recursos informáticos.
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.

Ejecute su código local como un trabajo híbrido

Amazon Braket Hybrid Jobs proporciona una orquestación totalmente gestionada de algoritmos híbridos cuántico-clásicos, que combina los recursos informáticos 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 sobre las tareas cuánticas individuales para que sus algoritmos no se vean interrumpidos por las fluctuaciones de 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:

- Cree un trabajo híbrido a partir del código Python local
- Instalar paquetes y código fuente de Python adicionales
- Guarde y cargue datos en una instancia de trabajo híbrida
- Mejores prácticas para decoradores de trabajos híbridos

Cree un trabajo híbrido a partir del código Python local

Puede ejecutar su código Python local como un Amazon Braket Hybrid Job. Para ello, anote el código con un `@hybrid_job` decorador, como se muestra en el siguiente ejemplo de código. Para

entornos personalizados, puede optar por utilizar un contenedor personalizado de Amazon Elastic Container Registry (ECR).

> **Note**
>
> De forma predeterminada, solo se admite Python 3.10.

Puede usar el `@hybrid_job` decorador para anotar una función. Braket transforma el código 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 Braket. El siguiente ejemplo de código muestra cómo ejecutar una secuencia de cinco estados en. 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 decoradora devuelve el identificador de 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 dispositivo del `@hybrid_job` decorador especifica el dispositivo al que la tarea híbrida tiene acceso prioritario, en este caso, el SV1 simulador. 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 de ayuda `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 contenerizado 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 `run_hybrid_job()` función utiliza el argumento `num_tasks` para controlar el número de tareas cuánticas creadas. El trabajo híbrido lo captura automáticamente como un hiperparámetro.

> **Note**
>
> Los hiperparámetros se muestran en la consola Braket como cadenas, con un límite de 2500 caracteres.

Métricas y registro

Dentro de la `run_hybrid_job()` función, se registran las métricas de los algoritmos iterativos. `log_metrics` Las métricas se grafican automáticamente en la página de la consola de Braket,

en la pestaña de tareas híbridas. Puede utilizar las métricas para realizar un seguimiento de los costes de las tareas cuánticas prácticamente en tiempo real durante la ejecución de un trabajo híbrido con el rastreador de costes de Braket. En el ejemplo anterior, se utiliza el nombre de métrica «probabilidad», que registra la primera probabilidad del tipo de resultado.

Recuperando los 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 que funciona y otro que no funciona.

```
@hybrid_job(device=Devices.Amazon.SV1)
def passing():
    np_array = np.random.rand(5)
    return np_array  # serializable

@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. También puede especificar un nombre personalizado de hasta 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 contenerizado en su entorno informático 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 Braket requeridas. El siguiente código crea un trabajo híbrido local con el dispositivo como simulador SV1.

```
@hybrid_job(device=Devices.Amazon.SV1, local=True)
def run_hybrid_job(num_tasks = 1):
    return ...
```

Se admiten todas las demás opciones de trabajos híbridos. Para ver una lista de opciones, consulte el módulo braket.jobs.quantum_job_creation.

Instalar paquetes y código fuente de Python adicionales

Puede personalizar su entorno de ejecución para usar sus paquetes de Python preferidos. Puedes usar un `requirements.txt` archivo, una lista de nombres de paquetes o traer tu propio contenedor (BYOC). Para personalizar un entorno de ejecución mediante un `requirements.txt` archivo, consulte el siguiente ejemplo de código.

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies="requirements.txt")
def run_hybrid_job(num_tasks = 1):
    return ...
```

Por ejemplo, el `requirements.txt` archivo puede incluir otros paquetes para instalar.

```
qiskit
pennylane >= 0.31
mitiq == 0.29
```

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 ...
```

Guarde y cargue datos en una instancia de trabajo híbrida

Especificar los datos de entrenamiento de entrada

Al crear un trabajo híbrido, puede proporcionar una entrada para los conjuntos de datos de entrenamiento especificando un bucket de Amazon Simple Storage Service (Amazon S3). También puede especificar una ruta local y, a continuación, Braket carga automáticamente los datos a Amazon S3 en. `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>` Si especificas una ruta local, el nombre del canal será «input» por defecto. El siguiente código muestra un archivo numérico de la ruta local. `data/file.npy`

```
@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 ...
```

En el caso de S3, debe utilizar la función `get_input_data_dir()` auxiliar.

```
s3_path = "s3://amazon-braket-us-west-1-961591465522/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 URI de S3 o rutas locales.

```
input_data = {
    "input": "data/file.npy",
    "input_2": "s3://my-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**
>
> Cuando los datos de entrada son grandes (más de 1 GB), hay 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 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 numérica y una figura de matplotlib.

```
@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()` Python o navegando 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 de `save_job_checkpoint()` ayuda integrada o guardar los archivos en la `AMZN_BRAKET_JOB_RESULTS_DIR` ruta. 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 trabajos 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 invoca 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 numéricas, 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 tus artefactos de trabajo híbridos, en una subcarpeta llamada «puntos de control».

Para crear un nuevo trabajo híbrido para continuar desde el punto de control, debemos pasar por `copy_checkpoints_from_job=job_arn` dónde `job_arn` está el ARN del trabajo híbrido del trabajo anterior. Luego solemos `load_job_checkpoint(job_name)` cargar desde el punto de control.

Mejores prácticas para decoradores de trabajos híbridos

Adopte la asincronicidad

Los trabajos híbridos creados con la anotación decorator son asíncronos: se ejecutan una vez que están disponibles los recursos clásicos y cuánticos. Usted monitorea el progreso del algoritmo utilizando Amazon Braket Management Console o Amazon CloudWatch. Cuando envías tu algoritmo para su ejecución, Braket lo ejecuta en un entorno contenerizado escalable y los resultados se recuperan cuando el algoritmo está completo.

Ejecuta algoritmos variacionales iterativos

Hybrid Jobs le proporciona las herramientas para ejecutar algoritmos iterativos clásicos cuánticos. Para problemas puramente cuánticos, utilice tareas cuánticas o un lote de tareas cuánticas. El acceso prioritario a determinadas QPU es más beneficioso para los algoritmos variacionales de larga duración que requieren múltiples llamadas iterativas a las QPU con el procesamiento clásico intermedio.

Depure utilizando el modo local

Antes de ejecutar un trabajo híbrido en una QPU, se recomienda ejecutar primero el simulador SV1 para confirmar que se ejecuta según lo esperado. Para pruebas a pequeña escala, puede ejecutarlas en modo local para una iteración y depuración rápidas.

Mejore la reproducibilidad con Bring your own container (BYOC)

Cree un experimento reproducible encapsulando su software y sus dependencias en un entorno contenerizado. Al empaquetar todo el código, las dependencias y la configuración en un contenedor, se evitan posibles conflictos y problemas de control de versiones.

Simuladores distribuidos de varias instancias

Para ejecutar una gran cantidad de circuitos, considere la posibilidad de utilizar el soporte MPI integrado para ejecutar simuladores locales en varias instancias dentro de un solo trabajo híbrido. Para obtener más información, consulte simuladores integrados.

Utilice circuitos paramétricos

Los circuitos paramétricos que envíe a partir de un trabajo híbrido se compilan automáticamente en determinadas QPU mediante la compilación paramétrica para mejorar los tiempos de ejecución de los algoritmos.

Compruebe periódicamente

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.

Realice un trabajo híbrido con Amazon Braket Hybrid Jobs

Para ejecutar un trabajo híbrido con Amazon Braket Hybrid Jobs, 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, consulta Bring your own container (BYOC).

En cualquier caso, a continuación, crea un trabajo híbrido con Amazon BraketAPI, donde proporciona el script o contenedor del algoritmo, selecciona el dispositivo cuántico de destino que va a utilizar el trabajo híbrido y, a continuación, elige entre una variedad de configuraciones opcionales. Los valores predeterminados que se proporcionan para estas configuraciones opcionales funcionan en la mayoría de los casos de uso. Para que el dispositivo de destino ejecute su Hybrid Job, puede elegir entre una QPU, un simulador bajo demanda (comoSV1, DM1 oTN1) o la propia instancia de trabajo híbrida clásica. Con un simulador bajo demanda o una QPU, su contenedor de trabajos híbrido realiza llamadas a la API a un dispositivo remoto. Con los simuladores integrados, el simulador está integrado en el mismo contenedor que el script del algoritmo. Los simuladores Lightning PennyLane vienen integrados en el contenedor de tareas híbridas 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. Consulta la página de precios Braket de Amazon para conocer los costes asociados a cada opción.

Si el dispositivo de destino es un simulador integrado o bajo demanda, Amazon Braket comienza a ejecutar el trabajo híbrido de inmediato. Activa la instancia de trabajo híbrida (puede personalizar el tipo de instancia en la API llamada), ejecuta su algoritmo, escribe los resultados en Amazon S3 y libera sus recursos. Esta versión de recursos garantiza que solo pague por lo que utilice.

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 dado. 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 híbrida necesaria y ejecuta tu trabajo híbrido en el dispositivo. Mientras dure 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 que están en cola en el dispositivo, siempre que las tareas cuánticas del trabajo se envíen a la QPU una vez cada pocos minutos. Una vez que haya completado su trabajo híbrido, se liberarán los recursos, lo que significa que solo pagará por lo que utilice.

> **Note**
>
> Los dispositivos son regionales y tu trabajo híbrido se ejecuta de la Región de AWS misma manera que tu dispositivo principal.

Tanto en el escenario objetivo del simulador como en el de la QPU, tiene la opción de definir métricas algorítmicas 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 deseas usar una instancia basada en GPU, asegúrate de usar uno de los simuladores basados en GPU disponibles con los simuladores integrados en Braket (por ejemplo,). `lightning.gpu` Si eliges uno de los simuladores integrados basados en la CPU (por ejemplo, o`braket:default-simulator`)`lightning.qubit`, la GPU no se utilizará y es posible que incurras en costes innecesarios.

Cree su primer trabajo híbrido

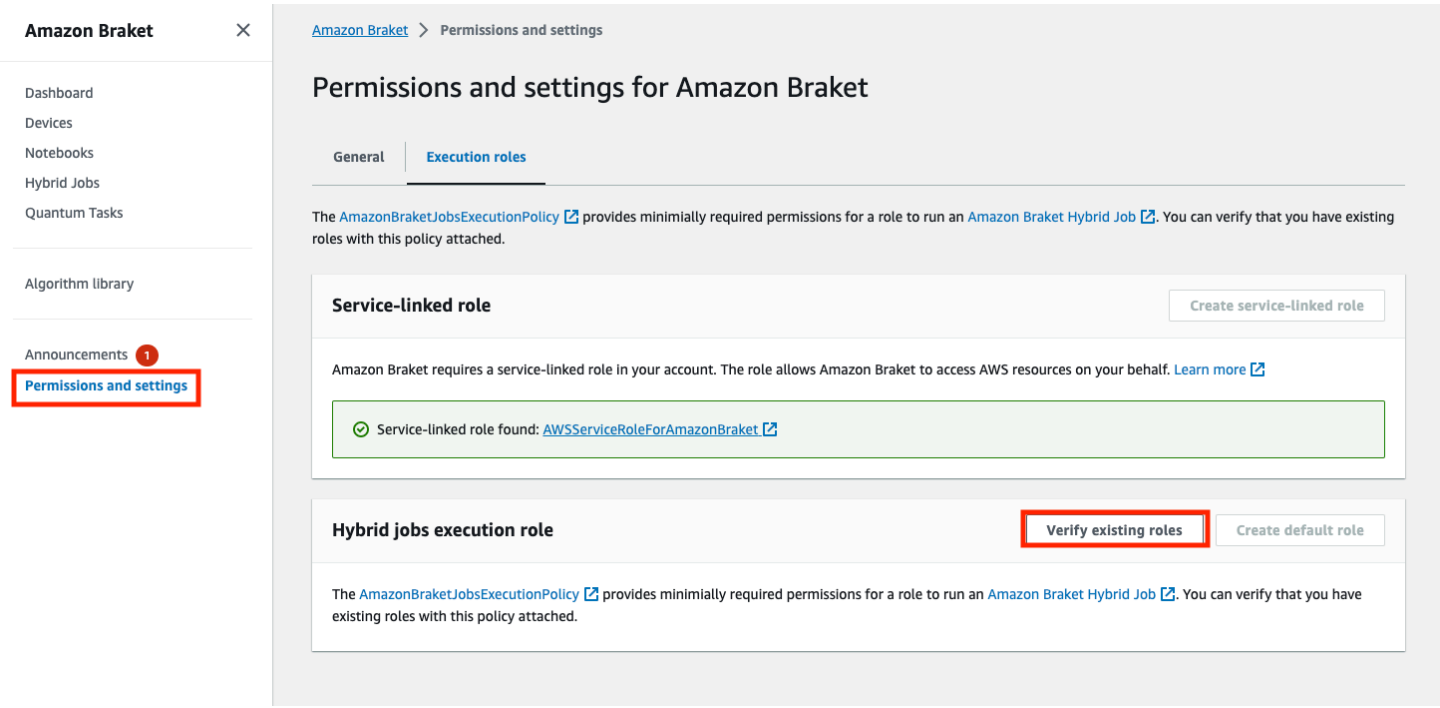
En esta sección se muestra cómo crear un Hybrid Job mediante un script de Python. Como alternativa, para crear un trabajo híbrido a partir del código Python local, como su entorno de desarrollo integrado (IDE) preferido o un bloc de notas Braket, consulteEjecute su código local como un trabajo híbrido.

En esta sección:

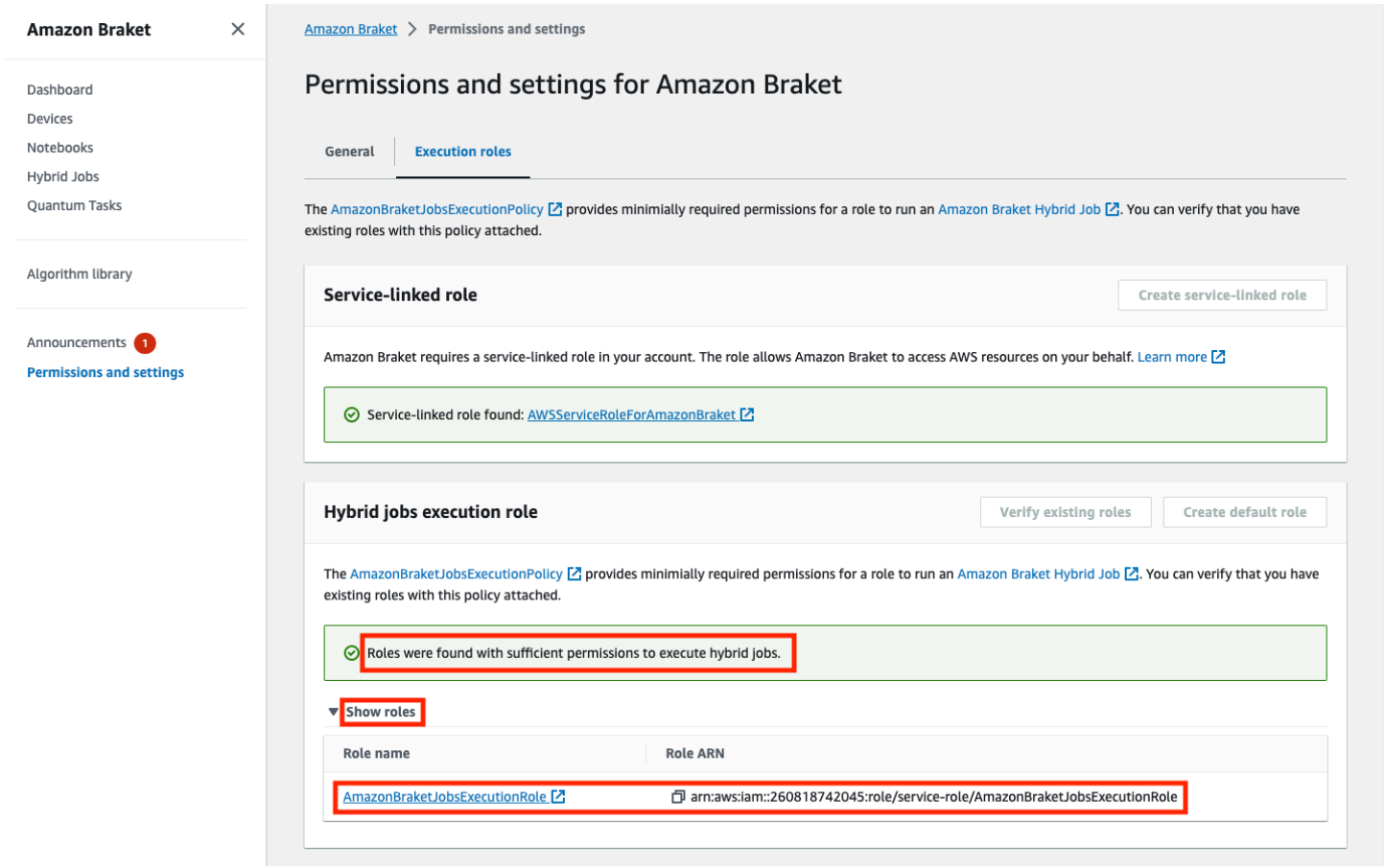
- Configure los permisos
- Crear y ejecutar
- Supervisar resultados

Configure los permisos

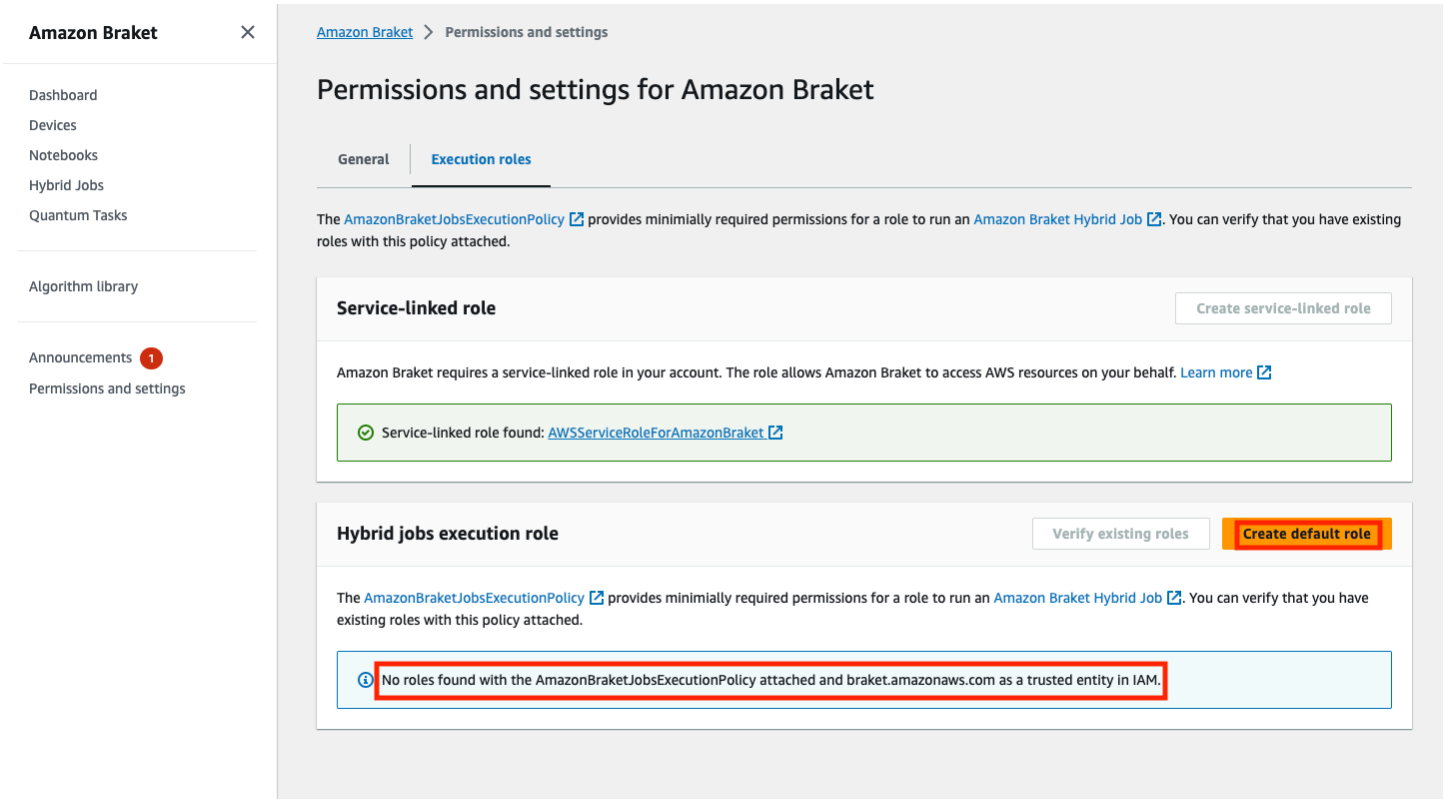
Antes de ejecutar su primer trabajo híbrido, debe asegurarse de tener los permisos suficientes para continuar con esta tarea. Para determinar si tiene los permisos correctos, seleccione Permisos en el menú de la parte izquierda de la consola Braket. La página de administración de permisos para Amazon Braket le ayuda a comprobar si uno de sus roles actuales tiene permisos suficientes para ejecutar su trabajo híbrido o lo guía a través de la creación de un rol predeterminado que pueda usarse para ejecutar su trabajo híbrido si aún no lo tiene.



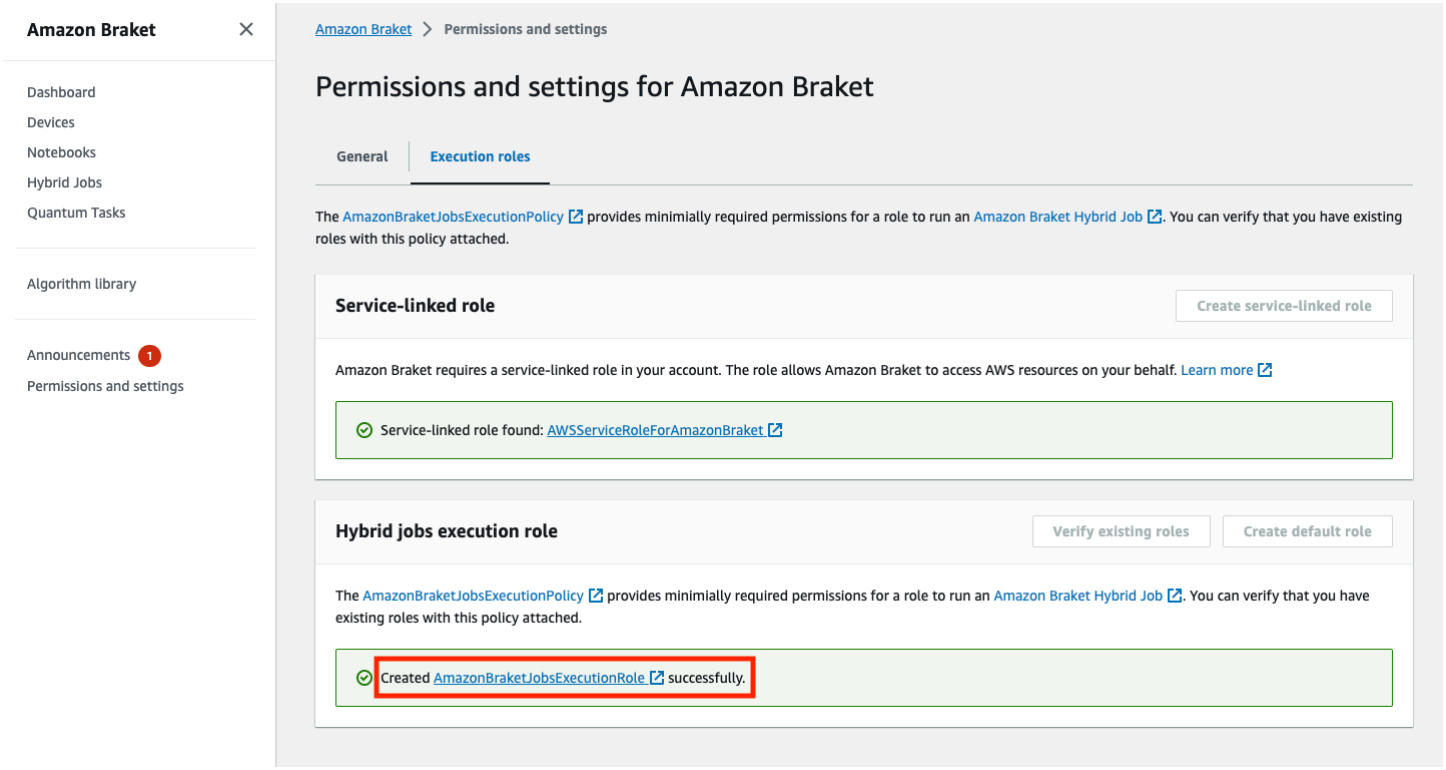
Para comprobar que tiene roles con permisos suficientes para ejecutar un trabajo híbrido, seleccione el botón Verificar el rol existente. Si lo hace, recibirá un mensaje en el que se indica que se han encontrado los roles. Para ver los nombres de las funciones y los ARN de sus funciones, seleccione el botón Mostrar funciones.



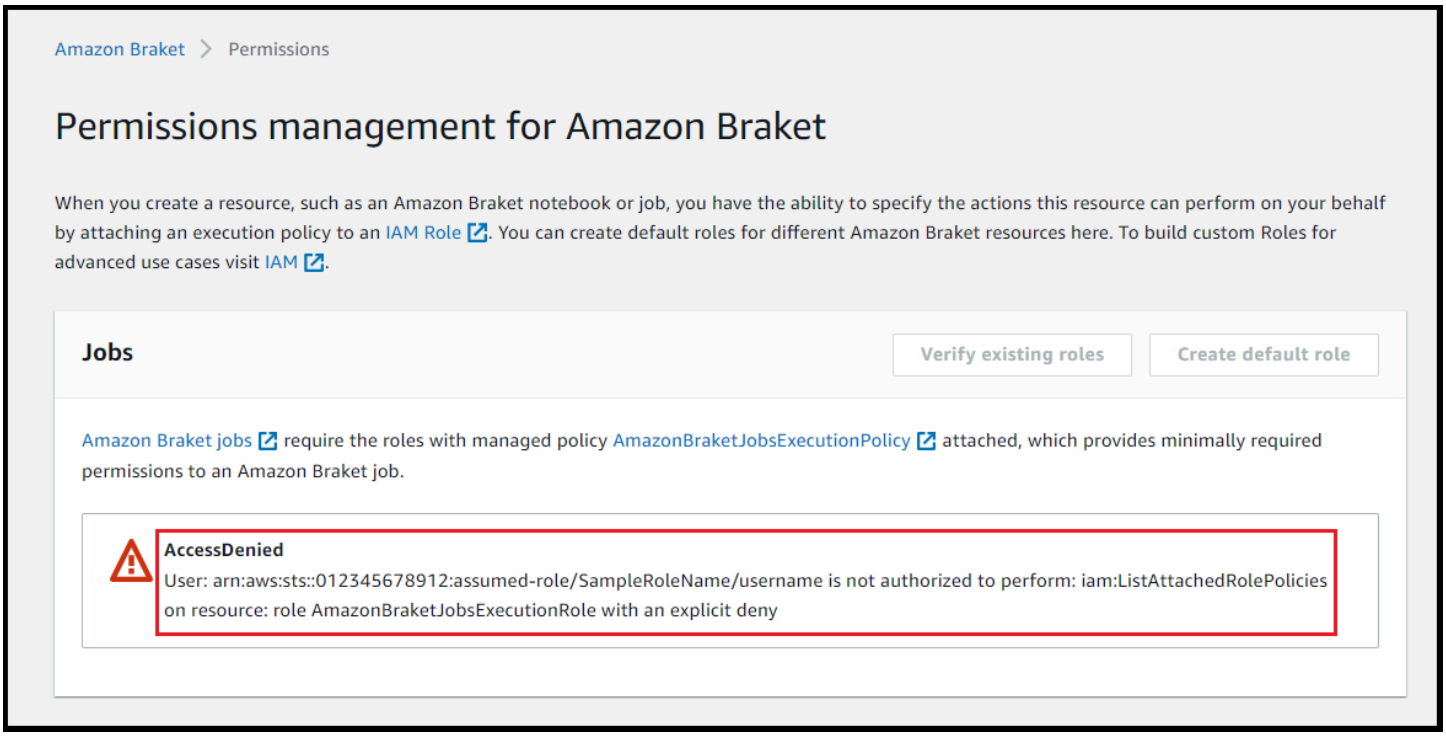
Si no tiene un rol con los permisos suficientes para ejecutar un trabajo híbrido, recibirá un mensaje en el que se indica que no se ha encontrado dicho rol. Seleccione el botón Crear un rol predeterminado para obtener un rol con permisos suficientes.



Si el rol se creó correctamente, recibirá un mensaje que lo confirma.

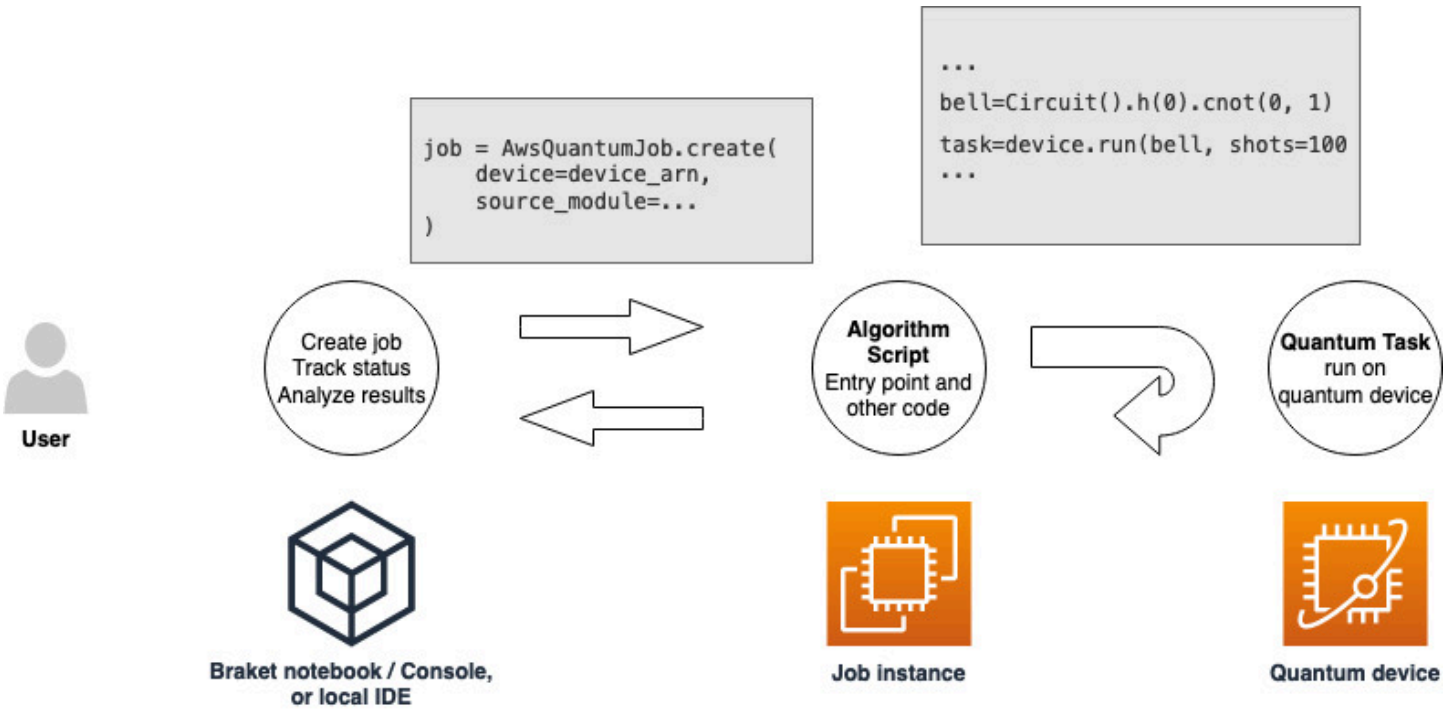


Si no tiene permisos para realizar esta consulta, se le denegará el acceso. En este caso, póngase en contacto con su AWS administrador interno.



Crear y ejecutar

Una vez que tenga un rol con permisos para ejecutar un trabajo híbrido, estará listo para continuar. La pieza clave de su primer trabajo híbrido de Braket es el script del algoritmo. Define el algoritmo que desea ejecutar y contiene las tareas lógicas y cuánticas clásicas que forman parte de su algoritmo. Además del script del algoritmo, puede proporcionar otros archivos de dependencia. El script del 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 de campana 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 bloc de notas 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 el script de trabajo híbrido y el dispositivo principal.

> **Note**
>
> Para obtener más información sobre cómo crear un bloc de notas Braket o cargar un archivo, como el archivo algorithm_script.py, en el mismo directorio que los cuadernos, consulte Ejecute su primer circuito con el SDK de Python de Amazon Braket

Para este primer caso básico, opta por un simulador. Independientemente del tipo de dispositivo cuántico al que apunte, ya sea un simulador o una unidad de procesamiento cuántico (QPU) real, el dispositivo que especifique `device` en el siguiente script se utiliza para programar el trabajo híbrido y está disponible para los scripts del algoritmo como variable `AMZN_BRAKET_DEVICE_ARN` de entorno.

> **Note**
>
> Solo puede usar los dispositivos que estén disponibles en Región de AWS su trabajo híbrido. El Amazon Braket SDK lo selecciona automáticamente. Región de AWS Por ejemplo, un trabajo híbrido en us-east-1 puede IonQ usar dispositivosSV1,, TN1 yDM1, pero no dispositivos. Rigetti

Si eliges un ordenador cuántico en lugar de un simulador, Braket programa tus trabajos híbridos para que ejecuten 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
)
```

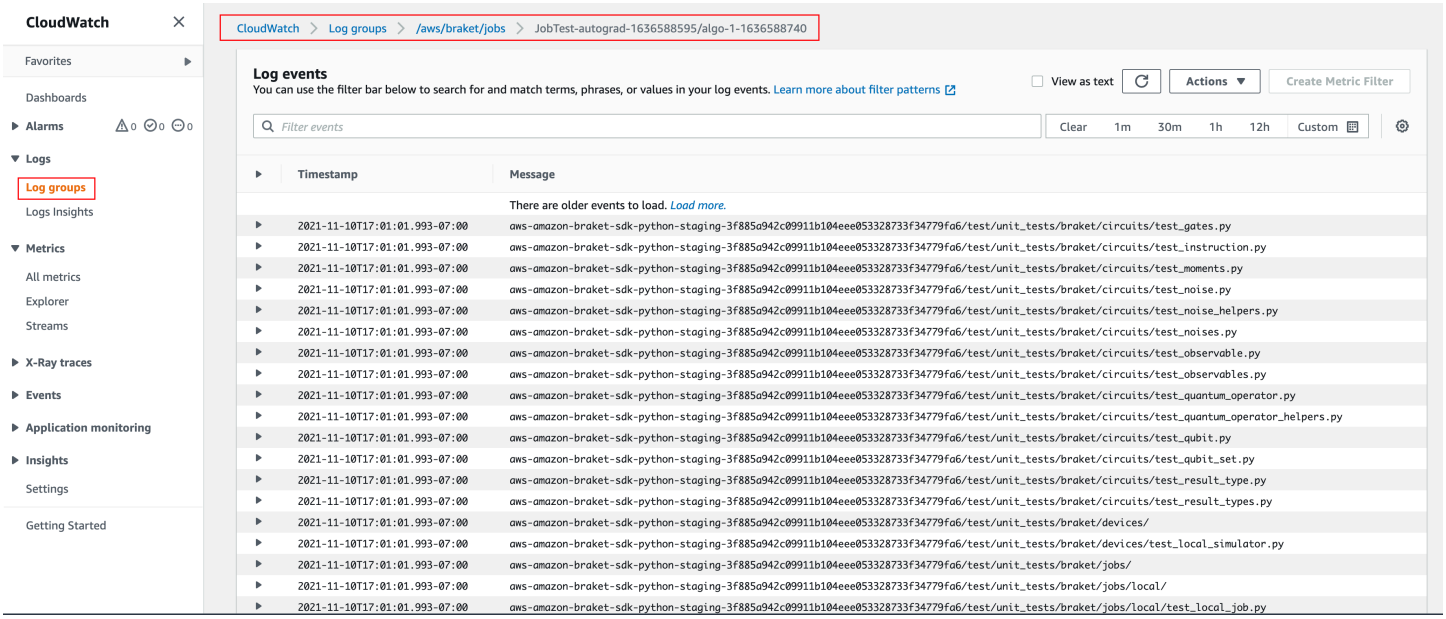
El parámetro `wait_until_complete=True` establece un modo detallado para que el trabajo imprima el resultado del trabajo real a medida que se ejecuta. Debería ver un resultado similar al siguiente ejemplo.

```
job = AwsQuantumJob.create(
    Devices.Amazon.SV1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=True,
)
Initializing Braket Job: arn:aws:braket:us-west-2:<accountid>:job/<UUID>
.....................................
.
.
.

Completed 36.1 KiB/36.1 KiB (692.1 KiB/s) with 1 file(s) remaining#015download:
 s3://braket-external-assets-preview-us-west-2/HybridJobsAccess/models/
braket-2019-09-01.normal.json to ../../braket/additional_lib/original/
braket-2019-09-01.normal.json
Running Code As Process
Test job started!!!!!
Counter({'00': 55, '11': 45})
Counter({'11': 59, '00': 41})
Counter({'00': 55, '11': 45})
Counter({'00': 58, '11': 42})
Counter({'00': 55, '11': 45})
Test job completed!!!!!
Code Run Finished
2021-09-17 21:48:05,544 sagemaker-training-toolkit INFO     Reporting training SUCCESS
```

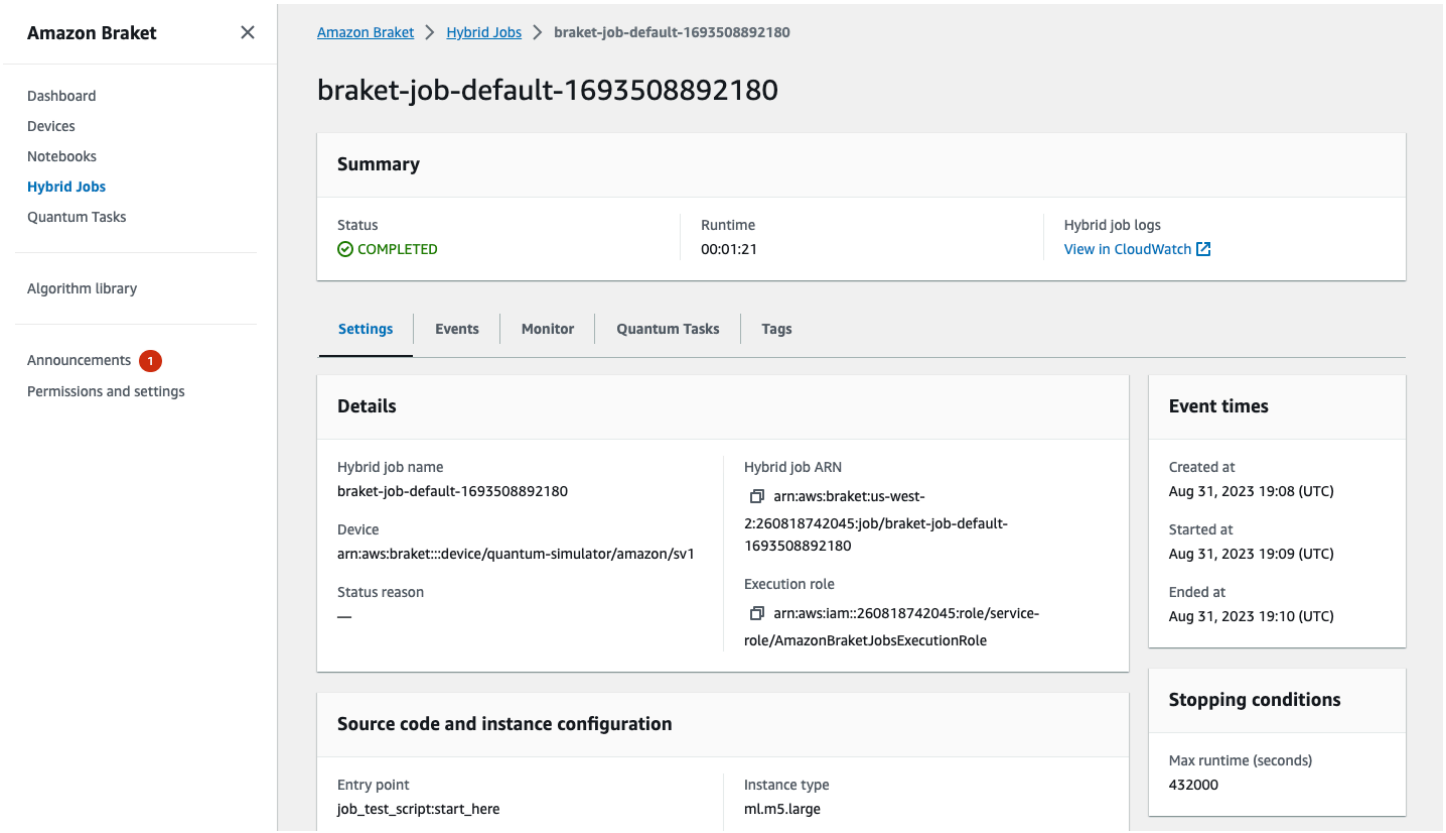
> **Note**
>
> También puedes usar tu módulo personalizado con el método AwsQuantumJob.create pasando su ubicación (la ruta a un directorio o archivo local o el URI de S3 de un archivo tar.gz). Para ver un ejemplo práctico, consulta el archivo Parallelize_Training_for_QML.ipynb en la carpeta de trabajos híbridos del repositorio Github de ejemplos de Amazon Braket.

Supervisar resultados

Como alternativa, puedes acceder a la salida del registro desde Amazon CloudWatch. Para ello, vaya a la pestaña Grupos de registros en el menú izquierdo de la página de detalles del trabajo, seleccione el grupo de registros `yaws/braket/jobs`, 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, seleccionando 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 diferente `code_location` 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 Región de AWS el script de trabajo.

El `jobs/<jobname>/<timestamp>` directorio contiene una subcarpeta con el resultado del script del punto de entrada en un `model.tar.gz` archivo. También hay un directorio llamado `script` que contiene los artefactos del script del algoritmo en un `source.tar.gz` archivo. Los resultados de sus tareas cuánticas reales se encuentran en el directorio nombrado`jobs/<jobname>/tasks`.

Entradas, salidas, variables de entorno y funciones auxiliares

Además del archivo o los archivos que componen el script de algoritmo completo, su 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 del 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.

AmazonBraket también proporciona varias variables de entorno y funciones auxiliares para simplificar las interacciones con las entradas y salidas de los contenedores.

En esta sección se explican los conceptos clave de la `AwsQuantumJob.create` función proporcionada por el SDK de Python de Amazon Braket y su asignación a la estructura del archivo contenedor.

En esta sección:

- Entradas
- Salidas
- 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 `input_data` argumento. El usuario define el `input_data` argumento dentro de la `AwsQuantumJob.create` función en el SDK. Esto copia los datos de entrada al sistema de archivos 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 (más de 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 cargan primero en un depósito de S3, después se añade la ruta de S3 a la solicitud de trabajo híbrido y, por último, la solicitud de trabajo híbrido se envía al servicio Braket.

Hiperparámetros: si los transfierehyperparameters, 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, después, pasar esta información al script de trabajo híbrido, consulta la sección Usar hiperparámetros y esta página de GitHub.

Puntos de control: para especificar qué job-arn punto de control desea utilizar en un nuevo trabajo híbrido, utilice el comando. copy_checkpoints_from_job Este comando copia los datos de los puntos de control a los checkpoint_configs3Uri del nuevo trabajo híbrido y los hace disponibles en la ruta indicada por la variable de entorno AMZN_BRAKET_CHECKPOINT_DIR mientras se ejecuta el trabajo. El valor predeterminado esNone, lo que significa que los datos de los puntos de control de otro trabajo híbrido no se utilizarán en el nuevo trabajo híbrido.

Salidas

Tareas cuánticas: los resultados de las tareas cuánticas se almacenan en la ubicación s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks S3.

Resultados del trabajo: todo lo que el script de su algoritmo guarda en el directorio indicado por la variable de entorno "AMZN_BRAKET_JOB_RESULTS_DIR" se copia en la ubicación S3 especificada enoutput_data_config. Si no especifica este 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 invoca desde el script de su 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"` En su lugar, también puedes usar la función de ayuda del SDK. `save_job_checkpoint`

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 los algoritmos, consulte Utilizar Amazon Braket Hybrid Jobs para ejecutar un algoritmo de QAOA.

Variables de entorno

AmazonBraket proporciona varias variables de entorno para simplificar las interacciones con las entradas y salidas de los contenedores. En el siguiente código se enumeran las variables de entorno que utiliza Braket.

```
# the input data directory opt/braket/input/data
os.environ["AMZN_BRAKET_INPUT_DIR"]
# the output directory opt/braket/model to write job results to
os.environ["AMZN_BRAKET_JOB_RESULTS_DIR"]
# the name of the job
os.environ["AMZN_BRAKET_JOB_NAME"]
# the checkpoint directory
os.environ["AMZN_BRAKET_CHECKPOINT_DIR"]
# the file containing the hyperparameters
os.environ["AMZN_BRAKET_HP_FILE"]
# the device ARN (AWS Resource Name)
os.environ["AMZN_BRAKET_DEVICE_ARN"]
# the output S3 bucket, as specified in the CreateJob request’s OutputDataConfig
os.environ["AMZN_BRAKET_OUT_S3_BUCKET"]
# the entry point as specified in the CreateJob request’s ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_ENTRY_POINT"]
# the compression type as specified in the CreateJob request’s ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE"]
# the S3 location of the user’s script as specified in the CreateJob request’s
 ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_S3_URI"]
# the S3 location where the SDK would store the quantum task results by default for the
 job
os.environ["AMZN_BRAKET_TASK_RESULTS_S3_URI"]
# the S3 location where the job results would be stored, as specified in CreateJob
 request’s OutputDataConfig
```

```
os.environ["AMZN_BRAKET_JOB_RESULTS_S3_PATH"]
# the string that should be passed to CreateQuantumTask's jobToken parameter for
 quantum tasks created in the job container
os.environ["AMZN_BRAKET_JOB_TOKEN"]
```

Funciones auxiliares

AmazonBraket proporciona varias funciones auxiliares para simplificar las interacciones con las entradas y salidas de los contenedores. Estas funciones auxiliares se llamarían desde el script del algoritmo que se utiliza para ejecutar su Hybrid Job. En el siguiente ejemplo, se muestra cómo utilizarlas.

```
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() # save hybrid job results
save_job_checkpoint() # save a checkpoint
load_job_checkpoint() # load a previously saved checkpoint
```

Guarde los resultados del trabajo

Puede guardar los resultados generados por el script del 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 numérica, puede incluir una opción para serializar utilizando un formato de datos decapado. 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 script del algoritmo.

```
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 guion 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="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, la hemos eliminado `wait_until_complete=True` para suprimir la salida detallada. Puedes volver a añadirlo para su depuración. Cuando ejecutas este trabajo híbrido, muestra el identificador y el `job-arn` estado del trabajo híbrido cada 10 segundos hasta que el

trabajo híbrido esté listo`COMPLETED`, tras lo cual te muestra los resultados del circuito de campana. Consulte el siguiente ejemplo.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-1234567890123
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Guarde y reinicie los trabajos híbridos mediante puntos de control

Puede guardar las iteraciones intermedias de sus trabajos híbridos mediante puntos de control. En el ejemplo del script de algoritmos de la sección anterior, añadiría las siguientes líneas comentadas con #ADD para crear archivos de puntos 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!!!!!")
```

Al ejecutar el trabajo híbrido, se crea el archivo -checkpoint-1.json <jobname>en los artefactos del trabajo híbrido del directorio de puntos de control con una ruta predeterminada. `/opt/jobs/checkpoints` 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, utilice `from braket.jobs import load_job_checkpoint` el script del algoritmo. La lógica que se debe cargar en el script de su algoritmo es la siguiente.

```
checkpoint_1 = load_job_checkpoint(
    "previous_job_name",
    checkpoint_file_suffix="checkpoint-1",
)
```

Tras cargar este punto de control, puede continuar con la lógica en función del contenido cargado en él. `checkpoint-1`

> **Note**
>
> El checkpoint_file_suffix debe coincidir con el sufijo previamente especificado al crear el punto de control.

El guion de orquestación debe especificar lo del trabajo híbrido anterior con la línea `job-arn` comentada con #ADD.

```
job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    copy_checkpoints_from_job="<previous-job-ARN>", #ADD
```

```
)
```

Defina el entorno para el script de su algoritmo

AmazonBraket admite tres entornos definidos por contenedores para el script de su algoritmo:

- Un contenedor base (el predeterminado, si no `image_uri` se especifica ningún valor)
- Un contenedor con Tensorflow y PennyLane
- Un contenedor con y PyTorch PennyLane

En la siguiente tabla se proporcionan detalles sobre los contenedores y las bibliotecas que incluyen.

Contenedores Amazon Braket

Tipo	PennyLane con TensorFlow	PennyLane con PyTorch	Pennylane
Base	292282985366.dkr.ecr.us-east-1.amazonaws.com /amazon-braket-tensorflow-jobs: latest	292282985366.dkr.ecr.us-west-2.amazonaws.com /amazon-braket-pytorch-jobs: más reciente	292282985366.dkr. ecr.us-west-2.amazonaws.com / amazon-braket-base-jobs:latest
Bibliotecas heredadas	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
Bibliotecas adicionales	• amazon-braket-default-simulator • plugin amazon-braket-pennylane • amazon-braket-esquemas • amazon-braket-sdk • ipykernel	• simulador Amazon Braket-Default • plugin amazon-braket-pennylane • amazon-braket-esquemas • amazon-braket-sdk • ipykernel	• simulador Amazon Braket-Default • plugin amazon-braket-pennylane • amazon-braket-esquemas • amazon-braket-sdk • awscli • boto3

Tipo	PennyLane con TensorFlow	PennyLane con PyTorch	Pennylane
	• keras • matplotlib • redes • openbabel • PennyLane • protobug • psi4 • rsa • PennyLane-Lightning-GPU • Cu Quantum	• keras • matplotlib • redes • openbabel • PennyLane • protobug • psi4 • rsa • PennyLane-Lightning-GPU • Cu Quantum	• ipykernel • matplotlib • redes • numpy • openbabel • pandas • PennyLane • protobug • psi4 • rsa • scipy

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. El contenedor debe estar en el lugar Región de AWS desde el que invoque su trabajo híbrido. Para especificar la imagen del contenedor al crear un trabajo híbrido, añada uno de los tres argumentos siguientes a la `create(…)` llamada en el script del trabajo híbrido. Puede instalar dependencias adicionales en el contenedor que elija durante el tiempo de ejecución (a costa del inicio o del tiempo de ejecución), ya que los contenedores Amazon Braket tienen conectividad a Internet. El siguiente ejemplo corresponde a la región us-west-2.

- Imagen base image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /amazon-braket-base-jobs:1.0-cpu-py39-ubuntu22.04»
- Imagen de Tensorflow image_uri="292282985366.dkr. ecr.us-east-1.amazonaws.com /amazon-braket-tensorflow-jobs:2.11.0-gpu-py39-cu112-ubuntu20.04»
- PyTorch imagen image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /amazon-braket-pytorch-jobs:1.13.1-gpu-py39-cu117-ubuntu20.04»

`image-uris`También se pueden recuperar mediante la función del SDK de Braket. `retrieve_image()` Amazon 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_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Usar hiperparámetros

Puede definir los hiperparámetros que necesita su algoritmo, como la tasa de aprendizaje o el tamaño de los pasos, al crear un trabajo híbrido. Los valores de los hiperparámetros se utilizan normalmente para controlar varios aspectos del algoritmo y, a menudo, se pueden ajustar 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. Tenga en cuenta que los valores deben ser del tipo de datos de cadena. Al buscar el conjunto de valores óptimo, debe especificar los valores de hiperparámetros que desea probar. El primer paso para utilizar los hiperparámetros es configurar y definir los hiperparámetros como un diccionario, como se puede ver en el siguiente código:

```
#defining the number of qubits used
n_qubits = 8
#defining the number of layers used
n_layers = 10
#defining the number of iterations used for your optimization algorithm
n_iterations = 10

hyperparams = {
    "n_qubits": n_qubits,
    "n_layers": n_layers,
    "n_iterations": n_iterations
}
```

A continuación, debe pasar los hiperparámetros definidos en el fragmento de código indicado anteriormente para utilizarlos en el algoritmo de su elección con algo parecido a lo siguiente:

```
import time
from braket.aws import AwsQuantumJob

#Name your job so that it can be later identified
job_name = f"qcbm-gaussian-training-{n_qubits}-{n_layers}-" + str(int(time.time()))
```

```
job = AwsQuantumJob.create(
    #Run this hybrid job on the SV1 simulator
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    #The directory or single file containing the code to run.
    source_module="qcbm",
    #The main script or function the job will run.
    entry_point="qcbm.qcbm_job:main",
    #Set the job_name
    job_name=job_name,
    #Set the hyperparameters
    hyperparameters=hyperparams,
    #Define the file that contains the input data
    input_data="data.npy", # or input_data=s3_path
    # wait_until_complete=False,
)
```

> **Note**
>
> Para obtener más información sobre los datos de entrada, consulte la sección Entradas.

A continuación, los hiperparámetros se cargarían en el script de trabajo híbrido mediante el siguiente código:

```
import json
import os

#Load the Hybrid Job hyperparameters
hp_file = os.environ["AMZN_BRAKET_HP_FILE"]
with open(hp_file, "r") as f:
    hyperparams = json.load(f)
```

> **Note**
>
> Para obtener más información sobre cómo pasar información como los datos de entrada y el arn del dispositivo al script de trabajo híbrido, consulta esta página de GitHub.

Los tutoriales QAOA con Amazon Braket Hybrid Jobs y Quantum machine learning en Amazon Braket Hybrid Jobs ofrecen un par de guías muy útiles para aprender a utilizar los hiperparámetros. PennyLane

Configure la instancia de trabajo híbrida para ejecutar el script de su algoritmo

En función del algoritmo, es posible que tenga requisitos diferentes. De forma predeterminada, Amazon Braket ejecuta el script del algoritmo en una `ml.m5.large` instancia. Sin embargo, puedes 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.p3.8xlarge"), # Use NVIDIA Tesla
 V100 instance with 4 GPUs.
    ...
    ),
```

Si está ejecutando una simulación integrada y ha especificado un dispositivo local en la configuración del dispositivo, también podrá solicitar más de una instancia en el 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.p3.8xlarge", instanceCount=3), #
 Use 3 NVIDIA Tesla V100
    ...
    ),
```

Cuando utilice varias instancias, considere la posibilidad de distribuir su trabajo híbrido mediante la función data parallel. Consulte el siguiente cuaderno de ejemplo para obtener más información sobre cómo ver este ejemplo de Braket.

En las tres tablas siguientes se enumeran los tipos de instancias y las especificaciones disponibles para las instancias de computación estándar, optimizada y acelerada.

> **Note**
>
> Para ver las cuotas predeterminadas de instancias informáticas clásicas para Hybrid Jobs, consulta esta página.

Instancia estándar	vCPU	Memoria
ml.m5.large (predeterminado)	2	8 GiB
ml.m5.xlarge	4	16 GiB
ml.m5.2xlarge	8	32 GiB
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB
ml.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

Instancias optimizadas para computación	vCPU	Memoria
ml.c4.xlarge	4	7,5 GiB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB

Instancias optimizadas para computación	vCPU	Memoria
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.xlarge	4	10,5 GiB
ml.c5n.2xlarge	8	21 GiB
ml.c5n.4xlarge	16	42 GiB
ml.c5n.9xlarge	36	96 GiB
ml.c5n.18xlarge	72	192 GiB

Instancias de computación acelerada	vCPU	Memoria
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB
ml.g4dn.xlarge	4	16 GiB

Instancias de computación acelerada	vCPU	Memoria
ml.g4dn.2xlarge	8	32 GiB
ml.g4dn.4xlarge	16	64 GiB
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

> **Note**
>
> Las instancias p3 no están disponibles en us-west-1. Si su trabajo híbrido no puede aprovisionar la capacidad informática de aprendizaje automático solicitada, utilice otra región.

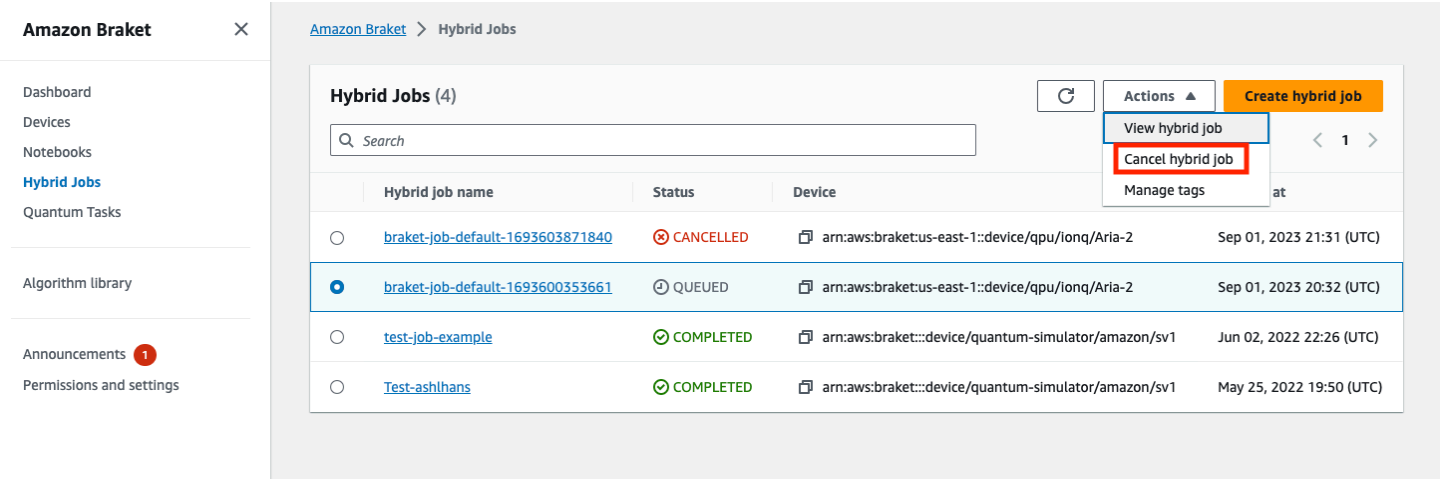
Cada instancia usa una configuración predeterminada de almacenamiento de datos (SSD) de 30 GB. Sin embargo, puede ajustar el almacenamiento de la misma manera que configura el`instanceType`. El siguiente ejemplo muestra cómo aumentar el almacenamiento total a 50 GB.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(
        instanceType="ml.p3.8xlarge",
        volumeSizeInGb=50,
    ),
    ...
    ),
```

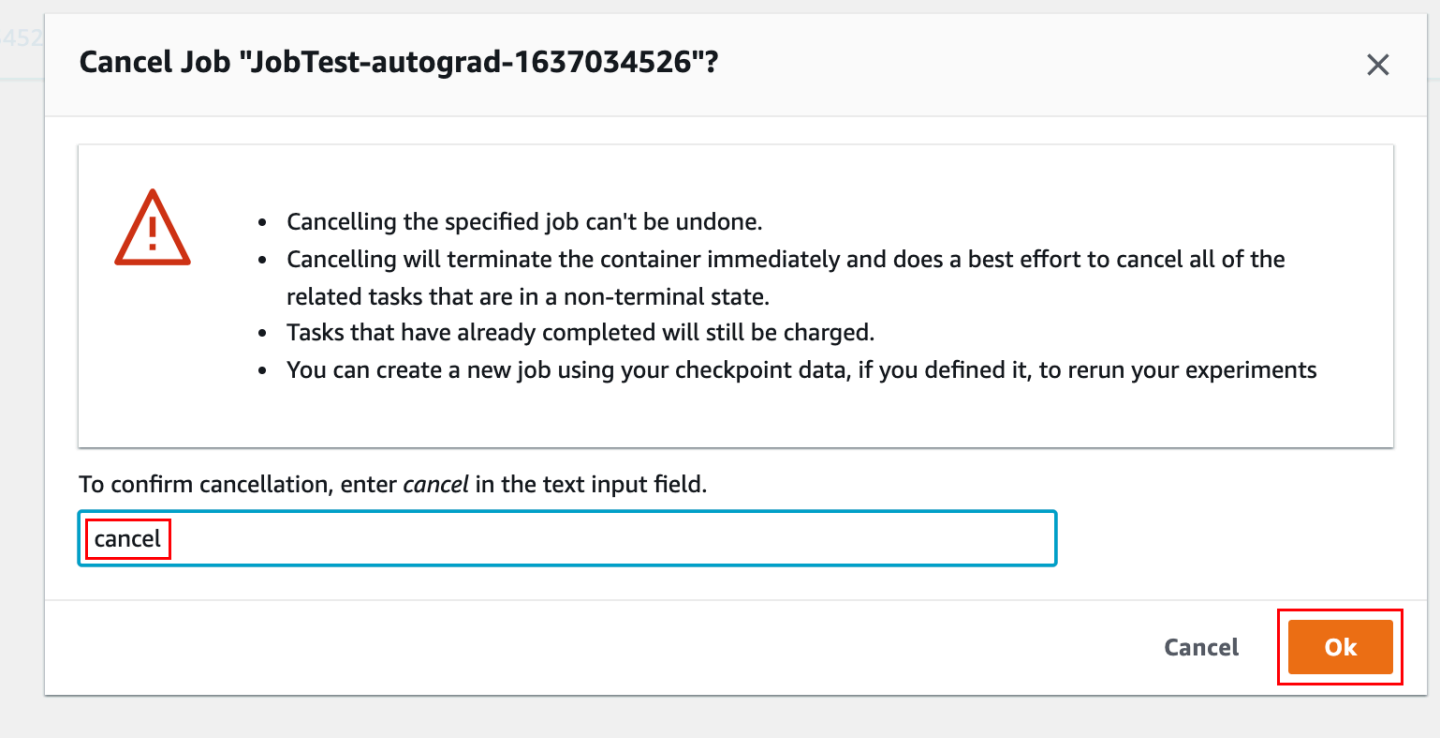
Cancelar un trabajo híbrido

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 su 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 el código del SDK de Python de Braket, utilice `job_arn` para identificar el trabajo híbrido y, a continuación, `cancel` ejecute el comando que contiene, como se muestra en el código siguiente.

```
job = AwsQuantumJob(arn=job_arn)
job.cancel()
```

El `cancel` comando cierra inmediatamente el contenedor de tareas híbrido clásico y hace todo lo posible por cancelar todas las tareas cuánticas relacionadas que aún se encuentran en un estado no terminal.

Uso de la compilación paramétrica para acelerar las tareas híbridas

Amazon Braket admite la compilación paramétrica en determinadas QPU. Esto le permite reducir la sobrecarga asociada al costoso paso de compilación desde el punto de vista computacional al compilar un circuito solo una vez y no para cada iteración de su algoritmo híbrido. Esto puede mejorar considerablemente los tiempos de ejecución de Hybrid Jobs, ya que evita la necesidad de volver a compilar el circuito en cada paso. Simplemente envíe los circuitos parametrizados a una de nuestras QPU compatibles como Braket Hybrid Job. Para trabajos híbridos de larga duración, Braket utiliza automáticamente los datos de calibración actualizados del proveedor del hardware al compilar el circuito para garantizar resultados de la más alta calidad.

Para crear un circuito paramétrico, primero debe proporcionar los parámetros como entradas en el script de su 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 del algoritmo para que se ejecute como un Hybrid Job con el siguiente script de trabajo. Al ejecutar el Hybrid Job 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 tiempo de ejecución del Hybrid Job 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 todas las QPU superconductoras basadas en puertas Rigetti Computing y Oxford Quantum Circuits con la excepción de los programas de nivel de pulso.

PennyLane Utilízalo 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 EC2 o un portátil) y las instrucciones cuánticas se ejecutan en un simulador o en un ordenador cuántico. Se recomienda ejecutar algoritmos híbridos mediante la función Hybrid Jobs. Para obtener más información, consulta Cuándo usar Amazon Braket Jobs.

AmazonBraket le permite configurar y ejecutar algoritmos cuánticos híbridos con la ayuda del PennyLane complemento Braket o con el SDK de Python de Amazon Amazon Braket y repositorios de cuadernos de ejemplo. Amazon Los cuadernos de ejemplo de Braket, basados en el SDK, permiten configurar y ejecutar determinados algoritmos híbridos sin el PennyLane complemento. 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 cálculo aumenta la probabilidad de añadir ruido; por lo tanto, los algoritmos de larga duración pueden verse abrumados por el ruido, lo que resulta en un cálculo defectuoso.

Los algoritmos cuánticos puros, como el de Shor (por ejemplo, la estimación de fase cuántica) o el de Grover (ejemplo de Grover), requieren miles o millones de operaciones. Por esta razón, 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 (QPU) funcionan como coprocesadores para las CPU clásicas, específicamente para acelerar ciertos cálculos en un algoritmo clásico. Las ejecuciones de los circuitos se acortan mucho, al alcance de las capacidades de los dispositivos actuales.

Amazon Braket con PennyLane

AmazonBraket proporciona soporte para PennyLaneun marco de software de código abierto creado en torno al concepto de programación cuántica diferenciable. Puedes usar este marco para entrenar circuitos cuánticos de la misma manera que entrenarías una red neuronal para encontrar soluciones a problemas computacionales en química cuántica, aprendizaje automático cuántico y optimización.

La PennyLane biblioteca proporciona interfaces con herramientas conocidas de aprendizaje automático, entre las que se incluyen PyTorch y TensorFlow, para 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 tutoriales le ayudan a empezar rápidamente. Como alternativa, puedes usarlo PennyLane en Amazon Braket desde el IDE que prefieras.

- 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 los QPUs y los simuladores
- `braket.local.qubit`para correr 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

AmazonBraket proporciona una variedad de cuadernos de ejemplo que no dependen del PennyLane complemento para ejecutar algoritmos híbridos. Puedes empezar con cualquiera de estos cuadernos de ejemplo híbridos de Amazon Braket que ilustran métodos variacionales, como el algoritmo de optimización aproximada cuántica (QAOA) o el solucionador propio cuántico variacional (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 Braket. Amazon 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

AmazonBraket Hybrid Jobs ahora viene con simuladores integrados de alto rendimiento basados en CPU y GPU de. PennyLane Esta familia de simuladores integrados se puede integrar directamente en tu contenedor de tareas híbridas e incluye el rápido `lightning.qubit` simulador vectorial de estado, el simulador acelerado mediante la biblioteca `lightning.gpu` CuQuantum de NVIDIA y otros. Estos simuladores integrados son ideales para algoritmos variacionales, como el aprendizaje automático 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, por ejemploSV1, o directamente utilizando el simulador integrado. PennyLane

El simulador integrado ya está disponible con el contenedor Hybrid Jobs, simplemente necesitas decorar tu función principal de Python con el `@hybrid_job` decorador. Para usar el PennyLane `lightning.gpu` simulador, también debes especificar una instancia de GPU `InstanceConfig` como se muestra en el siguiente fragmento de código:

```
import pennylane as qml
from braket.jobs import hybird_job
from braket.jobs.config import InstanceConfig


@hybrid_job(device="local:pennylane/lightning.gpu",
 instance_config=InstanceConfig(instanceType="ml.p3.8xlarge"))
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 en SV1.

Nota: Para utilizar el método de diferenciación adjunto, debe especificarlo **`diff_method='device'`** en el suyo y no. **`qnode`** `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 calculará los índices de agrupamiento para los hamiltonianos QAOA y los utilizará para dividir el hamiltoniano en varios valores esperados. Si desea utilizar la capacidad de diferenciación adjunta del SV1 al ejecutar el QAOA, necesitará reconstruir el coste hamiltoniano eliminando los índices de PennyLane 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)`

Utilice Amazon Braket Hybrid Jobs y ejecute un PennyLane algoritmo QAOA

En esta sección, utilizará lo que ha aprendido para escribir un programa híbrido real utilizando PennyLane la compilación paramétrica. Utilizará el script del algoritmo para solucionar un problema del algoritmo de optimización aproximada cuántica (QAOA). El programa crea una función de coste correspondiente a un problema de optimización de corte máximo clásico, especifica un circuito cuántico parametrizado y utiliza un método de descenso de gradiente simple para optimizar los parámetros de forma que se minimice la función de coste. En este ejemplo, generamos el gráfico del problema en el script del algoritmo para simplificar, pero para los casos de uso más típicos, 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 forma `parametrize_differentiable` predeterminada para `True` que pueda beneficiarse automáticamente de un mejor rendimiento en tiempo de ejecución gracias a la compilación paramétrica en las QPUs compatibles.

```
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 todas las QPU superconductoras basadas en compuertas Rigetti Computing y Oxford Quantum Circuits con la excepción de los programas de nivel de pulso.

Acelere sus cargas de trabajo híbridas con simuladores integrados de 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 la GPU de Pennylane utiliza la biblioteca Nvidia CuQuantum para `lightning.gpu` acelerar las simulaciones de circuitos. El simulador de GPU integrado viene preconfigurado en todos los contenedores de tareas de Braket, que los usuarios pueden utilizar de forma inmediata. En esta página, le mostramos cómo usarlo `lightning.gpu` para acelerar sus cargas de trabajo híbridas.

Utilización **`lightning.gpu`** para cargas de trabajo de algoritmos de optimización aproximada cuántica

Considere los ejemplos del algoritmo de optimización cuántica aproximada (QAOA) de este cuaderno. Para seleccionar un simulador integrado, debe especificar que el `device` argumento sea una cadena con la forma:. `"local:<provider>/<simulator_name>"` Por ejemplo, usted configuraría `"local:pennylane/lightning.gpu"` para`lightning.gpu`. La cadena de dispositivo que se proporciona al Hybrid Job al lanzarlo se transfiere al trabajo como variable de entorno`"AMZN_BRAKET_DEVICE_ARN"`.

```
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, comparemos los dos simuladores vectoriales de PennyLane estado integrados `lightning.qubit` (que están basados en la CPU) y `lightning.gpu` (que están basados

en la GPU). Deberás proporcionar a los simuladores algunas descomposiciones de compuertas personalizadas para poder calcular varios gradientes.

Ahora ya está listo para preparar el guion de lanzamiento de trabajos híbridos. Ejecutará el algoritmo QAOA mediante dos tipos de instancias: `m5.2xlarge` y. `p3.2xlarge` El tipo de `m5.2xlarge` instancia es comparable al de un portátil estándar para desarrolladores. `p3.2xlarge`Se trata de una instancia de computación acelerada que tiene una sola GPU NVIDIA Volta con 16 GB de memoria.

`hyperparameters`Para todos sus trabajos híbridos, será el mismo. Todo lo que necesita hacer para probar diferentes instancias y simuladores es cambiar dos líneas de la siguiente manera.

```
# Specify device that the hybrid job will primarily be targeting
device = "local:pennylane/lightning.qubit"
# Run on a CPU based instance with about as much power as a laptop
instance_config = InstanceConfig(instanceType='ml.m5.2xlarge')
```

o bien:

```
# Specify device that the hybrid job will primarily be targeting
device = "local:pennylane/lightning.gpu"
# Run on an inexpensive GPU based instance
instance_config = InstanceConfig(instanceType='ml.p3.2xlarge')
```

> **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. ¡Asegúrate de usar el simulador integrado basado en la GPU si quieres apuntar a la GPU!

En primer lugar, puedes crear dos tareas híbridas y resolver Max-Cut con QAOA en un gráfico con 18 vértices. Esto se traduce en un circuito de 18 qubits, relativamente pequeño y fácil de ejecutar rápidamente en el portátil o en la instancia. `m5.2xlarge`

```
num_nodes = 18
num_edges = 24
seed = 1967
```

```
graph = nx.gnm_random_graph(num_nodes, num_edges, seed=seed)

# And similarly for the p3 job
m5_job = AwsQuantumJob.create(
    device=device,
    source_module="qaoa_source",
    job_name="qaoa-m5-" + str(int(time.time())),
    image_uri=image_uri,
    # Relative to the source_module
    entry_point="qaoa_source.qaoa_algorithm_script",
    copy_checkpoints_from_job=None,
    instance_config=instance_config,
    # general parameters
    hyperparameters=hyperparameters,
    input_data={"input-graph": input_file_path},
    wait_until_complete=True,
)
```

El tiempo medio de iteración de la `m5.2xlarge` instancia es de unos 25 segundos, mientras que el de la `p3.2xlarge` instancia es de unos 12 segundos. Para este flujo de trabajo de 18 qubits, la instancia de GPU nos proporciona una aceleración del doble. Si echas un vistazo a la página de precios de Amazon Braket Hybrid Jobs, verás que el coste por minuto de una `m5.2xlarge` instancia es de 0,00768$, mientras que para la `p3.2xlarge` instancia es de 0,06375$. Ejecutarlo durante 5 iteraciones en total, como hiciste aquí, costaría 0,016$ con la instancia de CPU o 0,06375$ con la instancia de GPU, ¡ambas opciones bastante económicas!

Ahora vamos a complicar el problema e intentemos resolver un problema de corte máximo en un gráfico de 24 vértices, lo que se traducirá en 24 qubits. Vuelva a ejecutar los trabajos híbridos en las mismas dos instancias y compare el costo.

> **Note**
>
> ¡Verá que el tiempo necesario para ejecutar este trabajo híbrido en la instancia de CPU puede ser de unas cinco horas!

```
num_nodes = 24
num_edges = 36
seed = 1967
```

```
graph = nx.gnm_random_graph(num_nodes, num_edges, seed=seed)

# And similarly for the p3 job
m5_big_job = AwsQuantumJob.create(
    device=device,
    source_module="qaoa_source",
    job_name="qaoa-m5-big-" + str(int(time.time())),
    image_uri=image_uri,
    # Relative to the source_module
    entry_point="qaoa_source.qaoa_algorithm_script",
    copy_checkpoints_from_job=None,
    instance_config=instance_config,
    # general parameters
    hyperparameters=hyperparameters,
    input_data={"input-graph": input_file_path},
    wait_until_complete=True,
)
```

El tiempo medio de iteración de la `m5.2xlarge` instancia es de aproximadamente una hora, mientras que el de la `p3.2xlarge` instancia es de aproximadamente dos minutos. Para este problema mayor, ¡la instancia de la GPU es un orden de magnitud más rápida! Todo lo que tenías que hacer para beneficiarte de esta aceleración era cambiar dos líneas de código, sustituyendo el tipo de instancia por el simulador local utilizado. Ejecutarlo durante 5 iteraciones en total, como se hizo aquí, costaría unos 2.27072$ con la instancia de CPU o unos 0.775625$ con la instancia de GPU. El uso de la CPU no solo es más caro, sino que también lleva más tiempo ejecutarlo. Acelerar este flujo de trabajo con una instancia PennyLane de GPU disponible AWS, mediante el simulador integrado respaldado por NVIDIA CuQuantum, te permite ejecutar flujos de trabajo con recuentos de cúbits intermedios (entre 20 y 30) por un coste total menor y en menos tiempo. Esto significa que puedes experimentar con la computación cuántica incluso para problemas que son demasiado grandes como para ejecutarlos rápidamente en un portátil o en una instancia de tamaño similar.

Aprendizaje automático cuántico y paralelismo de datos

Si su tipo de carga de trabajo es el aprendizaje automático cuántico (QML) que se entrena con conjuntos de datos, puede acelerar aún más su carga de trabajo mediante 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 cálculos 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 aprovechar el 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 del conjunto de datos Sonar del conocido repositorio de la UCI como ejemplo de clasificación binaria. El conjunto de datos del sonar tiene 208 puntos de datos, cada uno con 60 características que se recopilan a partir de las señales del 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 ejemplos de cuadernos para obtener más información sobre la carga de trabajo. Al igual que en el ejemplo anterior de QAOA, puedes aprovechar la potencia de la GPU mediante el uso de simuladores integrados basados en la GPU, `lightning.gpu` para mejorar el rendimiento en comparación con los simuladores basados en la CPU integrada. PennyLane

Para crear un trabajo híbrido, puedes llamar `AwsQuantumJob.create` y especificar el script del algoritmo, el dispositivo y otras configuraciones mediante sus argumentos de palabras clave.

```
instance_config = InstanceConfig(instanceType='ml.p3.2xlarge')

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 SageMaker distribuida paralelice correctamente el entrenamiento. En primer lugar, importa el `smdistributed` paquete que se encarga de la mayor parte del trabajo pesado de distribuir las cargas de trabajo entre varias GPU y varias instancias. 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 GPU para el entrenamiento (`world_size`), así como el número `rank` y del 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 GPU 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 una en `DistributedSampler` función de `world_size` y `rank` y, a continuación, se pasa al cargador de datos. Este muestreador evita que las GPU accedan al mismo segmento 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, usa la `DistributedDataParallel` clase 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 utilizar el paralelismo de datos. En QML, a menudo quieres guardar los resultados e imprimir el progreso del entrenamiento. Si cada GPU ejecuta el comando de guardar e imprimir, el registro se inundará con la información repetida y los resultados se sobrescribirán entre sí. Para evitarlo, solo puedes guardar e imprimir desde la GPU que tenga `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.p3.16xlarge` instancias para la biblioteca paralela de datos SageMaker distribuidos. El tipo de instancia se configura mediante el `InstanceConfig` argumento de Hybrid Jobs. 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 utiliza estos dos hiperparámetros. `smdistributed` No es necesario que el script de su algoritmo los utilice de forma explícita. En Amazon Braket SDK, proporciona un práctico argumento de palabra clave. `distribution` `distribution="data_parallel"`En el caso de la creación de empleos híbridos, el SDK de Amazon Braket inserta automáticamente los dos hiperparámetros por ti. Si utilizas la API Amazon Braket, debes incluir estos dos hiperparámetros.

Con el paralelismo de instancias y datos configurado, ahora puede enviar su trabajo híbrido. Hay 8 GPU en una instancia. `ml.p3.16xlarge` Cuando lo configura`instanceCount=1`, la carga de trabajo se distribuye entre las 8 GPU de la instancia. Si configuras `instanceCount` más de una, la carga de trabajo se distribuye entre las GPU disponibles en todas las instancias. Al usar varias instancias, cada instancia conlleva un cargo en función del tiempo que la utilices. Por ejemplo, cuando usa cuatro instancias, el tiempo facturable es cuatro veces el tiempo de ejecución por instancia, ya que hay cuatro instancias que ejecutan sus cargas de trabajo al mismo tiempo.

```
instance_config = InstanceConfig(instanceType='ml.p3.16xlarge',
                                 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 de empleos híbridos anterior, `train_dp.py` se encuentra el script de algoritmo modificado para utilizar el paralelismo de datos. Tenga en cuenta que el paralelismo de datos solo funciona correctamente cuando modifica el script del algoritmo de acuerdo con la sección anterior. Si la opción de paralelismo de datos está habilitada sin un script de algoritmo modificado correctamente, es posible que el trabajo híbrido genere errores o que cada GPU procese repetidamente el mismo segmento de datos, lo que resulta ineficiente.

Comparemos el tiempo de ejecución y el coste en un ejemplo en el que entrenamos un modelo con un circuito cuántico de 26 qubits para resolver el problema de clasificación binaria mencionado anteriormente. La `ml.p3.16xlarge` instancia utilizada en este ejemplo cuesta 0,4692$ por minuto. Sin el paralelismo de datos, el simulador tarda unos 45 minutos en entrenar el modelo para una época (es decir, más de 208 puntos de datos) y cuesta unos 20 dólares. Con el paralelismo de datos en 1 instancia y 4 instancias, solo se necesitan 6 minutos y 1,5 minutos respectivamente, lo que se traduce en unos 2,8$ para ambas. Al utilizar el paralelismo de datos en 4 instancias, no solo se mejora el tiempo de ejecución en 30 veces, ¡sino que también se reducen los costes en un orden de magnitud!

Cree y depure un trabajo híbrido con el modo local

Si está creando un nuevo algoritmo híbrido, el modo local le ayuda a depurar y probar el script de su algoritmo. El modo local es una función que le permite ejecutar el código que planea usar en Amazon Braket Hybrid Jobs, pero sin necesidad de que Braket gestione la infraestructura necesaria para ejecutar el trabajo híbrido. En su lugar, ejecuta los trabajos híbridos de forma local en su instancia de Braket Notebook o en un cliente preferido, como un ordenador portátil o de sobremesa. En el modo local, puedes seguir enviando tareas cuánticas a dispositivos reales, pero no obtendrás las ventajas de rendimiento si ejecutas con una QPU real en modo local.

Para usar el modo local, modifíquelo `AwsQuantumJob` al `LocalQuantumJob` lugar en el que se produzca. Por ejemplo, para ejecutar el ejemplo de Cree su primer trabajo híbrido, edite el script del trabajo híbrido 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 portátiles Amazon Braket, debe estar instalado en su entorno local para poder utilizar esta función. Las instrucciones para instalar Docker se encuentran aquí. Además, no todos los parámetros son compatibles en el modo local.

Traiga su propio contenedor (BYOC)

Amazon Braket Hybrid Jobs proporciona tres contenedores prediseñados 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 su algoritmo cuando cree un trabajo híbrido. Las pequeñas dependencias que falten se pueden añadir desde el script del algoritmo o desde un `requirements.txt` archivo utilizando`pip`.

Si ninguno de estos contenedores es compatible con su caso de uso, o si desea ampliarlos, Braket Hybrid Jobs le permite ejecutar trabajos híbridos con su propia imagen de Docker contenedor

personalizada o usar su propio contenedor (BYOC). Pero antes de entrar en detalles, asegurémonos de que realmente es la función adecuada para su caso de uso.

¿Cuándo es la decisión correcta traer mi propio contenedor?

Traer su propio contenedor (BYOC) a Braket Hybrid Jobs ofrece la flexibilidad de usar su propio software al instalarlo en un entorno empaquetado. Según sus necesidades específicas, puede haber formas de lograr la misma flexibilidad sin tener que pasar por todo el ciclo completo de BYOC Docker creación (carga de Amazon ECR) y URI de imagen personalizada.

> **Note**
>
> Es posible que BYOC no sea la elección correcta si desea agregar una pequeña cantidad de paquetes de Python adicionales (generalmente menos de 10) que estén disponibles públicamente. Por ejemplo, si estás usando. PyPi

En este caso, puede utilizar una de las imágenes Braket prediseñadas y, a continuación, incluir un `requirements.txt` archivo en el directorio de origen al enviar el trabajo. El archivo se lee automáticamente e `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 Python y, si corresponde, CUDA del contenedor precompilado que desee utilizar para comprobar si el software funciona.

El BYOC es necesario cuando se quiere utilizar un lenguaje que no sea Python (como C++ o Rust) para el script de trabajo, o si se quiere utilizar una versión de Python que no esté disponible en los contenedores prediseñados de Braket. También es una buena opción si:

- Está utilizando un software con una clave de licencia y necesita autenticar esa clave en un servidor de licencias para ejecutar el software. Con BYOC, puede incrustar la clave de licencia en su Docker imagen e incluir un código para autenticarla.
- Estás utilizando un software que no está disponible públicamente. Por ejemplo, el software está alojado en un GitHub repositorio privado GitLab o al que se necesita una clave SSH específica para acceder.
- Necesitas instalar un paquete grande de software que no esté empaquetado en los contenedores proporcionados por Braket. El BYOC le permitirá eliminar los largos tiempos de arranque de sus contenedores de trabajos híbridos debido a la instalación del software.

BYOC también le permite poner su SDK o algoritmo personalizado a disposición de los clientes al crear un Docker contenedor 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.

Receta para traer tu propio contenedor

En esta sección, te ofrecemos una step-by-step guía con todo lo que necesitas bring your own container (BYOC) para crear Hybrid Jobs: los scripts, los archivos y los pasos para combinarlos y empezar a trabajar con tus Docker imágenes personalizadas. Proporcionamos recetas para dos casos comunes:

1. Instale software adicional en una Docker imagen y utilice únicamente scripts de algoritmos de Python en sus trabajos.
2. Utilice scripts de algoritmos escritos en un lenguaje que no sea Python con Hybrid Jobs o una arquitectura de CPU distinta de x86.

Definir el script de entrada del contenedor es más complejo en el caso 2.

Cuando Braket ejecuta su Hybrid Job, lanza el número y el tipo solicitados de instancias de Amazon EC2 y, a continuación, ejecuta Docker la imagen especificada en la entrada del URI de la imagen para crear el trabajo en ellas. Al utilizar la función BYOC, debe especificar un URI de imagen alojado en un repositorio privado de Amazon ECR al que tiene acceso de lectura. Braket Hybrid Jobs utiliza esa imagen personalizada para ejecutar el trabajo.

Los componentes específicos que necesita para crear una Docker imagen que pueda usarse con Hybrid Jobs. Si no está familiarizado con la escritura y la creación`Dockerfiles`, le sugerimos que consulte la documentación de Dockerfile y la Amazon ECR CLIdocumentación según sea necesario mientras lee estas instrucciones.

Aquí tienes un resumen de lo que necesitarás:

- Una imagen base para tu Dockerfile
- (Opcional) Un script de punto de entrada al contenedor modificado

- R: Dockerfile que instale todo el software necesario e incluya el script del contenedor

Una imagen base para tu Dockerfile

Si utilizas Python y quieres 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 BYOC Docker podría ser: `FROM [IMAGE_URI_HERE]`

A continuación, rellene el resto Dockerfile para instalar y configurar el software que desea añadir al contenedor. Las imágenes prediseñadas de Braket ya contienen el script de punto de entrada al contenedor adecuado, así que no tienes que preocuparte por incluirlo.

Si quieres usar un lenguaje que no sea Python, como C++, Rust o Julia, o si deseas crear una imagen para una arquitectura de CPU que no sea x86, como ARM, es posible que tengas que compilarla sobre 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 al contenedor modificado

> **Note**
>
> Si solo vas a añadir software adicional a una imagen de Braket prediseñada, puedes saltarte esta sección.

Para ejecutar código que no sea de Python como parte de su trabajo híbrido, tendrá que modificar el script de Python que define el punto de entrada del contenedor. Por ejemplo, el script de `braket_container.py` python del Github de Amazon Braket. Este es el script que utilizan las imágenes prediseñadas por Braket para lanzar el script de su 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 prediseñado, 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

thekick_off_customer_script()función para iniciar procesos de Rust en función del final de la extensión del archivo.

También puede optar por escribir una completamente nuevabraket_container.py. Debe copiar los datos de entrada, los archivos fuente y otros archivos necesarios de Amazon S3 al contenedor y definir las variables de entorno adecuadas.

R: **Dockerfile** que instale todo el software necesario e incluya el script del contenedor

> **Note**
>
> Si utiliza una imagen Braket prediseñada como imagen Docker base, el script del contenedor ya está presente.

Si creó un script de contenedor modificado en el paso anterior, tendrá que copiarlo en el contenedor y definir la variable de entorno o el nombre SAGEMAKER_PROGRAM con braket_container.py el que ha denominado el nuevo script de punto de entrada del contenedor.

El siguiente es un ejemplo de un ejemplo Dockerfile que te permite usar Julia en instancias de Jobs aceleradas por la 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 regido 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, Docker Compose utilízalas al compilar para permitir Docker el acceso a SSH en la máquina host en la que está integrado. Para obtener más información, consulta la guía Cómo usar claves SSH de forma segura en Docker para acceder a los repositorios privados de Github.

Cómo crear y cargar una imagen Docker

Una vez definido correctamente`Dockerfile`, ya puede seguir los pasos para crear un repositorio privado de Amazon ECR, si aún no existe uno. También puede crear, etiquetar y cargar la imagen de su contenedor en el repositorio.

Estás listo para crear, etiquetar e insertar la imagen. Consulta la documentación de compilación de Docker para obtener una explicación completa de las opciones `docker build` y algunos ejemplos.

Para el archivo de muestra definido anteriormente, puedes ejecutar:

```
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

Braket Hybrid Jobs Dockerlas imágenes deben estar alojadas en repositorios privados de Amazon ECR. De forma predeterminada, un repositorio privado de Amazon ECR no proporciona acceso de lectura 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. Por lo general, solo debes dar permiso a los usuarios y IAM roles específicos a los que desees acceder a tus imágenes, en lugar de permitir que cualquier persona que tenga los permisos image URI los extraiga.

Ejecuta las tareas híbridas de Braket en tu propio contenedor

Para crear un trabajo híbrido con su propio contenedor, llame `AwsQuantumJob.create()` con el argumento `image_uri` especificado. Puede utilizar una QPU, un simulador bajo demanda, o ejecutar el código localmente en el procesador clásico disponible en Braket Hybrid Jobs. Recomendamos probar el código en un simulador como SV1, DM1 o TN1 antes de ejecutarlo en una QPU real.

Para ejecutar el código en el procesador clásico, especifique el `instanceType` y el que va a `instanceCount` utilizar actualizando el. `InstanceConfig` Tenga en cuenta que si especifica un `instance_count` > 1, debe asegurarse de que el código pueda ejecutarse en varios hosts. El límite máximo de instancias que puedes 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.p3.8xlarge", 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`.`. La cadena está limitada a 256 caracteres.
>
> Si planea usar BYOC y no está usando el SDK de Braket para crear tareas cuánticas, debe pasar el valor de la variable de entorno `AMZN_BRAKET_JOB_TOKEN` al `jobToken` parámetro de la solicitud. `CreateQuantumTask` Si no lo haces, las tareas cuánticas no tienen prioridad y se consideran tareas cuánticas normales e independientes.

Configure el depósito predeterminado en **`AwsSession`**

Si proporciona el suyo propio`AwsSession`, tendrá mayor flexibilidad, por ejemplo, en la ubicación del depósito predeterminado. De forma predeterminada, an `AwsSession` tiene una ubicación de

depósito predeterminada def`"amazon-braket-{id}-{region}"`. Sin embargo, puede anular ese valor predeterminado al crear un`AwsSession`. Si lo desean, los usuarios pueden pasar un `AwsSession` objeto a `AwsQuantumJob.create` con el nombre del parámetro, tal y `aws_session` como se muestra en el siguiente ejemplo de código.

```
aws_session = AwsSession(default_bucket="other-default-bucket")

# then you can use that AwsSession when creating a hybrid job
job = AwsQuantumJob.create(
    ...
    aws_session=aws_session
)
```

Interactúe directamente con los trabajos híbridos mediante el API

Puede acceder a Amazon Braket Hybrid Jobs e interactuar con ellos directamente mediante el. API Sin embargo, los métodos predeterminados y prácticos no están disponibles cuando se utiliza directamente. API

> **Note**
>
> Le recomendamos encarecidamente que interactúe con Amazon Braket Hybrid Jobs mediante el SDK Amazon Braket Python. Ofrece ajustes y protecciones convenientes que ayudan a que sus trabajos híbridos se ejecuten correctamente.

En este tema se describen los aspectos básicos del uso deAPI. Si decide utilizar la API, tenga en cuenta que este enfoque puede ser más complejo y prepárese para varias iteraciones a fin de ejecutar su trabajo híbrido.

Para usar la API, tu cuenta debe tener una función en la política `AmazonBraketFullAccess` administrada.

> **Note**
>
> Para obtener más información sobre cómo obtener un rol con la política `AmazonBraketFullAccess` gestionada, consulta la página Habilitar Amazon Braket.

Además, necesita un rol de ejecución. Esta función se transferirá al servicio. Puede crear el rol mediante la consola Amazon Braket. Utilice la pestaña Funciones de ejecución de la página de permisos y configuración para crear una función predeterminada para los trabajos híbridos.

`CreateJob`APIRequiere que especifique todos los parámetros necesarios para el trabajo híbrido. Para usar Python, comprima los archivos de script del 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/Aspen-M-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 del trabajo híbrido a través de la consola `GetJob` API o la consola. Para obtener los detalles del trabajo híbrido de la sesión de Python en la que ejecutó el `createJob` código, como en el ejemplo anterior, utilice el siguiente comando de Python.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

Para cancelar un trabajo híbrido, llama a `CancelJob` API with the Amazon Resource Name of the job ('JobArn').

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

Puede especificar puntos de control como parte del `createJob` API uso del `checkpointConfig` parámetro.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> El LocalPath de `checkpointConfig` no puede comenzar con ninguna de las siguientes rutas reservadas:`/opt/ml`, `/opt/braket/tmp`, o. `/usr/local/nvidia`

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 cálculos realizados. Si bien la computación cuántica tolerante a fallos promete una solución a este problema, los dispositivos cuánticos actuales están limitados por el número de cúbits 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 ruidosa computación cuántica. 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.

Mitigación de errores en IonQ Aria

La mitigación de errores implica ejecutar varios circuitos físicos y combinar sus mediciones para obtener un resultado mejorado. El IonQ Aria dispositivo cuenta con un método de mitigación de errores denominado «debiasing».

Debiasing mapea un circuito en múltiples variantes que actúan sobre diferentes permutaciones de qubits o con diferentes descomposiciones de compuertas. Esto reduce el efecto de los errores sistemáticos, como las sobrerotaciones de las compuertas o un solo qubit defectuoso, al utilizar diferentes implementaciones de un circuito que, de otro modo, podrían sesgar los resultados de las mediciones. Esto supone una sobrecarga adicional para calibrar varios cúbits y compuertas.

Para obtener más información sobre la depuración, consulte Mejora del rendimiento de los ordenadores cuánticos mediante la simetrización.

> **Note**
>
> El uso de la depuración requiere un mínimo de 2500 tomas.

Puede ejecutar una tarea cuántica con el debiasing en un IoQ Aria dispositivo mediante el siguiente código:

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

```
from braket.circuits import Circuit
from braket.error_mitigation import Debias

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

Definición

También puede acceder a las probabilidades de medición calculadas con una estrategia de posprocesamiento diferente denominada nitidez. La nitidez compara los resultados de cada variante y descarta los disparos inconsistentes, lo que favorece el resultado de medición más probable en todas las variantes. Para obtener más información, consulte Mejorar el rendimiento de los ordenadores cuánticos mediante la simetrización.

Es importante destacar que la nitidez asume que la forma de la distribución de salida es dispersa, 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 más precisa en el `additional_metadata` campo del SDK de Python de Braket. `GateModelTaskResult` Tenga en cuenta que la precisión no devuelve los recuentos de mediciones, sino que devuelve una distribución de probabilidad renomalizada. El siguiente fragmento de código muestra cómo acceder a la distribución después de aplicar el enfoque.

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

Braket Direct

Con Braket Direct, puede reservar el 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.

En esta sección:

- Reservas
- Asesoramiento de expertos
- Capacidades experimentales

Reservas

Las reservas le dan acceso exclusivo al dispositivo cuántico de su elección. 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 y se pueden cancelar con hasta 48 horas de antelación, sin coste adicional. Puede optar por poner en cola las tareas cuánticas y los trabajos híbridos para una próxima reserva con antelación o enviar las cargas de trabajo durante la reserva.

El coste del acceso exclusivo a un dispositivo depende de la duración de la reserva, independientemente del número de tareas cuánticas e híbridas que ejecute en la unidad de procesamiento cuántico (QPU).

Se pueden reservar los siguientes ordenadores cuánticos:

- Aria de IonQ
- Garnet de IQM
- QuEra¿Es Aquila
- El Aspen-M-3 de Rigetti

¿Cuándo usar una reserva

Aprovechar el acceso dedicado a los dispositivos con reservas 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 la investigación, lo que permitirá una iteración rápida y rentable de los algoritmos. Cuando esté listo para obtener los resultados finales del experimento, considere la posibilidad de programar una reserva de dispositivos cuando le resulte más cómodo para asegurarse de que puede cumplir con los plazos del proyecto o de la publicación. También te recomendamos que utilices las reservas cuando desees ejecutar tareas en momentos específicos, como cuando estés organizando una demostración en directo o un taller en un ordenador cuántico.

En esta sección:

- Crea una reserva
- Gestiona tu carga de trabajo con una reserva
- Cancela o reprograma una reserva existente

Crea una reserva

Para crear una reserva, ponte en contacto con el equipo de Braket siguiendo estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Braket Direct en el panel izquierdo y, a continuación, en la sección Reservas, selecciona Reservar dispositivo.
3. Selecciona el dispositivo que deseas 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 periódicamente.
5. En Cuéntanos sobre tu carga de trabajo, proporciona cualquier detalle sobre la carga de trabajo que se va a ejecutar con tu reserva. Por ejemplo, la duración de la reserva deseada, las restricciones relevantes o el horario deseado.
6. Si estás interesado en ponerte en contacto con un experto de Braket para una sesión de preparación de reservas después de que se confirme tu reserva, si lo deseas, selecciona Estoy interesado en una sesión de preparación.

También puedes ponerte en contacto con nosotros para crear una reserva siguiendo estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Dispositivos en el panel izquierdo y elige el dispositivo que deseas reservar.
3. En la sección Resumen, selecciona Reservar dispositivo.
4. Siga los pasos 4 a 6 del procedimiento anterior.

Después de enviar el formulario, recibirá un correo electrónico del equipo de Braket con los siguientes pasos para crear su reserva. Una vez confirmada su reserva, recibirá el ARN de reserva por correo electrónico.

> **Note**
>
> Su reserva solo se confirma 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 comparte contigo cualquier información relevante antes de confirmar la reserva.

Si has indicado tu interés en una sesión de preparación de la reserva, el equipo de Braket se pondrá en contacto contigo por correo electrónico para concertar una sesión de 30 minutos con un experto de Braket.

Gestiona tu carga de trabajo con una reserva

Durante una reserva, solo tus cargas de trabajo se ejecutan en el dispositivo. Para designar las tareas cuánticas y las tareas híbridas que se ejecutarán durante la reserva de un dispositivo, debe utilizar un ARN de reserva válido.

> **Note**
>
> Las reservas son específicas de AWS la cuenta y del dispositivo. Solo la AWS cuenta que creó la reserva puede usar su ARN de reserva. Además, el ARN de reserva solo es válido en el dispositivo reservado a las horas de inicio y finalización elegidas.

Para aprovechar al máximo el tiempo reservado, puedes optar por poner en cola las tareas y los trabajos antes de la reserva. Estas cargas de trabajo permanecen en ese `QUEUED` estado hasta que

comience la reserva. Cuando se inicia la reserva, todas las cargas de trabajo en cola se ejecutan en el orden en que se enviaron. Las tareas de trabajo se priorizan antes que las tareas cuánticas independientes.

> **Note**
>
> Como solo sus cargas de trabajo se ejecutan durante la reserva, no hay visibilidad en las colas para las tareas y los trabajos enviados con un ARN de reserva.

Ejemplos de código para crear una tarea cuántica para una reserva:

1. Defina un circuito para preparar el estado de GHZ en formato OpenQASM.

```
// 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;
```

2. Cree una tarea cuántica con su circuito y el ARN de reserva.

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

# import the device module
from braket.aws import AwsDevice
from braket.ir.openqasm import Program

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

program = Program(source=ghz_qasm_string)
```

```
# Reservation ARN will be of the form arn:aws:braket:us-
east-1:<AccountId>:reservation/<ReservationId>
# Example: arn:aws:braket:us-east-1:123456789012:reservation/f17cc20b-1ba4-461f-8854-
de4bb2aa64c1
###############################################################################
# IMPORTANT: If the reservation ARN is not specified, the created task
# queues and runs outside of the reservation.
# (The only exception is when the task is created by the script of a hybrid
#  job that had the reservation ARN passed at the time of its creation.
#  See “Code example for creating a hybrid job for a Braket Direct reservation:”
#  in the following section.)
###############################################################################
my_task = device.run(
    program,
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)

# You can also specify a particular Amazon S3 bucket location
# and the desired number of shots, when running the program.
# If no S3 location is specified, a default Amazon S3 bucket is chosen at amazon-
braket-{region}-{account_id}
# If no shot count is specified, 1000 shots are applied by default.
s3_location = ("amazon-braket-my-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)
```

Ejemplo de código para crear un trabajo híbrido para una reserva de Braket Direct:

1. Defina el script de su algoritmo.

```
//algorithm_script.py

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!!!!!")
```

2. Cree el trabajo híbrido con el script de su algoritmo y el ARN de reserva.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)
```

3. Cree el trabajo híbrido con el decorador remoto.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.devices import Devices
from braket.jobs import hybrid_job, get_job_device_arn

@hybrid_job(device=Devices.IonQ.Aria1, reservation_arn="arn:aws:braket:us-
east-1:<AccountId>:reservation/<ReservationId>")
def sample_job():
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)
    task = device.run(bell, shots=10)
    measurements = task.result().measurements
    return measurements
```

¿Qué ocurre al final de tu reserva

Una vez finalizada la reserva, ya no tendrás acceso exclusivo al dispositivo. Las cargas de trabajo restantes que estén en cola con esta reserva se cancelarán automáticamente.

> **Note**
>
> Se cancela cualquier trabajo que estuviera en `RUNNING` estado al finalizar la reserva. Te recomendamos que utilices puntos de control para guardar y reiniciar los trabajos según te convenga.

Una reserva en curso, por ejemplo, una vez iniciada y antes de su finalización, no se puede prorrogar, ya que cada reserva representa un acceso exclusivo e independiente a un dispositivo. Por ejemplo, dos back-to-back reservas se consideran independientes y cualquier tarea pendiente de la primera reserva se cancela automáticamente. No se reanudan 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.

Cancela o reprograma una reserva existente

Puede cancelar su reserva al menos 48 horas antes de la hora de inicio programada. Para cancelar, responde al correo electrónico de confirmación de reserva que recibiste con tu solicitud de cancelación.

Para volver a programarla, debes cancelar tu reserva actual y, a continuación, crear una nueva.

Asesoramiento de expertos

Conéctese 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, elija Braket Direct en el panel izquierdo y vaya a la sección de asesoramiento de expertos. Están disponibles las siguientes opciones de asesoramiento experto:

- Horario de oficina de Braket: el horario de oficina de Braket consiste en sesiones individuales, por orden de llegada, y se llevan 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 de la idea a la ejecución con mayor rapidez, ya que explora use-case-to-device las opciones para aprovechar 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 apuntarse al horario de oficina de Braket, seleccione Registrarse y complete la información de contacto, los detalles de la carga de trabajo y los temas de debate que desee.
  - Recibirás por correo electrónico una invitación programada para el siguiente espacio disponible.

  > **Note**
  >
  > Para problemas emergentes o preguntas rápidas de solución de problemas, le recomendamos que se ponga en contacto con AWS Support. Si tiene preguntas que no sean urgentes, también puede utilizar el foro AWS Re:post o Quantum Computing Stack Exchange, donde puede consultar las preguntas respondidas anteriormente y formular otras nuevas.

- Ofertas de proveedores de hardware cuántico: iONQ, Oxford Quantum Circuits QuEra, y Rigetti ofrecen ofertas de servicios profesionales a través de. AWS Marketplace
  - Para explorar sus ofertas, selecciona Connect y navega por sus listados.
  - Para obtener más información sobre las ofertas de servicios profesionales disponibles en AWS Marketplace, consulte los productos de servicios profesionales.
- AmazonLaboratorio de soluciones cuánticas (QSL): el QSL es un equipo colaborativo de investigación y servicios profesionales compuesto por expertos en computación cuántica que pueden ayudarlo a explorar la computación cuántica de manera eficaz y a evaluar el rendimiento actual de esta tecnología.
  - Para ponerse en contacto con la QSL, seleccione Connect y complete la información de contacto y los detalles del caso de uso.
  - El equipo de QSL se pondrá en contacto contigo por correo electrónico para explicarte los pasos a seguir.

Capacidades experimentales

Para mejorar sus cargas de trabajo de investigación, es importante acceder rápidamente a nuevas capacidades innovadoras. Con Braket Direct, puede solicitar el acceso a las capacidades experimentales disponibles, como los nuevos dispositivos cuánticos de disponibilidad limitada, directamente en la consola Braket.

Algunas capacidades experimentales funcionan fuera de las especificaciones estándar de los dispositivos y requieren una orientación práctica adaptada a su caso de uso. Para garantizar que sus cargas de trabajo estén preparadas para el éxito, puede acceder a ellas previa solicitud a través de Braket Direct.

Acceso a IonQ Forte solo con reserva

Con Braket Direct, solo tiene acceso con reserva a la IonQ Forte QPU. Debido a su disponibilidad limitada, este dispositivo solo está disponible a través de Braket Direct.

Para obtener más información y solicitar acceso a IonQ Forte, sigue estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Braket Direct en el menú de la izquierda y, a continuación, en Capacidades experimentales, navega hasta IonQ Forte. Selecciona Ver dispositivo.
3. En la página de detalles del dispositivo Forte, en Resumen, selecciona Reservar dispositivo.
4. Proporcione su información de contacto, incluidos el nombre y el correo electrónico. Proporcione una dirección de correo electrónico válida que consulte periódicamente.
5. En la sección Háblenos de su carga de trabajo, proporcione detalles sobre la carga de trabajo que implica utilizar su reserva, como la duración deseada de la reserva, las restricciones pertinentes o el horario deseado.
6. (Opcional) Si estás interesado en ponerte en contacto con un experto de Braket para una sesión de preparación de reservas una vez confirmada tu reserva, selecciona Estoy interesado en una sesión de preparación.

Una vez enviado el formulario, el equipo de Braket se pondrá en contacto contigo para explicarte los pasos a seguir.

> **Note**
>
> Debido a la disponibilidad limitada de los dispositivos, el acceso a Forte es limitado. Póngase en contacto con nosotros para obtener más información.

Acceso a la sintonización local en Aquila QuEra

Con Braket Direct, puedes solicitar acceso para controlar la desafinación local al programar en la QPU. QuEra Aquila Con esta capacidad, puedes ajustar en qué medida el campo de conducción afecta a cada qubit específico.

Para obtener más información y solicitar el acceso a esta función, sigue estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Braket Direct en el menú de la izquierda y, a continuación, en Capacidades experimentales, navega hasta QuEra Aquila: desafinación local. Selecciona Obtener acceso.
3. Proporcione su información de contacto, incluidos el nombre y el correo electrónico. Proporcione una dirección de correo electrónico válida que consulte periódicamente.
4. En Háblenos de su carga de trabajo, proporcione detalles sobre la carga de trabajo y sobre dónde piensa utilizar esta capacidad.

Acceso a geometrías altas en Aquila QuEra

Con Braket Direct, puede solicitar el acceso a geometrías ampliadas al programar en la QPU. QuEra Aquila Con esta capacidad, puede experimentar más allá de las capacidades estándar de los dispositivos y especificar geometrías con una mayor altura de celosía.

Para obtener más información y solicitar acceso a esta función, sigue estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Braket Direct en el menú de la izquierda y, a continuación, en Capacidades experimentales, navega hasta QuEra Aquila: geometrías altas. Selecciona Obtener acceso.
3. Proporcione su información de contacto, incluidos el nombre y el correo electrónico. Proporcione una dirección de correo electrónico válida que consulte periódicamente.
4. En Háblenos de su carga de trabajo, proporcione detalles sobre la carga de trabajo y sobre dónde piensa utilizar esta capacidad.

Acceso a geometrías reducidas en Aquila QuEra

Con Braket Direct, puede solicitar el acceso a geometrías ampliadas al programar en la QPU. QuEra Aquila Con esta capacidad, puede experimentar más allá de las capacidades estándar de los dispositivos y organizar las filas de celosía con un espaciado vertical más reducido.

Para obtener más información y solicitar el acceso a esta función, sigue estos pasos:

1. Abre la consola Amazon Braket.
2. Selecciona Braket Direct en el menú de la izquierda y, a continuación, en Capacidades experimentales, navega hasta QuEra Aquila: geometrías altas. Selecciona Obtener acceso.
3. Proporcione su información de contacto, incluidos el nombre y el correo electrónico. Proporcione una dirección de correo electrónico válida que consulte periódicamente.
4. En Háblenos de su carga de trabajo, proporcione detalles sobre la carga de trabajo y sobre dónde piensa utilizar esta capacidad.

Registro y supervisión

Después de enviar una tarea cuántica, puedes realizar un seguimiento de su estado a través del SDK y la consola de Amazon Braket. Cuando se completa la tarea cuántica, Braket guarda los resultados en la ubicación de Amazon S3 que haya especificado. La finalización puede tardar algún tiempo, especialmente en el caso de los dispositivos QPU, en función de la longitud de la cola. Los tipos de estado incluyen:

- `CREATED`— Amazon Braket recibió tu tarea cuántica.
- `QUEUED`— Amazon Braket procesó tu tarea cuántica y ahora está esperando para ejecutarse en el dispositivo.
- `RUNNING`— Tu tarea cuántica se ejecuta 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 y falló. Dependiendo de la razón por la que haya fallado tu tarea cuántica, intenta volver a enviarla.
- `CANCELLED`— Cancelaste la tarea cuántica. La tarea cuántica no se ejecutó.

En esta sección:

- Seguimiento de tareas cuánticas desde el SDK de Amazon Braket
- Supervisión de tareas cuánticas a través de la consola Amazon Braket
- Etiquetado de los recursos de Amazon Braket
- Eventos y acciones automatizadas para Amazon Braket with Amazon EventBridge
- Monitorización de Amazon Braket con Amazon CloudWatch
- Registro de la API Amazon Braket con CloudTrail
- Cree una instancia de bloc de notas Amazon Braket usando AWS CloudFormation
- Registro avanzado

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 un seguimiento del estado con, tal y `task.state()` como se muestra en el siguiente ejemplo.

Nota: `task = device.run()` es una operación asíncrona, lo que significa que puedes seguir trabajando mientras el sistema procesa tu tarea cuántica en segundo plano.

Recupera un resultado

Cuando llamas`task.result()`, el SDK comienza a sondear a Amazon Braket para ver si la tarea cuántica está completa. El SDK usa los parámetros de sondeo que usted definió en`.run()`. Una vez completada la tarea cuántica, el SDK recupera el resultado del depósito de S3 y lo devuelve como un `QuantumTaskResult` objeto.

```
# 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
```

Cancela una tarea cuántica

Para cancelar una tarea cuántica, llame al `cancel()` método, como se muestra en el siguiente ejemplo.

```
# cancel quantum task
task.cancel()
status = task.state()
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

Compruebe 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
```

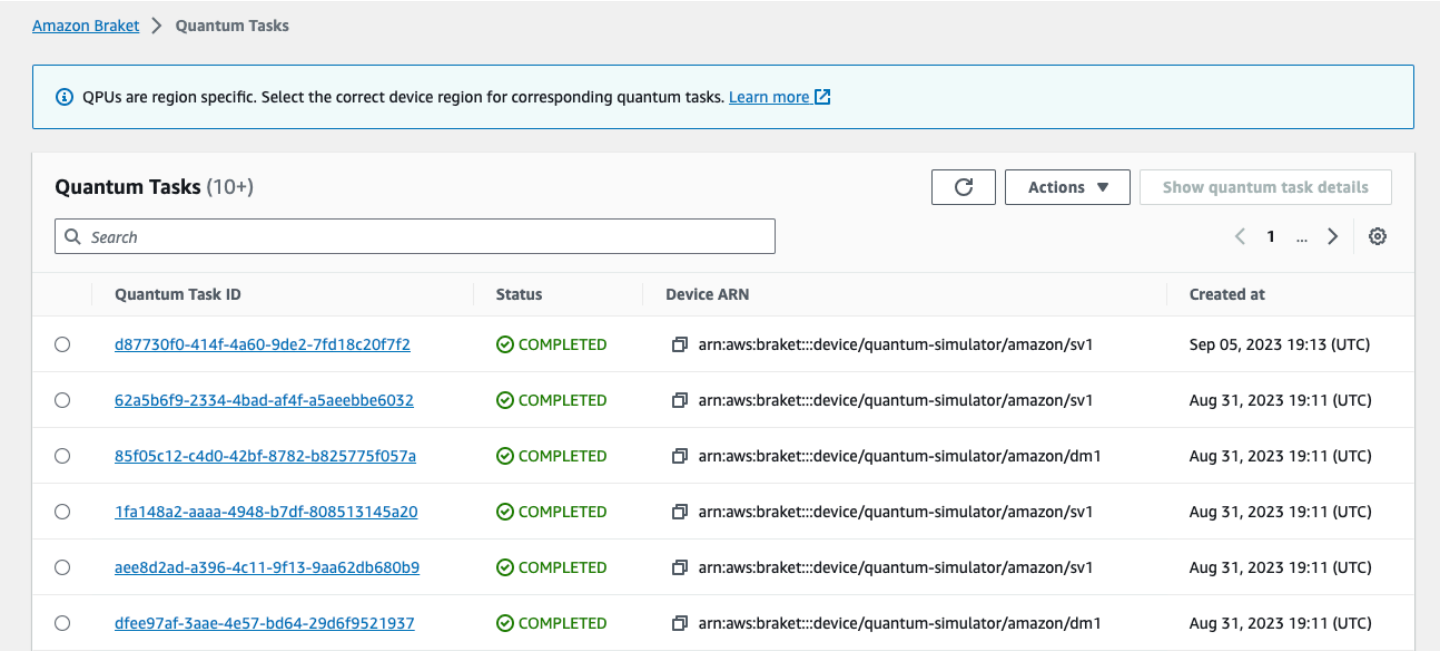
Recupera una tarea o resultado cuántico

Si su núcleo muere después de enviar la tarea cuántica o si cierra su cuaderno o computadora, puede reconstruir el `task` objeto con su ARN (ID de tarea cuántica) único. A continuación, puede llamar `task.result()` para obtener el resultado del depósito S3 en el que está almacenado.

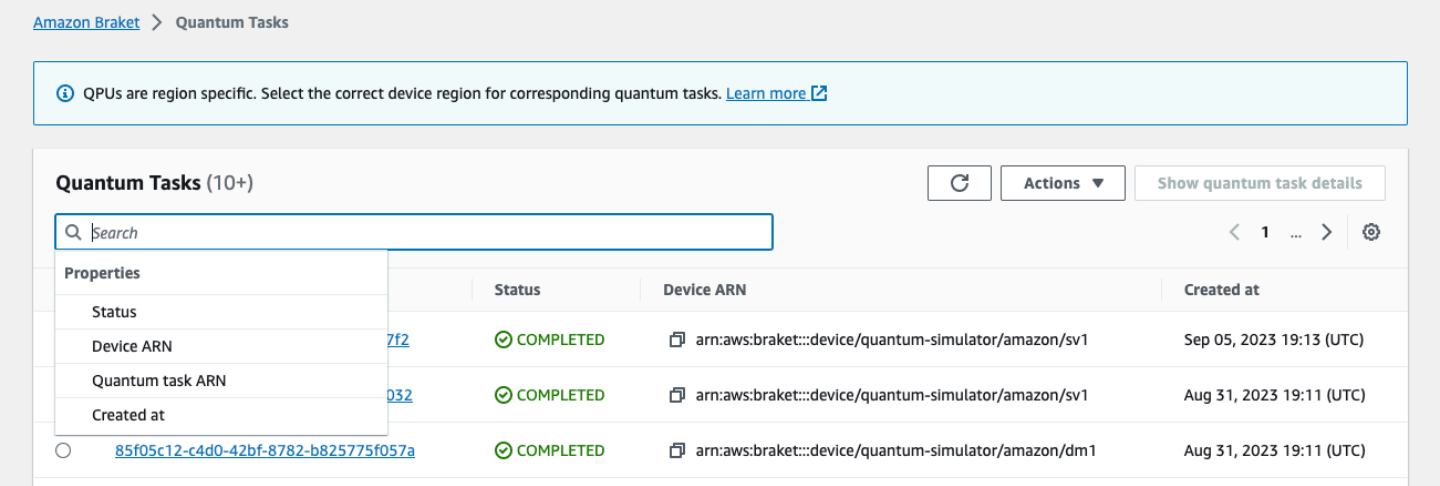
```
from braket.aws import AwsSession, AwsQuantumTask

# 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 Amazon Braket

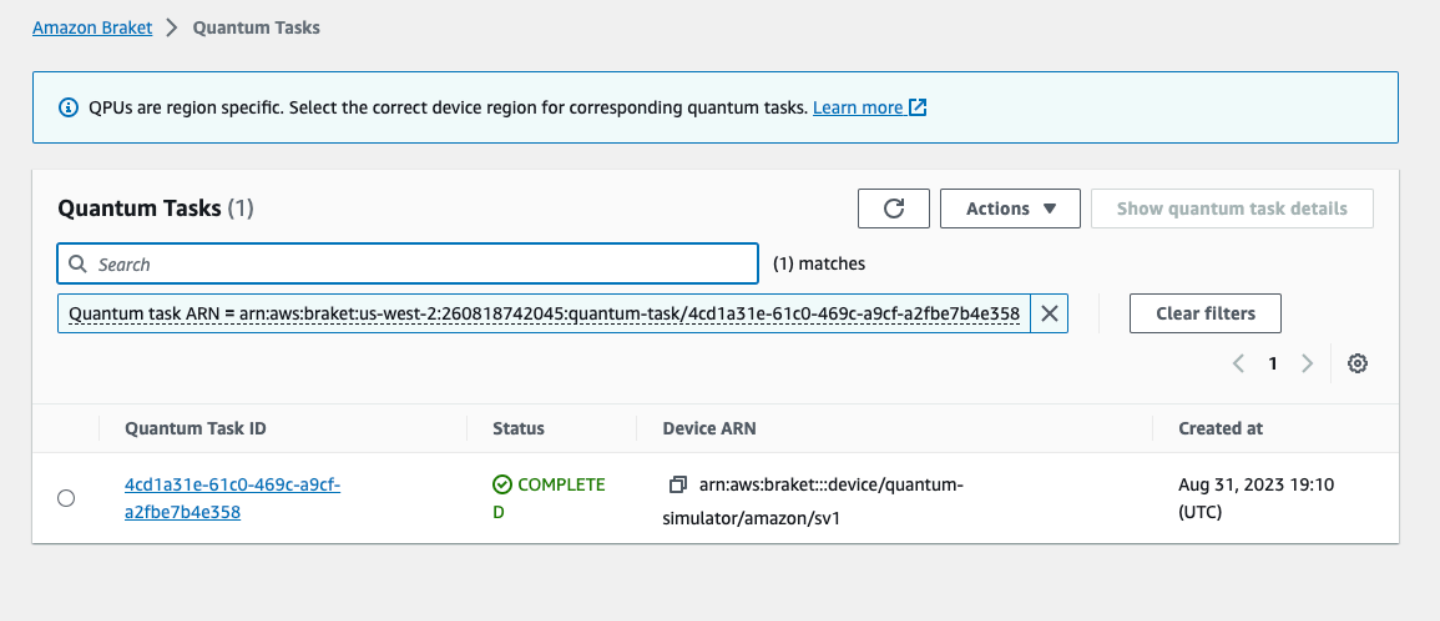
AmazonBraket ofrece una forma cómoda de monitorizar la tarea cuántica a través de la consola Amazon Braket. Todas las tareas cuánticas enviadas se muestran en el campo Tareas cuánticas, como se muestra en la siguiente figura. Este servicio es específico de una región, lo que significa que solo puede ver las tareas cuánticas creadas en una región específica. Región de AWS



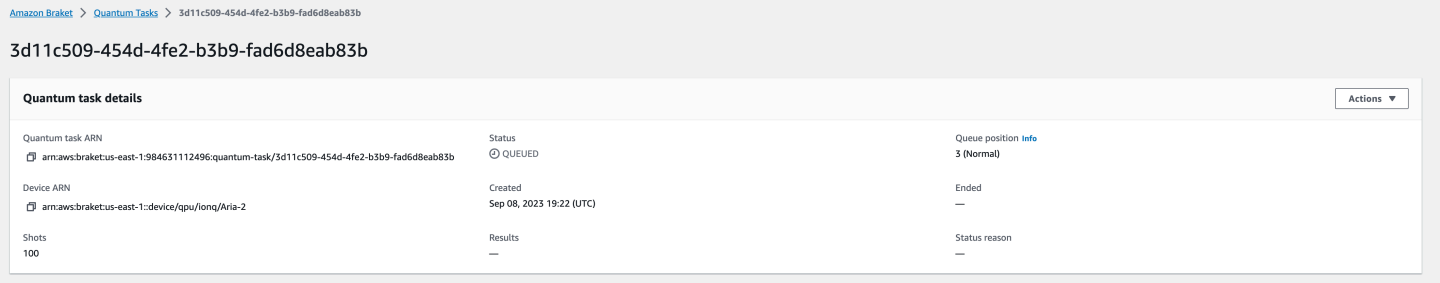
Puede buscar tareas cuánticas específicas a través de la barra de navegación. La búsqueda se puede basar en el ARN (ID) de Quantum Task, el estado, el dispositivo y la hora de creación. Las opciones aparecen automáticamente al seleccionar la barra de navegación, como se muestra en el siguiente ejemplo.



La siguiente imagen muestra un ejemplo de búsqueda de una tarea cuántica en función de su identificador único de tarea cuántica, que se puede obtener mediante una llamada`task.id`.



Además, en la siguiente figura, se puede monitorear el estado de una tarea cuántica mientras está en ese 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 de Quantum enviadas como parte de un trabajo híbrido tendrán prioridad cuando estén en cola. Las tareas de Quantum enviadas fuera de un trabajo híbrido tendrán prioridad normal en las listas de espera.

Los clientes que deseen consultar el SDK de Braket pueden obtener sus puestos en lista de tareas cuánticas y puestos en espera de trabajos híbridos mediante programación. Para obtener más información, consulte la página Cuándo se ejecutará mi tarea.

Etiquetado de los recursos de Amazon Braket

Una etiqueta es una etiqueta de atributo personalizada que usted asigna o que AWS asigna a un recurso. AWS Una etiqueta es un metadato que proporciona más información sobre el 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, puedes navegar hasta una tarea cuántica o un cuaderno y ver la lista de etiquetas asociadas a ella. 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.

Uso de etiquetas

Las etiquetas pueden organizar tus recursos en categorías que te resulten útiles. Por ejemplo, puede asignar una etiqueta de «Departamento» para especificar el departamento propietario de este recurso.

Cada etiqueta de tiene dos partes:

- Una clave de etiqueta (por ejemplo CostCenter, Medio ambiente o Proyecto). Las claves de etiqueta distinguen entre mayúsculas y minúsculas.

- Un campo opcional conocido como valor de etiqueta (por ejemplo, 111122223333 o Producción). Omitir el valor de etiqueta es lo mismo que utilizar una cadena vacía. Al igual que las claves de etiqueta, los valores de etiqueta distinguen entre mayúsculas y minúsculas.

Las etiquetas le ayudan a realizar las siguientes tareas:

- 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 AWS Billing and Cost Management 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 de costes en la Guía del usuario de AWS Billing and Cost Management .
- Controle el acceso a sus AWS recursos. Para obtener más información, consulte Controlar el acceso mediante etiquetas.

Más información sobre AWS las etiquetas y

- Para obtener información general sobre el etiquetado, incluidas las convenciones de nomenclatura y uso, consulte AWS los recursos de etiquetado en la Referencia AWS general.
- Para obtener información sobre las restricciones del etiquetado, consulte los límites y requisitos de denominación de las etiquetas en la AWS Referencia general.
- Para conocer las mejores prácticas y estrategias de etiquetado, consulte Mejores prácticas de etiquetado y AWS Estrategias de etiquetado.
- 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 proporciona información más específica sobre las etiquetas de Braket. Amazon

Recursos compatibles en Amazon Braket

El siguiente tipo de recurso de Amazon Braket admite el etiquetado:

- Recurso de `quantum-task`
- Nombre del recurso: `AWS::Service::Braket`

- Régex del 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 libretas, aunque las libretas en realidad son recursos de Amazon. SageMaker Para obtener más información, consulte los metadatos de las instancias de Notebook en la documentación. SageMaker

Restricciones de las etiquetas

Las siguientes restricciones básicas se aplican a las etiquetas de 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.

Gestión de etiquetas en Amazon Braket

Las etiquetas se configuran como propiedades en un recurso. Puede ver, añadir, modificar, enumerar y eliminar etiquetas a través de la consola Amazon Braket, Amazon Braket API o. AWS CLI Para obtener más información, consulta la referencia de la API Amazon Braket.

Agregue etiquetas

Puede añadir etiquetas a los recursos etiquetables en los siguientes momentos:

- Al crear el recurso: utilice la consola o incluya el `Tags` parámetro con la `Create` operación en la AWS API.
- Después de crear el recurso: usa la consola para navegar hasta la tarea cuántica o el recurso del bloc de notas, o llama a la `TagResource` operación en la AWS API.

Para añadir etiquetas a un recurso al crearlo, también necesitas permiso para crear un recurso del tipo especificado.

Ver 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 a la AWS `ListTagsForResource` API operación.

Puedes usar el siguiente AWS API comando para ver las etiquetas de un recurso:

- AWS API: `ListTagsForResource`

Editar etiquetas

Puede editar las etiquetas mediante la consola para navegar hasta la tarea cuántica o el recurso del bloc de notas, o puede utilizar el siguiente comando para modificar el valor de una etiqueta adjunta 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 las etiquetas de un recurso especificando las claves que desea eliminar, utilizando la consola para navegar hasta la tarea cuántica o el recurso del bloc de notas, o al llamar a la `UntagResource` operación.

- AWS API: `UntagResource`

Ejemplo de etiquetado CLI en Amazon Braket

Si está trabajando con la AWS CLI, aquí hay un comando de ejemplo que muestra cómo crear una etiqueta que se aplique a una tarea cuántica que cree SV1 con la configuración de los parámetros de la Rigetti QPU. Observe que la etiqueta se especifica al final del comando de ejemplo. En este caso, a Key se le da el valor **`state`** y a Value se le da el valor`Washington`.

```
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\"}
```

Etiquetado con la Amazon Braket API

- Si utilizas Amazon Braket API para configurar etiquetas en un recurso, llama al. TagResourceAPI

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tags {\"city\":
\"Seattle\"}
```

- Para eliminar etiquetas de un recurso, llama al UntagResourceAPI.

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- Para ver todas las etiquetas adjuntas a un recurso concreto, llama a ListTagsForResourceAPI.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city
\",\"state\"]"
```

Eventos y acciones automatizadas para Amazon Braket with Amazon 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 sencillas 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. Entre las acciones automáticas que se pueden activar se incluyen las siguientes:

- Invocar una función AWS Lambda
- Activar una máquina de AWS Step Functions estados
- Notificar un tema sobre Amazon SNS

EventBridge monitorea estos eventos de cambio de estado de Amazon Braket:

- El estado de la tarea cuántica cambia

AmazonBraket garantiza la entrega de eventos de cambio de estado de las tareas cuánticas. Estos eventos se entregan al menos una vez, pero es posible que estén fuera de servicio.

Para obtener más información, consulte los eventos y los patrones de eventos en 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, puede crear reglas que definan las acciones que se deben tomar cuando Amazon Braket envíe una notificación de un cambio de estado relacionado con una tarea cuántica de Braket. Por ejemplo, puedes crear una regla que te envíe un mensaje de correo electrónico cada vez que cambie el estado de una tarea cuántica.

1. Inicia sesión AWS con una cuenta que tenga permisos para usar EventBridge y Amazon Braket.
2. Abre la EventBridge consola de Amazon en https://console.aws.amazon.com/events/.
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 `detail-type` sección 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 objetivo se activa cuando Braket recibe de Amazon Braket un evento de cambio de estado cuántico en una tarea.

  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, cree primero un tema de Amazon SNS con 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 y patrones de eventos en. EventBridge

Los siguientes atributos aparecen en el campo «detalle» de JSON.

- **`quantumTaskArn`**(str): la tarea cuántica para la que se generó este evento.
- **`status`**(Opcional [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 shots de solicitudes del usuario.
- **`outputS3Bucket`**(str): el segmento de salida especificado por el usuario.
- **`outputS3Directory`**(str): el prefijo clave de salida especificado por el usuario.
- **`createdAt`**(str): el tiempo de creación de la tarea cuántica expresado en una cadena ISO-8601.
- **`endedAt`**(Opcional [str]): 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 un evento de cambio de estado de Amazon Braket Quantum Task.

```
{
    "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"
      }
}
```

Monitorización de Amazon Braket con Amazon 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.

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 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	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 `deviceArn` parámetro, que tiene el formato arn:aws:braket: ::device/xxx.

Dispositivos admitidos

Para obtener una lista de los dispositivos y los ARN de dispositivos compatibles, consulte Dispositivos Braket.

> **Note**
>
> Puedes ver las secuencias de CloudWatch registro de las libretas Amazon Braket accediendo a la página de detalles de las libretas en la consola de Amazon. SageMaker Los ajustes adicionales del portátil Amazon Braket están disponibles a través de la SageMaker consola.

Registro de la API Amazon Braket con CloudTrail

AmazonBraket 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 llamadas de Amazon Braket como eventos. Las llamadas capturadas incluyen llamadas desde la consola de Amazon Braket y llamadas en código a las operaciones de Amazon Braket 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 usted CloudTrail, puede 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.

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. Puedes ver, buscar y descargar eventos recientes en tu 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 un sendero. 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 lo 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 CloudTrail UserIdentity.

Descripción de las entradas de los archivos 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 `GetQuantumTask` acción, 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 `GetDevice` acción, 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"
}
```

Cree una instancia de bloc de notas Amazon Braket usando AWS CloudFormation

Puedes utilizarla AWS CloudFormation para gestionar las instancias de tu bloc de notas Amazon Braket. Las plantillas Braket Notebook están fabricadas en Amazon SageMaker. Con CloudFormation, puedes aprovisionar una instancia de notebook 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 Braket Notebook en su interior. Cuenta de AWS

Después de crear una CloudFormation plantilla para una libreta Braket, se utilizará AWS CloudFormation para implementar el recurso. Para obtener más información, consulte Crear una pila en la AWS CloudFormation consola en la guía del AWS CloudFormation usuario.

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

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

Puede reutilizar la configuración del ciclo de vida de todas las libretas Braket que cree. También puede reutilizar la función de IAM para las libretas Braket a las que asigne los mismos permisos de ejecución.

Paso 1: Crear un script de configuración SageMaker del ciclo de vida de Amazon

Utilice la siguiente plantilla para crear un script de configuración SageMaker del ciclo de vida. El script personaliza una instancia de SageMaker notebook para Braket. Para ver las opciones de configuración del CloudFormation recurso del ciclo de vida, consulte AWS::SageMaker::NotebookInstanceLifecycleConfigla guía del AWS 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
            aws s3 cp s3://braketnotebookcdk-prod-i-
notebooklccs3bucketb3089-1cysh30vzj2ju/notebook/braket-notebook-lcc.zip braket-
notebook-lcc.zip
            unzip braket-notebook-lcc.zip
            ./install.sh
            EOS

            exit 0
```

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

Cuando utilizas una instancia de Braket Notebook, SageMaker realiza operaciones en tu nombre. Por ejemplo, supongamos que utilizas un ordenador portátil Braket utilizando un circuito de un dispositivo compatible. En la instancia del portátil, SageMaker ejecuta la operación en Braket por usted. La función de ejecución del cuaderno define las operaciones exactas que SageMaker se pueden ejecutar en tu nombre. Para obtener más información, consulta SageMaker las funciones en la guía para SageMaker desarrolladores de Amazon.

Utilice el siguiente ejemplo para crear un rol de ejecución de Braket Notebook con los permisos necesarios. Puede modificar las políticas según sus necesidades.

> **Note**
>
> Asegúrese de que el rol tenga permiso para `s3:GetObject` las operaciones `s3:ListBucket` y 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 portátil 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 Amazon SageMaker Notebook con el prefijo **`amazon-braket-`**

Utilice el script de SageMaker ciclo de vida y el rol de IAM creados en los pasos 1 y 2 para crear una instancia de SageMaker bloc de notas. La instancia del portátil está personalizada para Braket y se puede acceder a ella desde la consola Amazon Braket. Para obtener más

información sobre las opciones de configuración de este CloudFormation recurso, consulte AWS::SageMaker::NotebookInstancela guía del AWS 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
```

Registro avanzado

Puede grabar todo el proceso de procesamiento de tareas mediante 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, se recomienda cambiar los `poll_interval_seconds` parámetros `poll_timeout_seconds` y, de forma que una tarea cuántica pueda durar mucho tiempo y su estado se registre de forma continua y 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.

Configura el registrador

En primer lugar, configure el registrador para que todos los registros se escriban automáticamente en un archivo de texto, 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
```

Cree y ejecute el circuito

Ahora puede crear un circuito, enviarlo a un dispositivo para que lo ejecute y ver qué sucede, 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
)
```

Compruebe el archivo de registro

Puede comprobar lo que está escrito en el archivo introduciendo 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})
```

Obtenga el ARN del archivo de registro

A partir del resultado 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()
```

Seguridad en Amazon Braket

Este capítulo le ayuda a entender cómo aplicar el modelo de responsabilidad compartida al utilizar Amazon Braket. Le muestra cómo configurar Amazon Braket para cumplir sus objetivos de seguridad y conformidad. También aprenderás a usar otros Servicios de AWS que te ayuden a monitorear y proteger tus recursos de Amazon Braket.

La seguridad en la nube de AWS es la máxima prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos diseñados para satisfacer los requisitos de seguridad de las organizaciones más exigentes. Usted es responsable de otros factores, como la confidencialidad de sus datos, los requisitos de su empresa y las leyes y reglamentos aplicables.

Responsabilidad compartida en materia de seguridad

La seguridad es una responsabilidad compartida entre AWS y el usuario. 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 Servicios de AWS en Nube de AWS. Además, AWS 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 más información sobre los programas de conformidad que se aplican a Amazon Braket, consulte AWSServicios incluidos en el ámbito de aplicación por 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 la configuración de seguridad y las tareas de administración para los Servicios de AWS que utiliza.

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 que ejecuta la totalidad de Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es 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 Modelo de responsabilidad compartida y GDPR de AWS en el Blog de seguridad de AWS.

Con fines de protección de datos, recomendamos proteger las credenciales de la cuenta de Cuenta de AWS y configurar cuentas de usuario individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se conceden a cada usuario los permisos necesarios para cumplir con sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos de AWS. reSe recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad del usuario con AWS CloudTrail.
- Utilice las soluciones de cifrado de AWS, junto con todos los controles de seguridad predeterminados dentro de los Servicios de AWS.
- Utilice servicios de seguridad gestionados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados FIPS 140-2 al acceder a AWS a través de una interfaz de la línea de comandos o una API, utilice un punto de conexión de FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

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 los AWS CLI SDK. Cualquier dato que introduzca en etiquetas o campos de formato libre utilizados para 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 todos los identificadores de tareas cuánticas 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 búsquedas desde la consola de Amazon Braket, aunque permanecen almacenados en su bucket de S3.

Si necesita acceder al historial de tareas y resultados cuánticos almacenados en su bucket de S3 durante más de 90 días, debe mantener un registro independiente del identificador de la tarea y del resto de metadatos asociados a esos datos. Asegúrese de guardar la información antes de los 90 días. Puede utilizar la información guardada para recuperar los datos históricos.

Administrar el acceso a Amazon Braket

En este capítulo se describen los permisos necesarios para ejecutar Amazon Braket o para restringir el acceso de usuarios y funciones específicos. Puedes conceder (o denegar) los permisos necesarios a cualquier usuario o función de tu cuenta. Para ello, adjunta la política de Amazon Braket correspondiente a ese usuario o rol en tu cuenta, tal y como se describe en las siguientes secciones.

Como requisito previo, debes 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
- Cuadernos y funciones
- Acerca de la política AmazonBraketFullAccess
- Acerca de la AmazonBraketJobsExecutionPolicy política
- Restrinja el acceso de los usuarios a determinados dispositivos
- Amazon Braket actualiza las políticas gestionadas AWS
- Restrinja el acceso de los usuarios a determinadas instancias de notebook
- Restrinja el acceso de los usuarios a determinados buckets de S3

Recursos de Amazon Braket

Braket crea un tipo de recurso: el recurso de tareas cuánticas. El nombre de recurso de Amazon (ARN) para este tipo de recurso es el siguiente:

- Nombre del recurso: :Service AWS: :Braket

- Régex de ARN: ARN: $ {Partition} :braket: $ {Region} :$ {Account} :quantum-task/$ {} RandomId

Cuadernos y funciones

Puede utilizar el tipo de recurso cuaderno en Braket. Una libreta es un SageMaker recurso de Amazon que Braket puede compartir. Para usar una libreta con Braket, debes especificar un rol de IAM con un nombre que comience por. `AmazonBraketServiceSageMakerNotebook`

Para crear un bloc de notas, debe usar un rol con permisos de administrador o que tenga asociada la siguiente política interna.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": "iam:CreateRole",
           "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
       },
       {
           "Effect": "Allow",
           "Action": "iam:CreatePolicy",
           "Resource": [
               "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
               "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": "iam:AttachRolePolicy",
           "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
           "Condition": {
               "StringLike": {
                   "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 Crear un bloc de notas 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 blocs de notas que lance en el futuro.

Acerca de la política 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 contenedores que se utilizan para la función Amazon Braket Hybrid Jobs. Los contenedores deben tener el formato «arn:aws:ecr: ::repository/amazon-braket».
- Guarde AWS CloudTrail registros: para todas las acciones de descripción, obtención y lista, además de iniciar y detener consultas, probar filtros de métricas y filtrar 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.
- Utilice roles para controlar los recursos: para crear un rol vinculado a un servicio en su cuenta. El rol vinculado al servicio tiene acceso a los AWS recursos en tu nombre. Solo lo puede utilizar el servicio 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.
- Cree grupos de registros, registre eventos y consulte grupos de registros 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 registros de trabajos híbridos. Utilice la ruta de Braket adecuada y permita colocar los datos de registro. Introduzca los datos métricos. CloudWatch

- Cree y almacene datos en buckets de Amazon S3 y enumere todos los buckets: para crear buckets S3, enumere los buckets S3 de su cuenta y coloque objetos en cualquier bucket de su cuenta cuyo nombre comience por amazon-braket- y obténgalos de ellos. Estos permisos son necesarios para que Braket coloque en el depósito los archivos que contienen los resultados de las tareas cuánticas procesadas y para recuperarlos del depósito.
- Transferir funciones de IAM: para transferir las funciones de IAM a. `CreateJob` API
- Amazon SageMaker Notebook: para crear y administrar 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 SageMaker libretas y trabajos híbridos de Amazon Braket, sus recuentos de recursos no pueden superar las cuotas de su cuenta.

Contenido de la política

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock",
                "s3:PutBucketPolicy"
            ],
            "Resource": "arn:aws:s3:::amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "servicequotas:GetServiceQuota",
                "cloudwatch:GetMetricData"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ],
            "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:ListNotebookInstances"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreatePresignedNotebookInstanceUrl",
                "sagemaker:CreateNotebookInstance",
                "sagemaker:DeleteNotebookInstance",
                "sagemaker:DescribeNotebookInstance",
                "sagemaker:StartNotebookInstance",
                "sagemaker:StopNotebookInstance",
                "sagemaker:UpdateNotebookInstance",
                "sagemaker:ListTags",
                "sagemaker:AddTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "arn:aws:sagemaker:*:*:notebook-instance/amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeNotebookInstanceLifecycleConfig",
                "sagemaker:CreateNotebookInstanceLifecycleConfig",
                "sagemaker:DeleteNotebookInstanceLifecycleConfig",
                "sagemaker:ListNotebookInstanceLifecycleConfigs",
                "sagemaker:UpdateNotebookInstanceLifecycleConfig"
            ],
            "Resource": "arn:aws:sagemaker:*:*:notebook-instance-lifecycle-config/
amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": "braket:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/braket.amazonaws.com/
AWSServiceRoleForAmazonBraket*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "braket.amazonaws.com"
                }
            }
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "sagemaker.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketJobsExecutionRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "braket.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
```

```
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "/aws/braket"
                }
            }
        }
    ]
}
```

Acerca de la AmazonBraketJobsExecutionPolicy política

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 contenedores que se utilizan para la función Amazon Braket Hybrid Jobs. Los contenedores deben tener el formato «arn:aws:ecr: *:*:repository/amazon-braket*».
- Cree grupos de registros y registre eventos y consulte grupos de registros para mantener los archivos de registro de uso de su cuenta: cree, almacene y vea la información de registro sobre el uso de Amazon Braket en su cuenta. Consulte las métricas de los grupos de registros de trabajos híbridos. Utilice la ruta de Braket adecuada y permita colocar los datos de registro. Introduzca los datos métricos. CloudWatch
- Almacene datos en buckets de Amazon S3: enumere los buckets de S3 de su cuenta, coloque objetos y obtenga objetos de cualquier bucket de su cuenta que comience por amazon-braket - en su nombre. Estos permisos son necesarios para que Braket coloque en el depósito los archivos que contienen los resultados de las tareas cuánticas procesadas y los recupere del depósito.
- Transferir las funciones de IAM: transferir las funciones de IAM a. CreateJob API Los roles deben tener el formato arn:aws:iam: :* *. :role/service-role/AmazonBraketJobsExecutionRole

```
"Version": "2012-10-17",
"Statement": [
```

```
{
 "Effect": "Allow",
 "Action": [
  "s3:GetObject",
  "s3:PutObject",
  "s3:ListBucket",
  "s3:CreateBucket",
  "s3:PutBucketPublicAccessBlock",
  "s3:PutBucketPolicy"
 ],
 "Resource": "arn:aws:s3:::amazon-braket-*"
},
{
 "Effect": "Allow",
 "Action": [
  "ecr:GetDownloadUrlForLayer",
  "ecr:BatchGetImage",
  "ecr:BatchCheckLayerAvailability"
 ],
 "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
},
{
 "Effect": "Allow",
 "Action": [
  "ecr:GetAuthorizationToken"
 ],
 "Resource": "*"
},
{
 "Effect": "Allow",
 "Action": [
  "braket:CancelJob",
  "braket:CancelQuantumTask",
  "braket:CreateJob",
  "braket:CreateQuantumTask",
  "braket:GetDevice",
  "braket:GetJob",
  "braket:GetQuantumTask",
  "braket:SearchDevices",
  "braket:SearchJobs",
  "braket:SearchQuantumTasks",
  "braket:ListTagsForResource",
  "braket:TagResource",
  "braket:UntagResource"
```

```
    ],
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "iam:PassRole"
    ],
    "Resource": "arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*",
    "Condition": {
     "StringLike": {
      "iam:PassedToService": [
       "braket.amazonaws.com"
      ]
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iam:ListRoles"
    ],
    "Resource": "arn:aws:iam::*:role/*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "logs:GetQueryResults"
    ],
    "Resource": [
     "arn:aws:logs:*:*:log-group:*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "logs:PutLogEvents",
     "logs:CreateLogStream",
     "logs:CreateLogGroup",
     "logs:GetLogEvents",
     "logs:DescribeLogStreams",
     "logs:StartQuery",
     "logs:StopQuery"
    ],
```

```
    "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
   },
   {
    "Effect": "Allow",
    "Action": "cloudwatch:PutMetricData",
    "Resource": "*",
    "Condition": {
     "StringEquals": {
      "cloudwatch:namespace": "/aws/braket"
     }
    }
   }
  ]
}
```

Restrinja el acceso de los usuarios a determinados dispositivos

Para restringir el acceso de determinados usuarios a determinados dispositivos Braket, puedes añadir una política de denegación de permisos a una IAM función específica.

Con estos permisos, se pueden restringir las siguientes acciones:

- `CreateQuantumTask`- denegar la creación de tareas cuánticas en dispositivos específicos.
- `CreateJob`- denegar la creación de empleos híbridos en dispositivos específicos.
- `GetDevice`- denegar la obtención de detalles de dispositivos específicos.

El siguiente ejemplo restringe el acceso a todas las QPU para el. Cuenta de AWS `123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
```

```
    }
  ]
}
```

Para adaptar este código, sustituya el número de Amazon recurso (ARN) del dispositivo restringido por la cadena que se muestra en el ejemplo anterior. Esta cadena proporciona el valor del recurso. En Braket, un dispositivo representa una QPU o un simulador al que se puede llamar para ejecutar tareas cuánticas. Los dispositivos disponibles aparecen en la página Dispositivos. Se utilizan dos esquemas para especificar el acceso a estos dispositivos:

- `arn:aws:braket:<region>:<account id>:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:<account id>:device/quantum-simulator/<provider>/<device_id>`

A continuación, se muestran ejemplos de varios tipos de acceso a dispositivos

- Para seleccionar todas las QPU de todas las regiones: `arn:aws:braket:*:*:device/qpu/*`
- Para seleccionar ÚNICAMENTE todas las QPU de la región us-west-2: `arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- Del mismo modo, para seleccionar ÚNICAMENTE todas las QPU de 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:* :123456789012:device/quantum-simulator/*`
- Para restringir el acceso al IonQ Harmony dispositivo en la región us-east-1: `arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- Para restringir el acceso a los dispositivos de un proveedor determinado (por ejemplo, a Rigetti QPU los dispositivos): `arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- Para restringir el acceso al TN1 dispositivo: `arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

Amazon Braket actualiza las políticas gestionadas AWS

En la siguiente tabla se proporcionan detalles sobre las actualizaciones de las políticas AWS gestionadas de Braket desde que este servicio comenzó a realizar un seguimiento de estos cambios.

Cambio	Descripción	Fecha
AmazonBraketFullAccess- Política de acceso completo a Braket	Se agregaron las GetMetric Data acciones servicequota GetServiceQuota y cloudwatc h, que se incluirán en la política. AmazonBraketFullAccess .	24 de marzo de 2023
AmazonBraketFullAccess- Política de acceso completo a Braket	Objetivo ajustado por corchetes: PassRole permisos AmazonBra ketFullAccess para incluir la ruta. `service-role/`	29 de noviembre de 2021
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 `service-role/` la ruta.	29 de noviembre de 2021
Braket comenzó a rastrear los cambios	Braket comenzó a realizar un seguimiento de los cambios en sus políticas AWS gestionadas.	29 de noviembre de 2021

Restrinja el acceso de los usuarios a determinadas instancias de notebook

Para restringir el acceso de determinados usuarios a instancias específicas de Braket Notebook, puedes añadir una política de denegación de permisos a un rol, usuario o grupo específico.

En el siguiente ejemplo, se utilizan variables de política para restringir de forma eficaz los permisos para iniciar, detener y acceder a determinadas instancias de 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`).

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

Restrinja 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 específico.

El siguiente ejemplo restringe los permisos para recuperar y colocar objetos en un S3 depósito específico (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) y también restringe la lista de esos objetos.

```
{
  "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 depósito a una instancia determinada del bloc de notas, puede añadir la política anterior a la función de ejecución del bloc de notas.

Función vinculada al servicio Amazon Braket

Cuando habilitas Amazon Braket, se crea un rol vinculado a un servicio en tu cuenta.

Un rol vinculado a un servicio es un tipo único de rol de IAM que, en este caso, está vinculado directamente a Amazon Braket. El rol vinculado al servicio Amazon Braket está predefinido para incluir todos los permisos que Braket necesita para llamar a otros Servicios de AWS en su nombre.

Un rol vinculado a un servicio facilita la configuración de Amazon Braket, ya que no es necesario añadir los permisos necesarios manualmente. Amazon Braket define los permisos de sus funciones

vinculadas a servicios. A menos que cambie estas definiciones, solo Amazon Braket puede asumir sus funciones. 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.

La función vinculada a servicios que establece Amazon Braket forma parte de la capacidad de funciones vinculadas a servicios AWS Identity and Access Management (IAM). Para obtener información sobre otros Servicios de AWS que admiten funciones vinculadas a servicios, consulte AWSServicios que funcionan con IAM y busque los servicios que tienen Sí en la columna Función vinculada a servicios. Seleccione una opción Sí con un enlace para ver la documentación acerca del rol vinculado al servicio en cuestión.

Permisos de rol vinculados a servicios para Amazon Braket

Amazon Braket usa el rol `AWSServiceRoleForAmazonBraket` vinculado al servicio que confía en que la entidad braket.amazonaws.com asuma el rol.

Debe configurar los permisos para permitir que una entidad de IAM (como un grupo o un rol) cree, edite o elimine un rol vinculado a un servicio. Para obtener más información, consulte Permisos de roles vinculados a servicios.

La función vinculada al servicio en Amazon Braket tiene los siguientes permisos de forma predeterminada:

- Amazon S3: permisos para enumerar los depósitos de su cuenta y colocar objetos en y obtener objetos de cualquier depósito de su cuenta con un nombre que comience por amazon-braket -.
- Amazon CloudWatch Logs: permisos para enumerar y crear grupos de registros, crear los flujos de registro asociados e incluir eventos en el grupo de registros creado para Amazon Braket.

La siguiente política se adjunta a la función **`AWSServiceRoleForAmazonBraket`**vinculada al servicio:

```
{"Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket"
            ],
```

```
            "Resource": "arn:aws:s3:::amazon-braket*"
        },
        {"Effect": "Allow",
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket/*"
        },
        {"Effect": "Allow",
            "Action": "braket:*",
            "Resource": "*"
        },
        {"Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/braket.amazonaws.com/
AWSServiceRoleForAmazonBraket*",
            "Condition": {"StringEquals": {"iam:AWSServiceName": "braket.amazonaws.com"
                }
            }
        }
    ]
}
```

Resiliencia en Amazon Braket

La infraestructura AWS global se basa en Regiones de AWS zonas de disponibilidad.

Cada región ofrece varias zonas de disponibilidad que están separadas y aisladas físicamente. Estas zonas de disponibilidad (AZ) se conectan a través de redes de baja latencia, alto rendimiento y alta redundancia. Como resultado, las zonas de disponibilidad tienen mayor disponibilidad, son más tolerantes a errores y son más escalables que las infraestructuras tradicionales de uno o varios centros de datos.

Puede diseñar y operar aplicaciones y bases de datos que se conmuten por error entre las AZ de forma automática, sin interrupciones.

Para obtener más información Regiones de AWS y sobre las zonas de disponibilidad, consulte Infraestructura AWS global.

Validación de conformidad para Amazon Braket

Los auditores externos evalúan periódicamente la seguridad y el cumplimiento de Amazon Braket y nuestra integración con proveedores de hardware de terceros. Para obtener una up-to-date lista de la información de cumplimiento de Braket, consulte Servicios de AWSel alcance por programa de cumplimiento. Para obtener información general, consulte AWSconformidad.

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

> **Note**
>
> AWSlos informes de cumplimiento no incluyen las QPU de proveedores de hardware externos que pueden optar por someterse a sus propias auditorías independientes.

Su responsabilidad de cumplimiento al utilizar Amazon Braket viene determinada por la confidencialidad de sus datos, los objetivos de cumplimiento de su empresa y las leyes y reglamentos aplicables. AWSproporciona los siguientes recursos para facilitar el cumplimiento:

- Guías de inicio rápido de seguridad y conformidad: estas guías de implementación tratan consideraciones sobre arquitectura y ofrecen pasos para implementar los entornos de referencia centrados en la seguridad y la conformidad en AWS.
- Recursos de conformidad de AWS: este conjunto de manuales y guías podría aplicarse a su sector y ubicación.

Seguridad de infraestructura en Amazon Braket

Como servicio gestionado, Amazon Braket está protegido por los procedimientos de seguridad de la red AWS global que se describen en el documento técnico AWS: Descripción general de los procesos de seguridad.

Para acceder a Amazon Braket a través de la red, debe realizar llamadas a las AWS API publicadas. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o una versión posterior. Los clientes también deben admitir conjuntos de cifrado con un secreto directo (PFS) perfecto, como

Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Seguridad de los proveedores de hardware de Amazon Braket

Las QPU de Amazon Braket están alojadas por proveedores de hardware externos. Cuando ejecutas 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 utilizas Amazon Braket para acceder al hardware de computación cuántica operado por uno de los proveedores de hardware externos, tu circuito y sus datos asociados son procesados por proveedores de hardware ajenos a las instalaciones operadas porAWS. Los datos sobre la ubicación física yAWSLa región en la que está disponible cada QPU se encuentra en elLos datos del dispositivosección de la consola Amazon Braket.

Su contenido está anonimizado. Solo el contenido necesario para procesar el circuito se envía a terceros.Cuenta de AWSla información no se transmite a terceros.

Los datos cifrados en reposo y en tránsito. Los datos se descifran únicamente para su procesamiento. Los proveedores externos de Amazon Braket no están autorizados a almacenar ni utilizar tu contenido para otros fines que no sean el procesamiento de tu circuito. Una vez que se completa el circuito, los resultados se devuelven a Amazon Braket y se almacenan en el depósito S3.

La seguridad de los proveedores externos de hardware cuántico de Amazon Braket se audita periódicamente para garantizar que se cumplan los estándares de seguridad de la red, control de acceso, protección de datos y seguridad física.

Puntos de enlace de Amazon VPC para Amazon Braket

Puede establecer una conexión privada entre su VPC y Amazon Braket mediante la creación de un punto final de VPC de interfaz. Los puntos finales de la interfaz funcionan con una tecnología que permite el acceso a las API de Braket sin necesidad de una pasarela de Internet, un dispositivo NAT, una conexión VPN o una conexión. AWS PrivateLinkAWS Direct Connect Las instancias de su VPC no necesitan direcciones IP públicas para comunicarse con las API de Braket.

Cada punto de conexión de la interfaz está representado por una o más interfaces de red elásticas en las subredes.

De este 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 Interface VPC endpoints (AWS PrivateLink) en la Guía del usuario de Amazon VPC.

Consideraciones sobre los puntos de conexión de Amazon Braket VPC

Antes de configurar un punto de enlace de VPC de interfaz para Braket, asegúrese de revisar las propiedades y limitaciones del punto de enlace de interfaz en la Guía del usuario de Amazon VPC.

Braket permite realizar llamadas a todas sus acciones de API desde su VPC.

De forma predeterminada, se permite el acceso total a Braket a través del punto final de la VPC. Puede controlar el acceso si especifica las políticas de punto final de la VPC. Para obtener más información, consulte Control del acceso a los servicios con puntos de conexión de VPC en la Guía del 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 Amazon del API servicio Braket.

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.
- Cree un punto de conexión de Amazon VPC para Braket
- Conecta y ejecuta las tareas cuánticas de Braket a través de tu terminal

Paso 1: Lance una Amazon VPC si es necesario

Recuerda que puedes saltarte este paso si tu cuenta ya tiene una VPC en funcionamiento.

Una VPC controla la configuración de la red, como el rango de direcciones IP, las subredes, las tablas de enrutamiento y las puertas de enlace de red. Básicamente, está lanzando sus AWS

recursos en una red virtual personalizada. Para obtener más información sobre VPC, consulte la Guía del usuario de Amazon VPC.

Abra la consola de Amazon VPC y cree una nueva VPC con subredes, grupos de seguridad y puertas de enlace de red.

Paso 2: Crear un punto final de VPC de interfaz 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 Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Para crear un punto de enlace de VPC en la consola, abra la consola de Amazon VPC, abra la página de puntos de enlace y proceda a crear el nuevo punto de enlace. Anote el ID del punto de conexión para consultarlo más adelante. Es obligatorio como parte de la `–endpoint-url` marca cuando se realizan determinadas llamadas al BraketAPI.

Cree el punto final de VPC para Braket con el siguiente nombre de servicio:

- `com.amazonaws.substitute_your_region.braket`

Nota: Si habilita el DNS privado para el punto final, puede realizar API solicitudes a Braket utilizando su nombre de DNS predeterminado para la región, por ejemplo. `braket.us-east-1.amazonaws.com`

Para obtener más información, consulte Acceso a un servicio a través de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.

Paso 3: Conecta y ejecuta las tareas cuánticas de Braket a través de tu terminal

Después de crear un punto de enlace de VPC, puede ejecutar comandos de CLI que incluyan el `endpoint-url` parámetro para especificar los puntos de enlace de la interfaz para el API 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 DNS privados para su punto de enlace de VPC, no necesita especificar el punto de enlace como una URL en los comandos de CLI. En su lugar, el nombre de

host API DNS de Amazon Braket, que la CLI y el SDK de Braket utilizan de forma predeterminada, se resuelve en el punto final de la VPC. Tiene el formulario que se muestra en el siguiente ejemplo:

```
https://braket.substituteYourRegionHere.amazonaws.com
```

La entrada del blog titulada Acceso directo a los SageMaker blocs de notas de Amazon 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 los blocs de notas, que son similares SageMaker Amazon a los blocs de notas Braket.

Si sigues los pasos de la entrada del blog, recuerda sustituir el nombre AmazonBraket por Amazon SageMaker. Si tu región no es us-east-1, introduce **`com.amazonaws.us-east-1.braket`** o sustituye la cadena por tu Región de AWS nombre correcto.

Más información sobre la creación de un punto final

- Para obtener información sobre cómo crear una VPC con subredes privadas, consulte Crear una VPC con subredes privadas
- Para obtener información sobre la creación y configuración de un punto de conexión mediante la consola de Amazon VPC o la AWS CLI, consulte Creación de un punto de conexión de interfaz en la Guía del usuario de Amazon VPC.
- Para obtener información sobre cómo crear y configurar un punto final medianteAWS CloudFormation, consulte el recurso AWS: :EC2: :VPCendpoint en la Guía del usuario. AWS CloudFormation

Controle el acceso con las políticas de puntos finales de Amazon VPC

Para controlar el acceso de conectividad a Amazon Braket, puede adjuntar una política de punto final AWS Identity and Access Management (IAM) a su punto de enlace de Amazon VPC. La política especifica la siguiente información:

- El principal (usuario o rol) que puede realizar acciones.
- Las acciones que se pueden realizar.
- Los recursos en los que se pueden llevar a cabo las acciones.

Para obtener más información, consulte Control del acceso a los servicios con puntos de conexión de VPC en la guía del usuario de Amazon VPC.

Ejemplo: política de puntos finales de VPC para acciones de Braket

El siguiente ejemplo muestra una política de puntos finales para Braket. Cuando se adjunta a un punto final, esta política otorga acceso a las acciones de Braket enumeradas a todos los principales de 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
- Control del acceso a los servicios con puntos de conexión de VPC

Solución de problemas de Amazon Braket

Usa la información y las soluciones de solución de problemas de esta sección para ayudarte a resolver los problemas con Amazon Braket.

En esta sección:

- AccessDeniedException
- Se ha producido un error (ValidationException) al llamar a la operación CreateQuantumTask
- Una función del SDK no funciona
- El trabajo híbrido falla debido a ServiceQuotaExceededException
- Los componentes dejaron de funcionar en una instancia de bloc de notas
- Cuotas de Amazon Braket
- Solucione los problemas de OpenQASM

AccessDeniedException

Si recibe un aviso AccessDeniedExceptional activar o utilizar Braket, es probable que esté intentando activar o utilizar Braket en una región a la que no tiene acceso su función restringida.

En esos casos, debe ponerse en contacto con su AWS administrador interno para saber cuáles de las siguientes condiciones se aplican:

- Si hay restricciones de roles que impidan el acceso a una región.
- Si el rol que intenta usar tiene permiso para usar Braket.

Si tu rol no tiene acceso a una región determinada cuando usas Braket, no podrás usar los dispositivos de esa región en particular.

Se ha producido un error (ValidationException) al llamar a la operación CreateQuantumTask

Si recibe un error similar al siguiente: `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to`

`amazon-braket-…` Compruebe que se refiere a una carpeta s3_existente. Braket no crea automáticamente nuevos buckets y prefijos de Amazon S3.

Si está accediendo API directamente y recibe un error similar al siguiente: `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` Compruebe que no está incluido `s3://` en la ruta del bucket de Amazon S3.

Una función del SDK no funciona

La versión de Python debe ser 3.9 o superior. Para Amazon Braket Hybrid Jobs, recomendamos Python 3.10.

Compruebe que su SDK y sus esquemas lo sean. up-to-date Para actualizar el SDK desde el bloc de notas o desde el editor de Python, ejecuta el siguiente comando:

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Para actualizar los esquemas, ejecuta el siguiente comando:

```
pip install amazon-braket-schemas --upgrade
```

Si accedes a Amazon Braket desde tu propio cliente, verifica que tu AWS región esté configurada 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 contra los simuladores Amazon Braket si se supera el límite de tareas cuánticas simultáneas del dispositivo simulador al que se dirige. Para obtener más información sobre los límites de servicio, consulte el tema Cuotas.

Si ejecutas tareas simultáneas en un dispositivo simulador en varias tareas híbridas desde tu cuenta, es posible que aparezca este error.

Para ver el número de tareas cuánticas simultáneas en un dispositivo simulador específico, usa el `search-quantum-tasks`API, 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 la cuota de servicio para el número de tareas cuánticas simultáneas para el dispositivo simulador. Esto solo se aplica al SV1 dispositivo.
2. Gestiona `ServiceQuotaExceeded` las excepciones en tu código y vuelve a intentarlo.

Los componentes dejaron de funcionar en una instancia de bloc de notas

Si algunos componentes de su portátil dejan de funcionar, intente lo siguiente:

1. Descarga todos los blocs de notas que hayas creado o modificado en una unidad local.
2. Detenga la instancia de su bloc de notas.
3. Elimine la instancia de su bloc de notas.
4. Crea una nueva instancia de bloc de notas con un nombre diferente.
5. Sube los cuadernos a la nueva instancia.

Cuotas de Amazon Braket

En la siguiente tabla se muestran 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 velocidad de ráfaga no se pueden aumentar.
- El aumento de velocidad máximo para las cuotas ajustables (excepto la velocidad de ráfaga, que no se puede ajustar) es el doble del límite de velocidad predeterminado especificado. Por ejemplo, una cuota predeterminada de 60 se puede ajustar a un máximo de 120.
- La cuota ajustable para las tareas cuánticas concurrentes SV1 (DM1) permite un máximo de 60 por cada Región de AWS tarea.
- La cantidad máxima permitida de instancias informáticas para un trabajo híbrido es de 5 y las cuotas son ajustables.

Recurso	Descripción	Límites	Ajustable
Tasa de solicitudes API	El número máximo de solicitudes por segundo que puede enviar desde esta cuenta en la región actual.	140	Sí
Velocidad de ráfaga de API solicitudes	El número máximo de solicitudes adicionales por segundo (RPS) que puede enviar en una ráfaga en esta cuenta en la región actual.	600	No
Tasa de solicitudes `CreateQuantumTask`	El número máximo de `CreateQuantumTask` solicitudes que puedes enviar por segundo en esta cuenta por región.	20	Sí
Velocidad de ráfaga de `CreateQuantumTask`	El número máximo de `CreateQuantumTask` solicitud	40	No

Recurso	Descripción	Límites	Ajustable
`ntumTask` solicitudes	es adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.		
Tasa de solicitudes `SearchQuantumTasks`	El número máximo de `SearchQuantumTasks` solicitudes que puedes enviar por segundo en esta cuenta por región.	5	Sí
Velocidad de ráfaga de `SearchQuantumTasks` solicitudes	El número máximo de `SearchQuantumTasks` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	50	No
Tasa de solicitudes `GetQuantumTask`	El número máximo de `GetQuantumTask` solicitudes que puedes enviar por segundo en esta cuenta por región.	100	Sí

Recurso	Descripción	Límites	Ajustable
Velocidad de ráfaga de `GetQuantumTask` solicitudes	El número máximo de `GetQuantumTask` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	500	No
Tasa de solicitudes `CancelQuantumTask`	El número máximo de `CancelQuantumTask` solicitudes que puedes enviar por segundo en esta cuenta por región.	2	Sí
Velocidad de ráfaga de `CancelQuantumTask` solicitudes	El número máximo de `CancelQuantumTask` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	20	No
Tasa de solicitudes `GetDevice`	El número máximo de `GetDevice` solicitudes que puedes enviar por segundo en esta cuenta por región.	5	Sí

Recurso	Descripción	Límites	Ajustable
Velocidad de ráfaga de `GetDevice` solicitudes	El número máximo de `GetDevice` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	50	No
Tasa de solicitudes `SearchDevices`	El número máximo de `SearchDevices` solicitudes que puedes enviar por segundo en esta cuenta por región.	5	Sí
Velocidad de ráfaga de `SearchDevices` solicitudes	El número máximo de `SearchDevices` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	50	No
Tasa de solicitudes `CreateJob`	El número máximo de `CreateJob` solicitudes que puedes enviar por segundo en esta cuenta por región.	1	Sí

Recurso	Descripción	Límites	Ajustable
Velocidad de ráfaga de `CreateJob` solicitudes	El número máximo de `CreateJob` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	5	No
Tasa de solicitudes `SearchJob`	El número máximo de `SearchJob` solicitudes que puedes enviar por segundo en esta cuenta por región.	5	Sí
Velocidad de ráfaga de `SearchJob` solicitudes	El número máximo de `SearchJob` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	50	No
Tasa de solicitudes `GetJob`	El número máximo de `GetJob` solicitudes que puedes enviar por segundo en esta cuenta por región.	5	Sí
Velocidad de ráfaga de `GetJob` solicitudes	El número máximo de `GetJob` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	25	No

Recurso	Descripción	Límites	Ajustable
Tasa de solicitudes `CancelJob`	El número máximo de `CancelJob` solicitudes que puedes enviar por segundo en esta cuenta por región.	2	Sí
Velocidad de ráfaga de `CancelJob` solicitudes	El número máximo de `CancelJob` solicitudes adicionales por segundo (RPS) que puedes enviar en una ráfaga en esta cuenta de la región actual.	5	No
Número de tareas cuánticas simultáneas SV1	El número máximo de tareas cuánticas simultáneas que se ejecutan en el simulador de vectores 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 simultáneas DM1	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

Recurso	Descripción	Límites	Ajustable
Número de tareas cuánticas simultáneas TN1	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-OEST-2, 5 EU-OEST-2,	Sí
Número de trabajos híbridos simultáneos	El número máximo de trabajos híbridos simultáneos en la región actual.	5	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 los trabajos híbridos. Para aumentar estas cuotas, póngase en contacto con AWS Support. Además, se especifican las regiones disponibles para cada instancia.

Recurso	Descripción	Límites	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 trabajos híbridos	El número máximo de instancias del tipo ml.c4.xlarge permitido	5	Sí	Sí	Sí	Sí	Sí	No

Recurso	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
	para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.							

Recurso	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.c4.2xlarge para 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	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 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	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p3.2xlarge para trabajos híbridos	El número máximo de instancias del tipo ml.p3.2xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Recurso	Descripción	Límites	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 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

Recurso	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p3dn.24xlarge para trabajos híbridos	El número máximo de instancias del tipo ml.p3dn.24xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	No	No

Recurso	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p3.8xlarge para trabajos híbridos	El número máximo de instancias del tipo ml.p3.8xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	Sí	No

Recurso	Descripción	Límites	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instancias de ml.p3.16xlarge para trabajos híbridos	El número máximo de instancias del tipo ml.p3.16xlarge permitido para todos los trabajos híbridos de Amazon Braket en esta cuenta y región.	0	Sí	Sí	No	Sí	Sí	No

Solicitar 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 informática de aprendizaje automático 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 Amazon Braket tiene un tamaño limitado a 3 MB.
- El número máximo de disparos permitido por SV1 tarea y Rigetti dispositivo es DM1 de 100 000.
- El número máximo de disparos por tarea permitido TN1 es de 1000.
- Para IonQ los dispositivos Aria-1 y Aria-2, el máximo es de 5000 disparos por tarea. Para los dispositivos IonQ Harmony y Forte y otros dispositivos, el máximo OQC es de 10 000.
- Para QuEra ello, el máximo permitido de disparos por tarea es de 1000.
- Para los dispositivos TN1 y los QPU dispositivos, los disparos por tarea deben ser > 0.

Solucione los problemas de OpenQASM

Esta sección proporciona consejos de solución de problemas que pueden resultar útiles cuando se producen errores al utilizar OpenQASM 3.0.

En esta sección:

- Incluya un error en la sentencia
- Error no contiguo qubits
- Mezclando un error físico qubits con qubits un error virtual
- Al solicitar los tipos de resultados y medirlos qubits en el mismo programa, se produjo un error
- Error: se superaron los límites clásicos y de qubit registro
- El cuadro no va precedido de un error pragmático literal
- Error en los cuadros literales sin puertas nativas
- Falta un error físico en las cajas textuales qubits
- Falta el error «braket» del pragma literal
- Un solo qubits error no se puede indexar

- El error físico qubits en una qubit puerta con dos puertas no está conectada
- GetDevice no devuelve el error de resultados de OpenQASM
- Advertencia de compatibilidad con simuladores locales

Incluya un error en la sentencia

Actualmente, Braket no tiene un archivo de biblioteca de puertas estándar para incluirlo en los programas de OpenQASM. Por ejemplo, el siguiente ejemplo genera un error en el 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 no contiguo qubits

El uso de dispositivos no contiguos qubits en dispositivos que están `requiresContiguousQubitIndices` configurados `true` en la capacidad del dispositivo provoca un error.

Al ejecutar tareas cuánticas en simuladoresIonQ, 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].`

Mezclando un error físico qubits con qubits un error virtual

No se permite mezclar lo físico qubits con lo virtual qubits 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.`

Al solicitar los tipos de resultados y medirlos qubits en el mismo programa, se produjo un error

Si se solicitan tipos de resultados que qubits se midan de forma explícita en el mismo programa, se produce 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: se superaron los límites clásicos y de qubit registro

Solo se permite un qubit registro clásico y un registro. 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.`

El cuadro no va precedido de un error pragmático literal

Todas las casillas deben ir precedidas de un pragma literal. 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 en los cuadros literales sin puertas nativas

Los buzones literales deben tener puertas nativas y físicas. qubits 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.`

Falta un error físico en las cajas textuales qubits

Las cajas textuales deben ser físicas. qubits El siguiente código genera el error físico qubits que falta.

```
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.`

Falta el error «braket» del pragma literal

Debe incluir «corchete» en el pragma literal. 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`

Un solo qubits error no se puede indexar

El único qubits no se puede indexar. 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 qubit matrices individuales se pueden indexar de la siguiente manera:

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

El error físico qubits en una qubit puerta con dos puertas no está conectada

Para utilizar la conexión físicaqubits, compruebe primero que el dispositivo utilice la conexión física qubits `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`

GetDevice no devuelve el error de resultados de OpenQASM

Si no ve los resultados de OpenQASM en la GetDevice respuesta cuando utiliza un SDK de Braket, puede que necesite configurar la variable de entorno AWS_EXECUTION_ENV para configurar user-agent. Consulta los ejemplos de código que aparecen a continuación para saber cómo hacerlo con los SDK de Go y Java.

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al utilizar: AWS CLI

```
% export AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0"
# Or for single execution
% AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0" aws braket <cmd> [options]
```

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al usar Boto3:

```
import boto3
import botocore

client = boto3.client("braket",
 config=botocore.client.Config(user_agent_extra="BraketSchemas/1.8.0"))
```

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al usar/(SDK v2): JavaScript TypeScript

```
import Braket from 'aws-sdk/clients/braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0'  });
```

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al usar/(SDK v3): JavaScript TypeScript

```
import { Braket } from '@aws-sdk/client-braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0' });
```

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al usar el SDK de Go:

```
os.Setenv("AWS_EXECUTION_ENV", "BraketGo BraketSchemas/1.8.0")
mySession := session.Must(session.NewSession())
svc := braket.New(mySession)
```

Para configurar la variable de entorno AWS_EXECUTION_ENV para configurar el agente de usuario al usar el SDK de Java:

```
ClientConfiguration config = new ClientConfiguration();
config.setUserAgentSuffix("BraketSchemas/1.8.0");
BraketClient braketClient =
 BraketClientBuilder.standard().withClientConfiguration(config).build();
```

Advertencia de compatibilidad con simuladores locales

`LocalSimulator`Es compatible con funciones avanzadas de OpenQASM que pueden no estar disponibles en las QPU o en los simuladores bajo demanda. Si su programa contiene funciones 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 las QPUs o los simuladores bajo demanda.

Para obtener más información sobre las funciones de OpenQASM compatibles, haga clic aquí.

Guía de referencia de API y SDK para Amazon Braket

Amazon Braket proporciona API, SDK y una interfaz de línea de comandos que puede utilizar para crear y administrar instancias de cuadernos y entrenar e implementar modelos.

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

También puede obtener ejemplos de código en el GitHub repositorio de tutoriales de Amazon Braket.

- Tutoriales sobre frenos GitHub

Historial del documento

En la siguiente tabla se describe la documentación de esta versión de Amazon Braket.

- APIversión: 28 de abril de 2022
- Última actualización API de referencia: 25 de septiembre de 2023
- Última actualización de la documentación: 22 de mayo de 2024

Cambio	Descripción	Fecha
Nuevo dispositivo IQM Garnet y región Europe North 1	Se ha añadido soporte para el dispositivo IQM Garnet. Un dispositivo de 20 qubits con una topología reticular cuadrada. Se ampliaron las regiones compatibles con Braket a Europe North 1 (Estocolmo).	22 de mayo de 2024
Publicada la desafinación local	Las capacidades experimentales ahora incluyen la función de desafinación local de la Aquila QPU. QuEra	11 de abril de 2024
Lanzamiento del gestor de inactividad de portátiles	Al crear una instancia de bloc de notas, active el administrador de inactividad y establezca un tiempo de inactividad para restablecer automáticamente la instancia de Braket Notebook.	27 de marzo de 2024
Reelaboración de la tabla de contenido	Se reorganizó el índice de Amazon Braket para cumplir con los requisitos de AWS la guía de estilo y mejorar el flujo	12 de diciembre de 2023

	de contenido para la experienc ia del cliente.	
Se ha lanzado Braket Direct	Se agregó compatibilidad con las funciones de Braket Direct, que incluyen: • Reservas • Asesoramiento de expertos • Capacidades experimen tales	27 de noviembre de 2023
Se ha actualizado Crear una instancia de bloc de notas Amazon Braket	Se actualizó la documenta ción para añadir información a fin de crear una instancia de bloc de notas para los clientes nuevos y existentes de Amazon Braket.	27 de noviembre de 2023
Se ha actualizado Traiga su propio contenedor (BYOC)	Se actualizó la documentación para añadir información sobre cuándo usar el BYOC, la receta para el BYOC y cómo ejecutar Braket Hybrid Jobs en el contenedor.	18 de octubre de 2023

Lanzamiento del decorador Hybrid Jobs	Ejecute su código local como un trabajo híbridoPágina agregada. Contiene ejemplos: • Cree un trabajo híbrido a partir del código Python local • Instalar paquetes y código fuente de Python adicionales • Guarde y cargue datos en una instancia de trabajo híbrida • Mejores prácticas para decoradores de trabajos híbridos	16 de octubre de 2023
Se ha añadido visibilidad a las colas	Se actualizó la documentación de la Guía del desarrollador para incluir queue depth yqueue 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
Estandariza la nomenclatura en la documentación	Se actualizó la documentación para cambiar cualquier instancia de «trabajo» a «trabajo híbrido» y «tarea» a «tarea cuántica»	11 de septiembre de 2023
Nuevo dispositivo IonQ Aria 2	Se agregó soporte para el IonQ Aria 2 dispositivo	8 de septiembre de 2023

Native Gates actualizado	Se actualizó la documentación para añadir información sobre el acceso programático a las puertas nativas desde Rigetti.	16 de agosto de 2023
Xanadusalida	Se actualizó la documenta ción para eliminar todos los Xanadu dispositivos	2 de junio de 2023
Nuevo dispositivo IonQ Aria	Se agregó soporte para el IonQ Aria dispositivo	16 de mayo de 2023
RigettiDispositivo retirado	Se interrumpió el soporte para Rigetti Aspen-M-2	2 de mayo de 2023
Información AmazonBra ketFullAccessde política actualizada	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 los viajes 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 agregó soporte para el Rigetti Aspen-M-3 dispositivo	17 de enero de 2023
Nueva función de gradiente adjunto	Se agregó información sobre la función de gradiente adjunto que ofrece SV1	7 de diciembre de 2022

Nueva función de biblioteca de algoritmos	Se agregó información sobre la biblioteca de algoritmo s Braket, que proporciona un catálogo de algoritmos cuánticos prediseñados	28 de noviembre de 2022
D-Wavesalida	Se actualizó la documentación para poder retirar todos los D-Wave dispositivos	17 de noviembre de 2022
Nuevo dispositivo QuEra Aquila	Se agregó soporte para el QuEra Aquila dispositivo	31 de octubre de 2022
Support para Braket Pulse	Se ha añadido soporte para Braket Pulse, que permite utilizar el control del pulso en cualquier dispositivo Rigetti OQC	20 de octubre de 2022
Support para puertas nativas de IonQ	Se agregó soporte para el conjunto de puertas nativo que ofrece el dispositivo IonQ	13 de septiembre de 2022
Nuevas cuotas de instancias	Se actualizaron las cuotas predeterminadas de las instancias de cómputo clásicas asociadas a los trabajos híbridos	22 de agosto de 2022
Nuevo panel de servicios	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 agregó soporte para el Rigetti Aspen-M-2 dispositivo	12 de agosto de 2022

Nuevas funciones de OpenQASM	Se han añadido funciones de OpenQASM compatibles con los simuladores locales (braket_sv y braket_dm)	4 de agosto de 2022
Nuevos procedimientos de seguimiento de costes	Se agregó la forma de 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 agregó soporte para el Xanadu Borealis dispositivo	2 de junio de 2022
Nuevos procedimientos de simplificación de la incorporación	Se agregó información sobre cómo funcionan los procedimientos de incorporación nuevos y simplificados	16 de mayo de 2022
Nuevo dispositivo D-Wave Advantage_system6.1	Se agregó soporte para el D-Wave Advantage_system6.1 dispositivo	12 de mayo de 2022
Support para simuladores embebidos	Se agregó cómo ejecutar simulaciones integradas con trabajos híbridos y cómo usar el simulador de PennyLane relámpagos	4 de mayo de 2022
AmazonBraketFullAccess - Política de acceso completo a Amazon Braket	Se agregaron ListAllMy Buckets permisos s3: para permitir a los usuarios ver e inspeccionar los depósitos creados y utilizados para Amazon Braket	31 de marzo de 2022

Support para OpenQASM	Se agregó 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 agregó soporte para 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 ha aumentado el número máximo de SV1 tareas DM1 y tareas simultáneas 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
RigettiDispositivo retirado	Se ha interrumpido el soporte para Rigetti Aspen-10 el dispositivo	20 de diciembre de 2021
Nuevo tipo de resultado	Tipo de resultado de matriz de densidad reducida compatible con DM1 dispositivos y simuladores de matriz de densidad local	20 de diciembre de 2021

Descripción de la política actualizada	Amazon Braket actualizó el ARN del rol para incluir la ruta. `servicerole/` Para obtener información sobre las actualizaciones de políticas, consulta la tabla Amazon Braket updates to AWS managed policies.	29 de noviembre de 2021
Amazon Braket Jobs	Guía de usuario para Amazon Braket Hybrid Jobs y añadidos API	29 de noviembre de 2021
Nuevo dispositivo Rigetti	Se ha agregado compatibi lidad para Rigetti Aspen-10	20 de noviembre de 2021
D-WaveDispositivo retirado	Se ha interrumpido el soporte para D-Wave QPU, Advantage _system1	4 de noviembre de 2021
Nuevo dispositivo D-Wave	Se agregó soporte para una D-Wave QPU adicional, Advantage_system4	5 de octubre de 2021
Nuevos simuladores de ruido	Se agregó soporte para 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 ha añadido soporte para PennyLane Amazon Braket	8 de diciembre de 2020
Nuevo simulador	Se agregó soporte para un simulador de red tensora (TN1), que permite circuitos más grandes	8 de diciembre de 2020

Agrupación de tareas	Braket admite la agrupación de tareas de los clientes	24 de noviembre de 2020
Asignación manual qubit	Braket admite la qubit asignación manual en el dispositivo Rigetti	24 de noviembre de 2020
Cuotas ajustables	Braket admite cuotas ajustable s de autoservicio para los recursos de sus tareas	30 de octubre de 2020
Support para PrivateLink	Puede configurar puntos finales de VPC privados para sus trabajos de Braket	30 de octubre de 2020
Admite etiquetas	Braket admite etiquetas API basadas en el recurso de tareas cuánticas	30 de octubre de 2020
Nuevo dispositivo D-Wave	Se agregó soporte para una D-Wave QPU adicional, Advantage_system1	29 de septiembre de 2020
Versión inicial	Publicación inicial de la documentación de Amazon Braket	12 de agosto de 2020

Glosario de AWS

Para obtener la AWS terminología más reciente, consulte el AWSglosario en la referencia del AWSglosario.

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.