aws

Guia do desenvolvedor

Amazon Braket

Amazon Braket: Guia do desenvolvedor

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o Amazon Braket? 1
Como funciona 3
Fluxo de tarefas quânticas do Amazon Braket 5
Processamento de dados por terceiros 6
Termos e conceitos do Amazon Braket 6
AWS terminologia e dicas para o Amazon Braket 11
Controle e economia de custos 12
Definindo limites de gastos para o Amazon Braket QPUs 12
Controle de custos quase em tempo real 16
Melhores práticas para redução de custos 18
Referências e repositórios de API 20
Repositórios principais 20
Plugins 21
Serviços e regiões compatíveis 21
Regiões e endpoints 25
Introdução 27
Habilitar o Amazon Braket 27
Pré-requisitos 28
Etapas para habilitar o Amazon Braket 28
Criar uma instância de caderno do Amazon Braket 29
(Avançado) Crie um caderno Braket usando CloudFormation 31
Etapa 1: criar um script de configuração do ciclo de vida da SageMaker IA 32
Etapa 2: criar a função do IAM assumida pela Amazon SageMaker AI 32
Etapa 3: criar uma instância de notebook de SageMaker IA com o prefixo `amazon-braket-` 34
Criar 35
Compilar seu primeiro circuito 35
Construindo seus primeiros algoritmos quânticos 40
Construir circuitos no SDK 41
Inspecionando o circuito 56
Lista de tipos de resultados 59
Obter aconselhamento especializado 64
(Avançado) Execute seus circuitos com o OpenQASM 3.0 65
O que é o OpenQASM 3.0? 66

Quando usar o OpenQASM 3.0 67
Como o OpenQASM 3.0 funciona 67
Pré-requisitos 67
Quais recursos do OpenQASM o Braket suporta? 68
Crie e envie um exemplo de tarefa quântica do OpenQASM 3.0 74
Suporte para OpenQASM em diferentes dispositivos Braket 76
Simular ruído 87
Reconexão Qubit 88
Compilação literal 89
O console do Braket 90
Recursos adicionais do 90
Gradientes de computação 90
Como medir qubits específicos 91
(Avançado) Explore as capacidades experimentais 92
Acesso ao desvio local em Aquila QuEra 93
Acesso a geometrias altas em Aquila QuEra 95
Acesso a geometrias estreitas em Aquila QuEra 96
Circuitos dinâmicos em dispositivos IQM 97
Controle de pulso (avançado) no Amazon Braket 100
Quadros 100
Portas 101
Formas de onda 101
Trabalhando com o Hello Pulse 103
Acessando portas nativas usando pulsos 106
Simulação hamiltoniana analógica (avançada) 108
Olá AHS: Execute sua primeira simulação hamiltoniana analógica 108
Envie um programa analógico usando Aquila QuEra 122
(Avançado) Trabalhando com o AWS Boto3 142
Ativar o cliente Amazon Braket Boto3 143
Configurar AWS CLI perfis para o Boto3 e o SDK do Braket 146
Testar 149
Envio de tarefas quânticas para simuladores 149
Simulador vetorial estadual local (`braket_sv`) 150
Simulador de matriz de densidade local (`braket_dm`) 151
Simulador AHS local (`braket_ahs`) 152
Simulador de vetores de estado (SV1) 152

Simulador de matriz de densidade (DM1) 153
Simulador de rede tensora (TN1) 154
Sobre simuladores incorporados 155
Comparar simuladores 156
Exemplo de tarefas quânticas no Amazon Braket 160
Testando uma tarefa quântica com o simulador local 166
Emulador de dispositivo quântico local 168
Benefícios da emulação local 168
Criar um emulador local 169
Executar 170
Enviando tarefas quânticas para QPUs 171
AQT 172
IonQ 173
IQM 174
Rigetti 174
QuEra 175
Exemplo: envio de uma tarefa quântica para uma QPU 176
Inspecionando circuitos compilados 179
Executar vários programas 180
Sobre o conjunto de programas e os custos 181
Sobre lotes e custos de tarefas quânticas 182
Tratamento por lotes de tarefas quânticas e PennyLane 182
Agrupamento de tarefas e circuitos parametrizados 183
Quando minha tarefa quântica será executada? 184
Janelas e status de disponibilidade da QPU 184
Visibilidade da fila 185
Configure notificações por e-mail ou SMS 186
(Avançado) Trabalhar com reservas 187
Como criar uma reserva 188
Executando tarefas quânticas durante uma reserva 189
Executando trabalhos híbridos durante uma reserva 193
O que acontece no final da sua reserva 194
Cancelar ou reagendar uma reserva existente 195
Técnicas (avançadas) de mitigação de erros 195
Técnicas de mitigação de erros em dispositivos IonQ 195
Amazon Braket Hybrid Jobs 198

Quando usar o Amazon Braket Hybrid Jobs 199
Executando um trabalho híbrido com o Amazon Braket Hybrid Jobs 200
Principais conceitos 202
Entradas 202
Saídas 203
Variáveis de ambiente 204
funções auxiliares 205
Pré-requisitos 205
Criar um trabalho híbrido 209
Criar e executar 210
Monitorar resultados 213
Salve seus resultados 215
Usando pontos de verificação 217
Execute seu código local como um trabalho híbrido 218
Usando a API com trabalhos híbridos 227
Crie e depure uma tarefa híbrida com o modo local 230
Cancelar um Hybrid Job 231
Personalizar seu Hybrid Job 233
Defina o ambiente para seu script de algoritmo 233
Usando hiperparâmetros 245
Configure sua instância de trabalho híbrida 247
Usar compilação paramétrica para acelerar trabalhos híbridos 250
(Avançado) PennyLane com Amazon Braket 252
Amazon Braket com PennyLane 253
Algoritmos híbridos em cadernos de exemplo do Amazon Braket 254
Algoritmos híbridos com PennyLane simuladores incorporados 255
Gradiente adjunto PennyLane com simuladores Amazon Braket 255
Usando trabalhos híbridos e PennyLane para executar um algoritmo QAOA 256
Execute cargas de trabalho híbridas com PennyLane simuladores incorporados 259
CUDA-Q (Avançado) com Amazon Braket 265
CUDA-Q em NBIs 266
CUDA-Q no Hybrid Jobs 266
Solução de problemas 270
AccessDeniedException 270
Ocorreu um erro (ValidationException) ao chamar a CreateQuantumTask operação 270
Um recurso de SDK não funciona 271

O trabalho híbrido falha devido a ServiceQuotaExceededException 271
Os componentes pararam de funcionar na instância do notebook 272
Solução de problemas da atualização do Python 3.12 272
Visão geral do 272
Mensagens de erro comuns 273
Notebooks gerenciados com suporte 273
Decorador de empregos híbrido 274
Bring-Your-Own-Container (BYOC) 275
Atualização da instância do Braket Notebook 276
Solução de problemas do OpenQASM 276
Incluir erro de declaração 277
Erro não contíguo do qubits 277
Erro de mistura do qubits físico com o qubits virtual 278
Erro de solicitação de tipos de resultados que o qubits mede no mesmo programa 278
Os limites clássicos e de registro qubit excederam o erro 278
Caixa não precedida por um erro de pragma literal 279
Erro de caixas textuais sem portas nativas 279
Caixas textuais sem erro qubits físico 279
O pragma literal não contém o erro “braket” 280
Erro único qubits não pode ser indexado 280
O qubits físico em um erro qubit de duas portas não está conectado 280
Aviso de suporte do simulador local 281
Segurança 282
Responsabilidade compartilhada pela segurança 283
Proteção de dados 283
Retenção de dados 284
Gerenciamento do acesso ao Amazon Braket 284
Recursos do Amazon Braket 285
Cadernos e perfis 285
AWS políticas gerenciadas 287
Restringir o acesso do usuário a determinados dispositivos 291
Restrinja o acesso do usuário a determinadas instâncias do caderno 293
Restringir o acesso do usuário a determinados buckets do S3 294
Perfil vinculado a serviço 295
Validação de compatibilidade 296
Segurança da infraestrutura 296

Segurança de terceiros 297
Endpoints da VPC (PrivateLink) 297
Considerações sobre endpoints da VPC do Amazon Braket 298
Configure o Braket e PrivateLink 298
Informações adicionais sobre como criar um endpoint 300
Controle o acesso com as políticas de endpoint da VPC Amazon 300
Registro em log e monitoramento 302
Rastreamento de tarefas quânticas a partir do Amazon Braket SDK 303
Monitoramento de tarefas quânticas por meio do console Amazon Braket 305
Marcar recursos 307
Usar tags 308
Recursos compatíveis para marcação no Amazon Braket 309
Marcar com o Amazon Braket API 309
Restrições de marcação 309
Gerenciamento de tags no Amazon Braket 310
Exemplo de AWS CLI marcação no Amazon Braket 311
Monitorando suas tarefas quânticas com EventBridge 312
Monitore o status da tarefa quântica com EventBridge 313
Exemplo de evento Amazon Braket EventBridge 314
Monitorando suas métricas com CloudWatch 315
Métricas e dimensões do Amazon Braket 316
Registrando suas tarefas quânticas com CloudTrail 316
Informações sobre o Amazon Braket em CloudTrail 317
Noções básicas sobre entradas de arquivos de log do Amazon Braket 318
Registro (avançado) 320
Cotas 323
Cotas e limites adicionais 364
Histórico do documento 365
..... ccclxxix

O que é o Amazon Braket?

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

O Amazon Braket é totalmente AWS service (Serviço da AWS) gerenciado que ajuda pesquisadores, cientistas e desenvolvedores a começarem a usar a computação quântica. A computação quântica tem o potencial de resolver problemas computacionais que estão além do alcance dos computadores clássicos porque aproveita as leis da mecânica quântica para processar informações de novas maneiras.

Obter acesso ao hardware de computação quântica pode ser caro e inconveniente. O acesso limitado dificulta a execução de algoritmos, a otimização de projetos, a avaliação do estado atual da tecnologia e o planejamento de quando investir seus recursos para obter o máximo benefício. O Braket ajuda você a superar esses desafios.

O Braket oferece um único ponto de acesso a uma variedade de tecnologias de computação quântica. Com o Braket, você pode:

- Explorar e projetar algoritmos quânticos e híbridos.
- Testar algoritmos de teste em diferentes simuladores de circuitos quânticos.
- Executar algoritmos em diferentes tipos de computadores quânticos.
- Criar aplicativos de prova de conceito.

Definir problemas quânticos e programar computadores quânticos para resolvê-los requer um novo conjunto de habilidades. Para ajudá-lo a adquirir essas habilidades, o Braket oferece diferentes ambientes para simular e executar seus algoritmos quânticos. Você pode encontrar a abordagem que melhor atende às suas necessidades e começar rapidamente com um conjunto de ambientes de exemplo chamado cadernos.

O desenvolvimento do braket tem três estágios:

- Build - Braket fornece ambientes de cadernos Jupyter totalmente gerenciados que facilitam o início. Os cadernos Braket são pré-instalados com exemplos de algoritmos, recursos e ferramentas para desenvolvedores, incluindo o Amazon Braket SDK. Com o Amazon Braket SDK, você pode criar algoritmos quânticos e depois testá-los e executá-los em diferentes computadores e simuladores quânticos alterando uma única linha de código.
- Teste - O Braket fornece acesso a simuladores de circuitos quânticos totalmente gerenciados e de alto desempenho. Você pode testar e validar seus circuitos. O Braket lida com todos os componentes de software subjacentes e com os clusters do Amazon Elastic Compute Cloud (Amazon EC2) para eliminar a carga de simular circuitos quânticos na infraestrutura clássica de computação de alta performance (HPC).
- Executar - Braket fornece acesso seguro e sob demanda a diferentes tipos de computadores quânticos. Você tem acesso a computadores quânticos baseados em portas deAQT,,, e IonQ IQMRigetti, bem como a um simulador hamiltoniano analógico de. QuEra Você também não tem nenhum compromisso inicial e não precisa obter acesso por meio de fornecedores individuais.

Sobre computação quântica e Braket

A computação quântica está em seu estágio inicial de desenvolvimento. É importante entender que nenhum computador quântico universal e tolerante a falhas existe atualmente. Portanto, certos tipos de hardware quântico são mais adequados para cada caso de uso e é crucial ter acesso a uma variedade de hardware de computação. Braket oferece uma variedade de hardware por meio de fornecedores terceirizados.

O hardware quântico existente é limitado devido ao ruído, que introduz erros. O setor está na era Noisy Intermediate Scale Quantum (NISQ). Na era do NISQ, os dispositivos de computação quântica são muito barulhentos para sustentar algoritmos quânticos puros, como o algoritmo de Shor ou o algoritmo de Grover. Até que uma melhor correção de erros quânticos esteja disponível, a computação quântica mais prática requer a combinação de recursos de computação clássicos (tradicionais) com computadores quânticos para criar algoritmos híbridos. O Braket ajuda você a trabalhar com algoritmos quânticos híbridos.

Em algoritmos quânticos híbridos, as unidades de processamento quântico (QPUs) são usadas como coprocessadores CPUs, acelerando assim cálculos específicos em um algoritmo clássico. Esses algoritmos utilizam processamento iterativo, no qual a computação se move entre computadores clássicos e quânticos. Por exemplo, as aplicações atuais da computação quântica em química, otimização e aprendizado de máquina são baseadas em algoritmos quânticos variacionais, que são um tipo de algoritmo quântico híbrido. Em algoritmos quânticos variacionais, as rotinas clássicas

de otimização ajustam os parâmetros de um circuito quântico parametrizado de forma iterativa, da mesma forma que os pesos de uma rede neural são ajustados iterativamente com base no erro em um conjunto de treinamento de aprendizado de máquina. O Braket oferece acesso à biblioteca de software de código PennyLane aberto, que ajuda você com algoritmos quânticos variacionais.

A computação quântica está ganhando força para cálculos em quatro áreas principais:

- Teoria dos números — incluindo fatoração e criptografia (por exemplo, o algoritmo de Shor é um método quântico primário para cálculos da teoria dos números)
- Otimização — incluindo satisfação com restrições, solução de sistemas lineares e aprendizado de máquina
- Computação oracular — incluindo pesquisa, subgrupos ocultos e localização de pedidos (por exemplo, o algoritmo de Grover é um método quântico primário para cálculos oraculares)
- Simulação — incluindo simulação direta, invariantes de nós e aplicativos de algoritmo de otimização quântica aproximada (QAOA)

Os aplicativos para essas categorias de cálculos podem ser encontrados em serviços financeiros, biotecnologia, manufatura e produtos farmacêuticos, para citar alguns. O Braket oferece recursos e exemplos de cadernos que já podem ser aplicados a muitos problemas de prova de conceito, além de certos problemas práticos.

Nesta seção:

- Como o Amazon Braket funciona
- Termos e conceitos do Amazon Braket
- Controle e economia de custos
- Referências e repositórios de API para Amazon Braket
- Regiões e dispositivos compatíveis com o Amazon Braket

Como o Amazon Braket funciona

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

O Amazon Braket fornece acesso sob demanda a dispositivos de computação quântica, incluindo simuladores de circuito sob demanda e diferentes tipos de unidades de processamento quântico (). QPUs No Amazon Braket, a solicitação atômica para um dispositivo é uma tarefa quântica. Para dispositivos baseados em portas, essa solicitação inclui o circuito quântico (incluindo as instruções de medição e o número de disparos) e outros metadados da solicitação. Para simuladores hamiltonianos analógicos, a tarefa quântica contém o layout físico do registro quântico e a dependência temporal e espacial dos campos manipuladores.

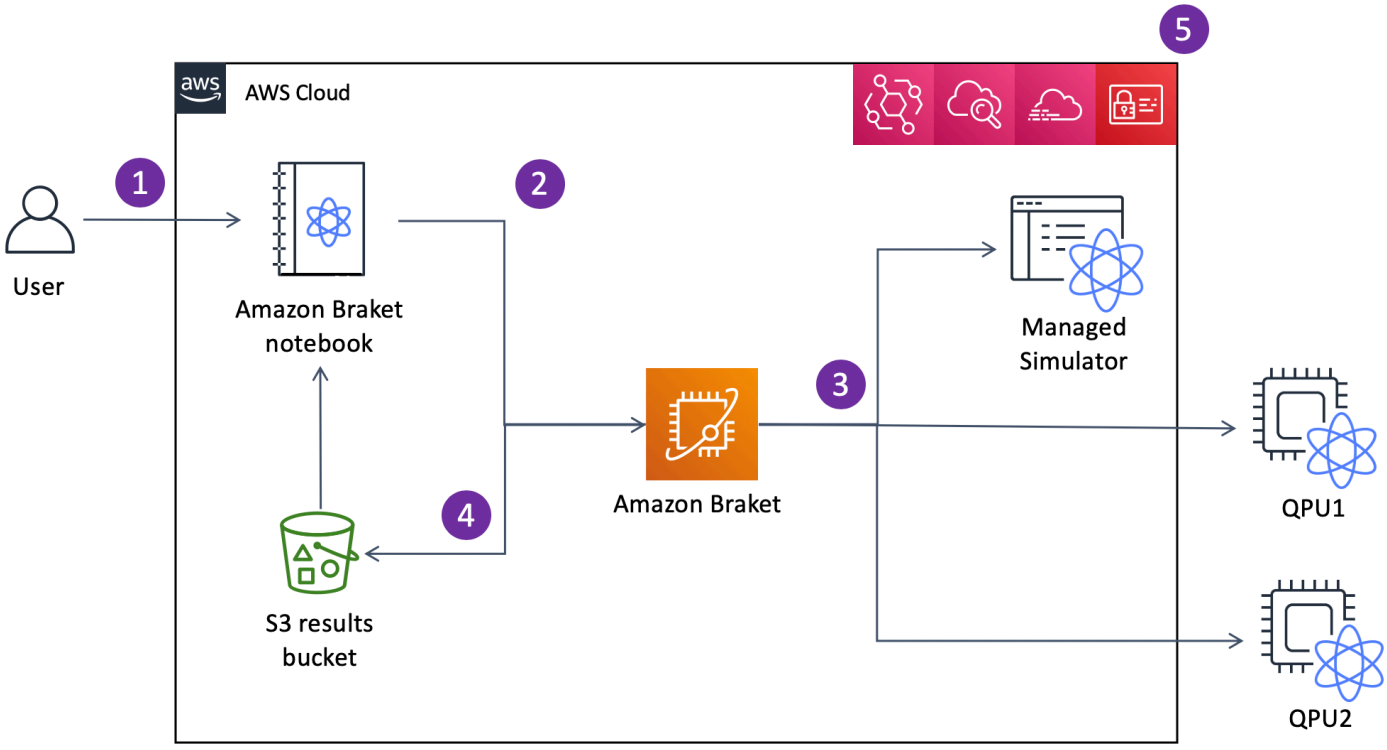
O Braket Direct é um programa que expande a forma como você pode explorar a computação quântica AWS, acelerando a pesquisa e a inovação. Você pode reservar capacidade dedicada em vários dispositivos quânticos, interagir diretamente com especialistas em computação quântica e ter acesso antecipado aos recursos da próxima geração, incluindo o mais recente dispositivo de íons presos da IonQ, Forte.

Nesta seção, aprenderemos sobre o fluxo de alto nível da execução de tarefas quânticas no Amazon Braket.

Nesta seção:

- Fluxo de tarefas quânticas do Amazon Braket
- Processamento de dados por terceiros

Fluxo de tarefas quânticas do Amazon Braket



Com Jupyter notebooks, você pode definir, enviar e monitorar suas tarefas quânticas a partir do Amazon Braket Console ou usando o Amazon Braket SDK. Você pode criar seus circuitos quânticos diretamente no SDK. No entanto, para simuladores hamiltonianos analógicos, você define o layout do registro e os campos de controle (1). Depois que sua tarefa quântica for definida, você poderá escolher um dispositivo para executá-la e enviá-la para a API Amazon Braket (2). Dependendo do dispositivo escolhido, a tarefa quântica é colocada em fila até que o dispositivo fique disponível e a tarefa seja enviada para a QPU ou simulador para implementação (3). O Amazon Braket oferece acesso a uma variedade de dispositivos quânticos compatíveis, QPUs incluindo simuladores sob demanda, simuladores locais e um simulador incorporado.

Depois de processar sua tarefa quântica, o Amazon Braket retorna os resultados para um bucket do Amazon S3, onde os dados são armazenados em seu (4). Conta da AWS Ao mesmo tempo, o SDK pesquisa os resultados em segundo plano e os carrega no caderno Jupyter na conclusão da tarefa quântica. Você também pode visualizar e gerenciar suas tarefas quânticas na página Quantum Tasks no console do Amazon Braket ou usando `GetQuantumTask` a operação do Amazon Braket. API

O Amazon Braket é integrado AWS Identity and Access Management com (IAM) CloudWatch, Amazon AWS CloudTrail e EventBridge Amazon para gerenciamento de acesso de usuários, monitoramento e registro, bem como para processamento baseado em eventos (5).

Processamento de dados por terceiros

As tarefas quânticas enviadas a um dispositivo QPU são processadas em computadores quânticos localizados em instalações operadas por fornecedores terceirizados. Para saber mais sobre segurança e processamento de terceiros no Amazon Braket, consulte Segurança dos fornecedores de hardware do Amazon Braket.

Termos e conceitos do Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

Os seguintes conceitos e termos são usados no Braket:

Simulação hamiltoniana analógica

A Simulação Hamiltoniana Analógica (AHS) é um paradigma de computação quântica distinto para simulação direta da dinâmica quântica dependente do tempo de sistemas de muitos corpos. Na AHS, os usuários especificam diretamente um hamiltoniano dependente do tempo e o computador quântico é ajustado de forma que emule diretamente a evolução contínua do tempo sob esse hamiltoniano. Os dispositivos AHS são normalmente dispositivos para fins especiais e não computadores quânticos universais, como dispositivos baseados em portas. Eles estão limitados a uma classe de hamiltonianos que eles podem simular. No entanto, como esses hamiltonianos são implementados naturalmente no dispositivo, o AHS não sofre com a sobrecarga necessária para formular algoritmos como circuitos e implementar operações de portas.

Braket

Chamamos o serviço Braket em homenagem à notação bra-ket, uma notação padrão na mecânica quântica. Foi introduzido por Paul Dirac em 1939 para descrever o estado dos sistemas quânticos e também é conhecido como notação de Dirac.

Braket Direct

Com o Braket Direct, você pode reservar acesso dedicado a diferentes dispositivos quânticos de sua escolha, conectar-se com especialistas em computação quântica para receber orientação sobre sua workload e obter acesso antecipado aos recursos da próxima geração, como novos dispositivos quânticos com disponibilidade limitada.

Trabalho híbrido Braket

O Amazon Braket tem um recurso chamado Amazon Braket Hybrid Jobs que fornece execuções totalmente gerenciadas de algoritmos híbridos. Uma tarefa híbrida Braket consiste em três componentes:

1. A definição do seu algoritmo, que pode ser fornecida como um script, módulo Python ou contêiner Docker.
2. A instância de trabalho híbrida, baseada no Amazon EC2, na qual executar seu algoritmo. O padrão é uma instância ml.m5.xlarge.
3. O dispositivo quântico no qual executar as tarefas quânticas que fazem parte do seu algoritmo. Um único trabalho híbrido normalmente contém uma coleção de muitas tarefas quânticas.

Dispositivo

No Amazon Braket, um dispositivo é um backend que pode executar tarefas quânticas. Um dispositivo pode ser um QPU ou um simulador de circuito quântico. Para saber mais, consulte Dispositivos compatíveis com o Amazon Braket.

Mitigação de erros

A mitigação de erros envolve a execução de vários circuitos físicos e a combinação de suas medições para obter um resultado aprimorado. Para obter mais informações, consulte Técnicas de mitigação de erros.

Computação quântica baseada em portas

Na computação quântica baseada em portas (QC), também chamada de QC baseada em circuitos, os cálculos são divididos em operações elementares (portas). Certos conjuntos de portas são universais, o que significa que cada cálculo pode ser expresso como uma sequência

finita dessas portas. As portas são os blocos de construção dos circuitos quânticos e são análogas às portas lógicas dos circuitos digitais clássicos.

Limite de Gateshot

Um limite de captura de porta se refere à contagem total de portas por disparo (a soma de todos os tipos de portas) e à contagem de disparos por tarefa. Matematicamente, o limite do gateshot pode ser expresso como:

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

Hamiltoniano

A dinâmica quântica de um sistema físico é determinada por seu hamiltoniano, que codifica todas as informações sobre as interações entre os constituintes do sistema e os efeitos das forças motrizes exógenas. O hamiltoniano de um sistema N-qubit é comumente representado como uma matriz 2^N por 2^N de números complexos em máquinas clássicas. Ao executar uma simulação hamiltoniana analógica em um dispositivo quântico, você pode evitar esses requisitos exponenciais de recursos.

Pulso

Um pulso é um sinal físico transitório transmitido aos qubits. É descrito por uma forma de onda reproduzida em um quadro que serve como suporte para o sinal da portadora e está vinculado ao canal ou porta do hardware. Os clientes podem projetar seus próprios pulsos fornecendo o envelope analógico que modula o sinal portador sinusoidal de alta frequência. O quadro é descrito exclusivamente por uma frequência e uma fase que geralmente são escolhidas para estarem em ressonância com a separação de energia entre os níveis de energia de |0# e |1# do qubit. As portas são, portanto, acionadas como pulsos com uma forma predeterminada e parâmetros calibrados, como amplitude, frequência e duração. Os casos de uso que não são cobertos pelas formas de onda do modelo serão habilitados por meio de formas de onda personalizadas, que serão especificadas na resolução de uma única amostra, fornecendo uma lista de valores separados por um tempo de ciclo físico fixo.

Circuito quântico

Um circuito quântico é o conjunto de instruções que define uma computação em um computador quântico baseado em portas. Um circuito quântico é uma sequência de portas quânticas, que são transformações reversíveis em um qubit registro, junto com instruções de medição.

Simulador de circuito quântico

Um simulador de circuito quântico é um programa de computador executado em computadores clássicos e calcula os resultados da medição de um circuito quântico. Para circuitos gerais, os requisitos de recursos de uma simulação quântica crescem exponencialmente com o número de qubits a serem simuladas. O Braket oferece acesso a simuladores de circuitos quânticos gerenciados (acessados através da API do Braket) e locais (parte do SDK do Amazon Braket).

Computação quântica

Um computador quântico é um dispositivo físico que usa fenômenos da mecânica quântica, como superposição e emaranhamento, para realizar cálculos. Existem diferentes paradigmas para a computação quântica (QC), como o QC baseado em portas.

Unidade de processamento quântico (QPU)

Um QPU é um dispositivo físico de computação quântica que pode ser executado em uma tarefa quântica. QPUs pode ser baseado em diferentes paradigmas de controle de qualidade, como controle de qualidade baseado em portas. Para saber mais, consulte Dispositivos compatíveis com o Amazon Braket.

Portões nativos da QPU

As portas nativas da QPU podem ser mapeadas diretamente para controlar os pulsos pelo sistema de controle da QPU. As portas nativas podem ser executadas no dispositivo QPU sem compilação adicional. Subconjunto de portas compatíveis com QPU. Você pode encontrar as portas nativas de um dispositivo na página Dispositivos no console do Amazon Braket e por meio do SDK do Braket.

Portas compatíveis com QPU

As portas compatíveis com QPU são as aceitas pelo dispositivo QPU. Essas portas podem não funcionar diretamente na QPU, o que significa que talvez precisem ser decompostas em portas nativas. Você pode encontrar as portas suportadas de um dispositivo na página Dispositivos no console do Amazon Braket e por meio do SDK do Amazon Braket.

Tarefa quântica

No Braket, uma tarefa quântica é a solicitação atômica a um dispositivo. Para dispositivos de controle de qualidade baseados em portas, isso inclui o circuito quântico (incluindo as instruções de medição e o número de shots) e outros metadados de solicitação. Você pode criar tarefas quânticas por meio do Amazon Braket SDK ou usando a operação CreateQuantumTask API

diretamente. Depois de criar uma tarefa quântica, ela ficará na fila até que o dispositivo solicitado fique disponível. Você pode visualizar suas tarefas quânticas na página Tarefas quânticas do console Amazon Braket ou usando as operações GetQuantumTask ou SearchQuantumTasks API.

Qubit

A unidade básica de informação em um computador quântico é chamada de qubit (bit quântico), assim como um bit na computação clássica. Um qubit é um sistema quântico de dois níveis que pode ser realizado por diferentes implementações físicas, como circuitos supercondutores ou íons e átomos individuais. Outros tipos de qubit são baseados em fótons, spins eletrônicos ou nucleares ou sistemas quânticos mais exóticos.

Queue depth

Queue depth refere-se ao número de tarefas quânticas e trabalhos híbridos em fila para um determinado dispositivo. As tarefas quânticas e a contagem de filas de tarefas híbridas de um dispositivo podem ser acessadas por meio do Braket Software Development Kit (SDK) ou Amazon Braket Management Console.

1. A profundidade da fila de tarefas se refere ao número total de tarefas quânticas esperando para serem executadas em prioridade normal.
2. A profundidade da fila de tarefas prioritárias se refere ao número total de tarefas quânticas enviadas aguardando execução do Amazon Braket Hybrid Jobs. Essas tarefas têm prioridade sobre as tarefas autônomas quando um trabalho híbrido é iniciado.
3. A profundidade da fila de trabalhos híbridos se refere ao número total de trabalhos híbridos atualmente em fila em um dispositivo. Quantum tasks enviadas como parte de um trabalho híbrido têm prioridade e são agregadas à Priority Task Queue.

Queue position

Queue position refere-se à posição atual de sua tarefa quântica ou trabalho híbrido em uma respectiva fila de dispositivos. Ele pode ser obtido para tarefas quânticas ou trabalhos híbridos por meio do Braket Software Development Kit (SDK) ou Amazon Braket Management Console.

Shots

Como a computação quântica é inerentemente probabilística, qualquer circuito precisa ser avaliado várias vezes para obter um resultado preciso. A execução e medição de um único circuito são chamadas de disparo. O número de disparos (execuções repetidas) para um circuito é escolhido com base na precisão desejada para o resultado.

AWS terminologia e dicas para o Amazon Braket

Políticas do IAM

Uma política do IAM é um documento que permite ou nega permissões a recursos e Serviços da AWS . As políticas do IAM permitem que você personalize os níveis de acesso dos usuários aos recursos. Por exemplo, você pode permitir que os usuários acessem todos os buckets do Amazon S3 dentro do seu Conta da AWS, ou somente um bucket específico.

- Prática recomendada: siga o princípio de segurança do privilégio mínimo ao conceder permissões. Ao seguir esse princípio, você ajuda a evitar que usuários ou funções tenham mais permissões do que o necessário para realizar suas tarefas quânticas. Por exemplo, se um funcionário precisar acessar somente um bucket específico, especifique-o na política do IAM em vez de conceder ao funcionário acesso a todos os buckets de sua Conta da AWS.

Funções do IAM

Um perfil do IAM é uma identidade que você pode assumir para obter acesso temporário às permissões. Para que um usuário, uma aplicação ou um serviço possa usar um perfil do IAM que você criou, você deverá conceder permissões para alternar para esse perfil. Quando alguém assume um perfil do IAM, abandona todas as permissões anteriores que tinha em uma função anterior e assume as permissões da nova função.

- Prática recomendada: os perfis do IAM são ideais para situações em que o acesso a serviços ou recursos precisa ser concedido temporariamente, em vez de a longo prazo.

Bucket do Amazon S

O Amazon Simple Storage Service (Amazon S3) permite armazenar dados como objetos em buckets. AWS service (Serviço da AWS) Os buckets Amazon S3 oferecem espaço de armazenamento ilimitado. O tamanho máximo de um objeto em um bucket do Amazon S3 é de 5 TB. Você pode fazer upload de qualquer tipo de dados de arquivo para um bucket do Amazon S3, como imagens, vídeos, arquivos de texto, arquivos de backup, arquivos de mídia para um site, documentos arquivados e os resultados da tarefa quântica do Braket.

- Prática recomendada: você pode definir permissões para controlar o acesso ao seu bucket do S3. Para obter mais informações, consulte Políticas de bucket na documentação do Amazon S3.

Controle e economia de custos

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

Com o Amazon Braket, você tem acesso a recursos de computação quântica sob demanda, sem compromisso prévio. Você paga somente pelo que usar. Para saber mais sobre precificação, consulte página de precificação.

Nesta seção:

- Definindo limites de gastos para o Amazon Braket QPUs
- Controle de custos quase em tempo real
- Melhores práticas para redução de custos

Definindo limites de gastos para o Amazon Braket QPUs

Os limites de gastos do Amazon Braket fornecem controles opcionais de custo por dispositivo para unidades de processamento quântico (). QPUs

Como os limites de gastos funcionam: o Amazon Braket monitora seus gastos cumulativos e valida cada solicitação de criação de tarefas em relação ao seu limite configurado. Se o custo estimado de uma tarefa exceder seu limite de gastos restante, o Amazon Braket rejeitará a tarefa imediatamente com um erro de validação. Opcionalmente, você pode configurar um período de tempo para seu limite de gastos. Ao configurar um período de tempo, você pode garantir que as tarefas só possam ser enviadas nesse período especificado. As tarefas enviadas fora do período de tempo serão rejeitadas.

Design opcional: os fluxos de trabalho existentes permanecerão inalterados, a menos que você habilite explicitamente os controles. Você pode remover todas as restrições excluindo o limite de gastos.

> **Note**
>
> Os limites de gastos se aplicam somente às tarefas de QPU de trabalho híbrido e sob demanda. Eles excluem simuladores, notebooks gerenciados, custos de instância do Hybrid Job EC2 e reservas do Braket Direct. Para um gerenciamento abrangente de custos em todos os serviços da AWS, continue usando AWS Budgets.

Lista de ações de limite de gastos

Pesquisar

Com o seguinte comando da AWS CLI, você pode pesquisar e listar os limites de gastos em uma região específica da AWS e para um dispositivo Braket específico.

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

Criar

Com o seguinte comando da AWS CLI, você pode criar um novo limite de gastos para um dispositivo quântico específico em uma região específica. A solicitação será rejeitada se já existir um limite de gastos para o dispositivo.

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

Atualizar

Com o seguinte comando da AWS CLI, você pode atualizar um limite de gastos existente para um novo valor máximo de gastos. A solicitação será rejeitada se a soma do gasto atual e do gasto em fila já for maior do que o novo gasto máximo solicitado.

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

Você pode fornecer um período de tempo em vez ou além do novo gasto máximo, como no exemplo acima.

Excluir

Com o seguinte comando da AWS CLI, você pode excluir um limite de gastos existente.

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

Você pode fornecer um período de tempo em vez ou além do novo gasto máximo, como no exemplo acima.

Embora opcional, sempre especifique o parâmetro da região como uma prática recomendada. Os comandos executados em uma região diferente da do dispositivo falharão ou, no caso de`SearchSpendingLimits`, retornarão resultados incorretos.

Para obter mais exemplos sobre como usar limites de gastos, consulte o exemplo de caderno.

Como funciona a validação de tarefas

Quando a conta da AWS envia uma `CreateQuantumTask` solicitação válida, ela está sujeita ao seguinte comportamento de bloqueio. Observação: o orçamento restante é a diferença entre o limite de gastos e a soma dos gastos atuais e em fila. (Veja a próxima seção)

- Caso 1: Não há limite de gastos para o dispositivo de tarefas: a tarefa foi criada.
- Caso 2: Há um limite de gastos para o dispositivo de destino e o horário atual está dentro do período de tempo do limite de gastos:
  - Se o custo estimado da tarefa for menor ou igual ao orçamento restante: CreateQuantumTask for bem-sucedido, a tarefa será criada.
  - Se o custo estimado for maior que o orçamento restante: `CreateQuantumTask` falha e nenhuma tarefa será criada.
- Caso 3: Há um limite de gastos para o dispositivo de destino e o horário atual está fora do período do limite de gastos: `CreateQuantumTask` falha e nenhuma tarefa é criada.

Como o orçamento restante é calculado

O orçamento restante é a diferença entre o limite de gastos e a soma dos gastos atuais e dos gastos em fila.

Quando uma tarefa é criada para um dispositivo com limite de gastos, o gasto na fila é aumentado pelo custo estimado da tarefa. Esse evento está listado na primeira linha da tabela a seguir. A tabela a seguir mostra o que acontece com o gasto na fila e o gasto atual, dependendo da progressão da tarefa.

Antigo estado de tarefa quântica	Novo estado de tarefa quântica	Alterar para gastos em fila	Alteração nos gastos atuais
-	CREATED	Aumentado pelo custo estimado	Nenhuma alteração
CREATED	QUEUED	Nenhuma alteração	Nenhuma alteração
Any	RUNNING (Em execução)	Nenhuma alteração	Nenhuma alteração
Any	CANCELANDO	Nenhuma alteração	Nenhuma alteração
CANCELANDO	CANCELADO	Reduzido pelo custo estimado	Sem alteração
Any	FAILED	Reduzido pelo custo estimado	Nenhuma alteração
RUNNING (Em execução)	CONCLUÍDO	Reduzido pelo custo estimado	Aumentado pelo custo estimado (ajustado adequadamente para tarefas parcialmente concluídas)

Casos Edge

P: Ao criar um limite de gastos, as tarefas que já estão na fila contam para o gasto na fila?

R: Não. Tarefas que já foram criadas, colocadas em fila ou em andamento não contam para os gastos em fila de um limite de gastos recém-criado.

P: Reduzir o limite de gastos, ao atualizá-lo, causa o encerramento antecipado de uma tarefa quântica criada, em fila ou em andamento?

R: Não.

P: Atingir a hora de término do limite de gastos causa o encerramento antecipado de uma tarefa quântica criada, em fila ou em andamento?

R: Não. Tarefas criadas, enfileiradas e em andamento podem ser concluídas independentemente do status do limite de gastos.

P: Como a falta de limite de gastos é diferente de um limite de gastos de zero dólares?

R: Nenhum limite de gastos permite criar tarefas quânticas sem restrições. Um limite de gastos com zero dólares bloqueia todas as tarefas quânticas.

P: Um limite de gastos de zero no passado ou no futuro bloqueia toda a criação de tarefas quânticas?

R: Sim.

P: Ao criar um limite de gastos, o custo estimado das tarefas que já estão na fila será contabilizado no gasto atual quando essas tarefas forem concluídas?

R: Não. Somente as tarefas enviadas enquanto o limite de gastos estiver ativo contam para o gasto acumulado.

Controle de custos quase em tempo real

O Braket SDK oferece a opção de adicionar um rastreamento de custos quase em tempo real às suas workloads quânticas. Cada um de nossos notebooks de exemplo inclui um código de rastreamento de custos para fornecer uma estimativa máxima de custo nas unidades de processamento quântico (QPUs) e nos simuladores sob demanda da Braket. As estimativas de custo máximo serão mostradas em USD e não incluem créditos ou descontos.

> **Note**
>
> As cobranças mostradas são estimativas com base no uso de tarefas do simulador Amazon Braket e da unidade de processamento quântico (QPU). As cobranças estimadas mostradas podem diferir das cobranças reais. As cobranças estimadas não incluem descontos ou créditos, e você pode receber cobranças adicionais com base no uso de outros serviços, como o Amazon Elastic Compute Cloud (Amazon EC2).

Rastreamento de custos para SV1

Para demonstrar como a função de controle de custos pode ser usada, construiremos um circuito Bell State e o executaremos em nosso SV1 simulador. Comece importando os módulos do SDK do Braket, definindo um estado de sino e adicionando a `Tracker()` função ao nosso circuito:

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

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

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

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

Ao executar seu Notebook, você pode esperar a seguinte saída para sua simulação do Bell State. A função de rastreamento mostrará o número de fotos enviadas, as tarefas quânticas concluídas, a duração da execução, a duração da execução faturada e seu custo máximo em dólares americanos. Seu tempo de execução pode variar para cada simulação.

```
import datetime

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

tracker.simulator_tasks_cost()
```

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

Usando o rastreador de custos para definir os custos máximos

Você pode usar o rastreador de custos para definir os custos máximos em um programa. Você pode ter um limite máximo de quanto deseja gastar em um determinado programa. Dessa forma, você pode usar o rastreador de custos para criar uma lógica de controle de custos em seu código de execução. O exemplo a seguir usa o mesmo circuito em uma Rigetti QPU e limita o custo a 1 USD. O custo para executar uma iteração do circuito em nosso código é de 0,30 USD. Definimos a lógica para repetir as iterações até que o custo total exceda 1 USD; portanto, o trecho de código será executado três vezes até que a próxima iteração exceda 1 USD. Geralmente, um programa continuaria a iterar até atingir o custo máximo desejado, neste caso, três iterações.

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

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

> **Note**
>
> O rastreador de custos não rastreará a duração de tarefas TN1 quânticas que falharam. Durante uma simulação TN1, se seu ensaio for concluído, mas a etapa de contração falhar, sua taxa de ensaio não será mostrada no rastreador de custos.

Melhores práticas para redução de custos

Considere as práticas recomendadas a seguir para usar o Amazon Braket. Economize tempo, minimize os custos e evite erros comuns.

Verificação com simuladores

- Verifique seus circuitos usando um simulador antes de executá-lo em uma QPU, para que você possa ajustar seu circuito sem incorrer em cobranças pelo uso da QPU.
- Embora os resultados da execução do circuito em um simulador possam não ser idênticos aos resultados da execução do circuito em uma QPU, você pode identificar erros de codificação ou problemas de configuração usando um simulador.

Restringir o acesso do usuário a determinados dispositivos

- Você pode configurar restrições que impeçam usuários não autorizados de enviar tarefas quânticas em determinados dispositivos. O método recomendado para restringir o acesso é com o AWS IAM. Para obter mais informações sobre como fazer isso, consulte Restringir acesso.
- Não recomendamos que você use sua conta de administrador como forma de conceder ou restringir o acesso do usuário aos dispositivos Amazon Braket.

Configurar alarmes de faturamento

- Você pode definir um alarme de cobrança para receber uma notificação quando sua fatura atingir um limite predefinido. A forma recomendada de configurar um alarme é por meio de AWS Budgets. Você pode definir orçamentos personalizados e receber alertas quando seus custos ou uso excederem o valor orçado. As informações estão disponíveis no AWS Budgets.

Teste tarefas quânticas TN1 com baixa contagem de disparos

- Os simuladores custam menos do que QPUs, mas certos simuladores podem ser caros se as tarefas quânticas forem executadas com altas contagens de disparos. Recomendamos que você teste suas tarefas TN1 com uma contagem baixa de shot. A contagem de Shot não afeta o custo do SV1 e as tarefas locais do simulador.

Verifique todas as regiões para tarefas quânticas

- O console exibe tarefas quânticas somente para a sua atual Região da AWS. Ao procurar tarefas quânticas faturáveis que foram enviadas, certifique-se de verificar todas as regiões.
- Você pode ver uma lista de dispositivos e suas regiões associadas na página de documentação de dispositivos compatíveis.

Referências e repositórios de API para Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

O Amazon Braket APIs fornece SDKs,, e uma interface de linha de comando que você pode usar para criar e gerenciar instâncias de notebook e treinar e implantar modelos.

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

Você também pode obter exemplos de código no repositório Amazon GitHub Braket Tutorials.

- Tutoriais do Braket GitHub

Repositórios principais

A seguir, é exibida uma lista dos repositórios principais que contêm pacotes de chaves usados para o Braket:

- SDK Braket Python - Use o SDK Braket Python para configurar seu código em cadernos na linguagem de programação Python. Jupyter Depois que seus cadernos Jupyter estiverem configurados, você poderá executar seu código em dispositivos e simuladores Braket
- Esquemas do Braket - O contrato entre o SDK do Braket e o serviço Braket.
- Braket Default Simulator - Todos os nossos simuladores quânticos locais para Braket (vetor de estado e matriz de densidade).

Plugins

Depois, há os vários plug-ins que são usados junto com vários dispositivos e ferramentas de programação. Isso inclui plug-ins compatíveis com Braket, bem como plug-ins suportados por terceiros, conforme mostrado abaixo.

O Amazon Braket dá suporte a:

- Biblioteca de algoritmos Amazon Braket — Um catálogo de algoritmos quânticos pré-criados escritos em Python. Execute-os como estão ou use-os como ponto de partida para criar algoritmos mais complexos.
- Braket- PennyLane plugin - Use PennyLane como estrutura QML no Braket.

Terceiros (a equipe do Braket monitora e contribui):

- Provedor Qiskit-Braket - Use o SDK para acessar os recursos do Qiskit Braket.
- SDK do Braket-Julia - (EXPERIMENTAL) Uma versão nativa de Julia do SDK do Braket

Regiões e dispositivos compatíveis com o Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

No Amazon Braket, um dispositivo representa uma unidade de processamento quântico (QPU) ou simulador que você pode chamar para executar tarefas quânticas. O Amazon Braket fornece

acesso a dispositivos QPU AQT deIonQ,,, e. IQM QuEra Rigetti Além disso, AWS oferece acesso a simuladores sob demanda, locais e incorporados. Para obter mais informações sobre simuladores incorporados, consulte Sobre simuladores incorporados.

Para obter informações sobre fornecedores de hardware quântico compatíveis, consulte Enviando tarefas quânticas para QPUs. Para obter informações sobre simuladores disponíveis, consulte Envio de tarefas quânticas para simuladores. A tabela a seguir exibe a lista de dispositivos e simuladores disponíveis.

Fornecedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
AQT	IBEX-Q1	Baseada em porta	QPU	arn: aws: braket:eu-north-1:: -Q1 device/qpu/aqt/Ibex	eu-north-1
IonQ	Forte-1	Baseada em porta	QPU	arn: aws: braket:us-east-1:: -1 device/qpu/ionq/Forte	us-east-1
IonQ	Forte-Enterprise-1	Baseada em porta	QPU	arn: aws:braket:us-east-1:: -Enterprise-1 device/qpu/ionq/Forte	us-east-1
IQM	Garnet	Baseada em porta	QPU	arn: aws: braket:eu-north-1:: device/qpu/iqm/Garnet	eu-north-1
IQM	Emerald	Baseada em porta	QPU	arn: aws: braket:eu-north-1:: device/qpu/iqm/Emerald	eu-north-1
QuEra	Aquila	Simulação hamiltoniana analógica	QPU	arn: aws: braket:us-east-1:: device/qpu/quera/Aquila	us-east-1

Fornecedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
Rigetti	Ankaa-3	Baseada em porta	QPU	arn: aws: braket:us-west-1:: -3 device/qpu/rigetti/Ankaa	us-west-1
AWS	braket_sv	Baseada em porta	Simulador local	N/A (simulador local no Braket SDK)	N/D
AWS	braket_dm	Baseada em porta	Simulador local	N/A (simulador local no Braket SDK)	N/D
AWS	braket_ahs	Simulação hamiltoniana analógica	Simulador local	N/A (simulador local no Braket SDK)	N/D
AWS	SV1	Baseada em porta	Simulador sob demanda	arn:aws:braket::: 1 device/quantum-simulator/amazon/sv	Valores válidos: us-east-1 | us-west-1 | us-west-2 | eu-west-1
AWS	DM1	Baseada em porta	Simulador sob demanda	arn:aws:braket::: 1 device/quantum-simulator/amazon/dm	Valores válidos: us-east-1 | us-west-1 | us-west-2 | eu-west-1

Fornecedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
AWS	TN1	Baseada em porta	Simulador sob demanda	arn:aws:braket::: 1 device/quantum-simulator/amazon/tn	us-east-1, us-west-2 e eu-west-2 e eu-west-2

> **Note**
>
> ARNs Os dispositivos diferenciam maiúsculas de minúsculas Por exemplo, ao usar o AQT IBEX-Q1 dispositivo, verifique se o ARN do dispositivo contém. 'Ibex-Q1'

Para ver detalhes adicionais sobre o QPUs que você pode usar com o Amazon Braket, consulte Amazon Braket Quantum Computers.

Propriedades do dispositivo

Para todos os dispositivos, você pode encontrar outras propriedades do dispositivo, como topologia do dispositivo, dados de calibração e conjuntos de portas nativos, na guia Dispositivos do console do Amazon Braket ou pela API do `GetDevice`. Ao construir um circuito com os simuladores, o Amazon Braket exige que você use qubits ou índices contíguos. Ao trabalhar com o SDK, o exemplo de código a seguir mostra como obter acesso às propriedades do dispositivo para cada dispositivo e simulador disponíveis.

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

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

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

# Get device properties
device.properties
```

Regiões e endpoints para o Amazon Braket

Para obter uma lista completa de regiões e pontos finais, consulte a General Reference AWS.

As tarefas quânticas executadas em um dispositivo QPU podem ser visualizadas no console Amazon Braket na região desse dispositivo. Ao usar o Amazon Braket SDK, você pode enviar tarefas quânticas para qualquer dispositivo QPU, independentemente da região em que você está trabalhando. O SDK cria automaticamente uma sessão na região para a QPU especificada.

O Amazon Braket está disponível nas seguintes opções: Regiões da AWS

Nome da região	Região	Endpoints do Braket
Leste dos EUA (Norte da Virgínia)	us-east-1	braket.us-east-1.amazonaws.com (IPv4 somente) braket.us-east-1.api.aws (pilha dupla)
Oeste dos EUA (N. da Califórnia)	us-west-1	braket.us-west-1.amazonaws.com (IPv4 somente) braket.us-west-1.api.aws (pilha dupla)
US West 2 (Oregon)	us-west-2	braket.us-west-2.amazonaws.com (IPv4 somente) braket.us-west-2.api.aws (pilha dupla)
EU North 1 (Estocolmo)	eu-north-1	braket.eu-north-1.amazonaws.com (IPv4 somente) braket.eu-north-1.api.aws (pilha dupla)
eu-west-2 (Londres)	eu-west-2	braket.eu-west-2.amazonaws.com (IPv4 somente) braket.eu-west-2.api.aws (pilha dupla)

Note

O Amazon Braket SDK não oferece suporte somente a redes. IPv6

Conceitos básicos do Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

Depois de seguir as instruções em Ativar o Amazon Braket, você pode começar a usar o Amazon Braket.

As etapas para começar incluem:

- Habilitar o Amazon Braket
- Criar uma instância de caderno do Amazon Braket
- Crie uma instância do notebook Braket usando CloudFormation

Habilitar o Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

Você pode ativar o Amazon Braket em sua conta por meio do console AWS.

Nesta seção:

- Pré-requisitos
- Etapas para habilitar o Amazon Braket

Pré-requisitos

Para habilitar e executar o Amazon Braket, você deve ter um usuário ou perfil com permissão para iniciar ações do Amazon Braket. Essas permissões estão incluídas na política do AmazonBraketFullAccess IAM (arn:aws:iam: :aws:policy/). AmazonBraketFullAccess

> **Note**
>
> Se você for administrador:
> Para dar a outros usuários acesso ao Amazon Braket, conceda permissões aos usuários anexando AmazonBraketFullAccessa política ou anexando uma política personalizada criada por você. Para saber mais sobre as permissões necessárias para usar o Amazon Braket, consulte Gerenciando o acesso ao Amazon Braket.

Etapas para habilitar o Amazon Braket

1. Faça login no console Amazon Braket com seu. Conta da AWS
2. Abra o console do Amazon Braket.
3. Na página inicial do Braket, clique em Começar para acessar a página do Service Dashboard. O alerta na parte superior do seu painel de serviços guiará você pelas três etapas a seguir:
   a. Criar perfis vinculados ao serviço (SLR)
   b. Habilitar acesso a computadores quânticos de terceiros
   c. Criar uma nova instancia de caderno Jupyter

Para usar dispositivos quânticos de terceiros, você precisa concordar com certas condições relacionadas à transferência de dados entre você e esses dispositivos. AWS Os termos e condições deste contrato são fornecidos na guia Geral da página de permissões e configurações no console do Amazon Braket.

> **Note**
>
> Dispositivos quânticos que não envolvem terceiros, como os simuladores locais Braket ou simuladores sob demanda, podem ser usados sem concordar com o contrato Habilitar de dispositivos de terceiros.

A aceitação desses termos para permitir o uso de dispositivos de terceiros só precisa ser feita uma vez por conta se você estiver acessando hardware de terceiros.

Criar uma instância de caderno do Amazon Braket

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

O Amazon Braket fornece cadernos Jupyter totalmente gerenciados para você começar. As instâncias do notebook Amazon Braket são baseadas nas instâncias do notebook SageMaker Amazon AI. As etapas a seguir descrevem como criar uma nova instancia de caderno para clientes novos e existentes.

Novos clientes do Amazon Braket:

1. Abra o console do Amazon Braket e navegue até a página Painel no painel esquerdo.
2. Clique em Começar no modal Bem-vindo ao Amazon Braket no centro da página do seu painel. Forneça um nome de caderno para criar um caderno Jupyter padrão.
   a. A criação do caderno pode demorar vários minutos.
   b. Seu caderno será listado na página Cadernos com o status Pendente.
   c. Quando a instância do notebook estiver pronta para uso, o status mudará para InService.
   d. Atualize a página para exibir o status atualizado do caderno.

Clientes existentes do Amazon Braket:

1. Abra o console Amazon Braket e selecione Cadernos no painel esquerdo.
2. Selecione Criar instância do caderno.
   a. Se você não tiver cadernos, selecione a configuração Padrão para criar um caderno Jupyter padrão.
3. Insira um nome de instância do caderno, usando somente caracteres alfanuméricos e hífen, e selecione seu Modo visual preferido.

4. Ative ou desative o gerenciador de inatividade do Caderno para seu caderno.
   a. Se ativado, selecione o tempo de inatividade desejado antes que o caderno seja reiniciado. Quando um caderno é reiniciado, as cobranças de computação param de ser cobradas, mas as cobranças de armazenamento continuarão.
   b. Para verificar quanto tempo ocioso resta para sua instância do caderno, navegue até a barra de comando, selecione a guia Braket e, em seguida, a guia Gerenciador de inatividade.

> **Note**
>
> Para salvar seu trabalho, integre sua instância de notebook de SageMaker IA a um repositório Git. Como alternativa, mova seu trabalho para fora dos `/Braket Examples` e pastas `/Braket Algorithms` para que não sejam substituídos pela reinicialização da instância do caderno.

5. (Opcional) Com a Configuração avançada, você pode criar um caderno com permissões de acesso, configurações adicionais e configurações de acesso à rede:
   a. Na configuração do Caderno, escolha seu tipo de instância.
      i. O tipo de instância padrão e econômico, ml.t3.medium, é escolhido por padrão. Para saber mais sobre os preços das instâncias, consulte os preços do Amazon SageMaker AI.
   b. Para associar um repositório público do Github à sua instância de notebook, clique no menu suspenso Repositório Git e selecione Clonar um repositório Git público a partir de um URL no menu suspenso Repositório. Insira a URL do repositório na barra de texto da URL do repositório Git.
   c. Em Permissões de acesso, configure todos os perfis opcionais do IAM, acesso root e chaves de criptografia.
   d. Em Acesso à rede, defina configurações personalizadas de rede e acesso para sua instância do Jupyter Notebook.
6. Revise suas configurações e defina quaisquer tags para identificar a instância do seu caderno. Clique em Iniciar.

> **Note**
>
> Visualize e gerencie suas instâncias de notebook Amazon Braket nos consoles Amazon Braket e Amazon AI. SageMaker Configurações adicionais do notebook Amazon Braket estão disponíveis no console. SageMaker

Se você estiver trabalhando no console AWS do Amazon Braket dentro do SDK do Amazon Braket, os plug-ins estão pré-carregados nos cadernos que você criou. Para executar em sua própria máquina, instale o SDK e os plug-ins ao executar o comando `pip install amazon-braket-sdk` ou ao executar o comando `pip install amazon-braket-pennylane-plugin` para PennyLane plug-ins.

Crie uma instância do notebook Braket usando CloudFormation

> **Tip**
>
> Aprenda os fundamentos da computação quântica com AWS! Inscreva-se no Amazon Braket Digital Learning Plan e ganhe seu próprio selo digital após concluir uma série de cursos de aprendizado e uma avaliação digital.

Você pode usar CloudFormation para gerenciar suas instâncias de notebook Amazon Braket. As instâncias do notebook Braket são criadas com base na Amazon SageMaker AI. Com CloudFormation, você pode provisionar uma instância de notebook com um arquivo de modelo que descreve a configuração pretendida. O arquivo de modelo é escrito no formato JSON ou YAML. É possível criar, editar e excluir instâncias de forma ordenada e repetível. Você pode achar isso útil ao gerenciar várias instâncias do caderno Braket em seu. Conta da AWS

Depois de criar um CloudFormation modelo para um notebook Braket, você usa CloudFormation para implantar o recurso. Para obter mais informações, consulte Criação de uma pilha no CloudFormation console no guia do CloudFormation usuário.

Para criar uma instância do notebook Braket usando CloudFormation, você executa estas três etapas:

1. Crie um script de configuração do ciclo de vida da SageMaker IA.
2. Crie uma função AWS Identity and Access Management (IAM) a ser assumida pela SageMaker IA.

3. Crie uma instância de notebook de SageMaker IA com o prefixo **`amazon-braket-`**

Você pode reutilizar a configuração do ciclo de vida de todos os cadernos Braket que você criar. Você também pode reutilizar o perfil do IAM para os cadernos Braket aos quais você atribui as mesmas permissões de execução.

Nesta seção:

- Etapa 1: criar um script de configuração do ciclo de vida da SageMaker IA
- Etapa 2: criar a função do IAM assumida pela Amazon SageMaker AI
- Etapa 3: criar uma instância de notebook de SageMaker IA com o prefixo amazon-braket-

Etapa 1: criar um script de configuração do ciclo de vida da SageMaker IA

Use o modelo a seguir para criar um script de configuração do ciclo de vida de SageMaker IA. O script personaliza uma instância de notebook de SageMaker IA para Braket. Para obter opções de configuração para o CloudFormation recurso de ciclo de vida, consulte o AWS::SageMaker::NotebookInstanceLifecycleConfigguia do CloudFormation usuário.

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

            exit 0
```

Etapa 2: criar a função do IAM assumida pela Amazon SageMaker AI

Quando você usa uma instância do notebook Braket, a SageMaker IA executa operações em seu nome. Por exemplo, suponha que você execute um caderno Braket usando um circuito em um

dispositivo compatível. Na instância do notebook, a SageMaker IA executa a operação no Braket para você. A função de execução do notebook define as operações exatas que a SageMaker IA tem permissão para executar em seu nome. Para obter mais informações, consulte as funções de SageMaker IA no guia do desenvolvedor de SageMaker IA da Amazon.

Use o exemplo a seguir para criar um perfil de execução do caderno Braket com as permissões necessárias. É possível modificar as políticas de acordo com as suas necessidades.

> **Note**
>
> Certifique-se de que o perfil tenha permissão para as operações `s3:GetObjecte s3:ListBucket` e nos buckets do Amazon S3 prefixados com `braketnotebookcdk-"`. O script de configuração do ciclo de vida requer essas permissões para copiar o script de instalação do caderno 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: "*"
```

Etapa 3: criar uma instância de notebook de SageMaker IA com o prefixo **`amazon-braket-`**

Use o script de ciclo de vida de SageMaker IA e a função do IAM criada nas etapas 1 e 2 para criar uma instância de notebook de SageMaker IA. A instância do caderno é personalizada para o Braket e pode ser acessada com o console Amazon Braket. Para obter mais informações sobre as opções de configuração desse CloudFormation recurso, consulte AWS::SageMaker::NotebookInstanceo guia CloudFormation do usuário.

```
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
```

Criando suas tarefas quânticas com o Amazon Braket

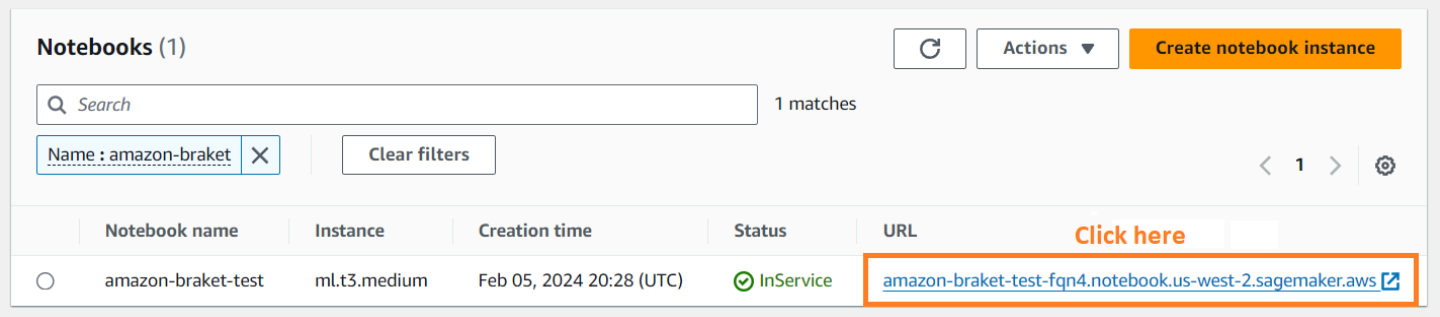
O Braket fornece ambientes de caderno Jupyter totalmente gerenciados que facilitam o início. Os cadernos Braket são pré-instalados com exemplos de algoritmos, recursos e ferramentas para desenvolvedores, incluindo o Amazon Braket SDK. Com o Amazon Braket SDK, você pode criar algoritmos quânticos e depois testá-los e executá-los em diferentes computadores e simuladores quânticos alterando uma única linha de código.

Nesta seção:

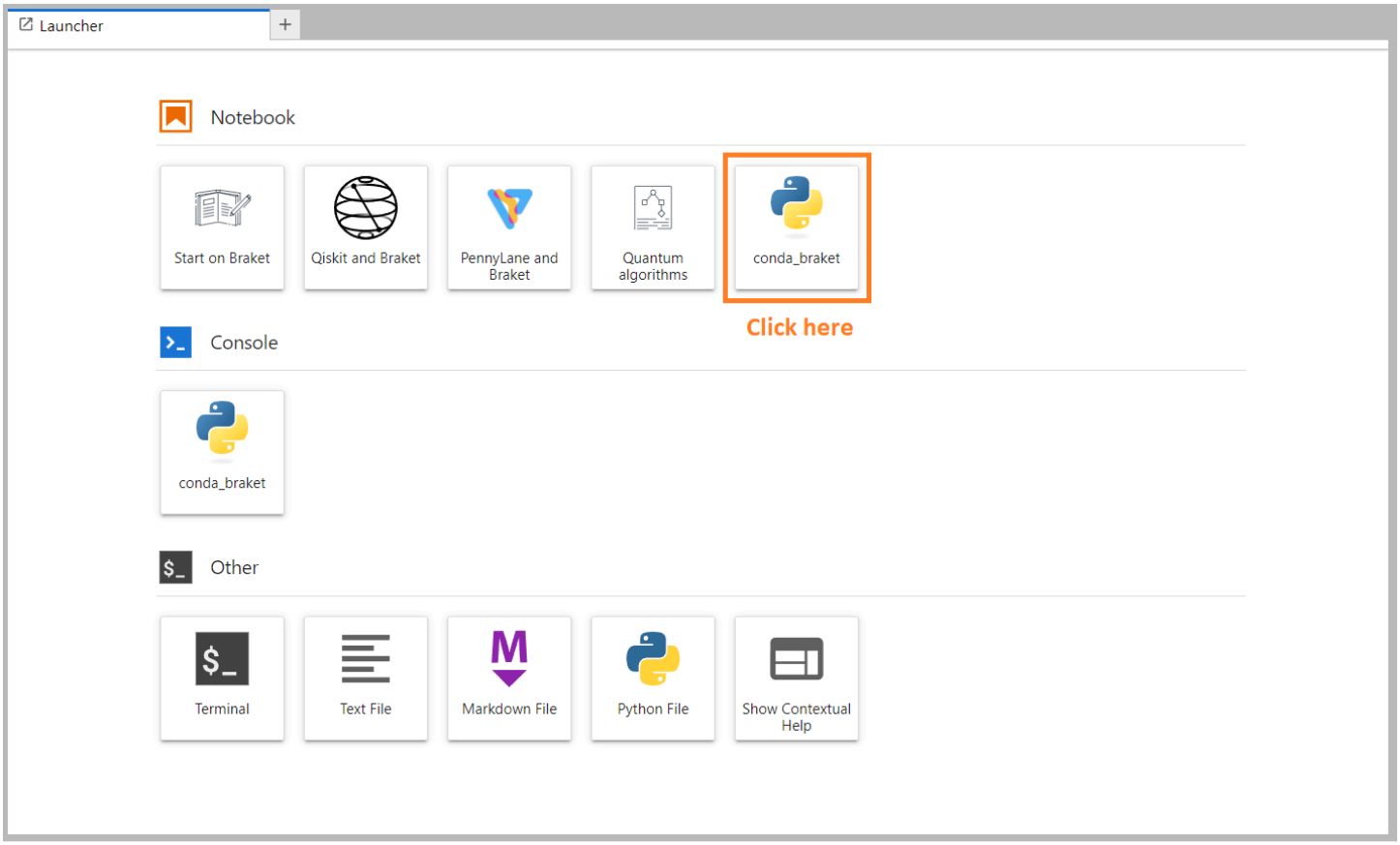
- Compilar seu primeiro circuito
- Obter aconselhamento especializado
- Execute seus circuitos com o OpenQASM 3.0
- Explorar capacidades experimentais
- Controle de pulso no Amazon Braket
- Simulação hamiltoniana analógica
- Trabalhando com o AWS Boto3

Compilar seu primeiro circuito

Depois que a instância do caderno for iniciada, abra a instância com uma interface Jupyter padrão escolhendo o caderno que você acabou de criar.



As instâncias do caderno Amazon Braket são pré-instaladas com o Amazon Braket SDK e todas as suas dependências. Comece criando um novo caderno com kernel `conda_braket`.



Você pode começar com um simples “Olá, mundo!” exemplo. Primeiro, construa um circuito que prepare um estado de Bell e, em seguida, execute esse circuito em dispositivos diferentes para obter os resultados.

Comece importando o Comece importando os módulos do Amazon Braket SDK e definindo um BRAKETlong simples; módulos SDK e definindo um circuito Bell State básico.

```
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)
```

É possível visualizar o circuito com este comando:

```
print(bell)
```

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

Execute seu circuito no simulador local

Em seguida, escolha o dispositivo quântico no qual executar o circuito. O Amazon Braket SDK vem com um simulador local para prototipagem e testes rápidos. Recomendamos usar o simulador local para circuitos menores, que podem ser de até 25 qubits (dependendo do hardware local).

Para instanciar o simulador local:

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

e executar o circuito:

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

Você deverá ver um resultado semelhante a este:

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

O estado de Bell específico que você preparou é uma superposição igual de |00# e |11# e uma distribuição quase igual (até ruído shot) de 00 e 11 como resultados de medição, conforme esperado.

Execute seu circuito em um simulador sob demanda

O Amazon Braket também fornece acesso a um simulador SV1 de alto desempenho sob demanda para executar circuitos maiores. SV1é um simulador vetorial de estado sob demanda que permite a simulação de circuitos quânticos de até 34 qubits. Você pode encontrar mais informações SV1 na seção Dispositivos compatíveis e no AWS console. Ao executar tarefas quânticas em SV1 (e sobre

TN1 ou em qualquer QPU), os resultados de sua tarefa quântica são armazenados em um bucket S3 em sua conta. Se você não especificar um bucket, o Braket SDK criará um bucket `amazon-braket-{region}-{accountID}` padrão para você. Para saber mais, consulte Gerenciando o acesso ao Amazon Braket.

> **Note**
>
> Preencha o nome real do bucket existente, onde o exemplo a seguir aparece `amazon-braket-s3-demo-bucket` como o nome do bucket. Os nomes de bucket para Amazon Braket sempre começam com `amazon-braket-`, seguidos por outros caracteres de identificação que você adiciona. Se você precisar de informações sobre como configurar um bucket do S3, consulte Conceitos básicos do Amazon S3.

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

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

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

Para executar um circuito SV1, você deve fornecer a localização do bucket S3 que você selecionou anteriormente como argumento posicional na chamada `.run()`.

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

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

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

O console Amazon Braket fornece mais informações sobre sua tarefa quântica. Navegue até a guia Tarefas quânticas no console e sua tarefa quântica deve estar no topo da lista. Como alternativa, você pode pesquisar sua tarefa quântica usando o ID exclusivo da tarefa quântica ou outros critérios.

> **Note**
>
> Depois de 90 dias, o Amazon Braket remove automaticamente todas as IDs tarefas quânticas e outros metadados associados às suas tarefas quânticas. Para obter mais informações, consulte Retenção de dados.

Executando em uma QPU

Com o Amazon Braket, você pode executar o exemplo anterior de circuito quântico em um computador quântico físico alterando apenas uma única linha de código. O Amazon Braket fornece acesso a uma variedade de dispositivos de unidade de processamento quântico (QPU). Você pode encontrar informações sobre os diferentes dispositivos e janelas de disponibilidade na seção Dispositivos compatíveis e no AWS console, na guia Dispositivos. O exemplo a seguir mostra como instanciar um dispositivo IQM.

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

Ou escolha um dispositivo IonQ com esse código:

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

Depois de selecionar um dispositivo e antes de executar sua workload, você pode consultar a profundidade da fila do dispositivo com o código a seguir para determinar o número de tarefas quânticas ou trabalhos híbridos. Além disso, os clientes podem visualizar as profundidades de fila específicas do dispositivo na página Dispositivos do Amazon Braket Management Console.

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

print(device.queue_depth().jobs)
# Returns the number of hybrid jobs queued on the device
# '2'
```

Quando você executa sua tarefa, o Amazon Braket SDK pesquisa um resultado (com um tempo limite padrão de 5 dias). Você pode alterar esse padrão modificando o parâmetro `poll_timeout_seconds` no comando `.run()`, conforme mostrado no exemplo a seguir. Lembre-se de que, se o tempo limite da pesquisa for muito curto, os resultados podem não ser retornados dentro do tempo da pesquisa, como quando uma QPU não está disponível e um erro de tempo limite local é retornado. Você pode reiniciar a pesquisa chamando a função `task.result()`.

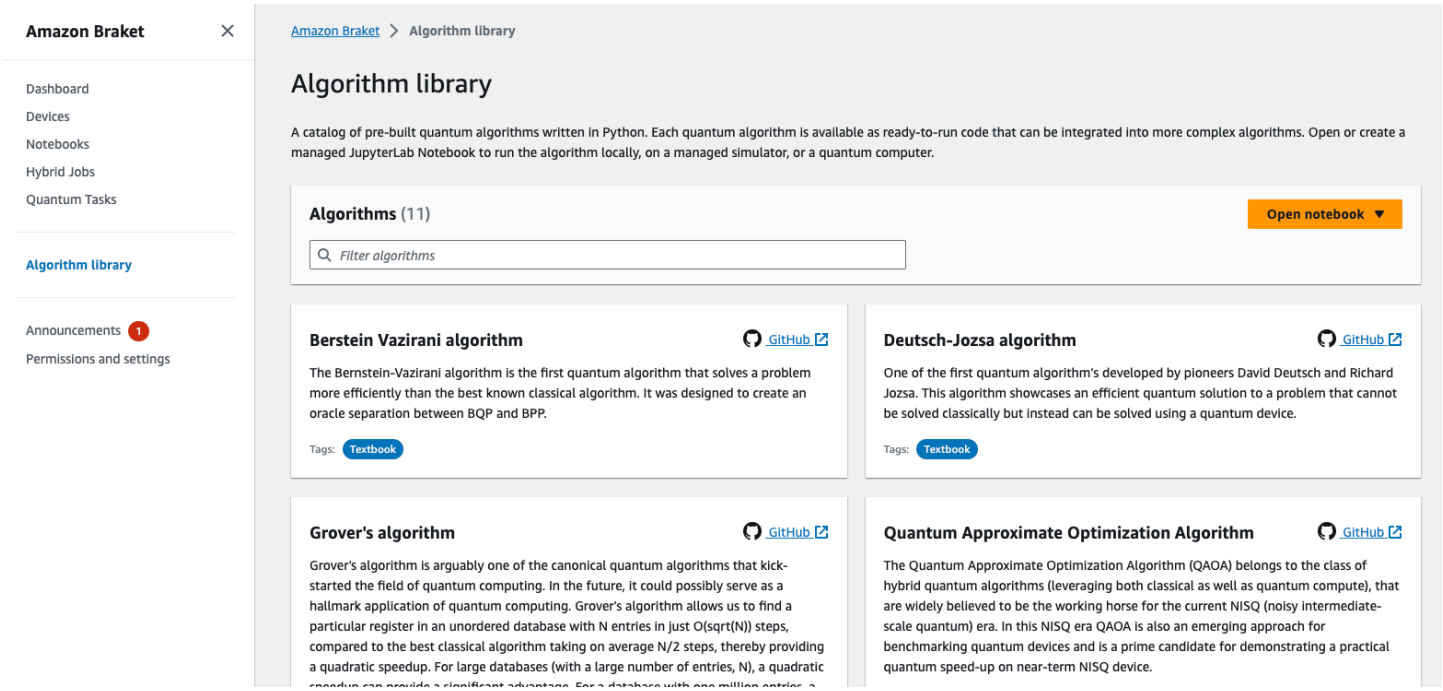
```
# Define quantum task with 1 day polling timeout
task = device.run(bell, s3_folder, poll_timeout_seconds=24*60*60)
print(task.result().measurement_counts)
```

Além disso, depois de enviar sua tarefa quântica ou trabalho híbrido, você pode chamar a função `queue_position()` para verificar a posição da fila.

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

Construindo seus primeiros algoritmos quânticos

A biblioteca de algoritmos Amazon Braket é um catálogo de algoritmos quânticos pré-criados escritos em Python. Execute esses algoritmos como estão ou use-os como pontos de partida para criar algoritmos mais complexos. Você pode acessar a biblioteca de algoritmos no console do Braket. Para obter mais informações, consulte a biblioteca de algoritmos do Braket no Github.



O console Braket fornece uma descrição de cada algoritmo disponível na biblioteca de algoritmos. Escolha um GitHub link para ver os detalhes de cada algoritmo ou escolha Abrir caderno para abrir ou criar um caderno que contenha todos os algoritmos disponíveis. Se você escolher a opção caderno, poderá encontrar a biblioteca de algoritmos Braket na pasta raiz do seu caderno.

Construir circuitos no SDK

Esta seção fornece exemplos de definição de um circuito, visualização das portas disponíveis, extensão de um circuito e visualização das portas compatíveis por cada dispositivo. Ele também contém instruções sobre como alocar manualmente o qubits, instruir o compilador a executar seus circuitos exatamente conforme definido e criar circuitos ruidosos com um simulador de ruído.

Você também pode trabalhar no nível de pulso no Braket para várias portas, com certeza. QPUs Para obter mais informações, consulte Controle de pulso no Amazon Braket.

Nesta seção:

- Portas e circuitos
- Conjuntos de programas
- Medição parcial
- Alocação manual de qubit
- Compilação literal

- Simulação de ruído

Portas e circuitos

As portas e circuitos quânticos são definidos na classe `braket.circuits` do Amazon Braket Python SDK. No SDK, você pode instanciar um novo objeto de circuito chamando `Circuit()`.

Exemplo: definir um circuito

O exemplo começa definindo um circuito de amostra de quatro qubits (`q0`, `q1`, `q2` e `q3` rotulados) consistindo em portas Hadamard padrão de um qubit e portas CNOT de dois qubits. Você pode visualizar esse circuito chamando a função `print`, conforme mostra o exemplo a seguir.

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

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

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

Exemplo: definir um circuito parametrizado

Neste exemplo, definimos um circuito com portas que dependem de parâmetros livres. Podemos especificar os valores desses parâmetros para criar um novo circuito ou, ao enviar o circuito, para ser executado como uma tarefa quântica em 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)
```

Você pode criar um novo circuito não parametrizado a partir de um parametrizado fornecendo um único `float` (que é o valor que todos os parâmetros livres terão) ou argumentos de palavra-chave especificando o valor de cada parâmetro para o circuito da seguinte forma.

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

Observe que `my_circuit` não foi modificado, então você pode usá-lo para instanciar muitos novos circuitos com valores de parâmetros fixos.

Exemplo: Modificar portas em um circuito

O exemplo a seguir define um circuito com portas que usam modificadores de controle e potência. Você pode usar essas modificações para criar novas portas, como a porta Ry controlada.

```
from braket.circuits import Circuit

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

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

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

print(my_circuit)
```

Os modificadores de porta são suportados somente no simulador local.

Exemplo: Veja todos as postas disponíveis

O exemplo a seguir mostra como analisar todas as portas disponíveis no 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)
```

A saída desse código lista todas as portas.

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

Qualquer uma dessas portas pode ser anexada a um circuito chamando o método para esse tipo de circuito. Por exemplo, chame `circ.h(0)` para adicionar uma porta Hadamard ao primeiro qubit.

> **Note**
>
> As portas são anexadas e o exemplo a seguir adiciona todas as portas listadas no exemplo anterior ao mesmo 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)
```

Além do conjunto de portas predefinido, você também pode aplicar portas unitárias autodefinidas ao circuito. Elas podem ser portas de um único qubit (conforme mostrado no código-fonte a seguir) ou portas de vários qubits aplicadas ao qubits definido pelo parâmetro `targets`.

```
import numpy as np

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

Exemplo: Estenda os circuitos existentes

Você pode estender os circuitos existentes adicionando instruções. Um `Instruction` é uma diretiva quântica que descreve a tarefa quântica a ser executada em um dispositivo quântico. Operadores `Instruction` incluem somente objetos do tipo `Gate`.

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

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

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

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

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

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

Exemplo: veja as portas que cada dispositivo suporta

Os simuladores oferecem suporte a todas as portas no SDK do Braket, mas os dispositivos QPU oferecem suporte a um subconjunto menor. Você pode encontrar as portas compatíveis de um dispositivo nas propriedades do dispositivo. O exemplo a seguir mostra um exemplo de um dispositivo IonQ:

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

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

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

```
Quantum Gates supported by Aria 1:
 ['x', 'y', 'z', 'h', 's', 'si', 't', 'ti', 'v', 'vi', 'rx', 'ry', 'rz', 'cnot',
 'swap', 'xx', 'yy', 'zz']
```

As portas compatíveis talvez precisem ser compiladas em portas nativas antes de poderem ser executadas em hardware quântico. Quando você envia um circuito, o Amazon Braket executa essa compilação automaticamente.

Exemplo: recupere programaticamente a fidelidade das portas nativas compatíveis por um dispositivo

Você pode ver as informações de fidelidade na página Dispositivos do console Braket. Às vezes, é útil acessar as mesmas informações programaticamente. O código a seguir mostra como extrair a fidelidade de duas portas qubit entre duas portas de uma QPU.

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

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

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

Conjuntos de programas

Os conjuntos de programas executam com eficiência vários circuitos quânticos em uma única tarefa quântica. Nessa tarefa, você pode enviar até 100 circuitos quânticos ou um único circuito paramétrico com até 100 conjuntos de parâmetros diferentes. Essa operação minimiza o tempo entre as execuções subsequentes do circuito e reduz a sobrecarga de processamento de tarefas quânticas. Atualmente, os conjuntos de programas são compatíveis com o Amazon Local Simulator Braket e outros AQT dispositivosIQM. Rigetti

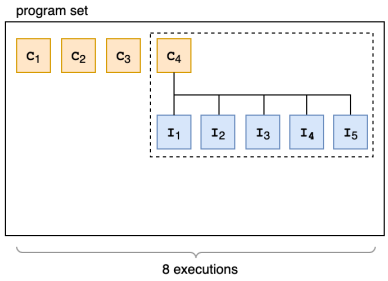
Definindo um ProgramSet

O primeiro exemplo de código a seguir demonstra como construir um `ProgramSet` usando circuitos parametrizados e circuitos sem parâmetros.

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

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

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

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

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

Esse conjunto de programas contém quatro programas exclusivos: `circ1`, `circ2`, `circ3` e `circuit_binding`. O programa `circuit_binding` é executado com cinco associações de parâmetros diferentes, criando cinco executáveis. Os outros três programas sem parâmetros criam um executável cada. Isso resulta em um total de oito executáveis, conforme mostrado na imagem a seguir.




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

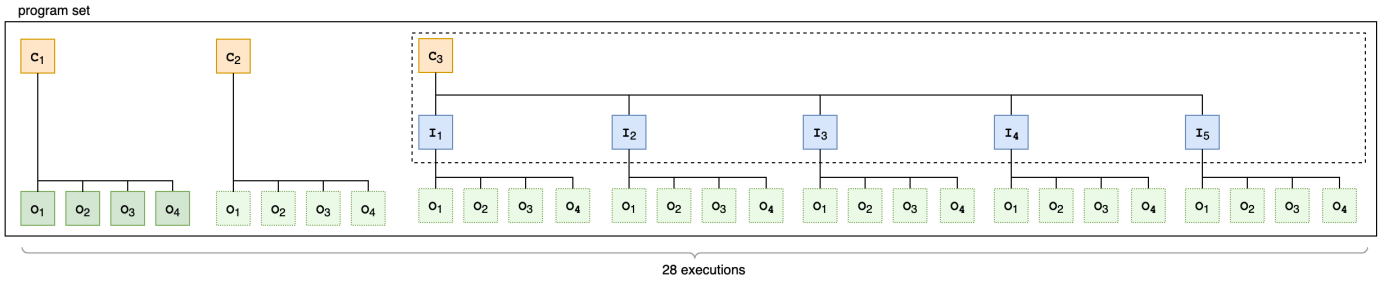
O segundo exemplo de código a seguir demonstra como usar o método `product()` para anexar o mesmo conjunto de observáveis a cada executável do conjunto de programas.

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

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

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

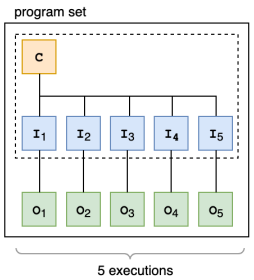
Para programas sem parâmetros, cada observável é medido para cada circuito. Para programas paramétricos, cada observável é medido para cada conjunto de entrada, conforme mostrado na imagem a seguir.

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



O terceiro exemplo de código a seguir demonstra como usar o método `zip()` para emparelhar observáveis individuais com conjuntos de parâmetros específicos no `ProgramSet`.

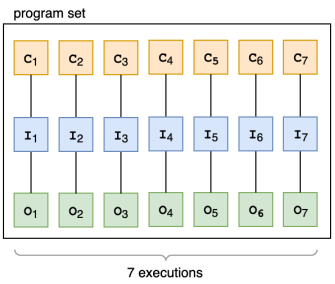
```
program_set_3 = ProgramSet.zip(
    circuits=circuit_binding,
    observables=observables + [Y(0) @ Y(1)]
)
```




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

Em vez do `CircuitBinding()`, você pode compactar diretamente uma lista de observáveis com uma lista de circuitos e conjuntos de entrada.

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




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

Para obter mais informações e exemplos sobre conjuntos de programas, consulte a pasta Conjunto de programas no amazon-braket-examples Github.

Inspecione e execute um conjunto de programas em um dispositivo

A contagem de executáveis de um conjunto de programas é igual ao número de circuitos exclusivos vinculados a parâmetros. Calcule o número total de executáveis e disparos do circuito usando o exemplo de código a seguir.

```
# Number of shots per executable
shots = 10
num_executables = program_set_1.total_executables

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

> **Note**
>
> Com conjuntos de programas, você paga uma taxa única por tarefa e uma taxa por tiro com base no número total de disparos em todos os circuitos em um conjunto de programas.

Para executar o conjunto de programas, use o exemplo de código a seguir.

```
# Run the program set
```

```
task = device.run(
   program_set_1, shots=total_num_shots,
)
```

Ao usar dispositivos Rigetti, seu conjunto de programas pode permanecer no estado `RUNNING` enquanto as tarefas estão parcialmente concluídas e parcialmente enfileiradas. Para obter resultados mais rápidos, considere enviar seu programa definido como um Hybrid Job.

Analisando resultados

Execute o código a seguir para analisar e medir os resultados dos executáveis em um `ProgramSet`.

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

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

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

Medição parcial

Em vez de medir todos os qubits em um circuito quântico, use a medição parcial para medir qubits individuais ou um subconjunto de qubits.

> **Note**
>
> Recursos adicionais, como medição de circuito médio e operações de avanço, estão disponíveis como capacidades experimentais, consulte Acesso a circuitos dinâmicos em dispositivos IQM.

Exemplo: medir um subconjunto de qubits

O exemplo de código a seguir demonstra a medição parcial medindo somente o qubit 0 em um circuito de estado Bell.

```
from braket.devices import LocalSimulator
```

```
from braket.circuits import Circuit

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

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

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

# Get the results
result = task.result()

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

Alocação manual de qubit

Ao executar um circuito quântico em computadores quânticos a partir do Rigetti, você pode, opcionalmente, usar a alocação manual de qubit para controlar quais qubits são usados em seu algoritmo. O Amazon Braket Console e o Amazon Braket SDK ajudam você a inspecionar os dados de calibração mais recentes do dispositivo de unidade de processamento quântico (QPU) selecionado, para que você possa selecionar o melhor para seu experimento de qubits.

A alocação manual de qubit permite que você execute circuitos com maior precisão e investigue propriedades de qubit individuais. Pesquisadores e usuários avançados otimizam o projeto do circuito com base nos dados mais recentes de calibração do dispositivo e podem obter resultados mais precisos.

O exemplo a seguir demonstra como alocar qubits de forma explícita.

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

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

# Set up S3 bucket (where results are stored)
my_bucket = "amazon-braket-s3-demo-bucket" # The name of the bucket
```

```
my_prefix = "your-folder-name" # The name of the folder in the bucket
s3_location = (my_bucket, my_prefix)

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

Para obter mais informações, consulte os exemplos do Amazon Braket GitHub em, ou mais especificamente, neste notebook: Alocação de qubits em dispositivos QPU.

Compilação literal

Quando você executa um circuito quântico em computadores quânticos baseados em portas, você pode direcionar o compilador para executar seus circuitos exatamente como definido, sem nenhuma modificação. Usando a compilação literal, você pode especificar que um circuito inteiro seja preservado com precisão conforme especificado ou que somente partes específicas dele sejam preservadas (compatíveis somente pelo Rigetti). Ao desenvolver algoritmos para benchmarking de hardware ou protocolos de mitigação de erros, você precisa ter a opção de especificar exatamente as portas e os layouts de circuito que estão sendo executados no hardware. A compilação integral oferece controle direto sobre o processo de compilação, desativando determinadas etapas de otimização, garantindo assim que seus circuitos funcionem exatamente como projetado.

A compilação literal é suportada nos Rigetti dispositivosAQT, IonQIQM, e e requer o uso de portas nativas. Ao usar a compilação literal, é aconselhável verificar a topologia do dispositivo para garantir que as portas sejam chamadas a qubits conectados e que o circuito use as portas nativas compatíveis no hardware. O exemplo a seguir mostra como acessar programática a lista de portas nativas compatíveis com um dispositivo.

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

Para Rigetti, a reconexão qubit deve ser desativada configurando `disableQubitRewiring=True` para uso com compilação literal. Se `disableQubitRewiring=False` for definido ao usar caixas textuais em uma compilação, o circuito quântico falha na validação e não será executado.

Se a compilação literal estiver habilitada para um circuito e executada em uma QPU não compatível, um erro é gerado indicando que uma operação não comptatível causou a falha da tarefa. À medida que mais hardware quântico oferece suporte nativo às funções do compilador, esse recurso será expandido para incluir esses dispositivos. Os dispositivos que oferecem suporte à compilação literal a incluem como uma operação compatível quando consultados com o código a seguir.

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

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

Não há custo adicional associado ao uso da compilação literal. Você continua sendo cobrado por tarefas quânticas executadas em dispositivos Braket QPU, instâncias de caderno e simuladores sob demanda com base nas taxas atuais, conforme especificado na página de preços do Amazon Braket. Para obter mais informações, consulte o exemplo de caderno de compilação literal.

> **Note**
>
> Se você estiver usando o OpenQASM para escrever seus circuitos para os IonQ dispositivos AQT e quiser mapear seu circuito diretamente para os qubits físicos, precisará usar o, `#pragma braket verbatim` pois o `disableQubitRewiring` sinalizador é ignorado pelo OpenQASM.

Simulação de ruído

Para instanciar o simulador de ruído local, você pode alterar o backend da seguinte maneira.

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

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

É possível fazer circuitos ruidosos de duas maneiras:

1. Construa o circuito ruidoso de baixo para cima.
2. Pegue um circuito existente e livre de ruído e injete ruído por toda parte.

O exemplo a seguir mostra as abordagens usando um circuito básico com ruído despolarizador e um canal Kraus personalizado.

```
import scipy.stats
import numpy as np

# Bottom up approach
# Apply depolarizing noise to qubit 0 with probability of 0.1
```

```
circ = Circuit().x(0).x(1).depolarizing(0, probability=0.1)

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

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

```
from braket.circuits import Noise

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

Executar um circuito é a mesma experiência de usuário de antes, conforme mostrado nos dois exemplos a seguir.

Exemplo 1

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

Ou

Exemplo 2

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

Para obter mais exemplos, consulte o exemplo introdutório do simulador de ruído do Braket

Inspecionando o circuito

Os circuitos quânticos no Amazon Braket têm um conceito de pseudotempo chamado `Moments`. Cada qubit pode experimentar uma única porta por `Moment`. O objetivo do `Moments` é facilitar o endereçamento dos circuitos e suas portas e fornecer uma estrutura temporal.

> **Note**
>
> Os momentos geralmente não correspondem ao tempo real em que as portas são executadas em uma QPU.

A profundidade de um circuito é dada pelo número total de momentos nesse circuito. É possível visualizar a profundidade do circuito chamando o método `circuit.depth` conforme mostrado no exemplo a seguir.

```
from braket.circuits import Circuit

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

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

A profundidade total do circuito acima é 3 (mostrada como momentos `0`, 1 e2). Você pode verificar a operação da porta a cada momento.

`Moments` funciona como um dicionário de pares de valores-chave.

- A chave é `MomentsKey()`, que contém pseudotempo e informações do qubit.
- O valor é atribuído no tipo de `Instructions()`.

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

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

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

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

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

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

Você também pode adicionar portas a um circuito `Moments`.

```
from braket.circuits import Instruction, Gate

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

new_circ.moments.add(instructions)
```

```
print(new_circ)
```

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

Lista de tipos de resultados

O Amazon Braket pode retornar diferentes tipos de resultados quando um circuito é medido usando o `ResultType`. Um circuito pode retornar os seguintes tipos de resultados.

- `AdjointGradient` retorna o gradiente (derivada vetorial) do valor esperado de um observável fornecido. Esse observável está agindo em um alvo fornecido em relação aos parâmetros especificados usando o método de diferenciação adjunta. Você só pode usar esse método quando tiros = 0.
- `Amplitude` retorna a amplitude dos estados quânticos especificados na função de onda de saída. Ele está disponível somente nos simuladores locais SV1 e nos simuladores.
- `Expectation` retorna o valor esperado de um determinado observável, que pode ser especificado com a classe `Observable` apresentada posteriormente neste capítulo. O alvo qubits usado para medir o observável deve ser especificado e o número de alvos especificados deve ser igual ao número qubits sobre o qual o observável atua. Se nenhum alvo for especificado, o observável deve operar somente em 1 qubit e é aplicado a todos os qubits em paralelo.
- `Probability` retorna as probabilidades de medir os estados de base computacional. Se nenhuma meta for especificada, `Probability` retorna a probabilidade de medir todos os estados básicos. Se os alvos forem especificados, somente as probabilidades marginais dos vetores básicos nos qubits especificados serão retornadas. Os simuladores gerenciados QPUs estão limitados a 15 qubits no máximo, e os simuladores locais estão limitados ao tamanho da memória do sistema.
- `Reduced density matrix` retorna uma matriz de densidade para um subsistema de qubits de destino especificado de um sistema de qubits. Para limitar o tamanho desse tipo de resultado, o Braket limita o número de alvos qubits a um máximo de 8.
- `StateVector` retorna o vetor de estado completo. Ele está disponível no simulador local.

- `Sample` retorna as contagens de medição de um alvo qubit especificado definido e observável. Se nenhum alvo for especificado, o observável deve operar somente em 1 qubit e é aplicado a todos os qubits em paralelo. Se os alvos forem especificados, o número de alvos especificados deverá ser igual ao número de qubits sobre os quais o observável atua.
- `Variance` retorna a variância (`mean([x-mean(x)]`^{2}`)`) do conjunto qubit de destino especificado e observável como o tipo de resultado solicitado. Se nenhum alvo for especificado, o observável deve operar somente em 1 qubit e é aplicado a todos os qubits em paralelo. Caso contrário, o número de alvos especificados deve ser igual ao número qubits ao qual o observável pode ser aplicado.

Os tipos de resultados compatíveis para diferentes provedores:

	sim local	SV1	DM1	TN1	AQT	IonQ	IQM	Rigetti
Gradient adjunto	N	S	N	N	N	N	N	N
Amplitud	S	S	N	N	N	N	N	N
Expectat va	S	S	S	S	S	S	S	S
Probabili dade	S	S	S	N	S	S	S	S
Matriz de densidac reduzida	S	N	S	N	N	N	N	N
Vetor de estado	S	N	N	N	N	N	N	N
Amostra	S	S	S	S	S	S	S	S
Variação	S	S	S	S	S	S	S	S

É possível verificar os tipos de resultados compatíveis ao examinar as propriedades do dispositivo, conforme mostrado no exemplo a seguir.

```
from braket.aws import AwsDevice

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

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

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

Para chamar `ResultType`, anexe-o a um circuito, conforme mostrado no exemplo a seguir.

```
from braket.circuits import Circuit, Observable

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

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

> **Note**
>
> Diferentes dispositivos quânticos fornecem resultados em vários formatos. Por exemplo, os dispositivos Rigetti retornam medições, enquanto os dispositivos IonQ fornecem probabilidades. O Amazon Braket SDK oferece uma propriedade de medidas para todos os resultados. No entanto, para dispositivos que retornam probabilidades, essas medidas são pós-computadas e baseadas nas probabilidades, pois as medições por disparo não estão disponíveis. Para determinar se um resultado foi pós-computado, verifique

`measurements_copied_from_device` no objeto resultante. Essa operação está detalhada no arquivo gate_model_quantum_task_result.py no repositório Amazon Braket SDK GitHub .

Observáveis

A classe `Observable` do Amazon Braket permite medir um observável específico.

Você pode aplicar somente uma única não identidade observável a cada qubit. Ocorre um erro se você especificar dois ou mais observáveis sem identidade diferentes para o mesmo qubit. Para isso, cada fator de um produto tensorial conta como um indivíduo observável. Isso significa que você pode ter vários produtos tensoriais no mesmo qubit, desde que os fatores que atuam sobre o qubit permaneçam os mesmos.

Um observável pode ser escalado e adicionar outros observáveis (escalonados ou não). Isso cria um Sum que pode ser usado no tipo de resultado `AdjointGradient`.

A classe `Observable` inclui os seguintes observáveis.

```
import numpy as np

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

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

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

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

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

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

```
Eigenvalue: [ 1. -1.]
Basis rotation gates: (Ry('angle': -0.7853981633974483, 'qubit_count': 1),)
The matrix form of the observable:
 [[ 1.+0.j  0.+0.j]
 [ 0.+0.j -1.+0.j]]
The matrix form of the tensor product:
 [[ 0.+0.j  0.+0.j  0.-1.j  0.+0.j]
 [ 0.+0.j -0.+0.j  0.+0.j  0.+1.j]
 [ 0.+1.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.-1.j  0.+0.j -0.+0.j]]
Factorize an observable: (Y('qubit_count': 1), Z('qubit_count': 1))
Self-defined Hermitian: Hermitian('qubit_count': 1, 'matrix': [[0.+0.j 1.+0.j], [1.+0.j
 0.+0.j]])
Sum of other (scaled) observables: Sum(TensorProduct(X('qubit_count': 1),
 X('qubit_count': 1)), TensorProduct(Z('qubit_count': 1), Z('qubit_count': 1)))
```

Parâmetros

Os circuitos podem incorporar parâmetros livres. Esses parâmetros livres só precisam ser construídos uma vez para serem executados várias vezes e podem ser usados para calcular gradientes.

Cada parâmetro livre usa um nome codificado em string que é usado para:

- Definir valores do parâmetro
- Identificar quais parâmetros usar

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

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

Gradiente adjunto

O dispositivo SV1 calcula o gradiente adjunto de um valor esperado observável, incluindo hamiltoniano de vários termos. Para diferenciar os parâmetros, especifique seu nome (em formato de string) ou por referência direta.

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

device = AwsDevice(Devices.Amazon.SV1)

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

Passar valores de parâmetros fixos como argumentos para um circuito parametrizado removerá os parâmetros livres. Executar esse circuito com `AdjointGradient` produz um erro, porque os parâmetros livres não existem mais. O exemplo de código a seguir demonstra o uso correto e incorreto:

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

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

Obter aconselhamento especializado

Conecte-se com especialistas em computação quântica diretamente no console de gerenciamento do Braket para obter orientação adicional sobre suas workloads.

Para explorar as opções de aconselhamento especializado por meio do Braket Direct, abra o console do Braket, escolha Braket Direct no painel esquerdo e navegue até a seção Aconselhamento especializado. As seguintes opções de aconselhamento especializado estão disponíveis:

- Horário de atendimento do Braket: O horário de atendimento do Braket é de sessões individuais, por ordem de chegada, e ocorre todos os meses. Cada horário de expediente disponível é de 30 minutos e é gratuito. Conversar com especialistas do Braket pode ajudá-lo a passar da concepção à execução com mais rapidez, explorando o use-case-to-device ajuste, identificando as opções para melhor usar o Braket em seu algoritmo e obtendo recomendações sobre como

usar determinados recursos do Braket, como Amazon Braket Hybrid Jobs, Braket Pulse ou Analog Hamiltonian Simulation.

- Para se inscrever no horário comercial da Braket, selecione Inscrever-se e preencha as informações de contato, detalhes da workload e os tópicos de discussão desejados.
- Você receberá um convite de calendário para o próximo espaço disponível por e-mail.

> **Note**
>
> Para problemas emergentes ou questões de solução rápida de problemas, recomendamos entrar em contato com o Suporte AWS. Para perguntas não urgentes, você também pode usar o fórum AWS re:Post ou o Stack Exchange de computação quântica, onde você pode procurar perguntas respondidas anteriormente e fazer novas.

- Ofertas de fornecedores de hardware Quantum: IonQ, QuEra e Rigetti fornecem ofertas de serviços profissionais por meio do AWS Marketplace.
  - Para explorar suas ofertas, selecione Connect e navegue em seus anúncios.
  - Para saber mais sobre as ofertas de serviços profissionais no AWS Marketplace, consulte Produtos de serviços profissionais.
- Laboratório de Soluções Quânticas da Amazon (QSL): O QSL é uma equipe colaborativa de pesquisa e serviços profissionais composta por especialistas em computação quântica que podem ajudá-lo a explorar com eficácia a computação quântica e avaliar o desempenho atual dessa tecnologia.
  - Para entrar em contato com o QSL, selecione Conectar-se e preencha as informações de contato e detalhes do caso de uso.
  - A equipe do QSL entrará em contato com você por e-mail com as próximas etapas.

Execute seus circuitos com o OpenQASM 3.0

Amazon Braket agora dá suporte ao OpenQASM 3.0 para dispositivos e simuladores quânticos baseados em portas. Este guia do usuário fornece informações sobre o subconjunto do OpenQASM 3.0 suportado pelo Braket. Os clientes do Braket agora têm a opção de enviar circuitos Braket com o SDK ou fornecendo diretamente strings OpenQASM 3.0 para todos os dispositivos baseados em portas lógicas com a Amazon Braket API e o Amazon Braket Python SDK.

Os tópicos deste guia explicam vários exemplos de como concluir as seguintes tarefas quânticas.

- Crie e envie tarefas quânticas do OpenQASM em diferentes dispositivos Braket
- Acesse as operações com suporte e os tipos de resultados
- Simule ruídos com o OpenQASM
- Use a compilação literal com o OpenQASM
- Solucionar problemas do OpenQASM

Este guia também fornece uma introdução a certos recursos específicos de hardware que podem ser implementados com o OpenQASM 3.0 no Braket e links para recursos adicionais.

Nesta seção:

- O que é o OpenQASM 3.0?
- Quando usar o OpenQASM 3.0
- Como o OpenQASM 3.0 funciona
- Pré-requisitos
- Quais recursos do OpenQASM o Braket suporta?
- Crie e envie um exemplo de tarefa quântica do OpenQASM 3.0
- Suporte para OpenQASM em diferentes dispositivos Braket
- Simule ruídos com o OpenQASM 3.0
- Reconexão Qubit com o OpenQASM 3.0
- Compilação literal com o OpenQASM 3.0
- O console do Braket
- Recursos adicionais do
- Gradientes de computação com o OpenQASM 3.0
- Medindo qubits específicos com o OpenQASM 3.0

O que é o OpenQASM 3.0?

A Open Quantum Assembly Language (OpenQASM) é uma representação intermediária para instruções quânticas. O OpenQASM é uma estrutura de código aberto e é amplamente usado para a especificação de programas quânticos para dispositivos baseados em portas. Com o OpenQASM, os usuários podem programar as portas quânticas e as operações de medição que formam os blocos de construção da computação quântica. A versão anterior do OpenQASM (2.0) foi usada por várias bibliotecas de programação quântica para descrever programas básicos.

A nova versão do OpenQASM (3.0) estende a versão anterior para incluir mais recursos, como controle de nível de pulso, temporização de portas e fluxo de controle clássico para preencher a lacuna entre a interface do usuário final e a linguagem de descrição do hardware. Detalhes e especificações da versão 3.0 atual estão disponíveis na GitHub OpenQASM 3.x Live Specification. O desenvolvimento futuro do OpenQASM é governado pelo Comitê de Direção Técnica do OpenQASM 3.0, do qual AWS é membro ao lado da IBM, da Microsoft e da Universidade de Innsbruck.

Quando usar o OpenQASM 3.0

O OpenQASM fornece uma estrutura expressiva para especificar programas quânticos por meio de controles de baixo nível que não são específicos da arquitetura, tornando-o adequado como representação em vários dispositivos baseados em portas. O suporte do Braket ao OpenQASM promove sua adoção como uma abordagem consistente para o desenvolvimento de algoritmos quânticos baseados em portas, reduzindo a necessidade de os usuários aprenderem e manterem bibliotecas em várias estruturas.

Se você tiver bibliotecas de programas existentes no OpenQASM 3.0, poderá adaptá-las para uso com o Braket em vez de reescrever completamente esses circuitos. Pesquisadores e desenvolvedores também devem se beneficiar de um número crescente de bibliotecas de terceiros disponíveis com suporte para desenvolvimento de algoritmos no OpenQASM.

Como o OpenQASM 3.0 funciona

O suporte para o OpenQASM 3.0 da Braket fornece paridade de recursos com a representação intermediária atual. Isso significa que tudo o que você pode fazer hoje em dispositivos de hardware e simuladores sob demanda com o Braket, você pode fazer com o OpenQASM usando o API Braket. Você pode executar programas OpenQASM 3.0 fornecendo diretamente cadeias de caracteres OpenQASM a todos os dispositivos baseados em portas de uma maneira semelhante à forma como os circuitos são fornecidos atualmente aos dispositivos no Braket. Os usuários do Braket também podem integrar bibliotecas de terceiros que suportam o OpenQASM 3.0. O restante deste guia detalha como desenvolver representações do OpenQASM para uso com o Braket.

Pré-requisitos

Para usar o OpenQASM 3.0 no Amazon Braket, você precisa ter a versão v1.8.0 dos esquemas do Amazon Braket Python e a versão v1.17.0 ou superior do Amazon Braket Python SDK.

Se você é um usuário iniciante do Amazon Braket, você precisa habilitar o Amazon Braket. Para obter instruções, consulte Habilitar o Amazon Braket.

Quais recursos do OpenQASM o Braket suporta?

A seção a seguir lista os tipos de dados, declarações e instruções pragmáticas do OpenQASM 3.0 suportados pelo Braket.

Nesta seção:

- Tipos de dados do OpenQASM com suporte
- Declarações do OpenQASM compatíveis
- Pragmas Braket OpenQASM
- Suporte de recursos avançados para o OpenQASM no simulador local
- Operações e gramática suportadas com OpenPulse

Tipos de dados do OpenQASM com suporte

Os seguintes tipos de dados são compatíveis com o Amazon Braket:

- Números inteiros não negativos são usados para índices de qubit (virtuais e físicos):
  - `cnot q[0], q[1];`
  - `h $0;`
- Números ou constantes de ponto flutuante podem ser usados para ângulos de rotação da porta:
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi é uma constante embutida no OpenQasm e não pode ser usada como nome de parâmetro.

- Matrizes de números complexos (com a notação `im` OpenQASM para parte imaginária) são permitidas em pragmas de tipo de resultado para definir observáveis hermitianos gerais e em pragmas unitários:
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

Declarações do OpenQASM compatíveis

As seguintes declarações do OpenQASM são apoiadas pelo Amazon Braket.

- `Header: OPENQASM 3;`
- Declarações de bits clássicas:
  - `bit b1;` (equivalentemente, `creg b1;`)
  - `bit[10] b2;` (equivalentemente, `creg b2[10];`)
- Declarações de qubit:
  - `qubit b1;` (equivalentemente, `qreg b1;`)
  - `qubit[10] b2;` (equivalentemente, `qreg b2[10];`)
- Indexação em matrizes: `q[0]`
- Entrada: `input float alpha;`
- especificação do qubits físico: `$0`
- Portas e operações compatíveis em um dispositivo:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> As portas compatíveis de um dispositivo podem ser encontradas nas propriedades do dispositivo para ações do OpenQASM; nenhuma definição de porta é necessária para usar essas portas.

- Declarações textuais. Atualmente, não oferecemos suporte à notação de duração da caixa. Portas qubits nativas e físicas são obrigatórias em caixas textuais.

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

- Medição e atribuição de medições em qubits ou em um registro qubit completo.

  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`
- As declarações de barreira fornecem controle explícito sobre a compilação e execução do circuito, impedindo a reordenação e as otimizações das portas além dos limites das barreiras. Eles também impõem uma ordem temporal estrita durante a execução, garantindo que todas as operações antes de uma barreira sejam concluídas antes do início das operações subsequentes.
  - `barrier;`
  - `barrier q[0], q[1];`
  - `barrier $3, $6;`

Pragmas Braket OpenQASM

As seguintes instruções do pragma OpenQASM são compatíveis pelo Amazon Braket.

- Pragmas de ruído
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- Pragmas literais
  - `#pragma braket verbatim`
- Pragmas de tipos de resultados
  - Tipos de resultados invariantes básicos:
    - Vetor de estado: `#pragma braket result state_vector`
    - Matriz de densidade: `#pragma braket result density_matrix`
  - Pragmas de computação de gradiente:
    - Gradiente adjunto: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Tipos de resultados básicos Z:
    - Amplitude `#pragma braket result amplitude "01"`

  - Probabilidade `#pragma braket result probability q[0], q[1]`
- Tipos de resultados rotacionados de base
  - Expectativa: `#pragma braket result expectation x(q[0]) @ y([q1])`
  - Variância: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
  - Amostra: `#pragma braket result sample h($1)`

> **Note**
>
> O OpenQASM 3.0 é compatível com versões anteriores do OpenQASM 2.0, portanto, programas escritos usando 2.0 podem ser executados no Braket. No entanto, os recursos do OpenQASM 3.0 suportados pelo Braket têm algumas pequenas diferenças de sintaxe, como `qreg` vs `creg` e `qubit` vs `bit`. Também há diferenças na sintaxe de medição, e elas precisam ser compatíveis com a sintaxe correta.

Suporte de recursos avançados para o OpenQASM no simulador local

`LocalSimulator` suporta recursos avançados do OpenQASM que não são oferecidos como parte dos QPUs ou simuladores sob demanda da Braket. A lista de recursos a seguir só é compatível no `LocalSimulator`:

- Modificadores de porta
- Portas embutidas OpenQASM
- Variáveis clássicas
- Operações clássicas
- Portas personalizadas
- Controle do Classic
- Arquivos ASM
- Sub-rotinas

Para ver exemplos de cada recurso avançado, consulte este exemplo de caderno. Para obter a especificação completa do OpenQASM, consulte o site do OpenQASM.

Operações e gramática suportadas com OpenPulse

Tipos OpenPulse de dados compatíveis

Blocos de chamadas:

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

Blocos decalques:

```
// 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 {
...
}
```

Quadros:

```
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};
```

Exemplo de calibração de porta 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;
```

Exemplo de pulso arbitrário:

```
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);
```

```
}
```

Crie e envie um exemplo de tarefa quântica do OpenQASM 3.0

Você pode usar o Amazon Braket Python SDK, o Boto3 AWS CLI ou o para enviar tarefas quânticas do OpenQASM 3.0 para um dispositivo Amazon Braket.

Nesta seção:

- Um exemplo do programa OpenQASM 3.0
- Use o SDK do Python para criar tarefas quânticas do OpenQASM 3.0
- Use o Boto3 para criar tarefas quânticas do OpenQASM 3.0
- Use o AWS CLI para criar tarefas do OpenQASM 3.0

Um exemplo do programa OpenQASM 3.0

Para criar uma tarefa OpenQASM 3.0, você pode começar com um programa OpenQASM 3.0 básico (ghz.qasm) que prepara um estado GHZ, conforme mostrado no exemplo a seguir.

```
// 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;
```

Use o SDK do Python para criar tarefas quânticas do OpenQASM 3.0

Você pode usar o Amazon Braket Python SDK para enviar esse programa para um dispositivo Amazon Braket com o código a seguir. Certifique-se de substituir o exemplo de localização de bucket do Amazon S3 “amzn-s3-demo-bucket” por seu próprio nome de bucket do Amazon S3.

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

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

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

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

Use o Boto3 para criar tarefas quânticas do OpenQASM 3.0

Você também pode usar o AWS Python SDK para Braket (Boto3) para criar as tarefas quânticas usando strings do OpenQASM 3.0, conforme mostrado no exemplo a seguir. O trecho de código a seguir faz referência ao ghz.qasm, que prepara um estado GHZ conforme mostrado acima.

```
import boto3
import json

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

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

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

```
braket_client = boto3.client('braket', region_name='us-west-1')
rsp = braket_client.create_quantum_task(
    action=json.dumps(
        action
    ),
    deviceParameters=json.dumps(
        device_parameters
    ),
    deviceArn=device_arn,
    shots=shots,
    outputS3Bucket=my_bucket,
    outputS3KeyPrefix=s3_prefix,
)
```

Use o AWS CLI para criar tarefas do OpenQASM 3.0

O AWS Command Line Interface (CLI) também pode ser usado para enviar programas OpenQASM 3.0, conforme mostrado no exemplo a seguir.

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

Suporte para OpenQASM em diferentes dispositivos Braket

Para dispositivos que suportam o OpenQASM 3.0, o campo `action` suporta uma nova ação por meio da resposta `GetDevice`, conforme mostrado no exemplo a seguir para os dispositivos Rigetti e 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"
            ],
            ….
        }
    }
}
```

Para dispositivos que suportam controle de pulso, o campo `pulse` é exibido na resposta `GetDevice`. O exemplo a seguir mostra esse campo `pulse` para o dispositivo Rigetti.

```
// Rigetti
{
  "pulse": {
    "braketSchemaHeader": {
```

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

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

Os campos anteriores detalham o seguinte:

Portas:

Descreve as portas de dispositivos externos (`extern`) pré-fabricadas declaradas na QPU, além das propriedades associadas à porta especificada. Todas as portas listadas nessa estrutura são pré-declaradas como identificadores válidos no programa `OpenQASM 3.0` enviado pelo usuário. As propriedades adicionais de uma porta incluem:

- ID da porta (portId)

  - O nome da porta declarado como identificador no OpenQASM 3.0.
- Direção (direção)
  - A direção da porta. As portas de acionamento transmitem pulsos (direção “tx”), enquanto as portas de medição recebem pulsos (direção “rx”).
- Tipo de porta (PortType)
  - O tipo de ação pela qual essa porta é responsável (por exemplo, drive, capture ou ff - fast-flux).
- Dt (dt)
  - O tempo em segundos que representa uma única etapa de tempo de amostra na porta especificada.
- Mapeamentos de Qubit (mapeamentos de qubit)
  - Os qubits associados à porta fornecida.
- Frequências centrais (frequências centrais)
  - Uma lista das frequências centrais associadas para todos os quadros pré-declarados ou definidos pelo usuário na porta. Para obter mais informações, consulte Quadros.
- Propriedades específicas do QHP () qhpSpecificProperties
  - Um mapa opcional detalhando as propriedades existentes sobre a porta específica do QHP.

Quadros:

Descreve os quadros externos pré-fabricados declarados na QPU, bem como as propriedades associadas aos quadros. Todos os quadros listados nessa estrutura são pré-declarados como identificadores válidos dentro do programa `0penQASM 3.0` enviado pelo usuário. As propriedades adicionais de uma moldura incluem:

- ID do quadro (frameId)
  - O nome do quadro declarado como um identificador no OpenQASM 3.0.
- ID da porta (portId)
  - A porta de hardware associada ao quadro.
- Frequência (frequência)
  - A frequência inicial padrão do quadro.
- Frequência central (centerFrequency)
  - O centro da largura de banda de frequência do quadro. Normalmente, os quadros só podem ser ajustados para uma determinada largura de banda em torno da frequência central. Como

resultado, os ajustes de frequência devem permanecer dentro de um determinado delta da frequência central. Você pode encontrar o valor da largura de banda nos parâmetros de validação.

- Fase (fase)
  - A fase inicial padrão do quadro.
- Porta associada (associatedGate)
  - As portas associadas ao quadro fornecido.
- Mapeamentos de Qubit (mapeamentos de qubit)
  - Os qubits associados ao frame fornecido.
- Propriedades específicas do QHP () qhpSpecificProperties
  - Um mapa opcional detalhando as propriedades existentes sobre o quadro específico do QHP.

SupportsDynamicFrames:

Descreve se um quadro pode ser declarado em blocos `cal` ou `defcal` por meio da função OpenPulse `newframe`. Se isso for falso, somente os quadros listados na estrutura do quadro poderão ser usados no programa.

SupportedFunctions:

Descreve as funções OpenPulse que são compatíveis pelo dispositivo, além dos argumentos, tipos de argumentos e tipos de retorno associados às funções fornecidas. Para ver exemplos de uso das OpenPulse funções, consulte a OpenPulseespecificação. No momento, o Braket suporta:

- shift_phase
  - Desloca a fase de um quadro por um valor especificado
- set_phase
  - Define a fase do quadro para o valor especificado
- fases_troca
  - Troca as fases entre dois quadros.
- frequência_de-turno
  - Desloca a frequência de um quadro por um valor especificado
- frequência_definida
  - Define a frequência do quadro para o valor especificado

- jogar
  - Agenda uma forma de onda
- capture_v0
  - Retorna o valor em um quadro de captura para um registro de bits

SupportedQhpTemplateWaveforms:

Descreve as funções de forma de onda pré-criadas disponíveis no dispositivo e os argumentos e tipos associados. Por padrão, o Braket Pulse oferece rotinas de forma de onda pré-criadas em todos os dispositivos, que são:

Constante

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

`τ` é o comprimento da forma de onda e `iq` é um número complexo.

```
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` é o comprimento da forma de onda, σ é a largura da gaussiana e `A` é a amplitude. Se definido `ZaE` como `True`, o gaussiano é deslocado e redimensionado de forma que seja igual a zero no início e no final da forma de onda e atinja o máximo `A`.

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

DRAG 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` é o comprimento da forma de onda, σ é a largura gaussiana, β é um parâmetro livre e `A` é a amplitude. Se definido `ZaE` como `True`, o Gaussiano de Remoção de Derivativa por Porta Adiabática

(DRAG) é deslocado e reescalado de forma que seja igual a zero no início e no fim da forma de onda, e a parte real atinge o máximo `A`. Para obter mais informações sobre a forma de onda DRAG, consulte o artigo Pulsos simples para eliminação de vazamento em qubits fracamente não lineares.

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

Quadrado Erf

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

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

Onde `L` é o comprimento, `W` é a largura da forma de onda, σ define a rapidez com que as bordas sobem e descem, t_1`=(L-W)/2` e t_2`2=(L+W)/2`, `A` é a amplitude. Se definido `ZaE` como `True`, o gaussiano é deslocado e redimensionado de forma que seja igual a zero no início e no final da forma de onda e atinja o máximo `A`. A equação a seguir é a versão redimensionada da forma de onda.

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

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

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

SupportsLocalPulseElements:

Descreve se elementos de pulso, como portas, estruturas e formas de onda, podem ser definidos localmente em blocos `defcal`. Se o valor for `false`, os elementos devem ser definidos em blocos `cal`.

SupportsNonNativeGatesWithPulses:

Descreve se podemos ou não usar portas não nativas em combinação com programas de pulso. Por exemplo, você não pode usar uma porta não nativa como uma porta `H` em um programa sem primeiro definir a porta de entrada `defcal` para o qubit usado. Você pode encontrar a lista de chaves `nativeGateSet` de portas nativas nos recursos do dispositivo.

ValidationParameters:

Descreve os limites de validação do elemento de pulso, incluindo:

- Valores de escala máxima/amplitude máxima para formas de onda (arbitrárias e pré-construídas)
- Largura de banda de frequência máxima da frequência central fornecida em Hz
- Pulso mínimo length/duration em segundos
- Pulso máximo length/duration em segundos

Operações, resultados e tipos de resultados suportados com o OpenQASM

Para descobrir quais recursos do OpenQASM 3.0 cada dispositivo suporta, você pode consultar a chave `braket.ir.openqasm.program` no campo `action` na saída de recursos do dispositivo. Por exemplo, a seguir estão as operações compatíveis e os tipos de resultados disponíveis para o simulador SV1 Braket State Vector.

```
...
  "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 ruídos com o OpenQASM 3.0

Para simular o ruído com o OpenQASM3, você usa instruções pragmáticas para adicionar operadores de ruído. Por exemplo, para simular a versão ruidosa do programa GHZ fornecida anteriormente, você pode enviar o seguinte 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;
```

As especificações de todos os operadores de ruído pragmático com suporte são fornecidas na lista a seguir.

```
#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 gerar um operador Kraus, você pode iterar por meio de uma lista de matrizes, imprimindo cada elemento da matriz como uma expressão complexa.

Ao usar operadores de Kraus, lembre-se do seguinte:

- O número de qubits não deve exceder 2. A definição atual nos esquemas define esse limite.
- O tamanho da lista de argumentos deve ser múltiplo de 8. Isso significa que ele deve ser composto apenas por matrizes 2x2.
- O comprimento total não excede matrizes 22^{*num_qubits}. Isso significa 4 matrizes para 1 qubit e 16 para 2 qubits.
- Todas as matrizes fornecidas preservam traços completamente positivos (CPTP).
- O produto dos operadores de Kraus com seus conjugados de transposição precisa se somar a uma matriz de identidade.

Reconexão Qubit com o OpenQASM 3.0

O Amazon Braket suporta a notação qubit física no OpenQASM em dispositivos (para saber mais, consulte Rigetti esta página). Ao usar o qubits físico com a estratégia de reconexão ingênua, certifique-se de que os qubits estejam conectados ao dispositivo selecionado. Como alternativa, se os registros qubit forem usados em vez disso, a estratégia de reconexão PARCIAL será ativada por padrão nos dispositivos Rigetti.

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

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

measure $0;
```

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

Compilação literal com o OpenQASM 3.0

Quando você executa um circuito quântico em computadores quânticos fornecidos por fornecedores como Rigetti e IonQ, você pode direcionar o compilador para executar seus circuitos exatamente como definido, sem nenhuma modificação. Esse recurso é conhecido como compilação literal. Com os dispositivos Rigetti, você pode especificar com precisão o que é preservado: um circuito inteiro ou apenas partes específicas dele. Para preservar somente partes específicas de um circuito, você precisará usar portas nativas dentro das regiões preservadas. Atualmente, o IonQ suporta apenas compilação literal para todo o circuito, portanto, todas as instruções no circuito precisam ser incluídas em uma caixa literal.

Com o OpenQASM, você pode especificar explicitamente um pragma literal em torno de uma caixa de código que é então deixada intocada e não otimizada pela rotina de compilação de baixo nível do hardware. O código de exemplo a seguir mostra como usar a diretiva `#pragma braket verbatim` para fazer isso.

```
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 obter informações mais detalhadas sobre o processo de compilação literal, incluindo exemplos e melhores práticas, consulte o caderno de amostra de compilação Verbatim disponível no repositório github. amazon-braket-examples

O console do Braket

As tarefas do OpenQASM 3.0 estão disponíveis e podem ser gerenciadas no console do Amazon Braket. No console, você tem a mesma experiência ao enviar tarefas quânticas no OpenQASM 3.0 que tinha ao enviar tarefas quânticas existentes.

Recursos adicionais do

O OpenQASM está disponível em todas as regiões do Amazon Braket.

Para ver um exemplo de caderno para começar a usar o OpenQASM no Amazon Braket, consulte Tutoriais do Braket. GitHub

Gradientes de computação com o OpenQASM 3.0

O Amazon Braket suporta o cálculo de gradientes em simuladores sob demanda e locais quando executado no modo (exato) `shots=0`. Isso é obtido por meio do uso do método de diferenciação adjunta. Para especificar o gradiente que você deseja calcular, você pode fornecer o pragma apropriado, conforme demonstrado no código no exemplo a seguir.

```
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
```

Em vez de listar explicitamente todos os parâmetros individuais, você também pode especificar a palavra-chave `all` dentro do pragma. Isso calculará o gradiente em relação a todos os parâmetros `input` listados, o que pode ser uma opção conveniente quando o número de parâmetros é muito grande. Nesse caso, o pragma será semelhante ao código do exemplo a seguir.

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

Todos os tipos observáveis são compatíveis com a implementação do OpenQASM 3.0 do Amazon Braket, incluindo operadores individuais, produtos tensores, observáveis Hermitian e `Sum` observáveis. O operador específico que você deseja usar ao calcular gradientes deve ser encapsulado na função `expectation()`, e os qubits sobre os quais cada termo do observável atua devem ser especificados explicitamente.

Medindo qubits específicos com o OpenQASM 3.0

O simulador vetorial estadual local e o simulador de matriz de densidade local fornecidos pela Amazon Braket oferecem suporte ao envio de programas em que um subconjunto OpenQASM dos qubits do circuito pode ser medido seletivamente. Essa capacidade, geralmente chamada de medição parcial, permite cálculos quânticos mais direcionados e eficientes. Por exemplo, no trecho de código a seguir, você pode criar um circuito de dois qubits e optar por medir somente o primeiro qubit, deixando o segundo qubit não medido.

```
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];
"""
```

Neste exemplo, temos um circuito quântico com dois qubits `q[0]` e `q[1]`, mas estamos interessados apenas em medir o estado do primeiro qubit. Isso é obtido pela linha `b[0] = measure q[0]`, que mede o estado do qubit [0] e armazena o resultado no bit clássico b [0]. Para executar esse cenário de medição parcial, podemos executar o código a seguir no simulador vetorial estadual local fornecido pela Amazon Braket.

```
from braket.devices import LocalSimulator

local_sim = LocalSimulator()
partial_measure_local_sim_task =
 local_sim.run(OpenQASMProgram(source=partial_measure_qasm), shots = 10)
partial_measure_local_sim_result = partial_measure_local_sim_task.result()
print(partial_measure_local_sim_result.measurement_counts)
```

```
print("Measured qubits: ", partial_measure_local_sim_result.measured_qubits)
```

Você pode verificar se um dispositivo suporta medição parcial inspecionando o campo `requiresAllQubitsMeasurement` em suas propriedades de ação; se for`False`, então a medição parcial é compatível.

```
from braket.devices import Devices

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

Aqui, `requiresAllQubitsMeasurement` é `False`, o que indica que nem todos os qubits devem ser medidos.

Explorar capacidades experimentais

Os recursos experimentais fornecem acesso a hardware com disponibilidade limitada e novos recursos de software emergentes. Esses recursos podem afetar o desempenho do dispositivo além das especificações padrão. Você pode habilitar automaticamente os recursos experimentais de software por tarefa por meio do Amazon Braket SDK.

Para usar recursos experimentais, especifique o `experimental_capabilities` parâmetro ao criar tarefas quânticas. Defina esse parâmetro `"ALL"` para ativar todos os recursos experimentais disponíveis para essa tarefa. O exemplo a seguir mostra como habilitar recursos experimentais ao executar um circuito em um dispositivo:

```
from braket.aws import AwsDevice

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

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

> **Note**
>
> Esses recursos são experimentais e podem ser alterados sem aviso prévio. O desempenho do dispositivo pode diferir das especificações publicadas e os resultados podem variar das

operações padrão. Você deve habilitar explicitamente os recursos experimentais para cada tarefa. As tarefas sem esse parâmetro usarão somente os recursos padrão do dispositivo.

Nesta seção:

- Acesso ao desvio local em Aquila QuEra
- Acesso a geometrias altas em Aquila QuEra
- Acesso a geometrias estreitas em Aquila QuEra
- Circuitos dinâmicos em dispositivos IQM

Acesso ao desvio local em Aquila QuEra

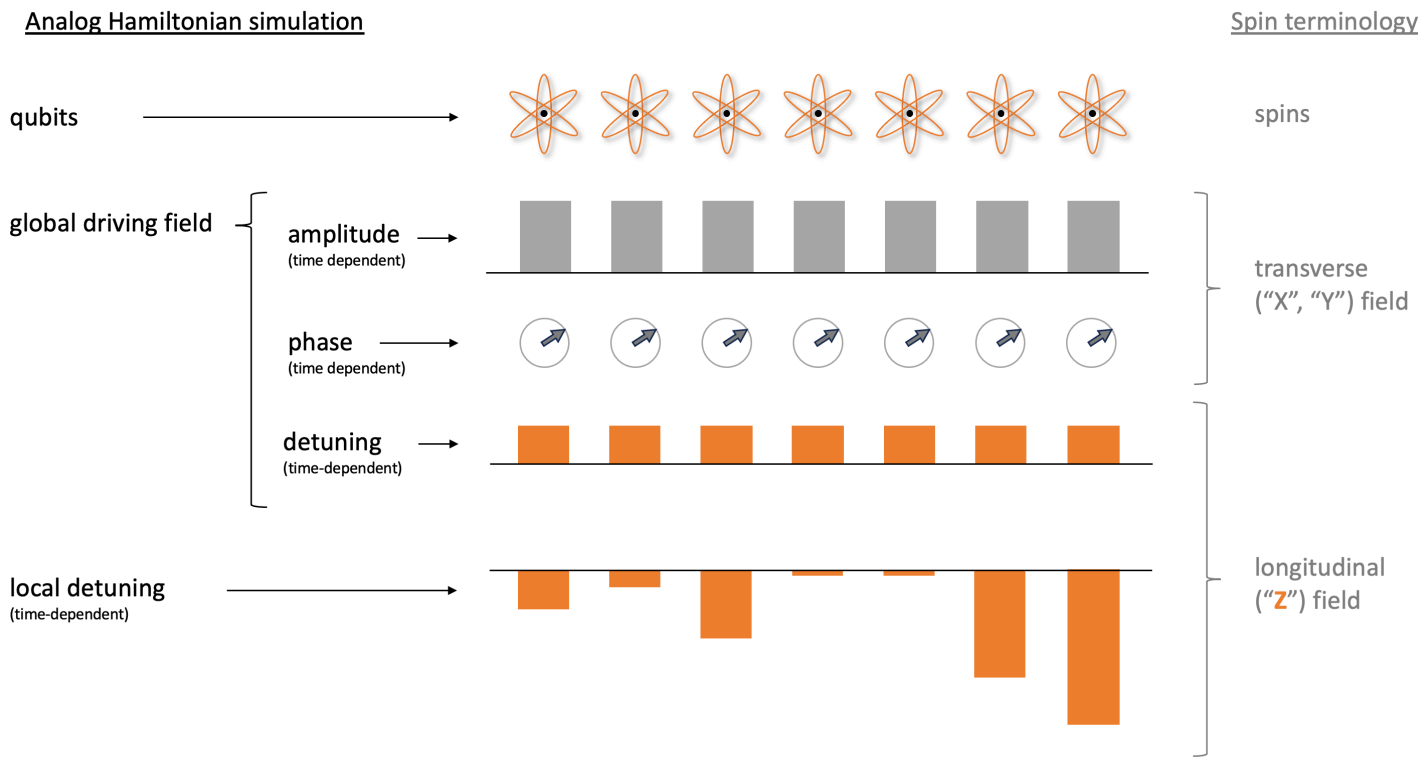
O desajuste local (LD) é um novo campo de controle dependente do tempo com um padrão espacial personalizável. O campo LD afeta os qubits de acordo com um padrão espacial personalizável, criando diferentes hamiltonianos para qubits diferentes, além do que o campo de condução uniforme e a interação Rydberg-Rydberg podem criar.

Restrições:

O padrão espacial do campo de desajuste local é personalizável para cada programa AHS, mas é constante ao longo de um programa. A série temporal do campo de desajuste local deve começar e terminar em zero, com todos os valores sendo menores ou iguais a zero. Além disso, os parâmetros do campo de desajuste local são limitados por restrições numéricas, que podem ser visualizadas por meio do SDK do Braket na seção `aquila_device.properties.paradigm.rydberg.rydbergLocal` de propriedades específicas do dispositivo.

Limitações:

Ao executar programas quânticos que usam o campo de desajuste local (mesmo que sua magnitude seja definida como zero constante no hamiltoniano), o dispositivo experimenta uma decoerência mais rápida do que o tempo T2 listado na seção de desempenho das propriedades do Aquila. Quando desnecessário, é uma boa prática omitir o campo de desajuste local do hamiltoniano do programa AHS.



Exemplos:

1. Simulação do efeito do campo magnético longitudinal não uniforme em sistemas de spin

   Embora a amplitude e a fase do campo de acionamento tenham o mesmo efeito nos qubits que o campo magnético transversal nos giros, a soma do desajuste do campo de acionamento e do desajuste local produz o mesmo efeito nos qubits que o campo longitudinal nos giros. Com o controle espacial sobre o campo de desajuste local, sistemas de rotação mais complexos podem ser simulados.
2. Preparando estados iniciais de não equilíbrio

   O exemplo de caderno Simulando a teoria do calibre de rede com átomos de Rydberg mostra como impedir que o átomo central de um arranjo linear de 9 átomos seja excitado durante o recozimento do sistema em direção à fase ordenada Z2. Após a etapa de preparação, o campo de desajuste local é reduzido e o programa AHS continua simulando a evolução temporal do sistema a partir desse estado específico de não equilíbrio.
3. Resolvendo problemas de otimização ponderada

   O exemplo de caderno Conjunto independente de peso máximo (MWIS) mostra como resolver um problema de MWIS no Aquila. O campo de desajuste local é usado para definir os pesos nos

nós do gráfico do disco unitário, cujas bordas são realizadas pelo efeito de bloqueio de Rybderg. Partindo do estado fundamental uniforme e aumentando gradualmente o campo de desajuste local, o sistema faz a transição do sistema para o estado fundamental do MWIS Hamiltonian para encontrar soluções para o problema.

Acesso a geometrias altas em Aquila QuEra

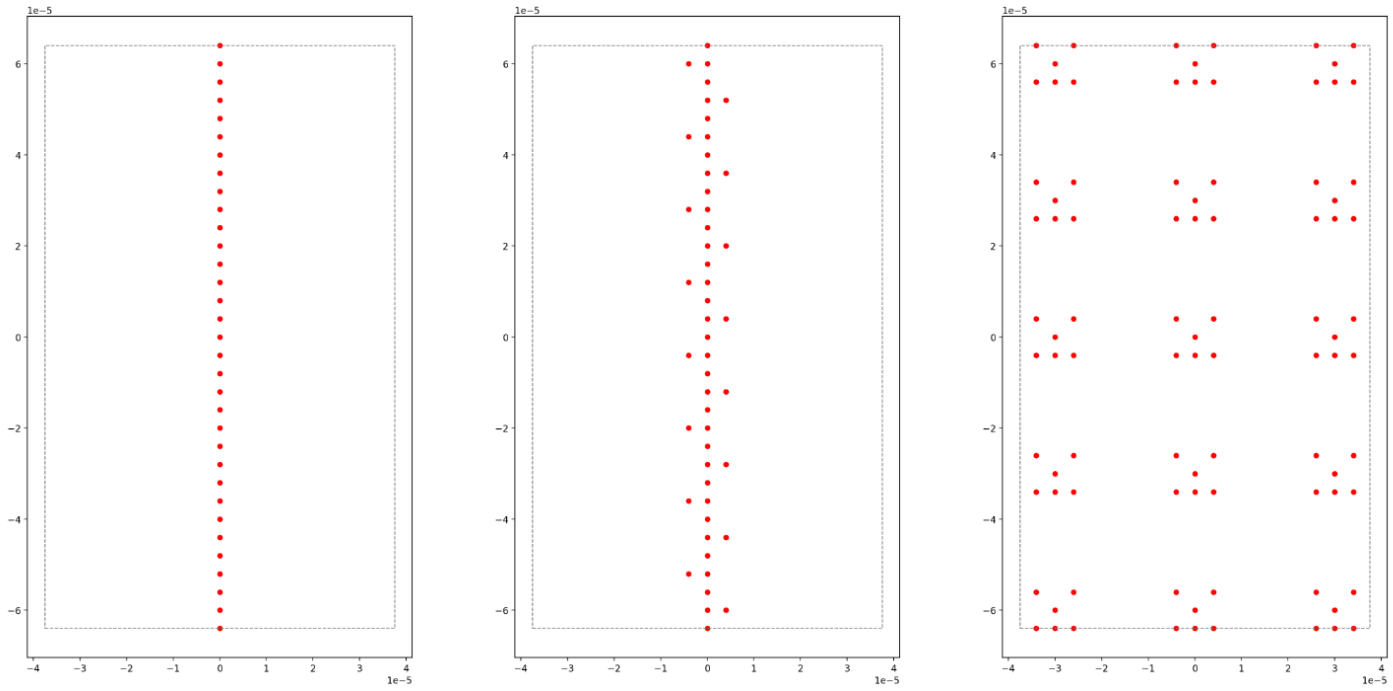
O recurso de geometrias altas permite especificar geometrias com maior altura. Com esse recurso, os arranjos de átomos de seus programas AHS podem abranger um comprimento adicional na direção y além dos recursos regulares do Aquila.

Restrições:

A altura máxima para geometrias altas é de 0,000128 m (128 um).

Limitações:

Quando esse recurso experimental estiver ativado em sua conta, os recursos mostrados na página de propriedades do dispositivo e na chamada `GetDevice` continuarão refletindo o limite inferior normal da altura. Quando um programa AHS usa arranjos de átomos que vão além das capacidades normais, espera-se que o erro de preenchimento aumente. Você encontrará um número elevado de 0s inesperados na parte `pre_sequence` do resultado da tarefa, diminuindo a chance de obter um arranjo perfeitamente inicializado. Esse efeito é mais forte em linhas com muitos átomos.



Exemplos:

1. Arranjos 1d e quase 1d maiores

   Cadeias de átomos e arranjos em forma de escada podem ser estendidos para números maiores de átomos. Ao orientar a direção longa paralelamente a y, é possível programar instâncias mais longas desses modelos.

2. Mais espaço para multiplexar a execução de tarefas com geometrias pequenas

   O exemplo de caderno Tarefas quânticas paralelas no Aquila mostra como tirar o máximo proveito da área disponível: colocando cópias multiplexadas da geometria em questão em um arranjo de átomos. Com a área mais disponível, mais cópias podem ser colocadas.

Acesso a geometrias estreitas em Aquila QuEra

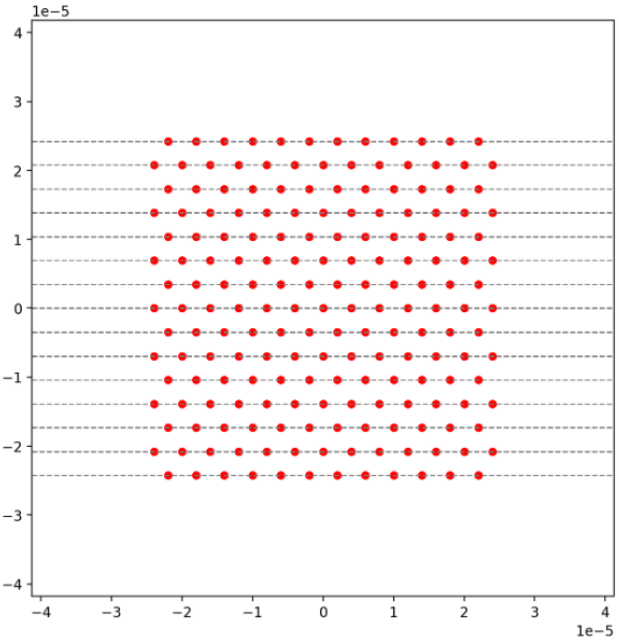
O recurso de geometrias estreitas permite especificar geometrias com menor espaçamento entre as linhas vizinhas. Em um programa AHS, os átomos são organizados em fileiras, separados por um espaçamento vertical mínimo. A coordenada y de quaisquer sítios de dois átomos deve ser zero (mesma linha) ou diferir em mais do que o espaçamento mínimo entre linhas (linha diferente). Com a capacidade de geometrias estreitas, o espaçamento mínimo entre linhas é reduzido, permitindo a criação de arranjos de átomos mais apertados. Embora essa extensão não altere o requisito mínimo de distância euclidiana entre os átomos, ela permite a criação de redes onde átomos distantes ocupam fileiras vizinhas mais próximas umas das outras, um exemplo notável é a rede triangular.

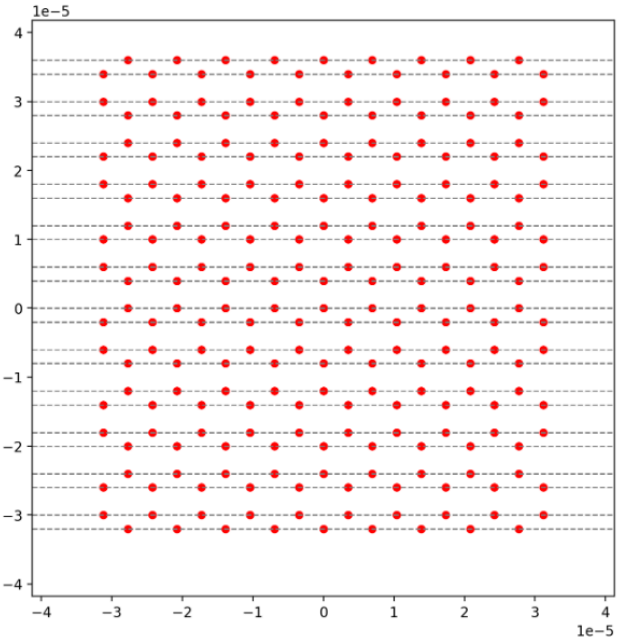
Restrições:

O espaçamento mínimo entre linhas para geometrias estreitas é de 0,000002 m (2 um).

Limitações:

Quando esse recurso experimental estiver ativado em sua conta, os recursos mostrados na página de propriedades do dispositivo e na chamada `GetDevice` continuarão refletindo o limite inferior normal da altura. Quando um programa AHS usa arranjos de átomos que vão além das capacidades normais, espera-se que o erro de preenchimento aumente. Os clientes encontrarão um número elevado de 0s inesperados na parte `pre_sequence` do resultado da tarefa, o que, por sua vez, diminuirá a chance de obter um arranjo perfeitamente inicializado. Esse efeito é mais forte em linhas com muitos átomos.





Exemplos:

1. Redes não retangulares com pequenas constantes de rede

   O espaçamento mais estreito entre linhas permite a criação de redes onde o vizinho mais próximo de alguns átomos está na direção diagonal. Exemplos notáveis são treliças triangulares, hexagonais e Kagome e alguns quase-cristais.
2. Família ajustável de treliças

   Nos programas AHS, as interações são ajustadas ajustando a distância entre pares de átomos. Um espaçamento mais estreito entre linhas permite ajustar as interações de diferentes pares de átomos em relação um ao outro com mais liberdade, uma vez que os ângulos e distâncias que definem a estrutura do átomo são menos limitados pela restrição mínima de espaçamento entre linhas. Um exemplo notável é a família de treliças Shastry-Sutherland com diferentes comprimentos de ligação.

Circuitos dinâmicos em dispositivos IQM

Os circuitos dinâmicos nos dispositivos IQM permitem medições de meio circuito (MCM) e operações de avanço. Esses recursos permitem que pesquisadores e desenvolvedores quânticos implementem algoritmos quânticos avançados com lógica condicional e recursos de reutilização de qubits. Esse

recurso experimental ajuda a explorar algoritmos quânticos com maior eficiência de recursos e a estudar esquemas de mitigação e correção de erros quânticos.

Instruções principais:

- `measure_ff`: implementa a medição para controle de avanço, medindo um qubit e armazenando o resultado com uma chave de feedback.
- `cc_prx`: implementa uma rotação controlada classicamente que se aplica somente quando o resultado associado à chave de feedback fornecida mede um estado |1#.

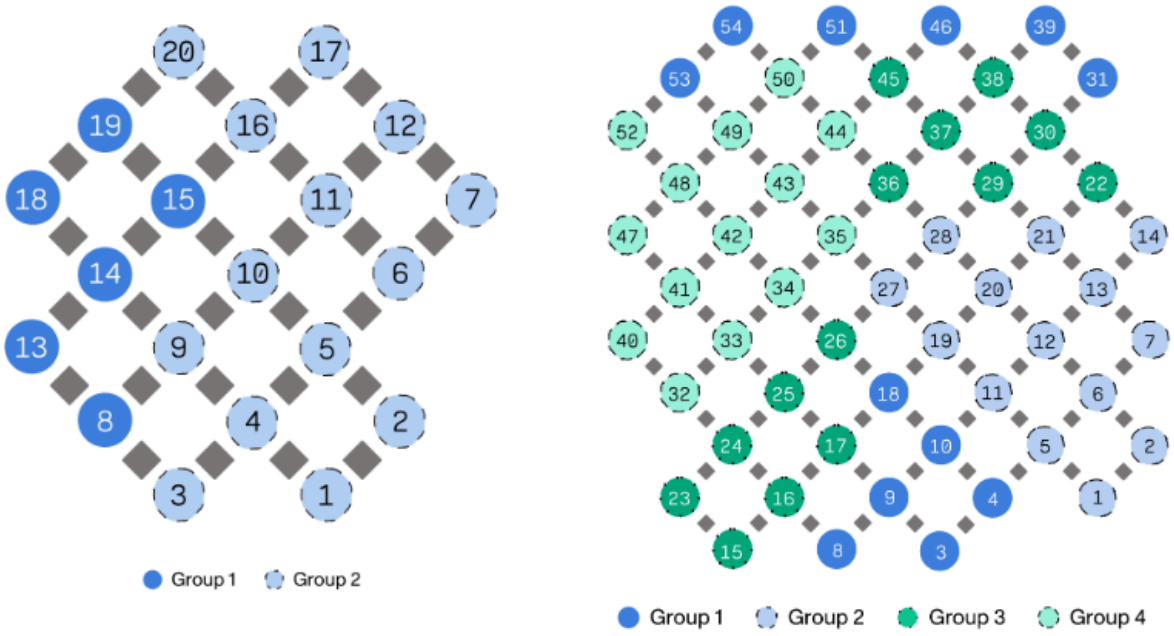
O Amazon Braket oferece suporte a circuitos dinâmicos por meio de OpenQASM, Amazon Braket SDK e Amazon Braket Qiskit Provider.

Restrições:

1. As chaves de feedback nas instruções `measure_ff` devem ser exclusivas.
2. A `cc_prx` deve acontecer depois de `measure_ff` com a mesma chave de feedback.
3. Em um único circuito, o avanço em um qubit só pode ser controlado por um qubit, sozinho ou por outro qubit. Em circuitos diferentes, você pode ter diferentes pares de controle.
   a. Por exemplo, se o qubit 1 for controlado pelo qubit 2, ele não poderá ser controlado pelo qubit 3 no mesmo circuito. Não há restrição sobre quantas vezes o controle é aplicado entre o qubit 1 e o qubit 2. O Qubit 2 pode ser controlado pelo qubit 3 (ou qubit 1), a menos que uma reinicialização ativa tenha sido executada no qubit 2.
4. O controle só pode ser aplicado a qubits dentro do mesmo grupo. Os grupos de qubits dos dispositivos Emerald e IQM Garnet estão nas imagens a seguir.
5. Os programas com esses recursos devem ser enviados como programas textuais. Para saber mais sobre programas textuais, consulte Compilação literal com o OpenQASM 3.0.

Limitações:

O MCM só pode ser usado para controle de avanço em um programa. Os resultados do MCM (0 ou 1) não são retornados como parte do resultado de uma tarefa.



Essas imagens exibem os agrupamentos de qubits dos dois dispositivos IQM. O dispositivo Garnet de 20 qubits contém 2 grupos de qubits, enquanto o dispositivo Emerald de 54 qubits contém 4 grupos de qubits.

Exemplos:

1. Reutilização de qubit por meio de reinicialização ativa

   O MCM com operações de reinicialização condicional permite a reutilização de qubits em uma única execução de circuito. Isso reduz os requisitos de profundidade do circuito e melhora a utilização dos recursos do dispositivo quântico.

2. Proteção ativa contra inversão de bits

   Os circuitos dinâmicos detectam erros de inversão de bits e aplicam operações corretivas com base nos resultados da medição. Essa implementação serve como um experimento de detecção de erros quânticos.

3. Experimentos de teletransporte

   O teletransporte estadual transfere estados de qubit usando operações quânticas locais e informações clássicas de. MCMs O teletransporte de portas implementa portas entre qubits sem operações quânticas diretas. Esses experimentos demonstram sub-rotinas fundamentais em

três áreas principais: correção de erros quânticos, computação quântica baseada em medição e comunicação quântica.

4. Simulação de sistemas quânticos abertos

   Os circuitos dinâmicos modelam o ruído em sistemas quânticos por meio de qubit de dados e entrelaçamento ambiental e medições ambientais. Essa abordagem usa qubits específicos para representar dados e elementos do ambiente. Um canal de ruído pode ser projetado pelas portas e medições aplicadas no ambiente.

Para obter mais informações sobre o uso de circuitos dinâmicos, consulte exemplos adicionais no repositório de cadernos do Amazon Braket.

Controle de pulso no Amazon Braket

Pulsos são os sinais analógicos que controlam os qubits em um computador quântico. Com determinados dispositivos no Amazon Braket, você pode acessar o recurso de controle de pulso para enviar circuitos usando pulsos. Você pode acessar o controle de pulso por meio do SDK do Braket, usando o OpenQASM 3.0 ou diretamente pelo Braket. APIs Primeiro, apresente alguns conceitos-chave para controle de pulso no Braket.

Nesta seção:

- Quadros
- Portas
- Formas de onda
- Trabalhando com o Hello Pulse
- Acessando portas nativas usando pulsos

Quadros

Um quadro é uma abstração de software que atua tanto como um relógio dentro do programa quântico quanto como uma fase. A hora do relógio é incrementada em cada uso e em um sinal portador com estado definido por uma frequência. Ao transmitir sinais para o qubit, um quadro determina a frequência portadora do qubit, o deslocamento de fase e a hora em que o envelope da forma de onda é emitido. No Braket Pulse, a construção de quadros depende do dispositivo, da frequência e da fase. Dependendo do dispositivo, você pode escolher um quadro predefinido ou instanciar novos quadros fornecendo uma porta.

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

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

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

Portas

Uma porta é uma abstração de software que representa qualquer componente de input/output hardware que controla qubits. Ele ajuda os fornecedores de hardware a fornecer uma interface com a qual os usuários podem interagir para manipular e observar qubits. As portas são caracterizadas por uma única string que representa o nome do conector. Essa string também expõe um incremento mínimo de tempo que especifica com que precisão podemos definir as formas de onda.

```
from braket.pulse import Port

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

Formas de onda

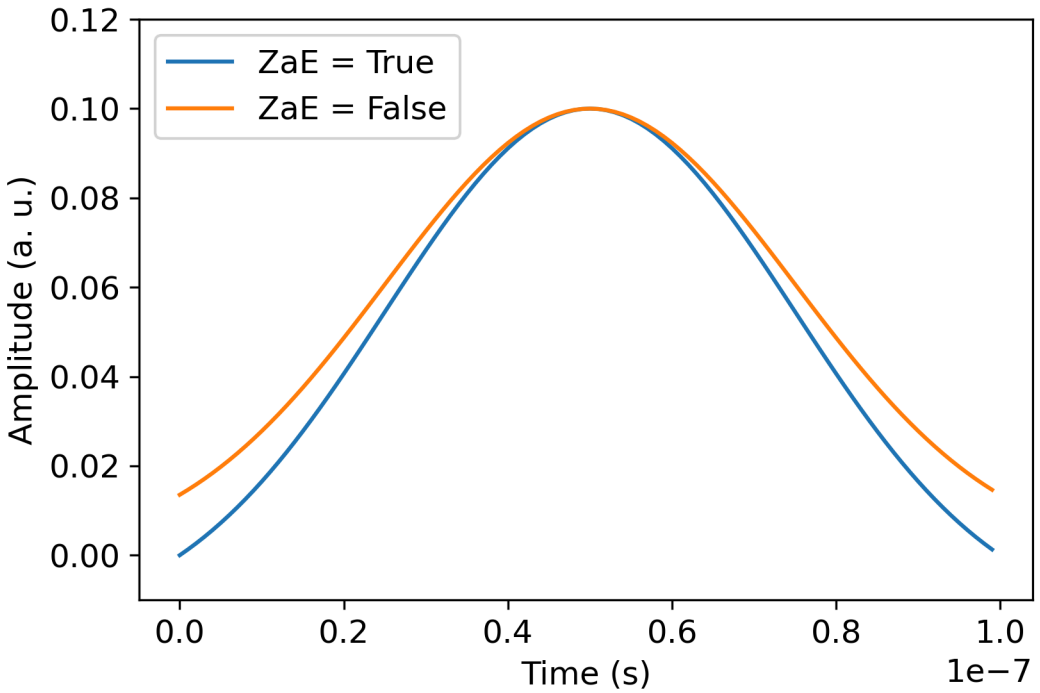
Uma forma de onda é um envelope dependente do tempo que podemos usar para emitir sinais em uma porta de saída ou capturar sinais por meio de uma porta de entrada. Você pode especificar suas formas de onda diretamente por meio de uma lista de números complexos ou usando um modelo de forma de onda para gerar uma lista do fornecedor do hardware.

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

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

O Braket Pulse fornece uma biblioteca padrão de formas de onda, incluindo uma forma de onda constante, uma forma de onda gaussiana e uma forma de onda de remoção derivada por porta

adiabática (DRAG). Você pode recuperar os dados da forma de onda por meio da função `sample` para desenhar a forma da onda, conforme mostrado no exemplo a seguir.

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

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

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

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



A imagem anterior mostra as formas de onda gaussianas criadas a partir de `GaussianWaveform`. Escolhemos um comprimento de pulso de 100 ns, uma largura de 25 ns e uma amplitude de 0,1 (unidades arbitrárias). As formas de onda estão centralizadas na janela de pulso. `GaussianWaveform`aceita um argumento booleano `zero_at_edges` (ZaE na legenda). Quando definido como `True`, esse argumento desloca a forma de onda gaussiana de forma que os pontos

em t=0 e t= `length` estejam em zero e redimensiona sua amplitude de forma que o valor máximo corresponda ao argumento `amplitude`.

Trabalhando com o Hello Pulse

Nesta seção, você aprenderá a caracterizar e a construir uma única porta de qubit usando a pulsação em um dispositivo Rigetti. A aplicação de um campo eletromagnético a um qubit leva à oscilação Rabi, alternando os qubits entre seu estado 0 e 1 estado. Com o comprimento e a fase calibrados do pulso, a oscilação Rabi pode calcular portas de um único qubit. Aqui, determinaremos o comprimento de pulso ideal para medir um pulso pi/2, um bloco elementar usado para criar sequências de pulso mais complexas.

Primeiro, para criar uma sequência de pulsos, importe a classe `PulseSequence`.

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

import numpy as np
```

Em seguida, instancie um novo dispositivo Braket usando o Amazon Resource Name (ARN) da QPU. O bloco de código a seguir usa Rigetti Ankaa-3.

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

A sequência de pulsos a seguir inclui dois componentes: reproduzir uma forma de onda e medir um qubit. A sequência de pulsos geralmente pode ser aplicada aos quadros. Com algumas exceções, como barreira e atraso, que podem ser aplicados aos qubits. Antes de construir a sequência de pulsos, você deve recuperar os quadros disponíveis. A estrutura de acionamento é usada para aplicar o pulso para a oscilação Rabi e a estrutura de leitura é para medir o estado do qubit. Este exemplo usa os quadros do qubit 25.

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

Agora, crie a forma de onda que será reproduzida na estrutura da unidade. O objetivo é caracterizar o comportamento dos qubits para diferentes comprimentos de pulso. Você reproduzirá uma forma de onda com comprimentos diferentes a cada vez. Em vez de instanciar uma nova forma de onda

a cada vez, use a sequência de pulsos `FreeParameter` compatível com o Braket. Você pode criar a forma de onda e a sequência de pulso uma vez com parâmetros livres e, em seguida, executar a mesma sequência de pulso com valores de entrada diferentes.

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

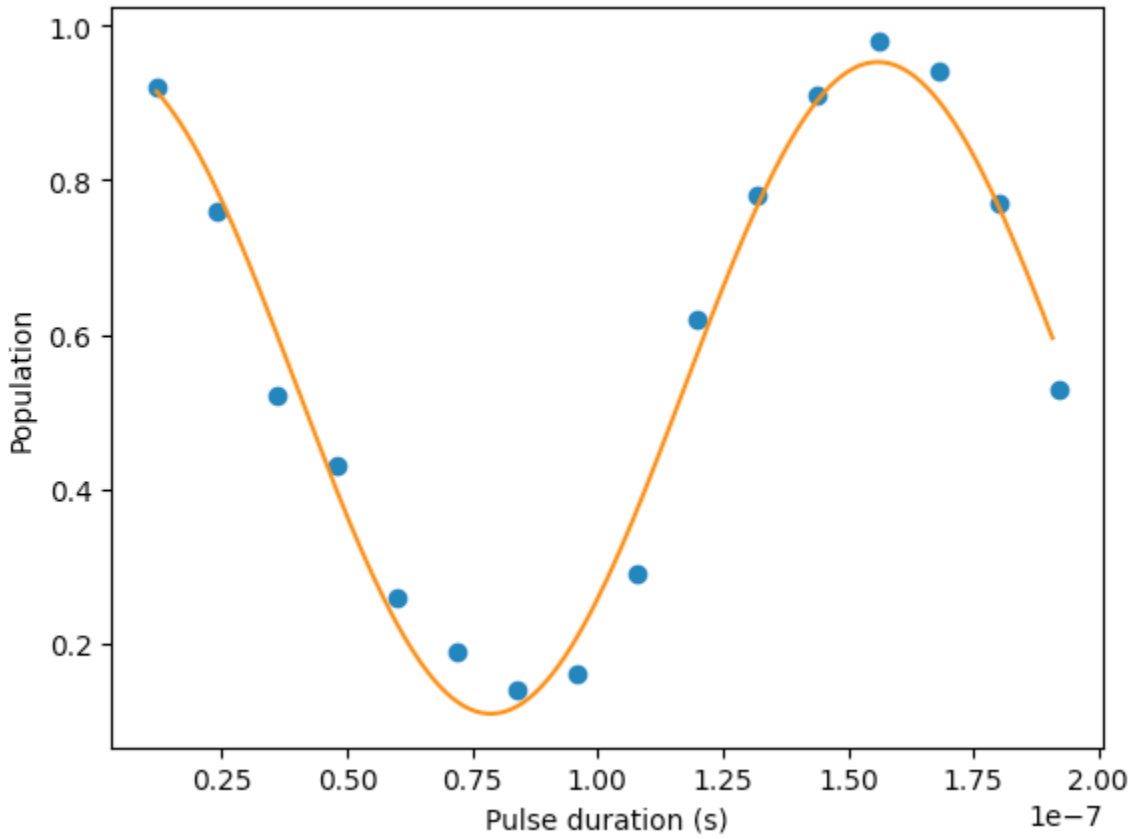
Finalmente, coloque-os juntos como uma sequência de pulsos. Na sequência de pulsos, `play` reproduz a forma de onda especificada na estrutura da unidade e `capture_v0` mede o estado a partir da estrutura de leitura.

```
pulse_sequence = (
    PulseSequence()
    .play(drive_frame, waveform)
    .capture_v0(readout_frame)
)
```

Digitalize uma faixa de comprimento de pulso e envie-os para a QPU. Antes de executar as sequências de pulso em uma QPU, vincule o valor dos parâmetros livres.

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

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

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

As estatísticas da medição do qubit exibem a dinâmica oscilatória do qubit que oscila entre o estado 0 e o estado 1. A partir dos dados de medição, você pode extrair a frequência Rabi e ajustar o comprimento do pulso para implementar uma porta específica de 1 qubit. Por exemplo, a partir dos dados na figura abaixo, a periodicidade é de cerca de 154 ns. Portanto, uma porta de rotação pi/2 corresponderia à sequência de pulso com comprimento = 38,5ns.



Hello Pulse usando OpenPulse

OpenPulseé uma linguagem para especificar o controle em nível de pulso de um dispositivo quântico geral e faz parte da especificação OpenQASM 3.0. O Amazon Braket suporta o OpenPulse para programação direta de pulsos usando a representação OpenQASM 3.0.

Braket usa OpenPulse como representação intermediária subjacente para expressar pulsos em instruções nativas. OpenPulse suporta a adição de calibrações de instruções na forma de declarações `defcal` (abreviação de “definir calibração”). Com essas declarações, você pode especificar a implementação de uma instrução gate dentro de uma gramática de controle de nível inferior.

Você pode visualizar o OpenPulse programa de um Braket `PulseSequence` usando o comando a seguir.

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

Você também pode construir um OpenPulse programa diretamente.

```
from braket.ir.openqasm import Program

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

Crie um objeto `Program` com seu script. Em seguida, envie o programa para uma QPU.

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

program = Program(source=openpulse_script)

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

Acessando portas nativas usando pulsos

Os pesquisadores geralmente precisam saber exatamente como as portas nativas compatíveis com uma determinada QPU são implementadas como pulsos. As sequências de pulsos são cuidadosamente calibradas pelos fornecedores de hardware, mas acessá-las oferece aos pesquisadores a oportunidade de projetar portas melhores ou explorar protocolos para mitigação de erros, como extrapolação de ruído zero, ampliando os pulsos de portas específicas.

O Amazon Braket oferece suporte ao acesso programático às portas nativas da Rigetti.

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

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

calibrations = device.gate_calibrations
```

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

> **Note**
>
> Os fornecedores de hardware calibram periodicamente a QPU, geralmente mais de uma vez por dia. O SDK do Braket permite que você obtenha as calibrações de portas mais recentes.

```
device.refresh_gate_calibrations()
```

Para recuperar uma determinada porta nativa, como a porta RX ou XY, você precisa passar o objeto `Gate` e os qubits de interesse. Por exemplo, você pode inspecionar a implementação de pulso do RX (π/2) aplicado em qubit 0.

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

Você pode criar um conjunto filtrado de calibrações usando a função `filter`. Você passa por uma lista de portas ou uma lista de `QubitSet`. O código a seguir cria dois conjuntos que contêm todas as calibrações para RX (π/2) e para qubit 0.

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

Agora você pode fornecer ou modificar a ação das portas nativas anexando um conjunto de calibração personalizado. Por exemplo, considere o seguinte circuito.

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

Você pode executá-lo com uma calibração de porta `rx` personalizada no `qubit 0` para o gate ligado, passando um dicionário de objetos `PulseSequence` para o argumento de palavra-chave

`gate_definitions`. Você pode construir um dicionário a partir do atributo `pulse_sequences` do objeto `GateCalibrations`. Todas as portas não especificadas são substituídas pela calibração de pulso do fornecedor de hardware quântico.

```
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)
```

Simulação hamiltoniana analógica

A Simulação Hamiltoniana Analógica (AHS) é um paradigma emergente na computação quântica que difere significativamente do modelo tradicional de circuito quântico. Em vez de uma sequência de portas, em que cada circuito atua apenas em alguns qubits por vez. Um programa AHS é definido pelos parâmetros dependentes do tempo e do espaço do hamiltoniano em questão. O hamiltoniano de um sistema codifica seus níveis de energia e os efeitos das forças externas, que juntas governam a evolução temporal de seus estados. Para um sistema de N qubits, o hamiltoniano pode ser representado por uma matriz quadrada de $2^N X 2^N$ de números complexos.

Os dispositivos quânticos capazes de realizar AHS são projetados para aproximar de perto a evolução temporal de um sistema quântico sob um hamiltoniano personalizado, ajustando cuidadosamente seus parâmetros de controle interno. Por exemplo, ajustar a amplitude e os parâmetros de desajuste de um campo de condução coerente. O paradigma AHS é adequado para simular as propriedades estáticas e dinâmicas de sistemas quânticos com muitas partículas em interação, como na física da matéria condensada ou na química quântica. Unidades de processamento quântico (QPUs) criadas especificamente, como o dispositivo Aquila daQuEra, foram desenvolvidas para usar o poder do AHS e resolver problemas além do alcance das abordagens convencionais de computação quântica digital de maneiras inovadoras.

Nesta seção:

- Olá AHS: Execute sua primeira simulação hamiltoniana analógica
- Envie um programa analógico usando Aquila QuEra

Olá AHS: Execute sua primeira simulação hamiltoniana analógica

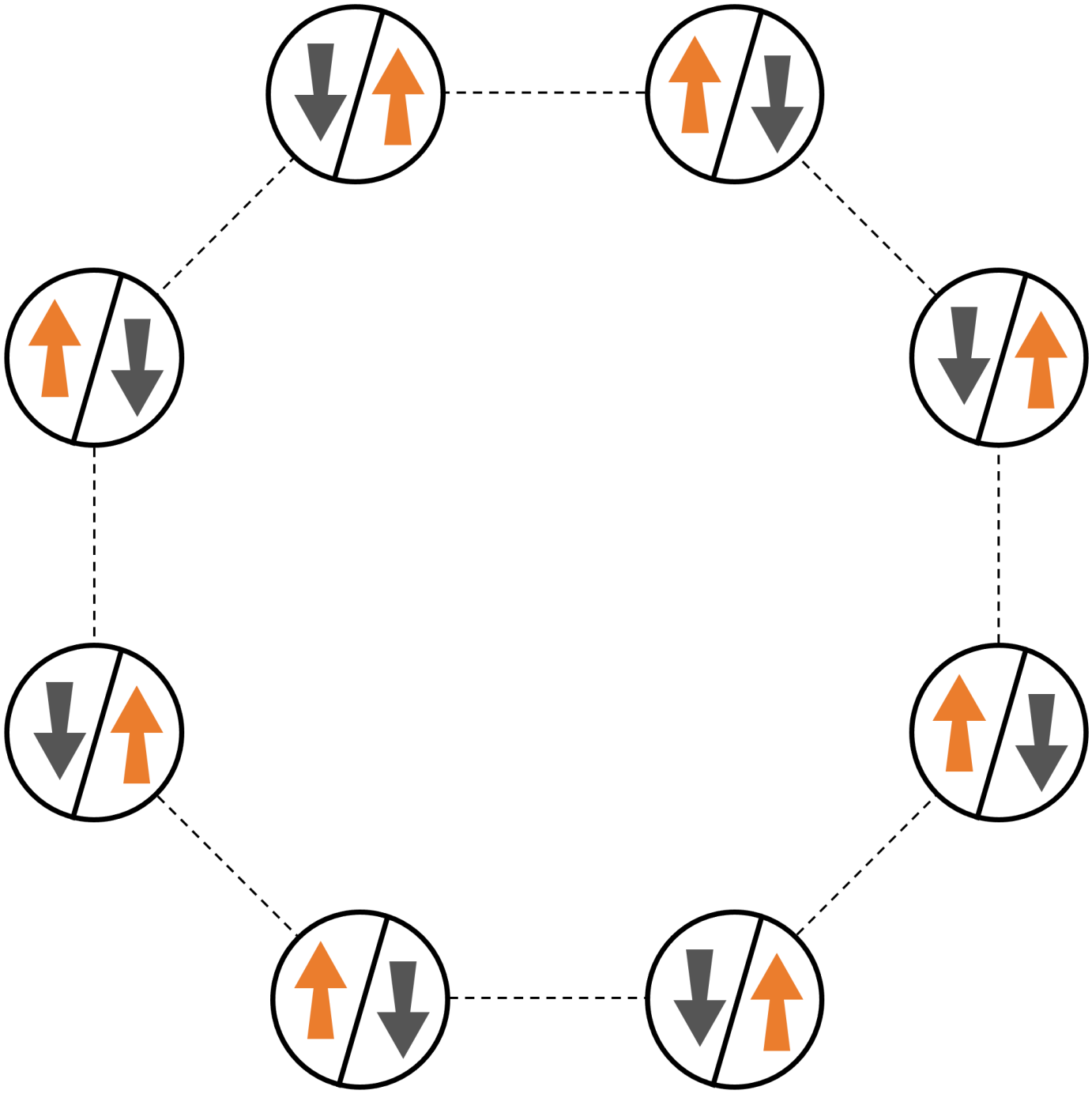
Esta seção fornece informações sobre como executar sua primeira simulação hamiltoniana analógica.

Nesta seção:

- Cadeia de rotação interativa
- Arranjo
- Interação
- Campo de condução
- Programa AHS
- Executando no simulador local
- Analisar resultados de simulação
- Executando no QuEra Aquila QPU
- Analisando os resultados da QPU
- Próximas etapas

Cadeia de rotação interativa

Para um exemplo canônico de um sistema de muitas partículas interagindo, vamos considerar um anel de oito giros (cada um dos quais pode estar nos estados “para cima” # e “para baixo” ↓#). Embora pequeno, esse sistema modelo já exibe um punhado de fenômenos interessantes de materiais magnéticos que ocorrem naturalmente. Neste exemplo, mostraremos como preparar a chamada ordem antiferromagnética, em que giros consecutivos apontam em direções opostas.



Arranjo

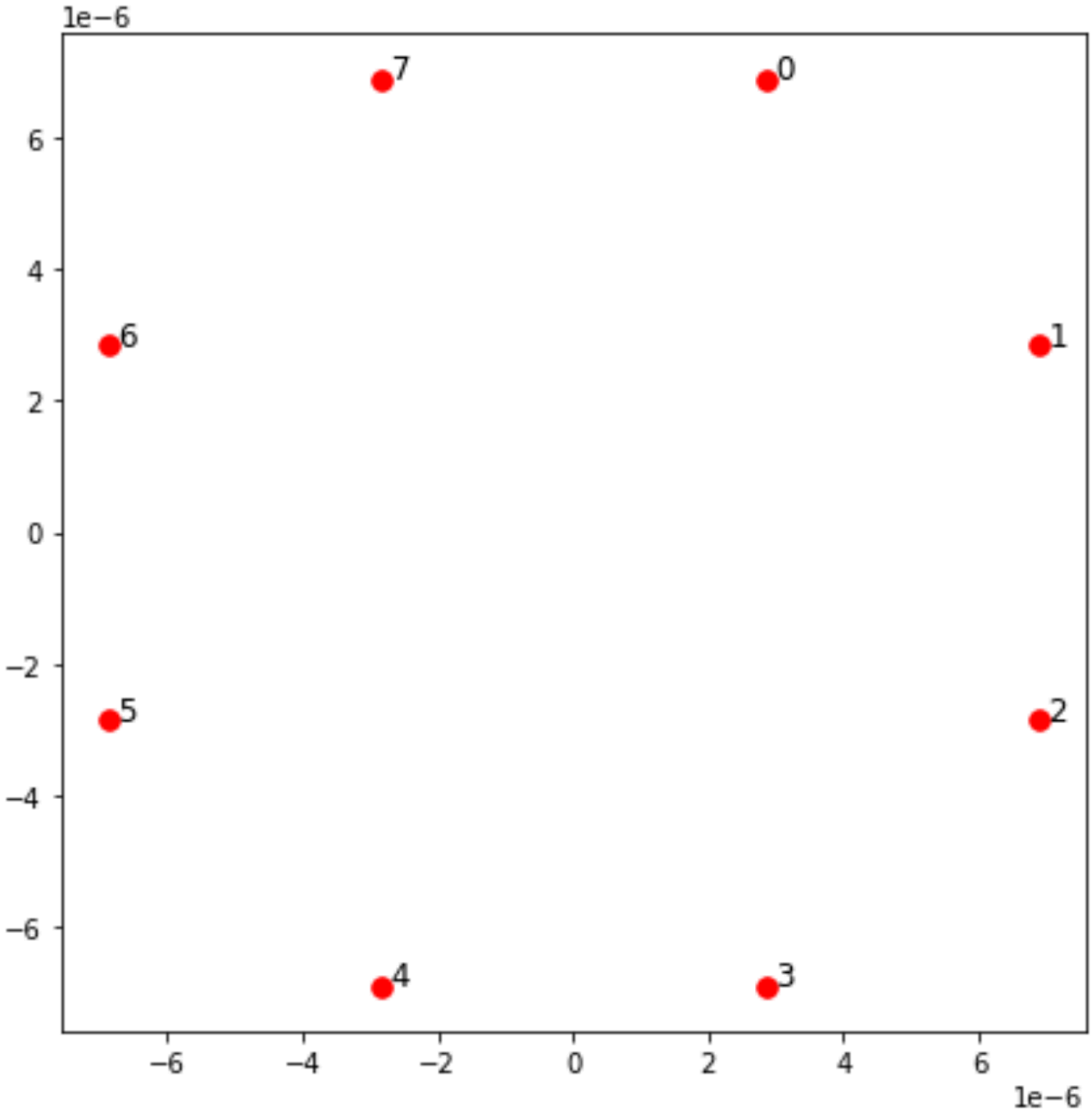
Usaremos um átomo neutro para representar cada spin, e os estados de spin “para cima” e “para baixo” serão codificados no estado excitado de Rydberg e no estado fundamental dos átomos, respectivamente. Primeiro, criaremos o arranjo 2-d. Podemos programar o anel de giros acima com o código a seguir.

Pré-requisitos: Você precisa instalar pip o SDK do Braket. (Se você estiver usando uma instância de caderno hospedada no Braket, esse SDK vem pré-instalado com os cadernos.) Para reproduzir os gráficos, você também precisa instalar separadamente o matplotlib com o comando shell `pip install matplotlib`.

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

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

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

com o qual também podemos traçar

```
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
```



Interação

Para preparar a fase antiferromagnética, precisamos induzir interações entre spins vizinhos. Usamos a interação van der Waals para isso, que é implementada nativamente por dispositivos de átomos neutros (como o dispositivo Aquila de QuEra). Usando a representação de spin, o termo hamiltoniano para essa interação pode ser expresso como uma soma de todos os pares de spin (j, k).

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

Aqui, nj=↑ j #↑ j é um operador que assume o valor de 1 somente se o spin j estiver no estado "ativo" e 0 caso contrário. A força é V j,k =C6/(dj,k) 6, onde C 6 é o coeficiente fixo e d j,k é a distância euclidiana entre os spins j e k. O efeito imediato desse termo de interação é que qualquer estado em que o spin j e o spin k estejam "para cima" tem energia elevada (na quantidade $V_{j,k}$). Ao projetar cuidadosamente o resto do programa AHS, essa interação evitará que ambas as rodadas vizinhas fiquem no estado "ativo", um efeito comumente conhecido como "bloqueio de Rydberg".

Campo de condução

No início do programa AHS, todos os giros (por padrão) começam em seu estado "inativo", eles estão na chamada fase ferromagnética. De olho em nosso objetivo de preparar a fase antiferromagnética, especificamos um campo de condução coerente dependente do tempo que faz a transição suave dos giros desse estado para um estado de muitos corpos, onde os estados "ascendentes" são preferidos. O hamiltoniano correspondente pode ser escrito como

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

onde Ω (t), θ (t), Δ (t) são a amplitude global dependente do tempo (também conhecida como frequência Rabi), a fase e o desajuste do campo de direção, afetando todos os giros de maneira uniforme. Aqui, S−,k=|↓k##↑k|e S+,k=(S−,k)†=|↑k##↓k| são os operadores de abaixamento e elevação do spin k, respectivamente, e nk=|↑k##↑k|é o mesmo operador de antes. A parte Ω do campo de condução acopla coerentemente os estados "para baixo" e "para cima" de todos os giros simultaneamente, enquanto a parte Δ controla a recompensa de energia para os estados "para cima".

Para programar uma transição suave da fase ferromagnética para a fase antiferromagnética, especificamos o campo de condução com o código a seguir.

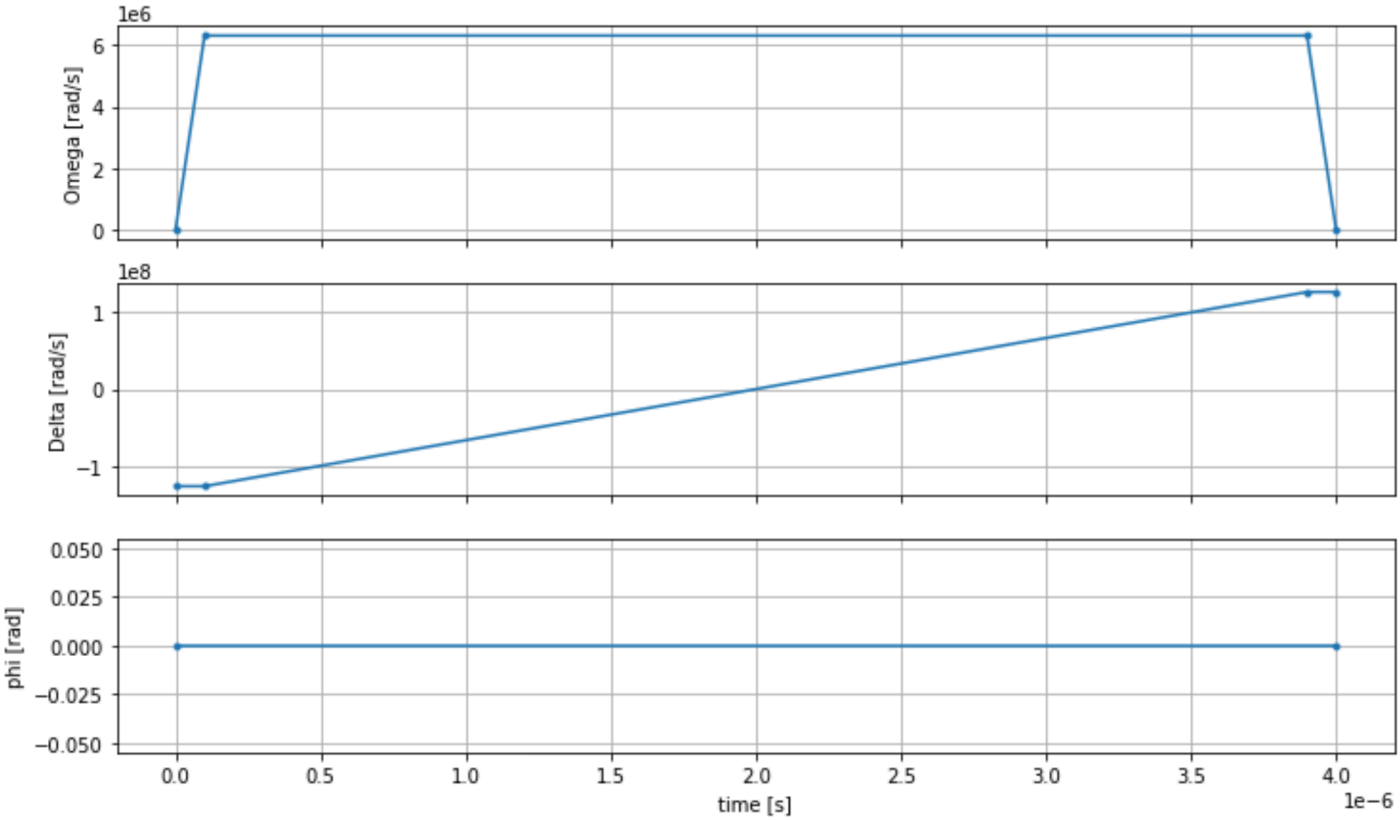
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField

# 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 a série temporal do campo de condução com o seguinte 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

O registro, o campo motriz (e as interações implícitas de van der Waals) compõem o programa de simulação hamiltoniana analógica `ahs_program`.

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

ahs_program = AnalogHamiltonianSimulation(
   register=register,
   hamiltonian=drive
)
```

Executando no simulador local

Como esse exemplo é pequeno (menos de 15 rodadas), antes de executá-lo em uma QPU compatível com AHS, podemos executá-lo no simulador AHS local que vem com o SDK Braket. Como o simulador local está disponível gratuitamente com o SDK do Braket, essa é a melhor prática para garantir que nosso código possa ser executado corretamente.

Aqui, podemos definir o número de fotos para um valor alto (digamos, 1 milhão) porque o simulador local rastreia a evolução temporal do estado quântico e extrai amostras do estado final; portanto, aumenta o número de fotos e aumenta o runtime total apenas 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
```

Analisar resultados de simulação

Podemos agregar os resultados da captura com a seguinte função que infere o estado de cada rotação (que pode ser “d” para “para baixo”, “u” para “para cima” ou “e” para local vazio) e conta quantas vezes cada configuração ocorreu nas fotos.

```
from collections import Counter


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

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

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

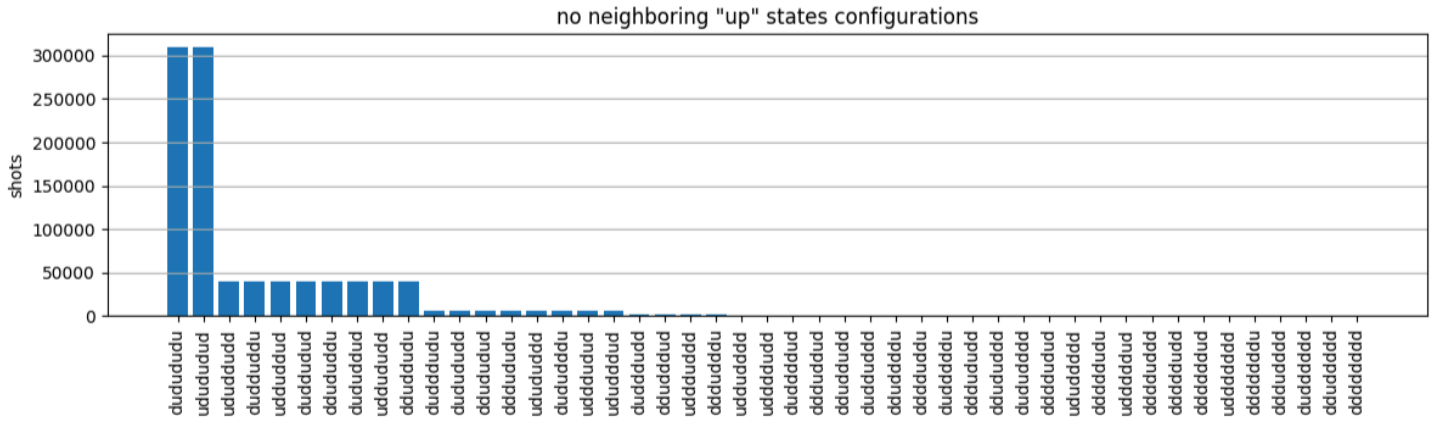
```
    Returns
       dict: number of times each state configuration is measured

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


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

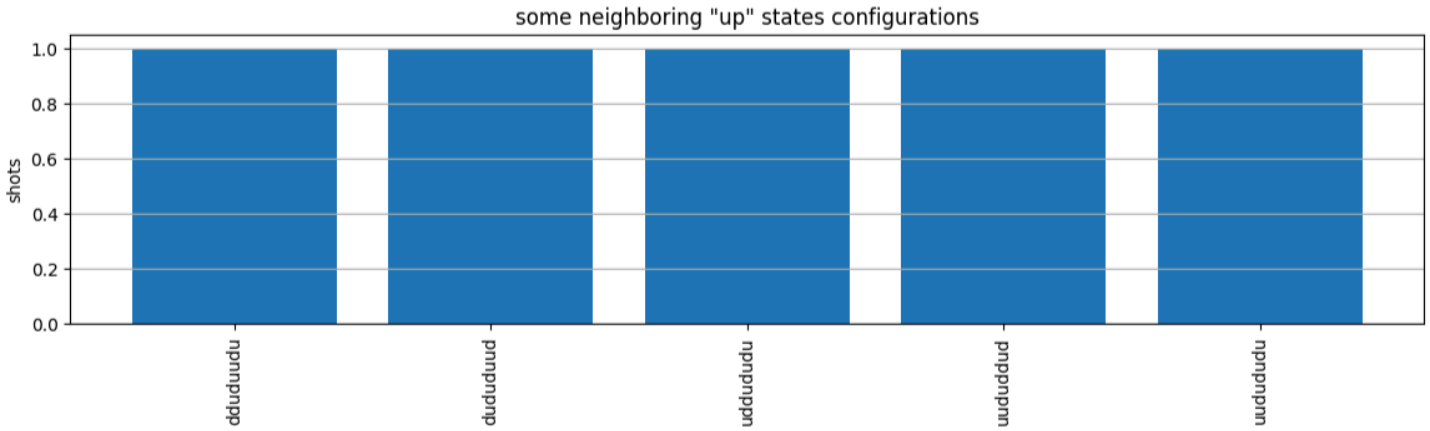
```
*[Output]*
{'dddddddd': 5, 'ddddddddu': 12, 'ddddddud': 15, ...}
```

Aqui `counts` está um dicionário que conta o número de vezes que cada configuração de estado é observada nas fotos. Também podemos visualizá-los com o código a seguir.

```
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 dos gráficos, podemos ler as seguintes observações para verificar se preparamos com sucesso a fase antiferromagnética.

1. Geralmente, estados sem bloqueio (onde não há dois giros vizinhos no estado “ativo”) são mais comuns do que estados em que pelo menos um par de giros vizinhos está no estado “ativo”.
2. Geralmente, estados com mais excitações “ascendentes” são favorecidos, a menos que a configuração esteja bloqueada.
3. Os estados mais comuns são, de fato, os estados antiferromagnéticos perfeitos `"dudududu"` e `"ududududud"`.
4. Os segundos estados mais comuns são aqueles em que há apenas 3 excitações “ascendentes” com separações consecutivas de 1, 2, 2. Isso mostra que a interação de van der Waals também afeta (embora muito menor) os vizinhos mais próximos.

Executando no QuEra Aquila QPU

Pré-requisitos: Além da instalação rápida do SDK do Braket, se você for novo no Amazon Braket, certifique-se de ter concluído as etapas necessárias do Passos para começar.

> **Note**
>
> Se você estiver usando uma instância de caderno hospedada no Braket, o SDK do Braket vem pré-instalado com a instância.

Com todas as dependências instaladas, podemos nos conectar à QPU Aquila.

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

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

Para tornar nosso programa AHS adequado para a máquina QuEra, precisamos arredondar todos os valores para cumprir os níveis de precisão permitidos pela QPU Aquila. Esses requisitos são regidos pelos parâmetros do dispositivo com “Resolução” no nome. Podemos vê-los executando `aquila_qpu.properties.dict()` em um caderno. Para obter mais detalhes sobre os recursos e requisitos do Aquila, consulte a Introdução ao caderno Aquila. Podemos fazer isso chamando o método `discretize`.

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

Agora podemos executar o programa (executando apenas 100 fotos por enquanto) na QPU Aquila.

> **Note**
>
> A execução desse programa no processador Aquila acarretará um custo. O Amazon Braket SDK inclui um rastreador de custos que permite aos clientes definir limites de custo e monitorar seus custos quase em tempo real.

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

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

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

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

Devido à grande variação de quanto tempo uma tarefa quântica pode levar para ser executada (dependendo das janelas de disponibilidade e da utilização da QPU), é uma boa ideia anotar o ARN da tarefa quântica, para que possamos verificar seu status posteriormente com o seguinte trecho de código.

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

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

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

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

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

Depois que o status for CONCLUÍDO (o que também pode ser verificado na página de tarefas quânticas do console Amazon Braket), podemos consultar os resultados com:

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

Analisando os resultados da QPU

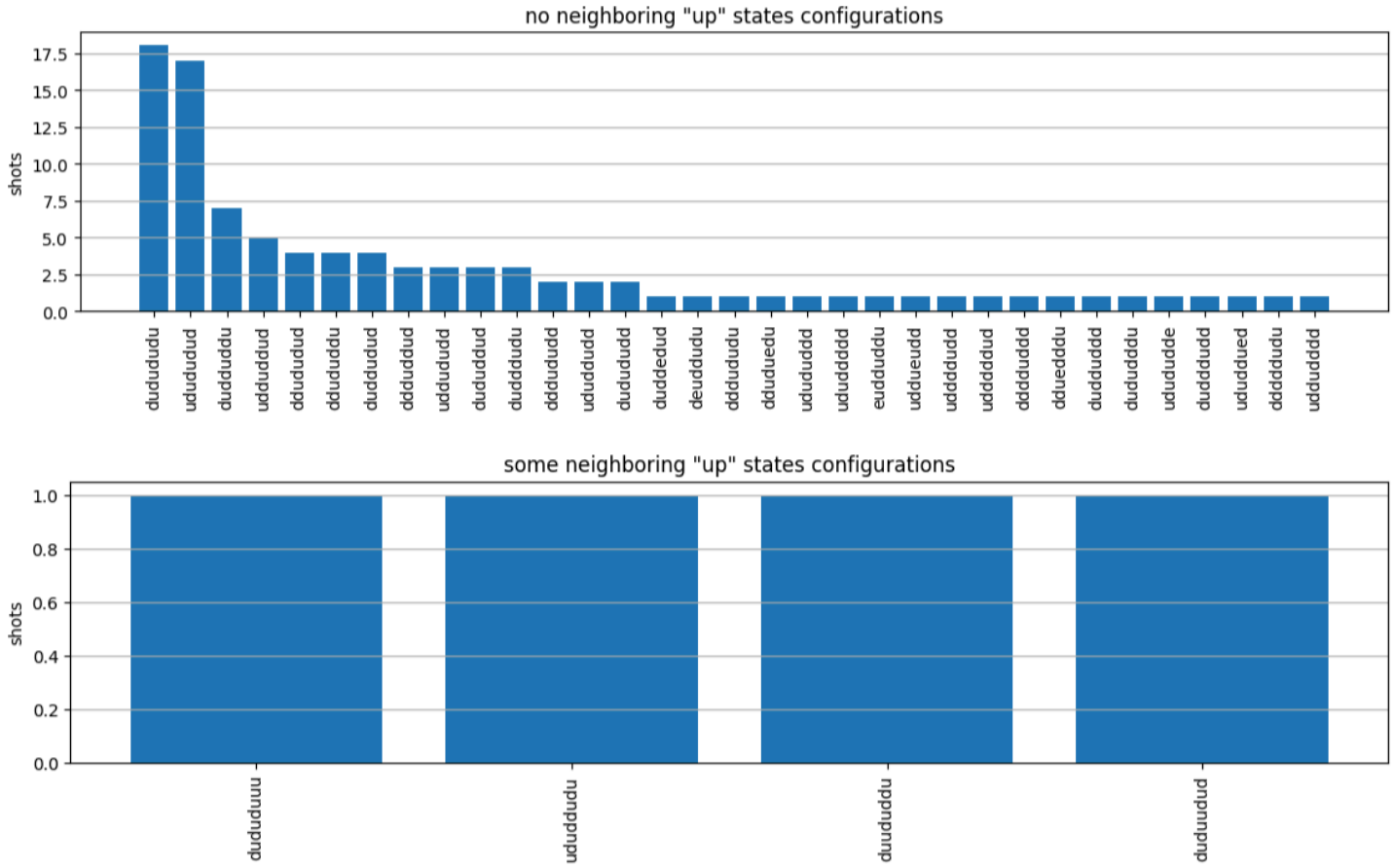
Usando as mesmas funções `get_counts` de antes, podemos calcular as contagens:

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

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

e plote-os com `plot_counts`:

```
plot_counts(counts_aquila)
```



Observe que uma pequena fração das fotos tem locais vazios (marcados com “e”). Isso se deve a imperfeições de 1 a 2% por átomo na preparação do QPU Aquila. Além disso, os resultados coincidem com a simulação dentro da flutuação estatística esperada devido ao pequeno número de disparos.

Próximas etapas

Parabéns, agora você executou sua primeira workload de AHS no Amazon Braket usando o simulador AHS local e a QPU Aquila.

Para saber mais sobre a física de Rydberg, a simulação hamiltoniana analógica e o dispositivo Aquila, consulte nossos exemplos de cadernos.

Envie um programa analógico usando Aquila QuEra

Esta página fornece uma documentação abrangente sobre os recursos da máquina Aquila a partir de QuEra. Os detalhes abordados aqui são os seguintes:

1. O hamiltoniano parametrizado simulado por Aquila

2. Parâmetros do programa AHS
3. Conteúdo do resultado do AHS
4. Parâmetro de capacidades Aquila

Nesta seção:

- Hamiltoniano
- Esquema do programa Braket AHS
- Esquema de resultados de tarefas do Braket AHS
- QuEra esquema de propriedades do dispositivo

Hamiltoniano

A máquina Aquila do QuEra simula o seguinte hamiltoniano (dependente do tempo) nativamente.

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

> **Note**
>
> O acesso ao desajuste local é um recurso experimental e está disponível mediante solicitação por meio do Braket Direct.

para onde

- $H_{drive,k}(t)=({}^{1}/_{2}\ \Omega(t)e^{i\phi(t)}S_{-,k} + {}^{1}/_{2}\ \Omega(t)e^{-i\phi(t)}\ S_{+,k}) + (-\Delta_{global}(t)n_k)$,
  - Ω (t) é a amplitude de condução global dependente do tempo (também conhecida como frequência Rabi), em unidades de (rad/s)
  - θ (t) é a fase global dependente do tempo, medida em radianos
  - $S_{-,k}$ e $S_{+,k}$ são os operadores de redução e aumento de spin do átomo k (na base |↓#=|g#, |↑#=|r#, são S_{-}=|g##r|, $S_{+}=(S_{-})^{\dagger}$=|r##g|)
  - Δ_{global} (t) é o desajuste global dependente do tempo
  - n_k é o operador de projeção no estado de Rydberg do átomo k (ou seja, n=|r#r|)
- $H_{local\ detuning,k}$ (t) =-Δ_{local} (t) h n_k k
  - Δ_{local} (t) é o fator dependente do tempo da mudança de frequência local, em unidades de (rad/s)

  - h_k é o fator dependente do local, um número adimensional entre 0,0 e 1,0
- $V_{vdw,k,l}=C_6/(d_{k,l})^6 n_k n_l$,
  - C_6 é o coeficiente de van der Waals, em unidades de (rad/s) * (m) ^6
  - $d_{k,l}$ é a distância euclidiana entre os átomos k e l, medida em metros.

Os usuários têm controle sobre os seguintes parâmetros por meio do esquema do programa Braket AHS.

- Arranjo de átomos 2-d (coordenadas x_k e y_k de cada átomo k, em unidade de um), que controla as distâncias atômicas entre pares $d_{k,l}$ com k,l=1,2,…N
- Ω (t), a frequência Rabi global dependente do tempo, em unidades de (rad/ s)
- θ (t), a fase global dependente do tempo, em unidades de (rad)
- Δ_{global} (t), o desajuste global dependente do tempo, em unidades de (rad/ s)
- Δ_{local} (t), o fator (global) dependente do tempo da magnitude do desajuste local, em unidades de (rad/ s)
- h_k, o fator (estático) dependente do local da magnitude do desajuste local, um número adimensional entre 0,0 e 1,0

> **Note**
>
> O usuário não pode controlar quais níveis estão envolvidos. (ou seja, operadores S_-,S_+, n são fixos) nem a intensidade do coeficiente de interação Rydberg-Rydberg (C_6).

Esquema do programa Braket AHS

braket.ir.ahs.program_v1.Program object (exemplo)

> **Note**
>
> Se o recurso de desajuste local não estiver habilitado para sua conta, use `localDetuning=[]` no exemplo a seguir.

```
Program(
```

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

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

JSON (exemplo)

> **Note**
>
> Se o recurso de desajuste local não estiver habilitado para sua conta, use `"localDetuning": []` no exemplo a seguir.

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

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

Principais campos

Campo do programa	type	descrição
setup.ahs_register.sites	Lista [Lista [Decimal]]	Lista de coordenadas 2-d em que as

Campo do programa	type	descrição
		pinças capturam átomos
setup.ahs_register.filling	List[int]	Marca os átomos que ocupam os locais da armadilha com 1 e os locais vazios com 0
hamiltonian.drivingFields[].amplitude.time_series.times	List[Decimal]	pontos de tempo da amplitude de condução, Omega (t)
hamiltonian.drivingFields[].amplitude.time_series.values	List[Decimal]	valores de amplitude de condução, Omega (t)
hamiltonian.drivingFields[].amplitude.pattern	str	padrão espacial de amplitude de condução, Omega (t); deve ser “uniforme”
hamiltonian.drivingFields[].phase.time_series.times	List[Decimal]	pontos temporais da fase de condução, phi (t)
hamiltonian.drivingFields[].phase.time_series.values	List[Decimal]	valores da fase de condução, phi (t)
hamiltonian.drivingFields[].phase.pattern	str	padrão espacial da fase de condução, phi (t); deve ser “uniforme”

Campo do programa	type	descrição
hamiltonian.drivingFields[].detuning.time_series.times	List[Decimal]	pontos temporais de desajuste, Delta_global(t)
hamiltonian.drivingFields[].detuning.time_series.values	List[Decimal]	valores do desajuste de direção, Delta_global(t)
hamiltonian.drivingFields[].detuning.pattern	str	padrão espacial de desajuste de direção, Delta_global(t); deve ser 'uniforme'
hamiltonian.localDetuning[].magnitude.time_series.times	List[Decimal]	pontos temporais do fator dependente do tempo da magnitude de desajuste local, Delta_local(t)
hamiltonian.localDetuning[].magnitude.time_series.values	List[Decimal]	valores do fator dependente do tempo da magnitude de desajuste local, Delta_local(t)

Campo do programa	type	descrição
hamiltonian.localDetuning[].magnitude.pattern	List[Decimal]	fator dependente do site da magnitude de desajuste local, h_k (os valores correspondem aos sites em setup.ahs_register.sites)

Campos de metadados

Campo do programa	type	descrição
braketSchemaHeader.name	str	nome do esquema; deve ser 'braket.ir.ahs.program'
braketSchemaHeader.versão	str	versão do esquema

Esquema de resultados de tarefas do Braket AHS

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQuantumTaskResult(exemplo)

```
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 (exemplo)

```
{
    "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
        }
    }
}
```

Principais campos

Campo de resultado da tarefa	type	descrição
measurements[].shotResult.preSequence	List[int]	Bits de medição de pré-sequência (um para cada local atômico) para cada disparo: 0 se o local estiver vazio, 1 se o local estiver cheio, medidos antes das sequências de pulsos que executam a evolução quântica
measurements[].shotResult.postSequence	List[int]	Bits de medição pós-sequência para cada disparo: 0 se o átomo estiver no estado de Rydberg ou o local estiver vazio, 1 se o átomo

Campo de resultado da tarefa	type	descrição
		estiver no estado fundamental, medidos no final das sequência s de pulsos que executam a evolução quântica

Campos de metadados

Campo de resultado da tarefa	type	descrição
braketSchemaHeader.name	str	nome do esquema; deve ser 'braket.t ask_resul t.analog_ hamiltoni an_simula tion_task _result'
braketSchemaHeader.versão	str	versão do esquema
Metadados da tarefa. braketSch emaHeader.nome	str	nome do esquema; deve ser 'braket.t ask_resul t.task_me tadata'
Metadados da tarefa. braketSch emaHeader.versão	str	versão do esquema
taskMetadata.id	str	A identific ação da tarefa quântica. Para tarefas AWS

Campo de resultado da tarefa	type	descrição
		quânticas, essa é a tarefa quântica ARN.
taskMetadata.shots	int	O número de disparos para a tarefa quântica
taskMetadata.shots.deviceId	str	O ID do dispositivo no qual a tarefa quântica foi executada . Para AWS dispositivos, esse é o ARN do dispositivo.
taskMetadata.shots.createdAt	str	O carimbo de data/hora da criação; o formato deve estar em ISO-8601/ string RFC3339 format:MM :SS.SSSZ. YYYY-MM-D DTHH Padrão é Nenhum.

Campo de resultado da tarefa	type	descrição
taskMetadata.shots.endedAt	str	A data e hora de quando a tarefa quântica terminou; o formato deve estar em ISO-8601/ string format:MM :SS.SSSZR FC3339 . YYYY-MM-D DTHH Padrão é Nenhum.
taskMetadata.shots.status	str	O status da tarefa quântica (CRIADA, ENFILEIRA DA, EM EXECUÇÃO, CONCLUÍDA , FALHA). Padrão é Nenhum.
taskMetadata.shots.failureReason	str	O motivo da falha da tarefa quântica. Padrão é Nenhum.

Campo de resultado da tarefa	type	descrição
additionalMetadata.action	braket.ir.ahs.program_v1.Programa	(Consulte a seção Esquema do programa Braket AHS)
Metadados adicionais. Ação. braketSchemaHeader.querametadata.name	str	nome do esquema; deve ser 'braket.task_result.quera_metadata'
Metadados adicionais. Ação. braketSchemaHeader.queraMetadata.Versão	str	versão do esquema
Metadados adicionais. Ação. numSuccessfulShots	int	número de disparos completamente bem-sucedidos; deve ser igual ao número solicitado de disparos
measurements[].shotMetadata.shotStatus	int	O status da foto (sucesso, sucesso parcial, falha); deve ser “Sucesso”

QuEra esquema de propriedades do dispositivo

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

```
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 (exemplo)

```
{
    "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 propriedades de serviços

Campo de propriedades do serviço	type	descrição
service.executionWindows[].executionDay	ExecutionDay	Dias da janela de execução; devem ser 'Todos os dias', 'Dias úteis', 'Fim de semana', 'Segunda-feira', 'Terça', 'Quarta', Quinta-feira ',' Sexta ',' Sábado 'ou' Domingo '

Campo de propriedades do serviço	type	descrição
Service.ExecutionWindows []. windowStartHour	datetime.time	Formato UTC de 24 horas da hora em que a janela de execução começa
Service.ExecutionWindows []. windowEndHour	datetime.time	Formato UTC de 24 horas da hora em que a janela de execução termina
service.qpu_capabilities.service.shotsRange	Tupla [int, int]	Número mínimo e máximo de fotos para o dispositivo
service.qpu_capabilities.service.deviceCost.price	flutuação	Preço do aparelho em dólares americanos
service.qpu_capabilities.service.deviceCost.unit	str	unidade para cobrar o preço, por exemplo: 'minuto', 'hora', 'tiro', 'tarefa'

Campos de metadados

Campos de metadados	type	descrição
ação [] .versão	str	versão do esquema do programa AHS
ação [] .actionType	ActionType	Nome do esquema do programa AHS; deve ser 'braket.ir.ahs.program'
serviço. braketSchemaHeader.nome	str	nome do esquema; deve ser 'braket.device_schema.device_service_properties'
serviço. braketSchemaHeader.versão	str	versão do esquema

Campos de metadados	type	descrição
service.deviceDocumentation.imageUrl	str	URL para a imagem do dispositivo
service.deviceDocumentation.summary	str	breve descrição no dispositivo
Documentação do serviço.dispositivo. externalD ocumentationUrl	str	URL de documenta ção externa
service.deviceLocation	str	localização geográfica do dispositivo
service.updatedAt	datetime	hora em que as propriedades do dispositivo foram atualizadas pela última vez

Trabalhando com o AWS Boto3

O Boto3 é o AWS SDK para Python. Com o Boto3, os desenvolvedores de Python podem criar, configurar e gerenciar Serviços da AWS, como o Braket. Amazon O Boto3 oferece uma API orientada a objetos, bem como acesso de baixo nível ao Amazon Braket.

Siga as instruções no Guia de início rápido do Boto3 para saber como instalar e configurar o Boto3.

O Boto3 fornece a funcionalidade principal que funciona junto com o SDK Amazon Braket Python para ajudar você a configurar e executar suas tarefas quânticas. Os clientes do Python sempre precisam instalar o Boto3, porque essa é a implementação principal. Se quiser usar métodos auxiliares adicionais, você também precisará instalar o Amazon Braket SDK.

Por exemplo, quando você chama `CreateQuantumTask`, o Amazon Braket SDK envia a solicitação para o Boto3, que então chama o AWS API.

Nesta seção:

- Ativar o cliente Amazon Braket Boto3

- Configurar AWS CLI perfis para o Boto3 e o SDK do Braket

Ativar o cliente Amazon Braket Boto3

Para usar o Boto3 com o Amazon Braket, você deve importar o Boto3 e depois definir um cliente que você usa para se conectar à API do Amazon Braket. No exemplo a seguir, o cliente Boto3 é nomeado `braket`.

```
import boto3
import botocore

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

> **Note**
>
> Suportes de suporte. IPv6 Se você estiver usando uma rede IPv6 somente ou quiser garantir que sua carga de trabalho use IPv6 tráfego, use os endpoints de pilha dupla conforme descrito no guia de endpoints de pilha dupla e FIPS.

Agora que você tem um cliente `braket` estabelecido, você pode fazer solicitações e processar respostas do serviço Amazon Braket. Você pode obter mais detalhes sobre os dados de solicitação e resposta na Referência da API.

Os exemplos a seguir mostram como trabalhar com dispositivos e tarefas quânticas.

- Pesquisar dispositivos
- Recuperar um dispositivo
- Crie uma tarefa quântica
- Recupere uma tarefa quântica
- Pesquise tarefas quânticas
- Cancele tarefa quântica

Pesquisar dispositivos

- `search_devices(**kwargs)`

Pesquise dispositivos usando os 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'])
```

Recuperar um dispositivo

- `get_device(deviceArn)`

Recupere os dispositivos disponíveis no 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']}")
```

Crie uma tarefa quântica

- `create_quantum_task(**kwargs)`

Crie uma tarefa quântica.

```
# 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")
```

Recupere uma tarefa quântica

- `get_quantum_task(quantumTaskArn)`

Recupere a tarefa quântica 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'])
```

Pesquise tarefas quânticas

- `search_quantum_tasks(**kwargs)`

Pesquise tarefas quânticas que correspondam aos 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']}")
```

Cancele tarefa quântica

- `cancel_quantum_task(quantumTaskArn)`

Cancele a tarefa quântica 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 perfis para o Boto3 e o SDK do Braket

O Amazon Braket SDK depende das credenciais AWS CLI padrão, a menos que você especifique explicitamente o contrário. Recomendamos que você mantenha o padrão ao executar em um caderno Amazon Braket gerenciado, pois você deve fornecer um perfil do IAM que tenha permissões para iniciar a instância do caderno.

Opcionalmente, se você executar seu código localmente (em uma instância do Amazon EC2, por exemplo), você pode estabelecer AWS CLI perfis nomeados. Você pode atribuir a cada perfil um conjunto de permissões diferente, em vez de substituir regularmente o perfil padrão.

Esta seção fornece uma breve explicação de como configurar esse `profile` CLI e como incorporá-lo ao Amazon Braket para que as chamadas de API sejam feitas com as permissões desse perfil.

Nesta seção:

- Etapa 1: Configurar uma AWS CLI local profile
- Etapa 2: estabelecer um objeto de sessão de Boto3
- Etapa 3: incorporar a sessão de Boto3 ao Braket AwsSession

Etapa 1: Configurar uma AWS CLI local **`profile`**

Está além do escopo deste documento explicar como criar um usuário e como configurar um perfil não padrão. Para mais informações sobre esses tópicos, consulte .

- Conceitos básicos
- Configurando o AWS CLI para usar Centro de Identidade do AWS IAM

Para usar o Amazon Braket, você deve fornecer a esse usuário — e ao `profile` CLI associado — as permissões necessárias do Braket. Por exemplo, você pode anexar a AmazonBraketFullAccesspolítica.

Etapa 2: estabelecer um objeto de sessão de Boto3

Para estabelecer um objeto de sessão do Boto3, utilize o exemplo de código a seguir.

```
from boto3 import Session

# Insert CLI profile name here
boto_sess = Session(profile_name=`profile`)
```

> **Note**
>
> Se as chamadas de API esperadas tiverem restrições baseadas na região que não estejam alinhadas com sua região `profile` padrão, você poderá especificar uma região para a sessão de Boto3, conforme mostrado no exemplo a seguir.

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

Para o argumento designado como`region`, substitua um valor que corresponda a um dos Regiões da AWS em que Amazon Braket está disponível`us-east-1`, como`us-west-1`, e assim por diante.

Etapa 3: incorporar a sessão de Boto3 ao Braket AwsSession

O exemplo a seguir mostra como inicializar uma sessão de Boto3 Braket e instanciar um dispositivo nessa sessão.

```
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)
```

Depois que essa configuração for concluída, você poderá enviar tarefas quânticas para esse objeto `AwsDevice` instanciado (chamando o comando `device.run(…)` , por exemplo). Todas as chamadas de API feitas por esse dispositivo podem usar as credenciais do IAM associadas ao perfil da CLI que você designou anteriormente como `profile`.

Testando suas tarefas quânticas com o Amazon Braket

O Amazon Braket fornece uma variedade de simuladores de circuitos quânticos de alto desempenho para ajudar você a testar e validar seus algoritmos quânticos antes de executá-los em um hardware quântico real. Esses simuladores lidam com o complexo software e a infraestrutura subjacentes e com clusters do Amazon Elastic Compute Cloud (Amazon EC2) para eliminar a carga de simular circuitos quânticos na infraestrutura clássica de computação de alta performance (HPC). Esses recursos permitem que você se concentre no desenvolvimento e na otimização de seus aplicativos quânticos.

Com os simuladores da Braket, você pode testar minuciosamente seus circuitos e algoritmos quânticos sem as restrições e limitações dos dispositivos quânticos físicos. Isso permite que você explore uma ampla variedade de conceitos de computação quântica, desde portas e circuitos quânticos básicos até algoritmos quânticos mais avançados e técnicas de mitigação de erros.

O Braket SDK simplifica o envio de suas tarefas quânticas aos simuladores, permitindo que você controle os parâmetros da simulação, como o número de disparos e o modelo de ruído, para entender melhor o comportamento de seus algoritmos quânticos. Você também pode usar os recursos do Amazon Braket Hybrid Job para combinar elementos de computação quântica e clássica, expandindo ainda mais o escopo de seus testes e validação.

Ao testar minuciosamente suas tarefas quânticas nos simuladores da Braket, você pode obter informações valiosas, refinar seus algoritmos e garantir sua exatidão antes de implantá-los em hardware quântico real. Isso ajuda a reduzir o tempo de desenvolvimento, minimizar erros e, por fim, acelerar seu progresso no campo da computação quântica.

Nesta seção:

- Envio de tarefas quânticas para simuladores
- Emulador de dispositivo quântico local

Envio de tarefas quânticas para simuladores

O Amazon Braket fornece acesso a vários simuladores que podem testar suas tarefas quânticas. Você pode enviar tarefas quânticas individualmente ou executar vários programas.

Simuladores

- Simulador de matriz de densidade, DM1 : `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- Simulador vetorial de estado, SV1 : `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Simulador de rede tensora, TN1 : `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- O simulador local: `LocalSimulator()`

> **Note**
>
> Você pode cancelar tarefas quânticas no `CREATED` estado para simuladores QPUs e sob demanda. Você pode cancelar tarefas quânticas no `QUEUED` estado com base no melhor esforço possível para simuladores sob demanda e. QPUs Observe que é improvável que as tarefas quânticas `QUEUED` da QPU sejam canceladas com sucesso durante as janelas de disponibilidade da QPU.

Nesta seção:

- Simulador vetorial estadual local (braket_sv)
- Simulador de matriz de densidade local (braket_dm)
- Simulador AHS local (braket_ahs)
- Simulador de vetores de estado (SV1)
- Simulador de matriz de densidade (DM1)
- Simulador de rede tensora (TN1)
- Sobre simuladores incorporados
- Comparação entre os simuladores Amazon Braket
- Exemplo de tarefas quânticas no Amazon Braket
- Testando uma tarefa quântica com o simulador local

Simulador vetorial estadual local (**`braket_sv`**)

O simulador vetorial estadual local (`braket_sv`) faz parte do SDK do Amazon Braket que é executado localmente em seu ambiente. Ele é adequado para prototipagem rápida em circuitos

pequenos (até 25 qubits), dependendo das especificações de hardware da instância do caderno do Braket ou do ambiente local.

O simulador local oferece suporte a todas as portas no SDK do Amazon Braket, mas os dispositivos QPU oferecem suporte a um subconjunto menor. Você pode encontrar as portas compatíveis de um dispositivo nas propriedades do dispositivo.

> **Note**
>
> O simulador local oferece suporte a recursos avançados do OpenQASM que podem não ser suportados em dispositivos QPU ou outros simuladores. Para obter mais informações sobre os recursos suportados, consulte os exemplos fornecidos no caderno do OpenQASM Local Simulator.

Para obter mais informações sobre como trabalhar com simuladores, consulte os Exemplos do Amazon Braket.

Simulador de matriz de densidade local (**`braket_dm`**)

O simulador de matriz de densidade local (`braket_dm`) faz parte do SDK do Amazon Braket que é executado localmente em seu ambiente. Ele é adequado para prototipagem rápida em pequenos circuitos com ruído (até 12 qubits), dependendo das especificações de hardware da instância do caderno do Braket ou do ambiente local.

Você pode criar circuitos ruidosos comuns do zero usando operações de ruído de porta, como inversão de bits e erro de despolarização. Você também pode aplicar operações de ruído a portas específicas de qubits e de circuitos existentes que devem funcionar com e sem ruído.

O simulador `braket_dm` local pode fornecer os seguintes resultados, considerando o número especificado de shots:

- Matriz de densidade reduzida: Shots = 0

> **Note**
>
> O simulador local oferece suporte a recursos avançados do OpenQASM, que podem não ser suportados em dispositivos QPU ou outros simuladores. Para obter mais informações sobre

os recursos suportados, consulte os exemplos fornecidos no caderno do OpenQASM Local Simulator.

Para saber mais sobre o simulador de matriz de densidade local, consulte Exemplo introdutório do simulador de ruído Braket.

Simulador AHS local (`braket_ahs`)

O simulador local AHS (Analog Hamiltonian Simulation) `braket_ahs` faz parte do Amazon Braket SDK que é executado localmente em seu ambiente. Ele pode ser usado para simular resultados de um programa AHS. Ele é adequado para prototipagem em pequenos registros (até 10 a 12 átomos), dependendo das especificações de hardware da instância do caderno do Braket ou do ambiente local.

O simulador local suporta programas AHS com um campo de condução uniforme, um campo de mudança (não uniforme) e arranjos de átomos arbitrários. Para obter detalhes, consulte a classe Braket AHS e o esquema do programa Braket AHS.

Para saber mais sobre o simulador AHS local, consulte a página Hello AHS: Execute sua primeira página de simulação hamiltoniana analógica e os Cadernos de exemplo de simulação hamiltoniana analógica.

Simulador de vetores de estado (SV1)

SV1 é um simulador vetorial de estado universal, de alto desempenho e sob demanda. Ele pode simular circuitos de até 34 qubits. Você pode esperar que um circuito de 34-qubit denso e quadrado (profundidade do circuito = 34) leve aproximadamente 1—2 horas para ser concluído, dependendo do tipo de portas usadas e de outros fatores. Circuitos com all-to-all portões são adequados paraSV1. Ele retorna resultados em formas como um vetor de estado completo ou uma matriz de amplitudes.

SV1 tem um runtime máximo de 6 horas. Ele tem um padrão de 35 tarefas quânticas simultâneas e um máximo de 100 (50 em us-west-1 e eu-west-2) tarefas quânticas simultâneas.

Resultados do SV1

SV1pode fornecer os seguintes resultados, dado o número especificado deshots:

- Amostra: Shots > 0

- Expectativa: Shots >= 0
- Variância: Shots >= 0
- Probabilidade: Shots > 0
- Amplitude: Shots = 0
- Gradiente adjunto: Shots = 0

Para saber mais sobre resultados, consulte Tipos de resultados.

SV1 está sempre disponível, ele executa seus circuitos sob demanda e pode executar vários circuitos em paralelo. O runtime é dimensionado linearmente com o número de operações e exponencialmente com o número de. qubits O número de shots tem um pequeno impacto no runtime. Para saber mais, acesse Compare simuladores.

Os simuladores oferecem suporte a todas as portas no SDK do Braket, mas os dispositivos QPU oferecem suporte a um subconjunto menor. Você pode encontrar as portas compatíveis de um dispositivo nas propriedades do dispositivo.

Simulador de matriz de densidade (DM1)

DM1é um simulador de matriz de densidade de alto desempenho e sob demanda. Ele pode simular circuitos de até 17qubits.

DM1 tem um runtime máximo de 6 horas, um padrão de 35 tarefas quânticas simultâneas e um máximo de 50 tarefas quânticas simultâneas.

Resultados do DM1

DM1pode fornecer os seguintes resultados, dado o número especificado deshots:

- Amostra: Shots > 0
- Expectativa: Shots >= 0
- Variância: Shots >= 0
- Probabilidade: Shots > 0
- Matriz de densidade reduzida: Shots = 0, até no máximo 8 qubits

Para obter mais informações sobre esses tipos de regra, consulte Resolvedores.

DM1 está sempre disponível, ele executa seus circuitos sob demanda e pode executar vários circuitos em paralelo. O runtime é dimensionado linearmente com o número de operações e exponencialmente com o número de. qubits O número de shots tem um pequeno impacto no runtime. Para saber mais, consulte Comparar simuladores.

Limitações e barreiras acústicas

```
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 rede tensora (TN1)

TN1é um simulador de rede tensorial sob demanda e de alto desempenho. TN1pode simular certos tipos de circuitos com até 50 qubits e uma profundidade de circuito de 100 ou menor. TN1é particularmente poderoso para circuitos esparsos, circuitos com portas locais e outros circuitos com estrutura especial, como circuitos de transformada quântica de Fourier (QFT). TN1opera em duas fases. Primeiro, a fase de ensaio tenta identificar um caminho computacional eficiente para seu circuito, para o TN1 poder estimar o runtime do próximo estágio, que é chamado de fase de contração. Se o tempo estimado de contração exceder o limite de runtime da simulação TN1, o TN1 não tentará contrair.

TN1 tem um limite de runtime de 6 horas. É limitado a um máximo de 10 (5 em eu-west-2) tarefas quânticas simultâneas.

Resultados do TN1

A fase de contração consiste em uma série de multiplicações de matrizes. A série de multiplicações continua até que um resultado seja alcançado ou até que seja determinado que um resultado não pode ser alcançado.

Nota: Shots deve ser > 0.

Os tipos de resultados incluem:

- Amostra
- Expectativa
- Variação

Para saber mais sobre resultados, consulte Tipos de resultados.

TN1 está sempre disponível, ele executa seus circuitos sob demanda e pode executar vários circuitos em paralelo. Para saber mais, consulte Comparar simuladores.

Os simuladores oferecem suporte a todas as portas no SDK do Braket, mas os dispositivos QPU oferecem suporte a um subconjunto menor. Você pode encontrar as portas compatíveis de um dispositivo nas propriedades do dispositivo.

Visite o GitHub repositório Amazon Braket para ver um TN1 exemplo de caderno para ajudá-lo a começar. TN1

Melhores práticas para trabalhar com TN1

- Evite all-to-all circuitos.
- Teste um novo circuito ou classe de circuitos com um pequeno número de shots, para saber qual é a “dureza” do circuito para TN1.
- Divida grandes simulações shot em várias tarefas quânticas.

Sobre simuladores incorporados

Os simuladores incorporados operam com a simulação incorporada diretamente no código do algoritmo. Além disso, ele está contido no mesmo contêiner e executa a simulação diretamente

na instância de trabalho híbrida. Essa abordagem é útil para remover gargalos normalmente associados à comunicação entre a simulação e um dispositivo remoto. Ao manter todos os cálculos em um ambiente único e coeso, os simuladores incorporados podem reduzir consideravelmente os requisitos de memória e diminuir o número de execuções de circuito necessárias para atingir o resultado desejado. Isso pode levar a melhorias substanciais no desempenho, geralmente por um fator de dez ou mais, em comparação com as configurações tradicionais que dependem da simulação remota. Para obter mais informações sobre como os simuladores incorporados melhoram o desempenho e permitem trabalhos híbridos simplificados, consulte a página de documentação Executar um trabalho híbrido com o Amazon Braket Hybrid Jobs.

PennyLanesimuladores de relâmpagos

Você pode usar os simuladores PennyLane de raios como simuladores incorporados no Braket. Com os simuladores PennyLane de raios, você pode usar métodos avançados de computação de gradientes, como diferenciação adjunta, para avaliar gradientes mais rapidamente. O simulador lightning.qubit está disponível como um dispositivo por meio do Braket NBIs e como um simulador incorporado, enquanto o simulador lightning.gpu precisa ser executado como um simulador incorporado com uma instância de GPU. Consulte os Simuladores incorporados no caderno do Braket Hybrid Jobs para ver um exemplo do uso de lightning.gpu.

Comparação entre os simuladores Amazon Braket

Esta seção ajuda você a selecionar o simulador Amazon Braket mais adequado para sua tarefa quântica, descrevendo alguns conceitos, limitações e casos de uso.

Escolha entre simuladores locais e sob demanda (SV1, TN1, DM1)

O desempenho dos simuladores locais depende do hardware que hospeda o ambiente local, como uma instância do caderno do Braket, usada para executar seu simulador. Os simuladores sob demanda são executados na AWS nuvem e projetados para escalar além dos ambientes locais típicos. Os simuladores sob demanda são otimizados para circuitos maiores, mas adicionam alguma sobrecarga de latência por tarefa quântica ou lote de tarefas quânticas. Isso pode implicar uma compensação se muitas tarefas quânticas estiverem envolvidas. Dadas essas características gerais de desempenho, a orientação a seguir pode ajudá-lo a escolher como executar simulações, inclusive aquelas com ruído.

Para simulações:

- Ao empregar menos de 18 qubits, use um simulador local.

- Ao empregar de 18 a 24 qubits, escolha um simulador com base na workload.
- Ao empregar mais de 24 qubits, use um simulador sob demanda.

Para simulações de ruído:

- Ao empregar menos de 9 qubits, use um simulador local.
- Ao empregar de 9 a 12 qubits, escolha um simulador com base na workload.
- Ao empregar mais de 12 qubits, use DM1.

O que é um simulador vetorial de estado?

SV1é um simulador vetorial de estado universal. Ele armazena a função de onda completa do estado quântico e aplica sequencialmente as operações de porta ao estado. Ele armazena todas as possibilidades, mesmo as extremamente improváveis. O tempo de execução do SV1 simulador para uma tarefa quântica aumenta linearmente com o número de portas no circuito.

O que é um simulador de matriz de densidade?

DM1 simula circuitos quânticos com ruído. Ele armazena a matriz de densidade total do sistema e aplica sequencialmente as portas e as operações de ruído do circuito. A matriz de densidade final contém informações completas sobre o estado quântico após a execução do circuito. O runtime geralmente é escalonado linearmente com o número de operações e exponencialmente com o número de. qubits

O que é um simulador de rede tensorial?

TN1 codifica circuitos quânticos em um gráfico estruturado.

- Os nós do gráfico consistem em portas quânticas, ou qubits.
- As bordas do gráfico representam conexões entre portas.

Como resultado dessa estrutura, o TN1 pode encontrar soluções simuladas para circuitos quânticos relativamente grandes e complexos.

TN1requer duas fases

Normalmente, TN1 opera em uma abordagem de duas fases para simular a computação quântica.

- A fase de ensaio: nesta fase, TN1 surge uma maneira de percorrer o gráfico de maneira eficiente, o que envolve visitar cada nó para que você possa obter a medida desejada. Como cliente, você não vê essa fase porque TN1 executa as duas fases juntas para você. Ele conclui a primeira fase e determina se deve realizar a segunda fase por conta própria, com base em restrições práticas. Você não tem nenhuma participação nessa decisão após o início da simulação.
- A fase de contração: Essa fase é análoga à fase de execução de uma computação em um computador clássico. A fase consiste em uma série de multiplicações de matrizes. A ordem dessas multiplicações tem um grande efeito na dificuldade do cálculo. Portanto, a fase de ensaio é realizada primeiro para encontrar os caminhos de computação mais eficazes no gráfico. Depois de encontrar o caminho da contração durante a fase de ensaio, o TN1 contrai as portas do circuito para produzir os resultados da simulação.

TN1gráficos são análogos a um mapa

Metaforicamente, você pode comparar o TN1 gráfico subjacente com as ruas de uma cidade. Em uma cidade com uma grade planejada, é fácil encontrar uma rota para seu destino usando um mapa. Em uma cidade com ruas não planejadas, nomes de ruas duplicados e assim por diante, pode ser difícil encontrar uma rota para seu destino consultando um mapa.

Se TN1 não realizasse a fase de ensaio, seria como andar pelas ruas da cidade para encontrar seu destino, em vez de olhar primeiro um mapa. Em termos de tempo de caminhada, pode realmente valer a pena passar mais tempo olhando o mapa. Da mesma forma, a fase de ensaio fornece informações valiosas.

Você pode dizer que o TN1 tem uma certa “consciência” da estrutura do circuito subjacente que ele atravessa. Ele ganha essa consciência durante a fase de ensaio.

Tipos de problemas mais adequados para cada um desses tipos de simuladores

SV1é adequado para qualquer classe de problemas que dependa principalmente de um certo número qubits de portas. Geralmente, o tempo necessário cresce linearmente com o número de portas, embora não dependa do número de shots. O SV1 geralmente é mais rápido do que o TN1 para circuitos abaixo de 28qubits.

SV1pode ser mais lento para qubit números maiores porque, na verdade, simula todas as possibilidades, mesmo as extremamente improváveis. Não há como determinar quais resultados são prováveis. Assim, para uma 30-qubit avaliação, é SV1 necessário calcular 2^30 configurações. O limite de 34 qubits para o SV1 simulador Amazon Braket é uma restrição prática devido às limitações

de memória e armazenamento. Você pode pensar assim: cada vez que você adiciona um qubit aSV1, o problema se torna duas vezes mais difícil.

Para muitas classes de problemas, o TN1 pode avaliar circuitos muito maiores em tempo real do que o SV1 porque o TN1 tira proveito da estrutura do gráfico. Essencialmente, ele acompanha a evolução das soluções desde o início e retém apenas as configurações que contribuem para uma travessia eficiente. Em outras palavras, ele salva as configurações para criar uma ordem de multiplicação de matrizes que resulta em um processo de avaliação mais simples.

PoisTN1, o número qubits e as portas são importantes, mas a estrutura do gráfico é muito mais importante. Por exemplo, o TN1 é muito bom para avaliar circuitos (gráficos) nos quais as portas são de curto alcance (ou seja, cada qubit é conectado por portas somente ao qubits vizinho mais próximo) e circuitos (gráficos) nos quais as conexões (ou portas) têm alcance semelhante. Um intervalo típico TN1 é fazer com que cada um qubit fale apenas com outro qubits que esteja a 5 qubits de distância. Se a maior parte da estrutura puder ser decomposta em relacionamentos mais simples, como esses, que podem ser representados em matrizes maiores, menores ou mais uniformes, TN1 realiza a avaliação com eficiência.

Limitações do TN1

TN1pode ser mais lento do que SV1 depender da complexidade estrutural do gráfico. Para determinados gráficos, TN1 encerra a simulação após a fase de ensaio e mostra um status de`FAILED`, por qualquer um desses dois motivos:

- Não é possível encontrar um caminho — Se o gráfico for muito complexo, é muito difícil encontrar um bom caminho de travessia e o simulador desiste do cálculo. TN1não consegue realizar a contração. Você pode ver uma mensagem de erro semelhante a esta: `No viable contraction path found.`
- O estágio de contração é muito difícil — Em alguns gráficos, é TN1 possível encontrar um caminho de travessia, mas é muito longo e demorado de avaliar. Nesse caso, a contração é tão cara que o custo seria proibitivo e, em vez disso, TN1 sai após a fase de ensaio. Você pode ver uma mensagem de erro semelhante a esta: `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> Você é cobrado pela fase de ensaio, TN1 mesmo que a contração não seja realizada e você veja um status. `FAILED`

O runtime previsto também depende da shot contagem. Na pior das hipóteses, o tempo de TN1 contração depende linearmente da contagem. shot O circuito pode ser contratável com menos shots. Por exemplo, você pode enviar uma tarefa quântica com 100 shots, que TN1 decide ser incontratável, mas se você reenviar com apenas 10, a contração prossegue. Nessa situação, para obter 100 amostras, você poderia enviar 10 tarefas quânticas de 10 shots para o mesmo circuito e combinar os resultados no final.

Como prática recomendada, recomendamos que você sempre teste seu circuito ou classe de circuito com alguns shots (por exemplo, 10) para descobrir a resistência do seu circuito do TN1, antes de prosseguir com um número maior de shots.

> **Note**
>
> A série de multiplicações que forma a fase de contração começa com pequenas matrizes NxN. Por exemplo, um 2-qubit portão requer uma matriz 4x4. As matrizes intermediárias necessárias durante uma contração considerada muito difícil são gigantescas. Esse cálculo exigiria dias para ser concluído. É por isso que o Amazon Braket não tenta contrações extremamente complexas.

Simultaneidade

Todos os simuladores Braket permitem executar vários circuitos ao mesmo tempo. Os limites de simultaneidade variam de acordo com o simulador e a região. Para obter mais informações sobre a duração das conexões, consulte Cotas e limites.

Exemplo de tarefas quânticas no Amazon Braket

Esta seção mostra os estágios da execução de um exemplo de tarefa quântica, desde a seleção do dispositivo até a visualização do resultado. Como prática recomendada para o Amazon Braket, recomendamos que você comece executando o circuito em um simulador, como o SV1.

Nesta seção:

- Especifique o dispositivo
- Envie um exemplo de tarefa quântica
- Enviar uma tarefa parametrizada
- Especifique shots
- Sondagem para obter os resultados
- Exemplo: visualizar os resultados

Especifique o dispositivo

Primeiro, selecione e especifique o dispositivo para sua tarefa quântica. Este exemplo mostra como escolher o simulador,SV1.

```
from braket.aws import AwsDevice

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

Você pode ver algumas das propriedades desse dispositivo da seguinte forma:

```
print(device.name)
for iter in device.properties.action['braket.ir.jaqcd.program']:
    print(iter)
```

```
SV1
('version', ['1.0', '1.1'])
('actionType', 'braket.ir.jaqcd.program')
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
 'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', 'ecr', 'h', 'i', 'iswap',
 'pswap', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'unitary', 'v',
 'vi', 'x', 'xx', 'xy', 'y', 'yy', 'z', 'zz'])
('supportedResultTypes', [ResultType(name='Sample', observables=['x', 'y', 'z', 'h',
 'i', 'hermitian'], minShots=1, maxShots=100000), ResultType(name='Expectation',
 observables=['x', 'y', 'z', 'h', 'i', 'hermitian'], minShots=0, maxShots=100000),
 ResultType(name='Variance', observables=['x', 'y', 'z', 'h', 'i', 'hermitian'],
 minShots=0, maxShots=100000), ResultType(name='Probability', observables=None,
 minShots=1, maxShots=100000), ResultType(name='Amplitude', observables=None,
 minShots=0, maxShots=0)])
('disabledQubitRewiringSupported', None)
```

Envie um exemplo de tarefa quântica

Envie um exemplo de tarefa quântica para ser executada no simulador sob demanda.

```
from braket.circuits import Circuit, Observable

# Create a circuit with a result type
circ = Circuit().rx(0, 1).ry(1, 0.2).cnot(0, 2).variance(observable=Observable.Z(),
 target=0)
# Add another result type
circ.probability(target=[0, 2])

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

# Submit the quantum task to run
my_task = device.run(circ, s3_location, shots=1000, poll_timeout_seconds=100,
 poll_interval_seconds=10)
# The positional argument for the S3 bucket is optional if you want to specify a bucket
 other than the default

# Get results of the quantum task
result = my_task.result()
```

O `device.run()` comando cria uma tarefa quântica por meio da `CreateQuantumTask` da API. Após um curto período de inicialização, a tarefa quântica é colocada em fila até que exista a capacidade de executar a tarefa quântica em um dispositivo. Neste caso, o dispositivo é SV1. Depois que o dispositivo conclui o cálculo, Amazon Braket grava os resultados no local do Amazon S3 especificado na chamada. O argumento posicional `s3_location` é necessário para todos os dispositivos, exceto para o simulador local.

> **Note**
>
> A ação da tarefa quântica do Braket é limitada a 3 MB de tamanho.

Enviar uma tarefa parametrizada

Os simuladores locais e sob demanda do Amazon Braket também oferecem suporte à especificação de valores QPUs de parâmetros gratuitos no envio da tarefa. Você pode fazer isso usando o `inputs` argumento to`device.run()`, conforme mostrado no exemplo a seguir. `inputs`Deve ser um dicionário de pares string-float, em que as chaves são os nomes dos parâmetros.

A compilação paramétrica pode melhorar o desempenho da execução de circuitos paramétricos em determinados. QPUs Ao enviar um circuito paramétrico como uma tarefa quântica para uma QPU compatível, o Braket compilará o circuito uma vez e armazenará o resultado em cache. Não há recompilação para atualizações subsequentes de parâmetros no mesmo circuito, resultando em runtime mais rápidos para tarefas que usam o mesmo circuito. O Braket usa automaticamente os dados de calibração atualizados do fornecedor de hardware ao compilar seu circuito para garantir resultados da mais alta qualidade.

> **Note**
>
> A compilação paramétrica é suportada em todos os formatos supercondutores baseados em portas, Rigetti Computing com exceção QPUs dos programas de nível de pulso.

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

# Create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')

# Create a circuit with a result type
circ = Circuit().rx(0, alpha).ry(1, alpha).cnot(0, 2).xx(0, 2, beta)
circ.variance(observable=Observable.Z(), target=0)

# Add another result type
circ.probability(target=[0, 2])

# Submit the quantum task to run
my_task = device.run(circ, inputs={'alpha': 0.1, 'beta': 0.2}, shots=100)
```

Especifique shots

O shots argumento se refere ao número de medidas desejadasshots. Simuladores como o SV1 suportam dois modos de simulação.

- Para shots = 0, o simulador executa uma simulação exata, retornando os valores reais para todos os tipos de resultados. (Não disponível no TN1.)
- Para valores diferentes de zeroshots, o simulador extrai amostras da distribuição de saída para emular o shot ruído real. QPUs Os dispositivos QPU permitem somente shots > 0.

Para obter informações sobre o número máximo de disparos por tarefa quântica, consulte Cotas do Braket.

Sondagem para obter os resultados

Durante a execução do `my_task.result()`, o SDK começa a pesquisar um resultado com os parâmetros que você define na criação da tarefa quântica:

- `poll_timeout_seconds` é o número de segundos para pesquisar a tarefa quântica antes que ela atinja o tempo limite ao executar a tarefa quântica no simulador sob demanda e/ou em dispositivos QPU. O valor padrão é 432.000 segundos, o que corresponde a 5 dias.
- Nota: Para dispositivos QPU como Rigetti e IonQ, recomendamos que você aguarde alguns dias. Se o tempo limite da pesquisa for muito curto, os resultados podem não ser retornados dentro do prazo da pesquisa. Por exemplo, quando uma QPU não está disponível, um erro de tempo limite local é retornado.
- `poll_interval_seconds` é a frequência com que a tarefa quântica é sondada. Ele especifica com que frequência você chama o API Braket para obter o status quando a tarefa quântica é executada no simulador sob demanda e em dispositivos QPU. O valor padrão de é 1 segundo.

Essa execução assíncrona facilita a interação com dispositivos QPU que nem sempre estão disponíveis. Por exemplo, um dispositivo pode ficar indisponível durante uma janela de manutenção regular.

O resultado retornado contém uma variedade de metadados associados à tarefa quântica. Você pode verificar o resultado da medição com os seguintes 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)
```

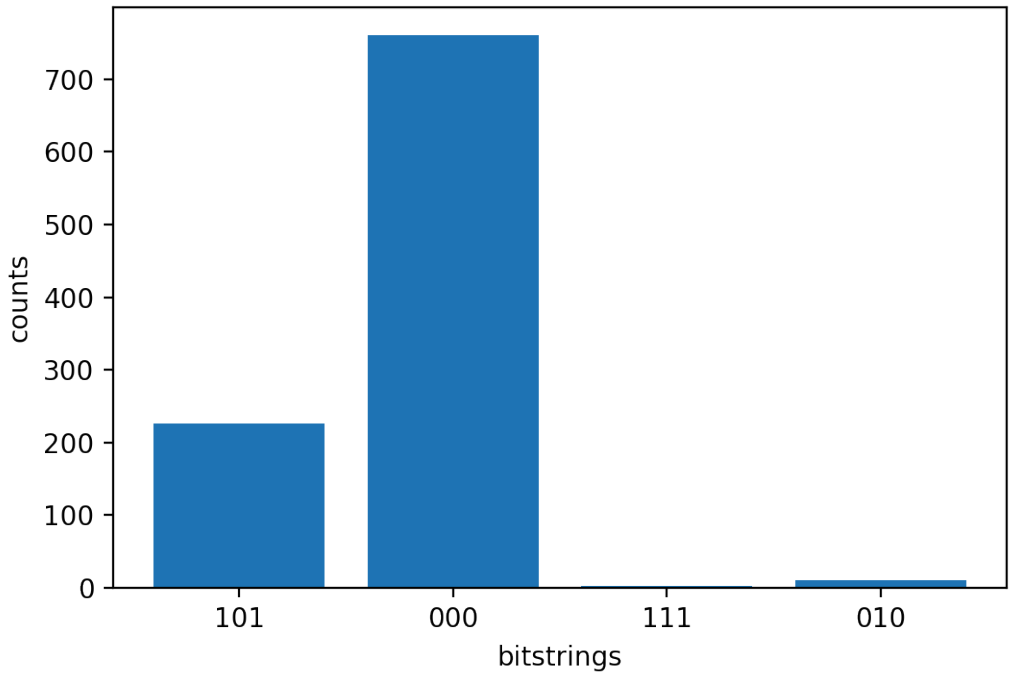
```
Measurement results:
 [[1 0 1]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [1 0 1]]
Counts for collapsed states:
 Counter({'000': 766, '101': 220, '010': 11, '111': 3})
Probabilities for collapsed states:
 {'101': 0.22, '000': 0.766, '010': 0.011, '111': 0.003}
```

Exemplo: visualizar os resultados

Como você também especificou o `ResultType`, você pode ver os resultados retornados. Os tipos de resultado aparecem na ordem em que foram adicionados ao circuito.

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

# Plot the result and do some analysis
import matplotlib.pyplot as plt
plt.bar(result.measurement_counts.keys(), result.measurement_counts.values())
plt.xlabel('bitstrings')
plt.ylabel('counts')
```

```
Result types include:
 [ResultTypeValue(type=Variance(observable=['z'], targets=[0], type=<Type.variance:
 'variance'>), value=0.693084), ResultTypeValue(type=Probability(targets=[0, 2],
 type=<Type.probability: 'probability'>), value=array([0.777, 0.   , 0.   , 0.223]))]
Variance= 0.693084
Probability= [0.777 0.    0.    0.223]
Text(0, 0.5, 'counts')
```



Testando uma tarefa quântica com o simulador local

Você pode enviar tarefas quânticas diretamente para um simulador local para prototipagem e testes rápidos. O simulador é executado em seu ambiente local, portanto, você não precisa especificar uma localização do Amazon S3. Os resultados são computados diretamente na sua sessão. Para executar uma tarefa quântica no simulador local, você deve especificar apenas o parâmetro shots.

> **Note**
>
> A velocidade de execução e o número máximo que simulador qubits o local pode processar dependem do tipo de instância do caderno Amazon Braket ou das especificações de hardware locais.

Os comandos a seguir são todos idênticos e instanciam o simulador local do vetor de estado (sem ruído).

```
# 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")
```

Em seguida, execute uma tarefa quântica com o seguinte.

```
my_task = device.run(circ, shots=1000)
```

Para instanciar o simulador de matriz de densidade local (ruído), os clientes alteram o back-end da seguinte maneira.

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

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

Medindo qubits específicos no simulador local

O simulador vetorial de estado local e o simulador de matriz de densidade local suportam circuitos em que um subconjunto dos qubits do circuito pode ser medido, o que geralmente é chamado de medição parcial.

Por exemplo, no código a seguir, você pode criar um circuito de dois qubits e medir somente o primeiro qubit adicionando uma instrução `measure` com os qubits de destino ao final do circuito.

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

# Use the local simulator device
device = LocalSimulator()

# Define a bell circuit and only measure
circuit = Circuit().h(0).cnot(0, 1).measure(0)

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

# Get the results
```

```
result = task.result()

# Print the measurement counts for qubit 0
print(result.measurement_counts)
```

Emulador de dispositivo quântico local

Com a ferramenta de emulador local do Amazon Braket, você pode emular seus programas quânticos textuais localmente antes de executá-los no hardware quântico real. O emulador usa dados de calibração do dispositivo para validar os circuitos textuais, permitindo que você detecte problemas de compatibilidade mais cedo.

Além disso, o emulador local simula o ruído quântico do hardware por meio do seguinte processo:

- Usando dados de calibração do dispositivo para construir o modelo de ruído
- Aplicando ruído despolarizador em cada porta em seu circuito
- Aplicando erro de leitura no final do circuito
- Simulando o circuito ruidoso com um simulador de matriz de densidade local

Para obter mais informações sobre o uso de um emulador local, consulte Emulação local para circuitos textuais no Amazon Braket no repositório. amazon-braket-examples GitHub

Nesta seção:

- Benefícios da emulação local
- Criar um emulador local

Benefícios da emulação local

- Valide circuitos textuais em relação às restrições do dispositivo usando dados de calibração históricos ou em tempo real.
- Depure problemas antes de enviar tarefas ao hardware quântico.
- Compare emulações silenciosas e ruidosas com resultados de hardware para entender os efeitos de ruído.
- Simplifique o fluxo de trabalho do desenvolvimento de algoritmos quânticos com reconhecimento de ruído.

Criar um emulador local

Um emulador de dispositivo quântico local pode ser criado diretamente de um dispositivo quântico ou de um conjunto de propriedades do dispositivo. Ao emular diretamente um dispositivo, o emulador usa os dados de calibração mais recentes do dispositivo instanciado. O exemplo de código a seguir demonstra como emular diretamente o dispositivo Rigetti's Ankaa-3.

```
from braket.aws.aws_device import AwsDevice

ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_emulator = ankaa3.emulator()
```

O exemplo a seguir mostra a criação de um emulador de dispositivo local a partir de um conjunto de propriedades do dispositivo Ankaa-3 no formato JSON.

```
from braket.aws import AwsDevice
from braket.emulation.local_emulator import LocalEmulator
import json

# Instantiate the device
ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_properties = ankaa3.properties

# Put the Ankaa-3 properties in a file named ankaa3_device_properties.json
with open("ankaa3_device_properties.json", "w") as f:
    json.dump(ankaa3_properties.json(), f)

# Load the json into the ankaa3_data_json variable
with open("ankaa3_device_properties.json", "r") as json_file:
    ankaa3_data_json = json.load(json_file)

# Create the Ankaa-3 local emulator from the json file you created
ankaa3_emulator = LocalEmulator.from_json(ankaa3_data_json)
```

Você pode personalizar o exemplo da seguinte forma:

- Usando propriedades de um dispositivo QPU diferente
- Especificando um arquivo JSON diferente para o emulador
- Alterando os valores das propriedades do dispositivo antes de instanciar o emulador

Executar tarefas quânticas com o Amazon Braket

O Braket fornece acesso seguro e sob demanda a diferentes tipos de computadores quânticos. Você tem acesso a computadores quânticos baseados em portas deAQT,,, e IonQ IQMRigetti, bem como a um simulador hamiltoniano analógico de. QuEra Você também não tem nenhum compromisso inicial e não precisa obter acesso por meio de fornecedores individuais.

- O Amazon Braket Console fornece informações e status do dispositivo para ajudá-lo a criar, gerenciar e monitorar seus recursos e tarefas quânticas.
- Envie e execute tarefas quânticas por meio do SDK Amazon Braket Python, bem como por meio do console. O SDK pode ser acessado por meio de cadernos do Amazon Braket pré-configurados.
- A API Amazon Braket pode ser acessada por meio do SDK e dos cadernos do Amazon Braket Python. Você pode fazer chamadas diretamente para a API se estiver criando aplicativos que funcionam com a computação quântica de forma programática.

Os exemplos desta seção demonstram como você pode trabalhar com a API do Amazon Braket diretamente usando o SDK do Amazon Braket para Python junto com o SDK do Python para AWS Braket (Boto3).

Mais sobre o SDK do Amazon Braket Python

Para trabalhar com o Amazon Braket Python SDK, primeiro AWS instale o Python SDK para Braket (Boto3) para que você possa se comunicar com o. AWS API Você pode pensar no SDK Amazon Braket Python como uma solução conveniente para o Boto3 para clientes quânticos.

- O Boto3 contém interfaces que você precisa acessar. AWS API (Observe que o Boto3 é um grande SDK para Python que se comunica com o. AWS API A maioria Serviços da AWS suporta uma interface Boto3.)
- O Amazon Braket Python SDK contém módulos de software para circuitos, portas, dispositivos, tipos de resultados e outras partes de uma tarefa quântica. Cada vez que você cria um programa, você importa os módulos necessários para essa tarefa quântica.
- O SDK Amazon Braket Python pode ser acessado por meio de cadernos, que são pré-carregados com todos os módulos e dependências de que você precisa para executar tarefas quânticas.
- Você pode importar módulos do Amazon Braket Python SDK para qualquer script Python se não quiser trabalhar com cadernos.

Depois de instalar o Boto3, uma visão geral das etapas para criar uma tarefa quântica por meio do SDK do Amazon Braket Python é semelhante à seguinte:

1. (Opcionalmente) Abra seu caderno.
2. Importe os módulos SDK necessários para seus circuitos.
3. Especifique uma QPU ou simulador.
4. Instancie o circuito.
5. Execute o circuito.
6. Colete os resultados.

Os exemplos nesta seção mostram detalhes de cada etapa.

Para obter mais exemplos, consulte o repositório Amazon Braket Examples em. GitHub

Nesta seção:

- Enviando tarefas quânticas para QPUs
- Executar vários programas
- Quando minha tarefa quântica será executada?
- Trabalhando com reservas
- Técnicas de mitigação de erros

Enviando tarefas quânticas para QPUs

O Amazon Braket fornece acesso a vários dispositivos que podem executar tarefas quânticas. Você pode enviar tarefas quânticas individualmente ou configurar o agrupamento de tarefas quânticas.

Unidades de processamento quântico (QPUs)

Você pode enviar tarefas quânticas a qualquer QPUs momento, mas a tarefa é executada dentro de determinadas janelas de disponibilidade que são exibidas na página Dispositivos do console Amazon Braket. Você pode recuperar os resultados da tarefa quântica com o ID da tarefa quântica, que é apresentado na próxima seção.

- AQT IBEX-Q1 : `arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1`
- IonQ Forte-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1`

- IonQ Forte-Enterprise-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1`
- IQM Garnet : `arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet`
- IQM Emerald : `arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Ankaa-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3`

> **Note**
>
> Você pode cancelar tarefas quânticas no `CREATED` estado para simuladores QPUs e sob demanda. Você pode cancelar tarefas quânticas no `QUEUED` estado com base no melhor esforço possível para simuladores sob demanda e. QPUs Observe que é improvável que as tarefas quânticas `QUEUED` da QPU sejam canceladas com sucesso durante as janelas de disponibilidade da QPU.

Nesta seção:

- AQT
- IonQ
- IQM
- Rigetti
- QuEra
- Exemplo: envio de uma tarefa quântica para uma QPU
- Inspecionando circuitos compilados

AQT

AQTO QPU IBEX-Q1 da IBEX-Q1 é baseado em um cristal de ^{40} íons Ca^{+} em uma armadilha macroscópica de radiofrequência localizada em uma câmara de vácuo ultra-alto. O dispositivo funciona em temperatura ambiente e cabe em dois racks compatíveis com datacenter de 19 polegadas.

As portas de alta fidelidade são ativadas pelas baixas taxas de aquecimento da armadilha e pelo uso de uma transição óptica direta para rotação de qubits. A transição de qubit é conduzida por um

laser de largura de linha estreita com uma estabilidade de frequência relativa muito alta. Os qubits também apresentam preparação e leitura eficientes do estado por meio de prateleiras ópticas. All-to-alla conectividade é alcançada pela interação de longo alcance de Coulomb no cristal de íons. O endereçamento e a leitura de íon único são obtidos pelo uso de uma lente de alta abertura numérica.

O AQT dispositivo suporta as seguintes portas quânticas.

```
'ccnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01', 'cphaseshift10',
 'cswap', 'swap', 'iswap', 'pswap', 'ecr', 'cy', 'cz', 'xy', 'xx', 'yy', 'zz', 'h',
 'i', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 't', 'ti', 'v', 'vi', 'x', 'y', 'z',
 'prx'
```

Com a compilação literal, o AQT dispositivo suporta as seguintes portas nativas.

```
'prx', 'xx', 'rz'
```

> **Note**
>
> A seguir, descrevemos portas equivalentes entre portas AQT nativas e o Amazon Braket:
>
> - O portão AQT Mølmer-Sørensen (MS ou RXX) corresponde ao portão de Braket `'xx'`
> - O portão AQT R corresponde ao portão de Braket `'prx'`
> - O nome do `'rz'` portão é o mesmo

IonQ

IonQoferece tecnologia baseada em portas com QPUs base na tecnologia de armadilha de íons. IonQ'síons aprisionados QPUs são construídos em uma cadeia de íons 171Yb+ aprisionados que são confinados espacialmente por meio de uma armadilha de eletrodo de superfície microfabricada dentro de uma câmara de vácuo.

Os dispositivos IonQ aceitam as seguintes portas quânticas.

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

Com a compilação literal, eles IonQ QPUs suportam as seguintes portas nativas.

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

Se você especificar apenas dois parâmetros de fase ao usar a porta MS nativa, uma porta MS totalmente entrelaçada será executada. Uma porta MS totalmente entrelaçada sempre executa uma rotação π/2. Para especificar um ângulo diferente e executar uma porta MS parcialmente entrelaçada, você especifica o ângulo desejado adicionando um terceiro parâmetro. Para obter mais informações, consulte o módulo braket.circuits.gate.

Esses portões nativos só podem ser usados com compilação literal. Para saber mais sobre compilação literal, consulte Compilação literal.

IQM

Processadores IQM quânticos são dispositivos do modelo de porta universal baseados em qubits transmon supercondutores. IQM Garnet é um dispositivo de 20 qubits, enquanto IQM Emerald é um dispositivo de 54 qubits. Ambos os dispositivos usam uma topologia de rede quadrada, também conhecida como topologia de rede cristalina.

Os dispositivos IQM suportam as seguintes portas quânticas.

```
"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"
```

Com a compilação literal, os dispositivos IQM suportam as seguintes portas nativas.

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

Rigetti

Os processadores Rigetti quânticos são máquinas universais, do tipo porta, baseadas em supercondutores totalmente ajustáveis qubits.

- O sistema Ankaa-3 é um dispositivo de 84 qubits que utiliza tecnologia escalável de vários chips.

O dispositivo Rigetti suporta as seguintes portas quânticas.

```
'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'
```

Com a compilação literal, o Ankaa-3 suporta as seguintes portas nativas.

```
'rx', 'rz', 'iswap'
```

Processadores Rigetti quânticos supercondutores podem executar a porta 'rx' com apenas os ângulos de ±π/2 ou ±π.

O controle de nível de pulso está disponível nos dispositivos Rigetti, que aceitam um conjunto de quadros predefinidos dos seguintes tipos para o sistema Ankaa-3.

```
`flux_tx`, `charge_tx`, `readout_rx`, `readout_tx`
```

O Ankaa-3 dispositivo tem um limite máximo de 20.000 portas por circuito. Os circuitos que excedem esse limite são rejeitados com um erro de validação. Esse é um limite fixo que não pode ser aumentado. A contagem de portas se refere ao circuito compilado, que pode ser diferente da contagem de portas do circuito não compilado original. Para estimar a contagem de portas compiladas antes de enviar para a QPU, você pode usar a compilação literal localmente ou transpilar seu circuito para o conjunto de portas nativo (,,). `rx` `rz` `iswap`

QuEra

QuEra oferece dispositivos baseados em átomos neutros que podem executar tarefas quânticas de Simulação Hamiltoniana Analógica (AHS). Esses dispositivos para fins especiais reproduzem fielmente a dinâmica quântica dependente do tempo de centenas de qubits que interagem simultaneamente.

Pode-se programar esses dispositivos no paradigma da simulação hamiltoniana analógica prescrevendo o layout do registro de qubits e a dependência temporal e espacial dos campos manipuladores. O Amazon Braket fornece utilitários para construir esses programas por meio do módulo AHS do SDK Python, `braket.ahs`.

Para obter mais informações, consulte os cadernos de exemplo de simulação hamiltoniana analógica ou a página Enviar um programa analógico usando o Aquila. QuEra

Exemplo: envio de uma tarefa quântica para uma QPU

O Amazon Braket permite que você execute um circuito quântico em um dispositivo QPU. O exemplo a seguir mostra como enviar uma tarefa quântica para nossos dispositivos Rigetti ou IonQ.

Escolha o dispositivo Rigetti Ankaa-3 e, em seguida, veja o gráfico de conectividade associado

```
# import the QPU module
from braket.aws import AwsDevice
# choose the Rigetti device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

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

```
{'fullyConnected': False,
 'connectivityGraph': {'0': ['1', '7'],
  '1': ['0', '2', '8'],
  '2': ['1', '3', '9'],
  '3': ['2', '4', '10'],
  '4': ['3', '5', '11'],
  '5': ['4', '6', '12'],
  '6': ['5', '13'],
  '7': ['0', '8', '14'],
  '8': ['1', '7', '9', '15'],
  '9': ['2', '8', '10', '16'],
  '10': ['3', '9', '11', '17'],
  '11': ['4', '10', '12', '18'],
  '12': ['5', '11', '13', '19'],
  '13': ['6', '12', '20'],
  '14': ['7', '15', '21'],
  '15': ['8', '14', '22'],
  '16': ['9', '17', '23'],
  '17': ['10', '16', '18', '24'],
  '18': ['11', '17', '19', '25'],
  '19': ['12', '18', '20', '26'],
  '20': ['13', '19', '27'],
  '21': ['14', '22', '28'],
  '22': ['15', '21', '23', '29'],
  '23': ['16', '22', '24', '30'],
  '24': ['17', '23', '25', '31'],
  '25': ['18', '24', '26', '32'],
  '26': ['19', '25', '33'],
```

```
    '27': ['20', '34'],
    '28': ['21', '29', '35'],
    '29': ['22', '28', '30', '36'],
    '30': ['23', '29', '31', '37'],
    '31': ['24', '30', '32', '38'],
    '32': ['25', '31', '33', '39'],
    '33': ['26', '32', '34', '40'],
    '34': ['27', '33', '41'],
    '35': ['28', '36', '42'],
    '36': ['29', '35', '37', '43'],
    '37': ['30', '36', '38', '44'],
    '38': ['31', '37', '39', '45'],
    '39': ['32', '38', '40', '46'],
    '40': ['33', '39', '41', '47'],
    '41': ['34', '40', '48'],
    '42': ['35', '43', '49'],
    '43': ['36', '42', '44', '50'],
    '44': ['37', '43', '45', '51'],
    '45': ['38', '44', '46', '52'],
    '46': ['39', '45', '47', '53'],
    '47': ['40', '46', '48', '54'],
    '48': ['41', '47', '55'],
    '49': ['42', '56'],
    '50': ['43', '51', '57'],
    '51': ['44', '50', '52', '58'],
    '52': ['45', '51', '53', '59'],
    '53': ['46', '52', '54'],
    '54': ['47', '53', '55', '61'],
    '55': ['48', '54', '62'],
    '56': ['49', '57', '63'],
    '57': ['50', '56', '58', '64'],
    '58': ['51', '57', '59', '65'],
    '59': ['52', '58', '60', '66'],
    '60': ['59'],
    '61': ['54', '62', '68'],
    '62': ['55', '61', '69'],
    '63': ['56', '64', '70'],
    '64': ['57', '63', '65', '71'],
    '65': ['58', '64', '66', '72'],
    '66': ['59', '65', '67'],
    '67': ['66', '68'],
    '68': ['61', '67', '69', '75'],
    '69': ['62', '68', '76'],
    '70': ['63', '71', '77'],
```

```
  '71': ['64', '70', '72', '78'],
  '72': ['65', '71', '73', '79'],
  '73': ['72', '80'],
  '75': ['68', '76', '82'],
  '76': ['69', '75', '83'],
  '77': ['70', '78'],
  '78': ['71', '77', '79'],
  '79': ['72', '78', '80'],
  '80': ['73', '79', '81'],
  '81': ['80', '82'],
  '82': ['75', '81', '83'],
  '83': ['76', '82']}}
```

O dicionário `connectivityGraph` anterior lista os qubits vizinhos para cada qubit no dispositivo Rigetti.

Escolha o dispositivo IonQ Forte-Enterprise-1

Para o IonQ Forte-Enterprise-1 dispositivo, o `connectivityGraph` está vazio, conforme mostrado no exemplo a seguir, porque o dispositivo oferece all-to-allconectividade. Portanto, um `connectivityGraph` detalhado não é necessário.

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

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

```
{'fullyConnected': True, 'connectivityGraph': {...}}
```

Conforme mostrado no exemplo a seguir, você tem a opção de ajustar o shots (padrão = 1000), o `poll_timeout_seconds` (padrão = 432000 = 5 dias), o `poll_interval_seconds` (padrão = 1) e o local do bucket do S3 (`s3_location`) em que seus resultados serão armazenados se você optar por especificar um local diferente do bucket padrão.

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

Os dispositivos Rigetti e IonQ compilam automaticamente o circuito fornecido em seus respectivos conjuntos de portas nativas e mapeiam os índices qubit abstratos para qubits físicos na respectiva QPU.

> **Note**
>
> Os dispositivos QPU têm capacidade limitada. Pode haver tempos de espera mais longos quando a capacidade for atingida.

O Amazon Braket pode executar tarefas quânticas de QPU dentro de determinadas janelas de disponibilidade, mas você ainda pode enviar tarefas quânticas a qualquer momento (24 horas por dia, 7 dias por semana) porque todos os dados e metadados correspondentes são armazenados de forma confiável no bucket S3 apropriado. Conforme mostrado na próxima seção, você pode recuperar sua tarefa quântica usando `AwsQuantumTask` e seu ID exclusivo de tarefa quântica.

Inspecionando circuitos compilados

Quando um circuito quântico precisa ser executado em um dispositivo de hardware, como uma unidade de processamento quântico (QPU), o circuito deve primeiro ser compilado em um formato aceitável que o dispositivo possa entender e processar. Por exemplo, transpilar o circuito quântico de alto nível até as portas nativas específicas suportadas pelo hardware de QPU de destino. Inspecionar a saída real compilada do circuito quântico pode ser extremamente útil para fins de depuração e otimização. Esse conhecimento pode ajudar a identificar possíveis problemas, gargalos ou oportunidades para melhorar o desempenho e a eficiência da aplicação quântica. Você pode visualizar e analisar a saída compilada de seus circuitos quânticos, tanto para dispositivos Rigetti e IQM de computação quântica quanto para dispositivos físicos, usando o código fornecido abaixo.

```
task = AwsQuantumTask(arn=task_id, aws_session=session)
# After the task has finished running
task_result = task.result()
compiled_circuit = task_result.get_compiled_circuit()
```

> **Note**
>
> Atualmente, a visualização da saída do circuito compilado para dispositivos IonQ não tem suporte.

Executar vários programas

O Amazon Braket oferece duas abordagens para executar vários programas quânticos de forma eficiente: conjuntos de programas e agrupamento de tarefas quânticas.

Os conjuntos de programas são a forma preferida de executar workloads com vários programas. Eles permitem que você empacote vários programas em uma única tarefa quântica do Amazon Braket. Os conjuntos de programas oferecem melhorias de desempenho e economia de custos em comparação ao envio de programas individualmente, especialmente quando o número de execuções do programa se aproxima de 100.

Atualmente, os dispositivos IQM e Rigetti suportam conjuntos de programas. Antes de enviar o programa definido para QPUs, é recomendável testar primeiro no Amazon Braket Local Simulator. Para verificar se um dispositivo suporta conjuntos de programas, você pode visualizar as propriedades do dispositivo usando o Amazon Braket SDK ou visualizar a página do dispositivo no Amazon Braket Console.

O exemplo a seguir mostra como executar um conjunto de programas.

```
from math import pi
from braket.devices import LocalSimulator
from braket.program_sets import ProgramSet
from braket.circuits import Circuit

program_set = ProgramSet([
    Circuit().h(0).cnot(0,1),
    Circuit().rx(0, pi/4).ry(1, pi/8).cnot(1,0),
    Circuit().t(0).t(1).cz(0,1).s(0).cz(1,2).s(1).s(2),
])

device = LocalSimulator()
result = device.run(program_set, shots=300).result()
print(result[0][0].counts)  # The result of the first program in the program set
```

Para saber mais sobre as diferentes formas de construir um conjunto de programas (por exemplo, construir um conjunto de programas a partir de vários observáveis ou parâmetros com um único programa) e recuperar os resultados do conjunto de programas, consulte a seção de conjuntos de programas no Guia do desenvolvedor do Amazon Braket e a pasta de conjuntos de programas no repositório Github de exemplos do Braket.

O agrupamento quântico de tarefas está disponível em todos os dispositivos Amazon Braket. O agrupamento em lotes é especialmente útil para tarefas quânticas que você executa nos simuladores sob demanda (SV1, DM1 ou TN1) porque eles podem processar várias tarefas quânticas em paralelo. O processamento em lotes permite que você inicie tarefas quânticas em paralelo. Por exemplo, se você deseja fazer um cálculo que exija 10 tarefas quânticas e os programas nessas tarefas quânticas são independentes uns dos outros, é recomendável usar o agrupamento de tarefas. Use o agrupamento quântico de tarefas ao executar workloads com vários programas em um dispositivo que não suporta conjuntos de programas.

O exemplo a seguir mostra como executar um lote de tarefas quânticas.

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

bell = Circuit().h(0).cnot(0, 1)
circuits = [bell for _ in range(5)]

device = LocalSimulator()
batch = device.run_batch(circuits, shots=100)
print(batch.results()[0].measurement_counts)  # The result of the first quantum task in
 the batch
```

Para obter informações mais específicas sobre agrupamento em lotes, consulte os exemplos do Amazon GitHub Braket em.

Nesta seção:

- Sobre o conjunto de programas e os custos
- Sobre lotes e custos de tarefas quânticas
- Tratamento por lotes de tarefas quânticas e PennyLane
- Agrupamento de tarefas e circuitos parametrizados

Sobre o conjunto de programas e os custos

Os conjuntos de programas executam com eficiência vários programas quânticos, agrupando até 100 programas ou conjuntos de parâmetros em uma única tarefa quântica. Com os conjuntos de programas, você paga apenas uma taxa por tarefa mais as taxas por dose com base no total de fotos em todos os programas, reduzindo significativamente os custos em comparação com o envio de programas individualmente. Essa abordagem é particularmente benéfica para workloads com muitos

programas e com baixo número de disparos por programa. Atualmente, os conjuntos de programas são suportados em dispositivos Rigetti e IQM, e no Amazon Braket Local Simulator.

Para obter mais informações, consulte a seção Conjuntos de programas para ver as etapas detalhadas de implementação, as melhores práticas e exemplos de código.

Sobre lotes e custos de tarefas quânticas

Algumas ressalvas a serem lembradas em relação aos custos de agrupamento e cobrança de tarefas quânticas:

- Por padrão, o agrupamento de tarefas quânticas tenta novamente o tempo limite ou falha nas tarefas quânticas 3 vezes.
- Um lote de tarefas quânticas de longa execução, como 34 qubits paraSV1, pode gerar grandes custos. Certifique-se de verificar cuidadosamente os valores da atribuição `run_batch` antes de iniciar um lote de tarefas quânticas. Não recomendamos usar TN1 com `run_batch`.
- TN1podem incorrer em custos por falhas nas tarefas da fase de ensaio (consulte a TN1 descrição para obter mais informações). Novas tentativas automáticas podem aumentar o custo e, portanto, recomendamos definir o número de 'max_retries' no lote como 0 durante ao usar TN1 (consulte Agrupamento de tarefas quânticas, Linha 186).

Tratamento por lotes de tarefas quânticas e PennyLane

Aproveite o agrupamento em lotes ao usar PennyLane no Amazon Braket `parallel = True` configurando quando você instancia um dispositivo Amazon Braket, conforme mostrado no exemplo a seguir.

```
import pennylane as qml

# Define the number of wires (qubits) you want to use
wires = 2  # For example, using 2 qubits

# Define your S3 bucket
my_bucket = "amazon-braket-s3-demo-bucket"
my_prefix = "pennylane-batch-output"
s3_folder = (my_bucket, my_prefix)

device = qml.device("braket.aws.qubit",
                    device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
```

```
                    wires=wires,
                    s3_destination_folder=s3_folder,
                    parallel=True)
```

Para obter mais informações sobre o agrupamento em lotes com PennyLane, consulte Otimização paralelizada de circuitos quânticos.

Agrupamento de tarefas e circuitos parametrizados

Ao enviar um lote de tarefas quânticas que contém circuitos parametrizados, você pode fornecer um dicionário `inputs`, que é usado para todas as tarefas quânticas do lote, ou um dicionário `list` de entrada. Nesse caso, o `i`-ésimo dicionário é emparelhado com a `i`-ésima tarefa, conforme mostrado no exemplo a seguir.

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

# Define your quantum device
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")

# Create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')

# Create two circuits
circ_a = Circuit().rx(0, alpha).ry(1, alpha).cnot(0, 2).xx(0, 2, beta)
circ_a.variance(observable=Observable.Z(), target=0)

circ_b = Circuit().rx(0, alpha).rz(1, alpha).cnot(0, 2).zz(0, 2, beta)
circ_b.expectation(observable=Observable.Z(), target=2)

# Use the same inputs for both circuits in one batch
tasks = device.run_batch([circ_a, circ_b], inputs={'alpha': 0.1, 'beta': 0.2})

# Or provide each task its own set of inputs
inputs_list = [{'alpha': 0.3, 'beta': 0.1}, {'alpha': 0.1, 'beta': 0.4}]

tasks = device.run_batch([circ_a, circ_b], inputs=inputs_list)
```

Você também pode preparar uma lista de dicionários de entrada para um único circuito paramétrico e enviá-los como um lote de tarefas quânticas. Se houver N dicionários de entrada na lista, o lote

contém N tarefas quânticas. A `i` -ésima tarefa quântica corresponde ao circuito executado com o `i` -ésimo dicionário 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, shots=100)
```

Quando minha tarefa quântica será executada?

Quando você envia um circuito, o Amazon Braket o envia para o dispositivo que você especificar. As tarefas quânticas da Unidade de Processamento Quântico (QPU) e do simulador sob demanda são colocadas em fila e processadas na ordem em que são recebidas. O tempo necessário para processar sua tarefa quântica após enviá-la varia dependendo do número e da complexidade das tarefas enviadas por outros clientes do Amazon Braket e da disponibilidade da QPU selecionada.

Nesta seção:

- Janelas e status de disponibilidade da QPU
- Visibilidade da fila
- Configure notificações por e-mail ou SMS

Janelas e status de disponibilidade da QPU

A disponibilidade da QPU varia de dispositivo para dispositivo.

Na página Dispositivos do console Amazon Braket, você pode ver as janelas de disponibilidade atuais e futuras e o status do dispositivo. Além disso, cada página do dispositivo mostra profundidades de fila individuais para tarefas quânticas e trabalhos híbridos.

Um dispositivo é considerado offline se não estiver disponível para os clientes, independentemente da janela de disponibilidade. Por exemplo, ele pode estar offline devido a manutenção programada, atualizações ou problemas operacionais.

Visibilidade da fila

Antes de enviar uma tarefa quântica ou um trabalho híbrido, você pode ver quantas tarefas quânticas ou trabalhos híbridos estão à sua frente verificando a profundidade da fila do dispositivo.

Profundidade da fila

Queue depth refere-se ao número de tarefas quânticas e trabalhos híbridos em fila para um determinado dispositivo. As tarefas quânticas e a contagem de filas de tarefas híbridas de um dispositivo podem ser acessadas por meio do Braket Software Development Kit (SDK) ou Amazon Braket Management Console.

1. A profundidade da fila de tarefas se refere ao número total de tarefas quânticas atualmente esperando para serem executadas em prioridade normal.
2. A profundidade da fila de tarefas prioritárias se refere ao número total de tarefas quânticas enviadas aguardando execução do Amazon Braket Hybrid Jobs. Essas tarefas são executadas antes das tarefas autônomas.
3. A profundidade da fila de trabalhos híbridos se refere ao número total de trabalhos híbridos atualmente em fila em um dispositivo. Quantum tasks enviadas como parte de um trabalho híbrido têm prioridade e são agregadas à Priority Task Queue.

Os clientes que desejam visualizar a profundidade da fila por meio do Braket SDK podem modificar o seguinte trecho de código para obter a posição na fila de sua tarefa quântica ou trabalho híbrido:

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

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


# returns the number of hybrid jobs queued on the device
print(device.queue_depth().jobs)
'3'
```

Enviar uma tarefa quântica ou um trabalho híbrido para uma QPU pode fazer com que sua workload fique em estado `QUEUED`. O Amazon Braket oferece aos clientes visibilidade de suas tarefas quânticas e da posição híbrida na fila de trabalhos.

Posição da fila

Queue position refere-se à posição atual de sua tarefa quântica ou trabalho híbrido em uma respectiva fila de dispositivos. Ele pode ser obtido para tarefas quânticas ou trabalhos híbridos por meio do Braket Software Development Kit (SDK) ou Amazon Braket Management Console.

Os clientes que desejam visualizar a posição da fila por meio do Braket SDK podem modificar o seguinte trecho de código para obter a posição na fila de sua tarefa quântica ou trabalho híbrido:

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

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

# retrieve the queue position information
print(task.queue_position().queue_position)

# Returns the number of Quantum Tasks queued ahead of you
'2'


from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=False
)

# retrieve the queue position information
print(job.queue_position().queue_position)
'3' # returns the number of hybrid jobs queued ahead of you
```

Configure notificações por e-mail ou SMS

O Amazon Braket envia eventos para a EventBridge Amazon quando a disponibilidade de uma QPU muda ou quando o estado de sua tarefa quântica muda. Siga estas etapas para receber notificações de alteração do status do dispositivo e da tarefa quântica por e-mail ou mensagem SMS:

1. Criar um tópico do Amazon SNS e uma assinatura para e-mail ou SMS. A disponibilidade de e-mail ou SMS depende da sua região. Para obter mais informações, consulte Introdução ao Amazon SNS e Envio de mensagens SMS.
2. Crie uma regra EventBridge que acione as notificações para seu tópico do SNS. Para obter mais informações, consulte Monitoramento do Amazon Braket com a Amazon. EventBridge

(Opcional) Configure as notificações de SNS

Você pode configurar notificações por meio do Amazon Simple Notification Service (SNS) para receber um alerta quando sua tarefa quântica do Amazon Braket for concluída. As notificações ativas são úteis se você espera um longo tempo de espera; por exemplo, quando você envia uma grande tarefa quântica ou quando envia uma tarefa quântica fora da janela de disponibilidade de um dispositivo. Se você não quiser esperar a conclusão da tarefa quântica, você pode configurar uma notificação do SNS.

Um caderno Amazon Braket orienta você nas etapas de configuração. Para obter mais informações, consulte os exemplos do Amazon Braket GitHub em e, especificamente, o exemplo de caderno para configurar notificações.

Trabalhando com reservas

As reservas oferecem acesso exclusivo ao dispositivo quântico de sua escolha. Você pode agendar uma reserva conforme sua conveniência, para saber exatamente quando sua workload começa e termina a execução. As reservas estão disponíveis em incrementos de 1 hora para todos os dispositivos Braket e podem ser canceladas com até 48 horas de antecedência, sem custo adicional. Recomendamos enfileirar tarefas quânticas e trabalhos híbridos para uma reserva futura com antecedência, usando seu ARN de reserva direta do Braket ou enviando cargas de trabalho durante sua reserva.

O custo do acesso dedicado ao dispositivo é baseado na duração da sua reserva, independentemente de quantas tarefas quânticas e trabalhos híbridos você executa na unidade de processamento quântico (QPU). Uma lista atualizada de computadores quânticos disponíveis para reservas pode ser encontrada em nossa página de preços ou por meio do console de gerenciamento do Amazon Braket.

> **Note**
>
> Em uma reserva, não há limites de gateshot. Além disso, para IonQ dispositivos, a contagem mínima de disparos para tarefas de mitigação de erros é reduzida para 500 (versus 2500 para tarefas sob demanda).

Quando usar uma reserva

Aproveitar o acesso à reserva oferece a conveniência e a previsibilidade de saber exatamente quando sua carga de trabalho quântica começa e termina a execução. Em comparação com o envio de tarefas e trabalhos híbridos sob demanda, você não precisa esperar na fila com outras tarefas do cliente. Como você tem acesso exclusivo ao dispositivo durante sua reserva, somente suas workloads são executadas no dispositivo durante toda a reserva.

Recomendamos usar o acesso sob demanda para a fase de projeto e prototipagem de sua pesquisa, permitindo uma iteração rápida e econômica de seus algoritmos. Quando estiver pronto para produzir os resultados finais do experimento, considere agendar uma reserva de dispositivo conforme sua conveniência para garantir que você possa cumprir os prazos do projeto ou da publicação. Também recomendamos o uso de reservas quando você desejar executar tarefas em horários específicos, como quando estiver executando uma demonstração ao vivo ou um workshop em um computador quântico.

Nesta seção:

- Como criar uma reserva
- Executando tarefas quânticas durante uma reserva
- Executando trabalhos híbridos durante uma reserva
- O que acontece no final da sua reserva
- Cancelar ou reagendar uma reserva existente

Como criar uma reserva

Para criar uma reserva, entre em contato com a equipe Braket seguindo estas etapas:

1. Abra o console do Amazon Braket.
2. Escolha Braket Direct no painel esquerdo e, na seção Reservas, escolha Reservar dispositivo.

3. Selecione o Dispositivo que você gostaria de reservar.
4. Forneça suas informações de contato, incluindo Nome e E-mail. Certifique-se de fornecer um endereço de e-mail válido que você verifique regularmente.
5. Em Conte-nos sobre sua workload, forneça todos os detalhes sobre a workload a ser executada usando sua reserva. Por exemplo, duração desejada da reserva, restrições relevantes ou cronograma desejado.

Depois de enviar o formulário, você receberá um e-mail da equipe Braket com as próximas etapas. Assim que sua reserva for confirmada, você receberá o ARN da reserva por meio de seu e-mail. Você precisará usar o ARN da reserva para criar tarefas de reserva. As tarefas criadas sem o ARN da reserva serão enviadas para a fila normal sob demanda e NÃO serão executadas durante sua reserva.

> **Note**
>
> Sua reserva só é confirmada quando você recebe o ARN da reserva.

As reservas estão disponíveis em incrementos mínimos de 1 hora e alguns dispositivos podem ter restrições adicionais de duração da reserva (incluindo durações mínimas e máximas de reserva). A equipe Braket compartilha todas as informações relevantes com você antes de confirmar a reserva.

A equipe da Braket entrará em contato com você por e-mail para agendar uma sessão de 30 minutos com um especialista da Braket.

Executando tarefas quânticas durante uma reserva

Depois de obter um ARN de reserva válido em Criar uma reserva, você pode criar tarefas quânticas para serem executadas durante a reserva. Tarefas quânticas e trabalhos híbridos enviados com um ARN de reserva não aparecerão na fila do dispositivo. As tarefas enviadas antes do horário de início da reserva permanecerão no `QUEUED` estado até o início da reserva.

> **Note**
>
> As reservas são específicas AWS da conta e do dispositivo. Somente a AWS conta que criou a reserva pode usar o ARN da sua reserva.
> Durante uma reserva, tanto a reserva quanto as tarefas regulares podem ser criadas. Para verificar se uma tarefa quântica criada pelo Braket está associada a uma reserva, verifique

o campo “ARN da reserva” na página da tarefa quântica no console do Braket ou consulte o mesmo campo nos metadados da tarefa usando o SDK. O restante desta página descreve como especificar quais tarefas estão associadas à reserva.

Você pode criar tarefas quânticas usando Python SDKs como Braket,,, CUDA-QPennyLaneQiskit, ou diretamente com o boto3 (Trabalhando com o Boto3). Para usar reservas, você deve ter a versão v1.79.0 ou superior do Amazon Braket Python SDK Você pode atualizar para o SDK, o provedor Qiskit e o plug-in PennyLane mais recentes do Braket com o código a seguir.

```
pip install --upgrade amazon-braket-sdk amazon-braket-pennylane-plugin qiskit-braket-
provider
```

Execute tarefas com o gerenciador de contexto **`DirectReservation`**

A forma recomendada de executar uma tarefa dentro da sua reserva agendada é usar o gerenciador de contexto `DirectReservation`. Ao especificar seu dispositivo de destino e o ARN de reserva, o gerenciador de contexto garante que todas as tarefas criadas na instrução `with` Python sejam executadas com acesso exclusivo ao dispositivo.

Primeiro, defina um circuito quântico e o dispositivo. Em seguida, use o contexto de reserva e execute a tarefa. Certifique-se de que toda a sua carga de trabalho seja executada dentro do **`with`** bloco; qualquer coisa executada fora do escopo do **`with`** bloco não será associada à sua reserva!

```
from braket.aws import AwsDevice, DirectReservation
from braket.circuits import Circuit
from braket.devices import Devices

bell = Circuit().h(0).cnot(0, 1)
device = AwsDevice(Devices.IonQ.ForteEnterprise1)

# run the circuit in a reservation
with DirectReservation(device, reservation_arn="<my_reservation_arn>"):
    task = device.run(bell, shots=100)
```

Você pode criar tarefas quânticas em uma reserva usando CUDA-QPennyLane, e Qiskit plug-ins, desde que o `DirectReservation` contexto esteja ativo durante a criação de tarefas quânticas. Por exemplo, com o provedor Qiskit-Braket, você pode executar tarefas da seguinte maneira.

```
from braket.devices import Devices
```

```
from braket.aws import DirectReservation
from qiskit import QuantumCircuit
from qiskit_braket_provider import BraketProvider


qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)

qpu = BraketProvider().get_backend("Forte Enterprise 1")

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    qpu_task = qpu.run(qc, shots=10)
```

Da mesma forma, o código a seguir executa um circuito durante uma reserva usando o plug-in Braket-PennyLane.

```
from braket.devices import Devices
from braket.aws import DirectReservation
import pennylane as qml


dev = qml.device("braket.aws.qubit", device_arn=Devices.IonQ.ForteEnterprise1.value,
 wires=2, shots=10)

@qml.qnode(dev)
def bell_state():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    return qml.probs(wires=[0, 1])

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    probs = bell_state()
```

Configurando manualmente o contexto da reserva

Como alternativa, você pode definir manualmente o contexto da reserva com o seguinte código.

```
# set reservation context
```

```
reservation_context = DirectReservation(device,
 reservation_arn="<my_reservation_arn>").start()

# run circuit during reservation
task = device.run(bell, shots=100)
```

Isso é ideal para cadernos Jupyter, onde o contexto pode ser executado na primeira célula e todas as tarefas subsequentes serão executadas na reserva.

> **Note**
>
> A célula que contém a chamada `.start()` só deve ser executada uma vez.

Para voltar ao modo sob demanda: reinicie o caderno Jupyter ou ligue para o seguinte para alterar o contexto de volta ao modo sob demanda.

```
reservation_context.stop()  # unset reservation context
```

> **Note**
>
> As reservas têm um horário de início e término predeterminado (consulte Criar uma reserva). Os métodos `reservation_context.stop()` e `reservation_context.start()` não iniciam nem encerram uma reserva. Em vez disso, enquanto o contexto estiver ativo, todas as tarefas quânticas que você criar serão associadas à sua reserva e serão executadas somente durante a reserva agendada. O contexto da reserva não tem efeito no horário agendado da reserva.

Passe explicitamente o ARN da reserva ao criar a tarefa

Outra forma de criar tarefas durante uma reserva é passar explicitamente o ARN da reserva ao chamar `device.run()`.

```
task = device.run(bell, shots=100, reservation_arn="<my_reservation_arn>")
```

Esse método associa diretamente a tarefa quântica ao ARN da reserva, garantindo que ela seja executada durante o período reservado. Para essa opção, adicione o ARN da reserva a cada tarefa

que você planeja executar durante uma reserva. No entanto, observe que, ao usar bibliotecas de terceirosPennyLane, como Qiskit ou, pode ser difícil garantir que as tarefas enviadas estejam usando o ARN de reserva correto. Por esse motivo, é recomendável usar o gerenciador de DirectReservation contexto.

Ao usar o boto3 diretamente, passe o ARN da reserva como uma associação ao criar uma tarefa.

```
import boto3

braket_client = boto3.client("braket")


kwargs["associations"] = [
    {
        "arn": "<my_reservation_arn>",
        "type": "RESERVATION_TIME_WINDOW_ARN",
    }
]

response = braket_client.create_quantum_task(**kwargs)
```

Executando trabalhos híbridos durante uma reserva

Depois de ter uma função Python para executar como uma tarefa híbrida, você pode executar a tarefa híbrida em uma reserva passando o argumento `reservation_arn` da palavra-chave. Todas as tarefas dentro do trabalho híbrido usam o ARN da reserva. É importante ressaltar que o trabalho híbrido com `reservation_arn` só ativa a computação clássica quando sua reserva começa.

> **Note**
>
> Uma tarefa híbrida executada durante uma reserva só executa com êxito tarefas quânticas no dispositivo reservado. A tentativa de usar um dispositivo Braket sob demanda diferente gerará um erro. Se você precisar executar tarefas em um simulador sob demanda e no dispositivo reservado na mesma tarefa híbrida, use `DirectReservation` em vez disso.

O código a seguir demonstra como executar um trabalho híbrido durante uma reserva.

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

```
from braket.devices import Devices
from braket.jobs import get_job_device_arn, hybrid_job

@hybrid_job(device=Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>")
def example_hybrid_job():
    # declare AwsDevice within the hybrid job
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)

    task = device.run(bell, shots=10)
```

Para trabalhos híbridos que usam um script Python (consulte a seção Criando seu primeiro trabalho híbrido no Guia do desenvolvedor), você pode executá-los dentro da reserva transmitindo o argumento `reservation_arn` de palavra-chave ao criar o trabalho.

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

job = AwsQuantumJob.create(
    Devices.IonQ.ForteEnterprise1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="<my_reservation_arn>"
)
```

O que acontece no final da sua reserva

Depois que sua reserva terminar, você não terá mais acesso dedicado ao dispositivo. Todas as workloads restantes que estão na fila com essa reserva são automaticamente canceladas.

> **Note**
>
> Qualquer trabalho que estava em status `RUNNING` quando a reserva termina é cancelado. Recomendamos o uso de pontos de verificação para salvar e reiniciar trabalhos conforme sua conveniência.

Uma reserva em andamento, como após o início da reserva e antes do final da reserva, não pode ser estendida porque cada reserva representa acesso independente a um dispositivo dedicado. Por

exemplo, duas back-to-back reservas são consideradas separadas e todas as tarefas pendentes da primeira reserva são automaticamente canceladas. Eles não são retomados na segunda reserva.

> **Note**
>
> As reservas representam acesso a dispositivos dedicados à sua AWS conta. Mesmo que o dispositivo permaneça ocioso, nenhum outro cliente poderá usá-lo. Portanto, você é cobrado pela duração do tempo reservado, independentemente do tempo utilizado.

Cancelar ou reagendar uma reserva existente

Você pode cancelar sua reserva pelo menos 48 horas antes do horário de início da reserva programada. Para cancelar, responda ao e-mail de confirmação da reserva que você recebeu com sua solicitação de cancelamento.

Para reagendar, você precisa cancelar sua reserva existente e, em seguida, criar uma nova.

Técnicas de mitigação de erros

A mitigação de erros quânticos é um conjunto de técnicas que visa reduzir os efeitos dos erros em computadores quânticos.

Os dispositivos quânticos estão sujeitos a ruídos ambientais que degradam a qualidade dos cálculos realizados. Embora a computação quântica tolerante a falhas prometa uma solução para esse problema, os dispositivos quânticos atuais são limitados pelo número de qubits e taxas de erro relativamente altas. Para combater isso no curto prazo, os pesquisadores estão investigando métodos para melhorar a precisão da computação quântica ruidosa. Essa abordagem, conhecida como mitigação de erros quânticos, envolve o uso de várias técnicas para extrair o melhor sinal de dados de medição ruidosos.

Nesta seção:

- Técnicas de mitigação de erros em dispositivos IonQ

Técnicas de mitigação de erros em dispositivos IonQ

A mitigação de erros envolve a execução de vários circuitos físicos e a combinação de suas medições para obter um resultado aprimorado.

> **Note**
>
> Para todos os dispositivos IonQ: ao usar um modelo sob demanda, há um limite de 1 milhão de disparos e um mínimo de 2.500 disparos para tarefas de Mitigação de erros. Para uma reserva direta, não há limite de disparos e há um mínimo de 500 disparos para tarefas de mitigação de erros.

Eliminação de preconceitos

Dispositivos IonQapresentam um método de mitigação de erros chamado debiasing.

O debiasing mapeia um circuito em várias variantes que atuam em diferentes permutações de qubits ou com diferentes decomposições de portas. Isso reduz o efeito de erros sistemáticos, como sobre-rotações de portas ou um único qubit defeituoso, usando diferentes implementações de um circuito que, de outra forma, poderiam distorcer os resultados da medição. Isso ocorre às custas de uma sobrecarga extra para calibrar vários qubits e portas.

Para obter mais informações sobre despolarização, consulte Aprimoramento do desempenho do computador quântico por meio da simetrização.

> **Note**
>
> Usar a eliminação de polarização requer um mínimo de 2500 disparos.

Você pode executar uma tarefa quântica com despolarização em um dispositivo IonQ usando o seguinte código:

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.error_mitigation import Debias

# choose an IonQ device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1")
circuit = Circuit().h(0).cnot(0, 1)

task = device.run(circuit, shots=2500, device_parameters={"errorMitigation": Debias()})

result = task.result()
```

```
print(result.measurement_counts)
>>> {"00": 1245, "01": 5, "10": 10 "11": 1240} # result from debiasing
```

Quando a tarefa quântica estiver concluída, você poderá ver as probabilidades de medição e qualquer tipo de resultado da tarefa quântica. As probabilidades de medição e as contagens de todas as variantes são agregadas em uma única distribuição. Todos os tipos de resultados especificados no circuito, como valores esperados, são calculados usando as contagens de medição agregadas.

Nitidez

Você também pode acessar as probabilidades de medição calculadas com uma estratégia de pós-processamento diferente chamada nitidez. A nitidez compara os resultados de cada variante e descarta fotos inconsistentes, favorecendo o resultado de medição mais provável entre as variantes. Para obter mais informações, consulte Aprimoramento do desempenho do computador quântico por meio da simetrização.

É importante ressaltar que a nitidez pressupõe que a forma da distribuição de saída seja esparsa, com poucos estados de alta probabilidade e muitos estados de probabilidade zero. Isso pode distorcer a distribuição de probabilidade se essa suposição não for válida.

Você pode acessar as probabilidades a partir de uma distribuição aprimorada no campo `additional_metadata` do `GateModelTaskResult` do SDK Braket Python. Observe que a nitidez não retorna as contagens de medição, mas retorna uma distribuição de probabilidade renormalizada. O trecho de código a seguir mostra como acessar a distribuição após a nitidez.

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

Trabalhando com o Amazon Braket Hybrid Jobs

O Amazon Braket Hybrid Jobs oferece uma maneira de você executar algoritmos clássicos quânticos híbridos que exigem recursos AWS clássicos e unidades de processamento quântico (). QPUs O Hybrid Jobs foi projetado para ativar os recursos clássicos solicitados, executar seu algoritmo e liberar as instâncias após a conclusão, para que você pague apenas pelo que usar.

O Hybrid Jobs é ideal para algoritmos iterativos de longa duração que envolvem o uso de recursos de computação clássica e recursos de computação quântica. Com o Hybrid Jobs, depois de enviar seu algoritmo para execução, o Braket executará seu algoritmo em um ambiente escalável e em contêineres. Depois que o algoritmo for concluído, você poderá recuperar os resultados.

Além disso, as tarefas quânticas criadas a partir de um trabalho híbrido se beneficiam do enfileiramento de maior prioridade no dispositivo QPU de destino. Essa priorização garante que seus cálculos quânticos sejam processados e executados antes de outras tarefas que aguardam na fila. Isso é particularmente vantajoso para algoritmos híbridos iterativos, nos quais os resultados de uma tarefa quântica dependem dos resultados de tarefas quânticas anteriores. Exemplos desses algoritmos incluem o Algoritmo de Otimização Aproximada Quântica (QAOA), o autosolucionador quântico variacional ou o machine learning quântico. Você também pode monitorar o progresso do algoritmo quase em tempo real, permitindo que você acompanhe os custos, o orçamento ou as métricas personalizadas, como perda de treinamento ou valores esperados.

Você pode acessar trabalhos híbridos no Braket usando:

- O SDK Amazon Braket Python.
- O console Amazon Braket.
- O Amazon Braket API.

Nesta seção:

- Quando usar o Amazon Braket Hybrid Jobs
- Executando um trabalho híbrido com o Amazon Braket Hybrid Jobs
- Conceitos principais do Hybrid Jobs
- Pré-requisitos
- Criar um trabalho híbrido
- Cancelar um Hybrid Job

- Personalizar seu Hybrid Job
- Usando PennyLane com o Amazon Braket
- Usando o CUDA-Q com o Amazon Braket

Quando usar o Amazon Braket Hybrid Jobs

O Amazon Braket Hybrid Jobs permite que você execute algoritmos clássicos quânticos híbridos, como o Variational Quantum Eigensolver (VQE) e o Quantum Approximate Optimization Algorithm (QAOA), que combinam recursos computacionais clássicos com dispositivos de computação quântica para otimizar o desempenho dos sistemas quânticos atuais. O Amazon Braket Hybrid Jobs oferece três benefícios principais:

1. Desempenho: o Amazon Braket Hybrid Jobs oferece melhor desempenho do que executar algoritmos híbridos em seu próprio ambiente. Enquanto seu trabalho está em execução, ele tem acesso prioritário à QPU de destino selecionada. As tarefas do seu trabalho são executadas antes de outras tarefas enfileiradas no dispositivo. Isso resulta em runtimes mais curtos e previsíveis para algoritmos híbridos. O Amazon Braket Hybrid Jobs também oferece suporte à compilação paramétrica. Você pode enviar um circuito usando parâmetros livres e o Braket compila o circuito uma vez, sem a necessidade de recompilar para atualizações subsequentes de parâmetros no mesmo circuito, resultando em runtimes ainda mais rápidos.
2. Conveniência: o Amazon Braket Hybrid Jobs simplifica a configuração e o gerenciamento do seu ambiente computacional e o mantém em execução enquanto seu algoritmo híbrido é executado. Basta fornecer seu script de algoritmo e selecionar um dispositivo quântico (uma unidade de processamento quântico ou um simulador) no qual executar. O Amazon Braket espera que o dispositivo de destino fique disponível, acelera os recursos clássicos, executa a workload em ambientes de contêineres pré-criados, retorna os resultados para o Amazon Simple Storage Service (Amazon S3) e libera os recursos computacionais.
3. Métricas: o Amazon Braket Hybrid Jobs on-the-fly fornece informações sobre a execução de algoritmos e fornece métricas de algoritmo personalizáveis quase em tempo real para a Amazon e para CloudWatch o console Amazon Braket, para que você possa acompanhar o progresso de seus algoritmos.

Executando um trabalho híbrido com o Amazon Braket Hybrid Jobs

Para executar um trabalho híbrido com o Amazon Braket Hybrid Jobs, primeiro você precisa definir seu algoritmo. Você pode defini-lo escrevendo o script do algoritmo e, opcionalmente, outros arquivos de dependência usando o Amazon Braket Python SDK ou. PennyLane Se quiser usar outras bibliotecas (de código aberto ou proprietárias), você pode definir sua própria imagem de contêiner personalizada usando o Docker, que inclui essas bibliotecas. Para obter mais informações, consulte Trazer seu próprio contêiner (BYOC).

Em ambos os casos, em seguida, você cria um trabalho híbrido usando o Amazon Braket API, onde você fornece seu script ou contêiner de algoritmo, seleciona o dispositivo quântico de destino que o trabalho híbrido deve usar e, em seguida, escolhe entre uma variedade de configurações opcionais. Os valores padrão fornecidos para essas configurações opcionais funcionam na maioria dos casos de uso. Para que o dispositivo de destino execute seu Hybrid Job, você pode escolher entre uma QPU, um simulador sob demanda (como SV1, DM1 ou TN1) ou a própria instância de trabalho híbrida clássica. Com um simulador sob demanda ou QPU, seu contêiner de tarefas híbrido faz chamadas de API para um dispositivo remoto. Com os simuladores incorporados, o simulador é incorporado no mesmo contêiner do seu script de algoritmo. Os simuladores de relâmpagos do PennyLane são incorporados ao contêiner de trabalhos híbrido pré-construído padrão para você usar. Se você executar seu código usando um PennyLane simulador incorporado ou personalizado, poderá especificar um tipo de instância e quantas instâncias deseja usar. Consulte a página de Preços do Amazon Braket para ver os custos associados a cada opção.

Se seu dispositivo de destino for um simulador sob demanda ou incorporado, o Amazon Braket começará a executar a tarefa híbrida imediatamente. Ele ativa a instância de trabalho híbrida (você pode personalizar o tipo de instância na chamada API), executa seu algoritmo, grava os resultados no Amazon S3 e libera seus recursos. Essa liberação de recursos garante que você pague apenas pelo que usar.

O número total de trabalhos híbridos simultâneos por unidade de processamento quântico (QPU) é restrito. Atualmente, somente uma tarefa híbrida pode ser executada em uma QPU a qualquer momento. As filas são usadas para controlar o número de trabalhos híbridos que podem ser executados de forma a não exceder o limite permitido. Se o dispositivo de destino for uma QPU, sua tarefa híbrida primeiro entrará na fila de tarefas da QPU selecionada. O Amazon Braket cria a instância de trabalho híbrida necessária e executa sua tarefa híbrida no dispositivo. Durante a duração do seu algoritmo, sua tarefa híbrida tem acesso prioritário, o que significa que as tarefas quânticas de sua tarefa híbrida são executadas antes de outras tarefas quânticas do Braket enfileiradas no dispositivo, desde que as tarefas quânticas da tarefa sejam enviadas à QPU uma vez a cada poucos minutos. Quando o trabalho híbrido é concluído, os recursos são liberados, o que significa que você paga apenas pelo que usa.

> **Note**
>
> Os dispositivos são regionais e sua tarefa híbrida é executada da Região da AWS mesma forma que seu dispositivo principal.

Tanto no simulador quanto nos cenários de destino da QPU, você tem a opção de definir métricas personalizadas do algoritmo, como a energia do seu hamiltoniano, como parte do seu algoritmo. Essas métricas são automaticamente reportadas à Amazon CloudWatch e, a partir daí, são exibidas quase em tempo real no console do Amazon Braket.

> **Note**
>
> Se você quiser usar uma instância baseada em GPU, certifique-se de usar um dos simuladores baseados em GPU disponíveis com os simuladores incorporados no Braket (por exemplo, `lightning.gpu`). Se você escolher um dos simuladores incorporados baseados em CPU (por exemplo, `lightning.qubit` ou `braket:default-simulator`), a GPU não será usada e você poderá incorrer em custos desnecessários.

Conceitos principais do Hybrid Jobs

Esta seção explica os principais conceitos da função `AwsQuantumJob.create` fornecida pelo SDK do Amazon Braket Python e o mapeamento para a estrutura do arquivo de contêiner.

Além do arquivo ou arquivos que compõem seu script de algoritmo completo, sua tarefa híbrida pode ter entradas e saídas adicionais. Quando sua tarefa híbrida é iniciada, o Amazon Braket copia as entradas fornecidas como parte da criação da tarefa híbrida no contêiner que executa o script do algoritmo. Quando o trabalho híbrido é concluído, todas as saídas definidas durante o algoritmo são copiadas para o local especificado no Amazon S3.

> **Note**
>
> As métricas do algoritmo são relatadas em tempo real e não seguem esse procedimento de saída.

O Amazon Braket também fornece várias variáveis de ambiente e funções auxiliares para simplificar as interações com entradas e saídas de contêineres. Para obter mais informações, consulte o pacote braket.jobs no Amazon Braket SDK.

Nesta seção:

- Entradas
- Saídas
- Variáveis de ambiente
- funções auxiliares

Entradas

Dados de entrada: os dados de entrada podem ser fornecidos ao algoritmo híbrido especificando o arquivo de dados de entrada, que é configurado como um dicionário, com o argumento `input_data`. O usuário define o argumento `input_data` dentro da função `AwsQuantumJob.create` no SDK. Isso copia os dados de entrada para o sistema de arquivos do contêiner no local fornecido pela variável de ambiente `"AMZN_BRAKET_INPUT_DIR"`. Para ver alguns exemplos de como os dados de entrada são usados em um algoritmo híbrido, consulte QAOA com Amazon Braket Hybrid Jobs PennyLane e Quantum machine learning nos notebooks Amazon Braket Hybrid Jobs Jupyter.

> **Note**
>
> Quando os dados de entrada forem grandes (>1 GB), haverá um longo tempo de espera até que o trabalho híbrido seja enviado. Isso se deve ao fato de que os dados de entrada locais serão primeiro carregados em um bucket do S3, depois o caminho do S3 será adicionado à solicitação de trabalho híbrida e, por fim, a solicitação de trabalho híbrida será enviada ao serviço Braket.

Hiperparâmetros: se você passar os `hyperparameters`, eles estarão disponíveis na variável de ambiente `"AMZN_BRAKET_HP_FILE"`.

> **Note**
>
> Para obter mais informações sobre como criar hiperparâmetros e dados de entrada e, em seguida, passar essas informações para o script de trabalho híbrido, consulte a seção Usar hiperparâmetros e esta página do GitHub.

Pontos de verificação: Para especificar um ponto de verificação `job-arn` cujo você deseja usar em um novo trabalho híbrido, use o comando `copy_checkpoints_from_job`. Esse comando copia os dados do ponto de verificação `checkpoint_configs3Uri` para o novo trabalho híbrido, disponibilizando-os no caminho fornecido pela variável de ambiente `AMZN_BRAKET_CHECKPOINT_DIR` enquanto o trabalho é executado. O padrão é `None`, o que significa que os dados do ponto de verificação de outro trabalho híbrido não serão usados no novo trabalho híbrido.

Saídas

Tarefas quânticas: os resultados das tarefas quânticas são armazenados no local `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks` do S3.

Resultados do trabalho: tudo o que seu script de algoritmo salva no diretório fornecido pela variável de ambiente `"AMZN_BRAKET_JOB_RESULTS_DIR"` é copiado para o local do S3 especificado em `output_data_config`. Se o valor não for especificado, o padrão será. `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data` Fornecemos a função auxiliar do SDK **`save_job_result`**, que você pode usar para armazenar resultados convenientemente na forma de um dicionário quando chamada a partir do seu script de algoritmo.

Pontos de verificação: Se você quiser usar pontos de verificação, poderá salvá-los no diretório fornecido pela variável de ambiente `"AMZN_BRAKET_CHECKPOINT_DIR"`. Você também pode usar `save_job_checkpoint` para invocar a função.

Métricas de algoritmo: você pode definir métricas de algoritmo como parte do seu script de algoritmo que são emitidas para a Amazon CloudWatch e exibidas em tempo real no console do Amazon Braket enquanto sua tarefa híbrida está em execução. Para obter um exemplo de como usar métricas de algoritmo, consulte Usar Amazon Braket Hybrid Jobs para executar um algoritmo QAOA.

Para obter mais informações sobre como salvar suas saídas de trabalho, consulte Salvar seus resultados na documentação do Hybrid Jobs.

Variáveis de ambiente

O Amazon Braket fornece várias variáveis de ambiente para simplificar as interações com entradas e saídas de contêineres. O código a seguir lista as variáveis ambientais que Braket usa.

- `AMZN_BRAKET_INPUT_DIR`— O diretório de dados de entradaopt/braket/input/data.
- `AMZN_BRAKET_JOB_RESULTS_DIR`— O diretório de saída no opt/braket/model qual gravar os resultados do trabalho.
- `AMZN_BRAKET_JOB_NAME`: o nome do trabalho.
- `AMZN_BRAKET_CHECKPOINT_DIR`— O diretório do ponto de verificação.
- `AMZN_BRAKET_HP_FILE`— O arquivo contendo os hiperparâmetros.
- `AMZN_BRAKET_DEVICE_ARN`— O ARN (nome do AWS recurso) do dispositivo.
- `AMZN_BRAKET_OUT_S3_BUCKET`— O bucket de saída do Amazon S3, conforme especificado na solicitação `CreateJob` do `OutputDataConfig`.
- `AMZN_BRAKET_SCRIPT_ENTRY_POINT`— O ponto de entrada, conforme especificado na solicitação `CreateJob` do `ScriptModeConfig`.
- `AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE`— O tipo de compressão conforme especificado na solicitação `CreateJob` do `ScriptModeConfig`.
- `AMZN_BRAKET_SCRIPT_S3_URI`— A localização do script do usuário no Amazon S3, conforme especificado na solicitação `CreateJob` do `ScriptModeConfig`.
- `AMZN_BRAKET_TASK_RESULTS_S3_URI`— O local do Amazon S3 em que o SDK armazenaria os resultados da tarefa quântica por padrão para o trabalho.
- `AMZN_BRAKET_JOB_RESULTS_S3_PATH`— O local do Amazon S3 onde os resultados do trabalho seriam armazenados, conforme especificado na solicitação `CreateJob` do `OutputDataConfig`.

- `AMZN_BRAKET_JOB_TOKEN`— A string que deve ser passada para o parâmetro `jobToken` do `CreateQuantumTask` para tarefas quânticas criadas no contêiner de tarefas.

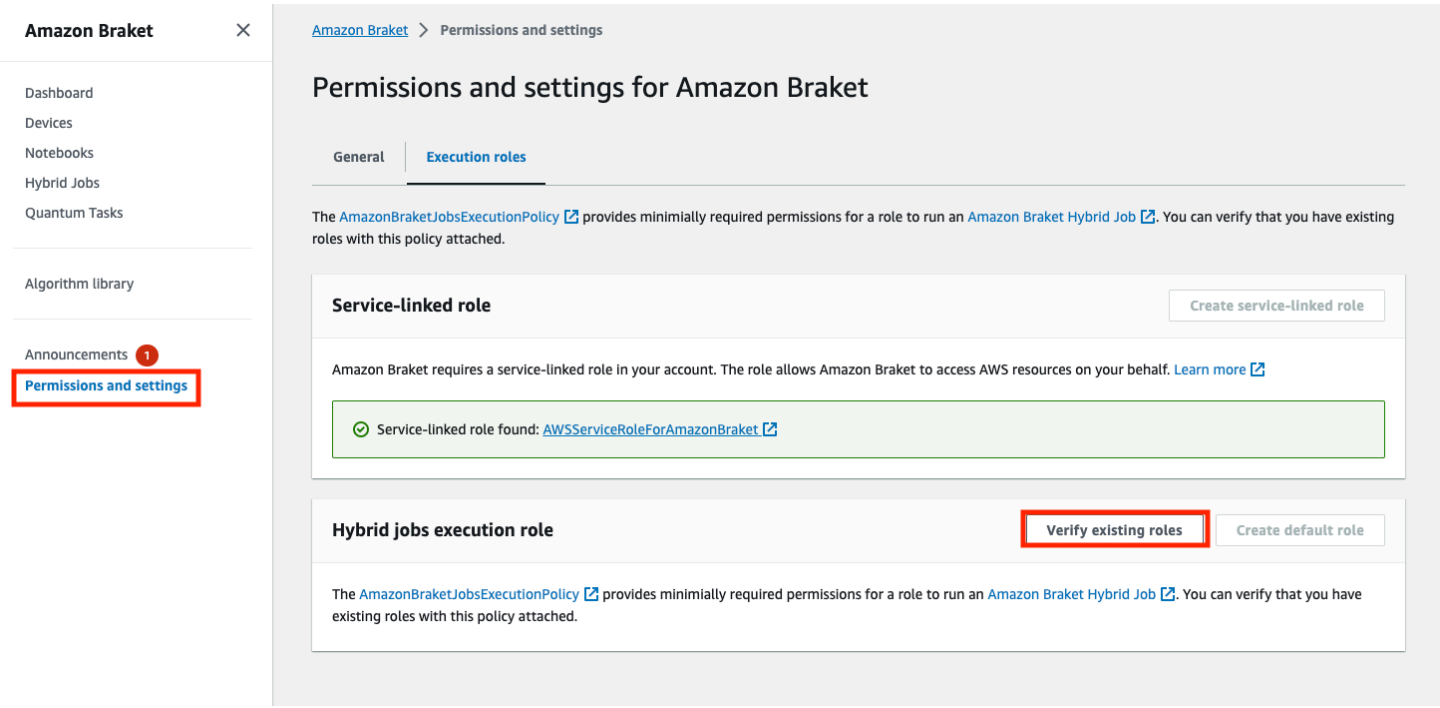
funções auxiliares

O Amazon Braket fornece várias funções auxiliares para simplificar as interações com entradas e saídas de contêineres. Essas funções auxiliares seriam chamadas de dentro do script de algoritmo usado para executar seu Hybrid Job. O exemplo a seguir demonstra como utilizá-las.

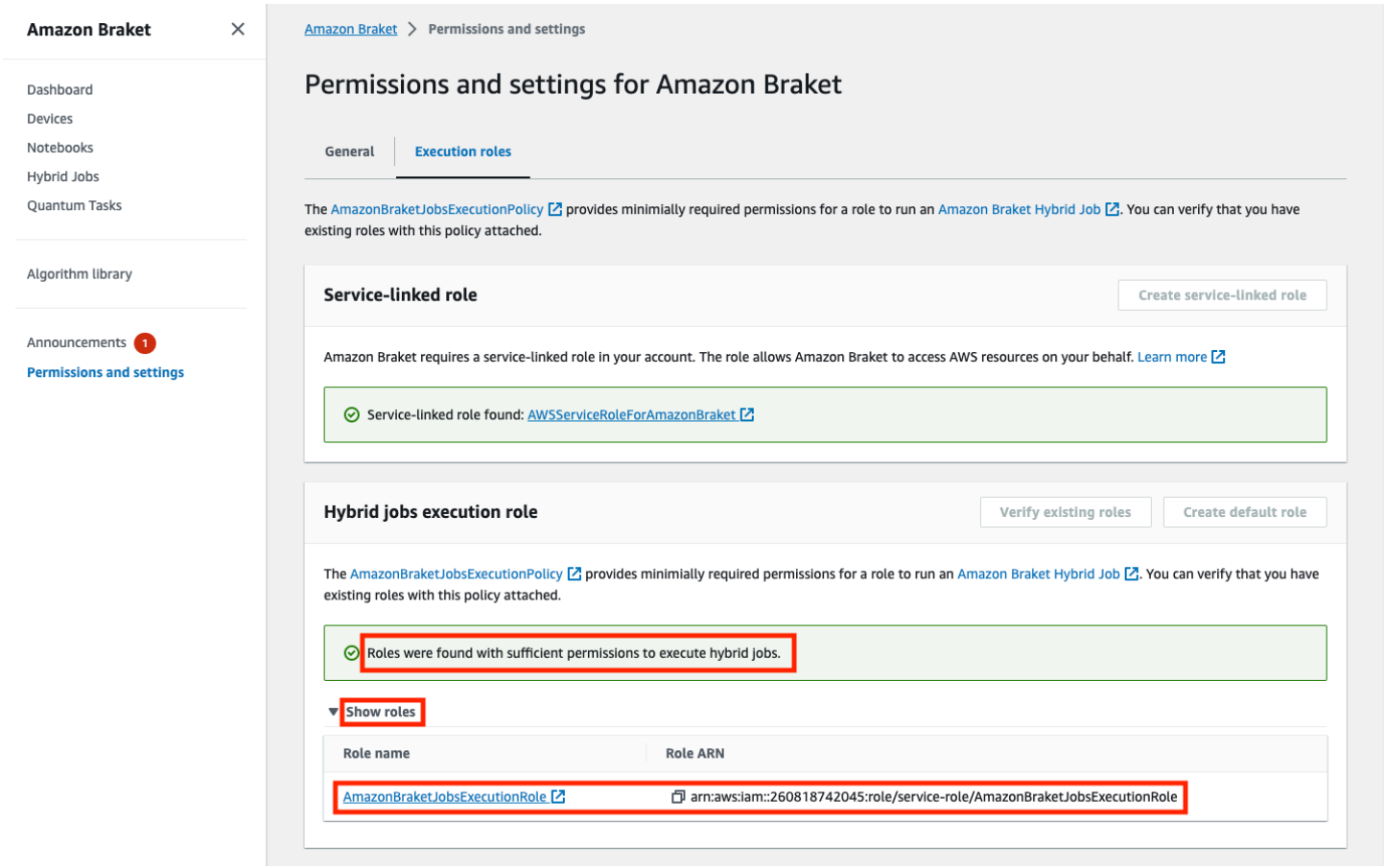
```
from braket.jobs import get_checkpoint_dir, get_hyperparameters, get_input_data_dir,
 get_job_device_arn, get_job_name, get_results_dir, save_job_result,
 save_job_checkpoint, load_job_checkpoint

get_checkpoint_dir() # Get the checkpoint directory
get_hyperparameters() # Get the hyperparameters as strings
get_input_data_dir() # Get the input data directory
get_job_device_arn() # Get the device specified by the hybrid job
get_job_name() # Get the name of the hybrid job.
get_results_dir() # Get the path to a results directory
save_job_result(result_data='data') # Save hybrid job results
save_job_checkpoint(checkpoint_data={'key': 'value'}) # Save a checkpoint
load_job_checkpoint() # Load a previously saved checkpoint
```

Pré-requisitos

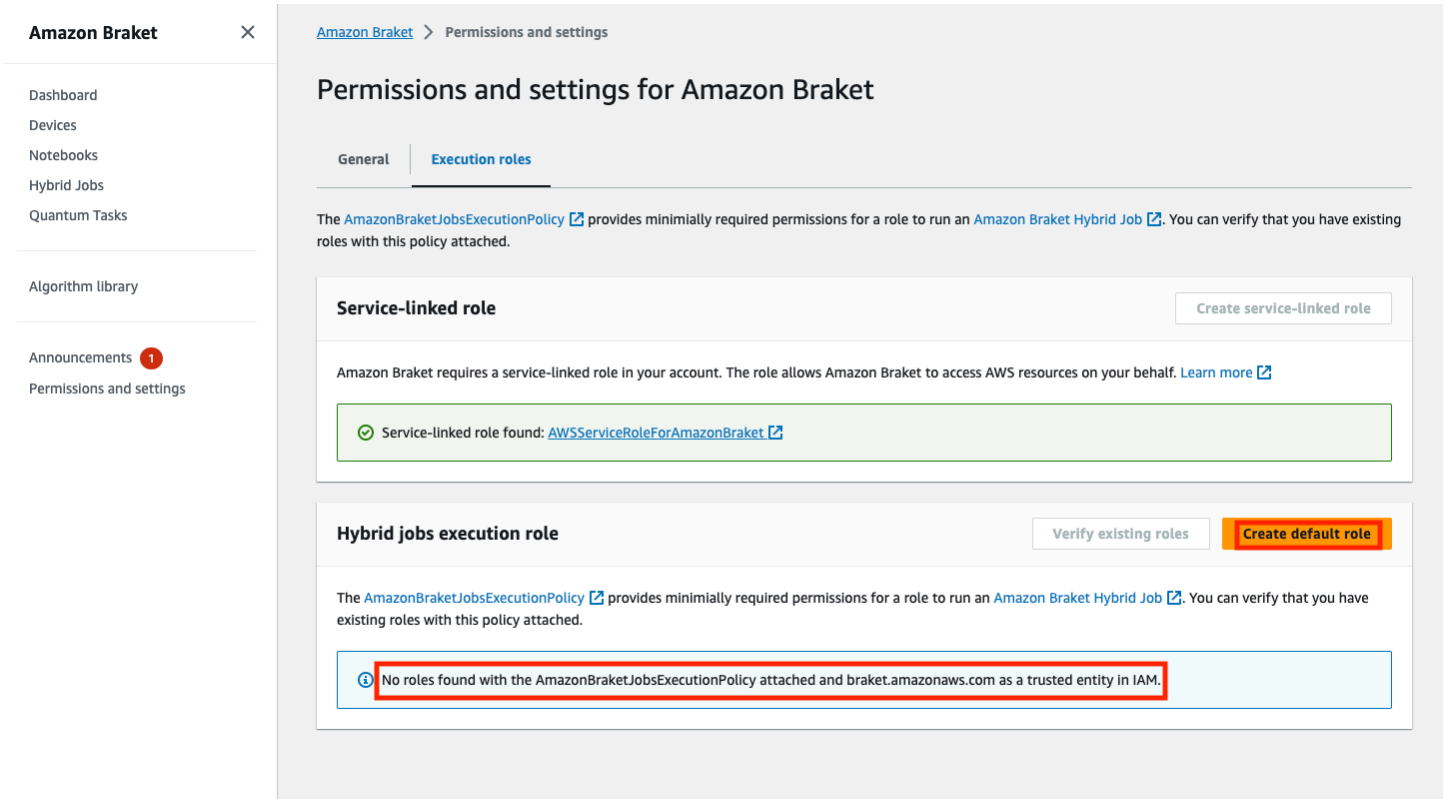
Antes de executar a primeira tarefa híbrida, você deverá garantir que tenha permissões suficientes para realizar essa tarefa. Para determinar se você tem as permissões corretas, selecione Permissões no menu à esquerda do console Braket. A página Gerenciamento de permissões para Amazon Braket ajuda você a verificar se uma de suas funções existentes tem permissões suficientes para executar seu trabalho híbrido ou orienta você na criação de uma função padrão que pode ser usada para executar seu trabalho híbrido, caso você ainda não tenha essa função.



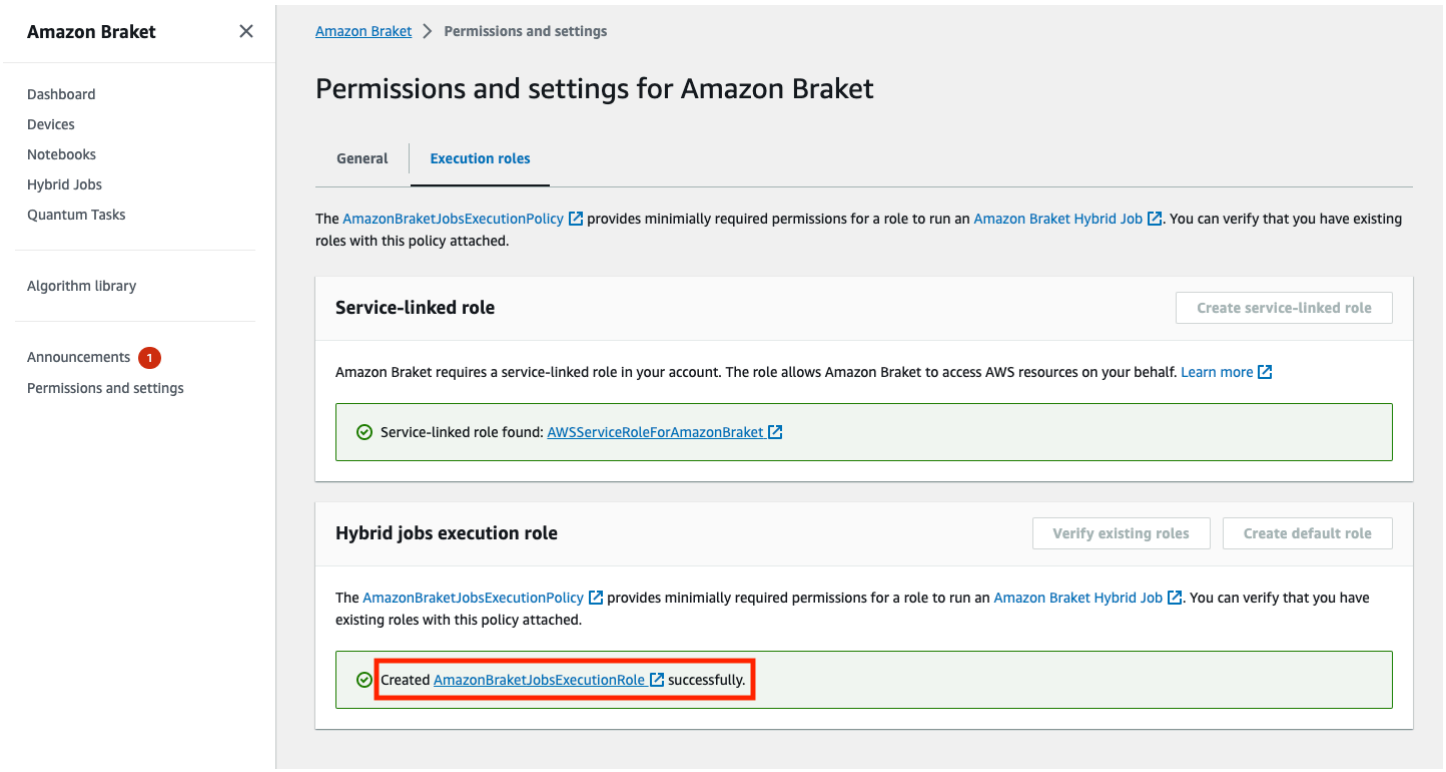
Para verificar se você tem funções com permissões suficientes para executar um trabalho híbrido, selecione o botão Verificar função existente. Quando usa, você obtém uma mensagem de que as funções foram encontradas. Para ver os nomes das funções e suas funções ARNs, selecione o botão Mostrar funções.



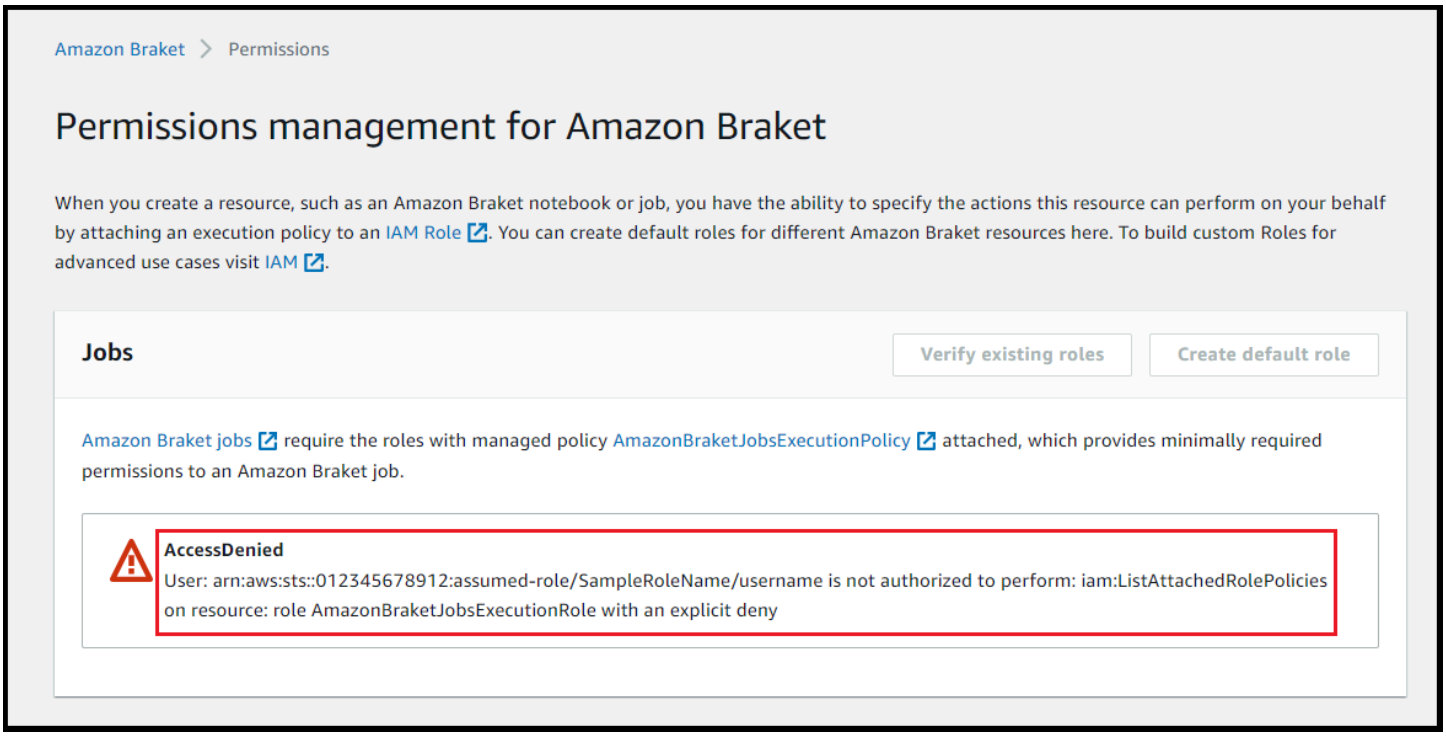
Se você não tiver uma função com permissões suficientes para executar uma tarefa híbrida, você receberá uma mensagem informando que essa função não foi encontrada. Selecione o botão Criar função padrão para obter uma função com permissões suficientes.



Se a função foi criada com sucesso, você receberá uma mensagem confirmando isso.



Se você não tiver permissão para fazer essa consulta, seu acesso será negado. Nesse caso, entre em contato com seu AWS administrador interno.



Criar um trabalho híbrido

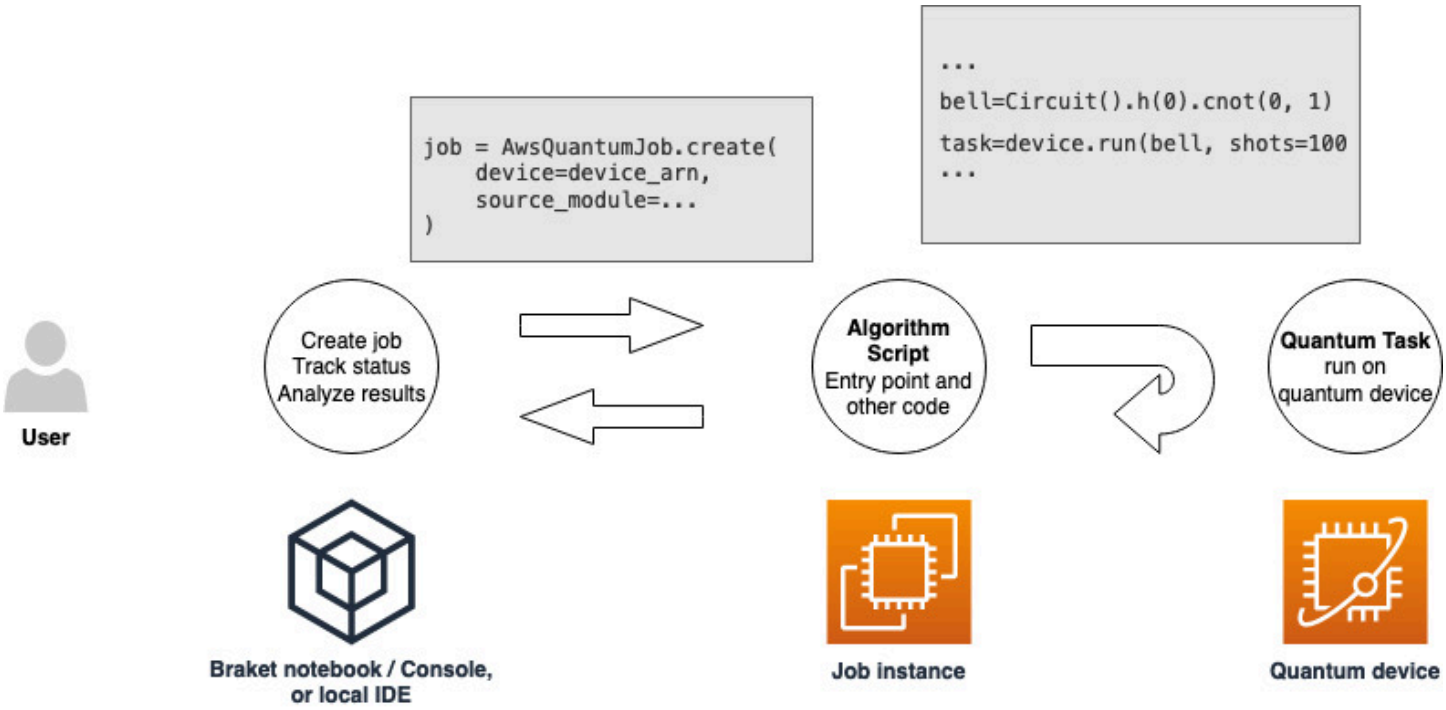
Esta seção mostra como criar um Hybrid Job usando um script do Python. Como alternativa, para criar um trabalho híbrido a partir do código Python local, como seu ambiente de desenvolvimento integrado (IDE) preferido ou um caderno do Braket, consulte Execute seu código local como um trabalho híbrido.

Nesta seção:

- Criar e executar
- Monitorar resultados
- Salve seus resultados
- Usando pontos de verificação
- Execute seu código local como um trabalho híbrido
- Usando a API com trabalhos híbridos
- Crie e depure uma tarefa híbrida com o modo local

Criar e executar

Depois de ter um perfil com permissões para executar um trabalho híbrido, você estará pronto para continuar. A peça-chave do seu primeiro trabalho híbrido do Braket é o script do algoritmo. Ele define o algoritmo que você deseja executar e contém a lógica clássica e as tarefas quânticas que fazem parte do seu algoritmo. Além do script do algoritmo, você pode fornecer outros arquivos de dependência. O script do algoritmo, junto com suas dependências, é chamado de módulo de origem. O ponto de entrada define o primeiro arquivo ou função a ser executado em seu módulo de origem quando o trabalho híbrido é iniciado.



Primeiro, considere o seguinte exemplo básico de um script de algoritmo que cria cinco estados de sino e imprime os resultados de medição correspondentes.

```
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!")
```

Salve esse arquivo com o nome algorithm_script.py em seu diretório de trabalho atual no notebook Braket ou no ambiente local. O arquivo algorithm_script.py tem `start_here()` como ponto de entrada planejado.

Em seguida, crie um arquivo Python ou um caderno Python no mesmo diretório do arquivo algorithm_script.py. Esse script inicia o trabalho híbrido e processa qualquer processamento assíncrono, como imprimir o status ou os principais resultados nos quais estamos interessados. No mínimo, esse script precisa especificar seu script de trabalho híbrido e seu dispositivo principal.

> **Note**
>
> Para obter mais informações sobre como criar um caderno Braket ou fazer upload de um arquivo, como o arquivo algorithm_script.py, no mesmo diretório dos cadernos, consulte Executar seu primeiro circuito usando o Amazon Braket Python SDK.

Para esse primeiro caso básico, você escolhe um simulador. Qualquer que seja o tipo de dispositivo quântico que você almeje, um simulador ou uma unidade de processamento quântico (QPU) real, o dispositivo especificado com `device` no script a seguir é usado para agendar a tarefa híbrida e está disponível para os scripts do algoritmo como a variável de ambiente `AMZN_BRAKET_DEVICE_ARN`.

> **Note**
>
> Você só pode usar dispositivos que estejam disponíveis no Região da AWS seu trabalho híbrido. O Amazon Braket SDK seleciona automaticamente a Região da AWS. Por exemplo, uma tarefa híbrida em us-east-1 pode usar dispositivos IonQ, SV1, DM1, e TN1, mas não Rigetti.

Se você escolher um computador quântico em vez de um simulador, o Braket agenda seus trabalhos híbridos para executar todas as tarefas quânticas com acesso prioritário.

```
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
)
```

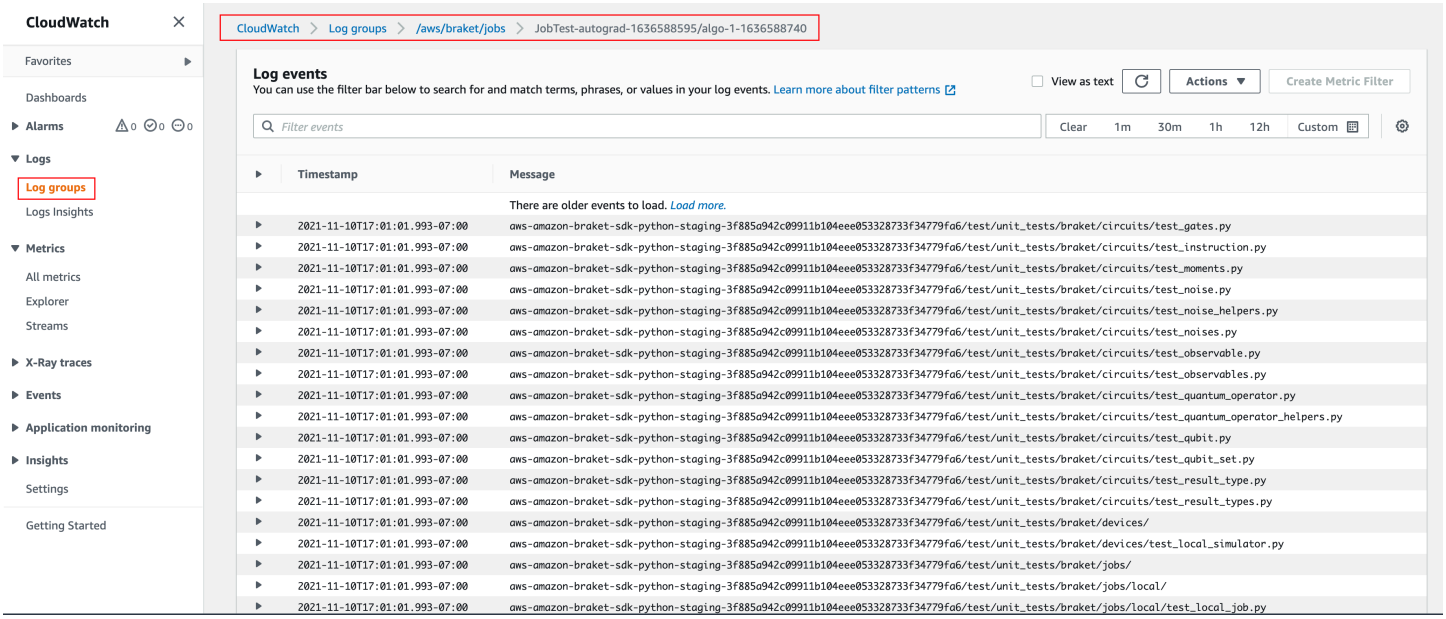
O parâmetro `wait_until_complete=True` define um modo detalhado para que seu trabalho imprima a saída do trabalho real enquanto ele está sendo executado. Você deverá ver uma saída semelhante ao exemplo a seguir.

```
Initializing Braket Job: arn:aws:braket:us-west-2:111122223333:job/braket-job-
default-123456789012
Job queue position: 1
Job queue position: 1
Job queue position: 1
..............
.
.
.
Beginning Setup
Checking for Additional Requirements
Additional Requirements Check Finished
Running Code As Process
Test job started!
Counter({'00': 58, '11': 42})
Counter({'00': 55, '11': 45})
Counter({'11': 51, '00': 49})
Counter({'00': 56, '11': 44})
Counter({'11': 56, '00': 44})
Test job completed!
Code Run Finished
2025-09-24 23:13:40,962 sagemaker-training-toolkit INFO     Reporting training SUCCESS
```

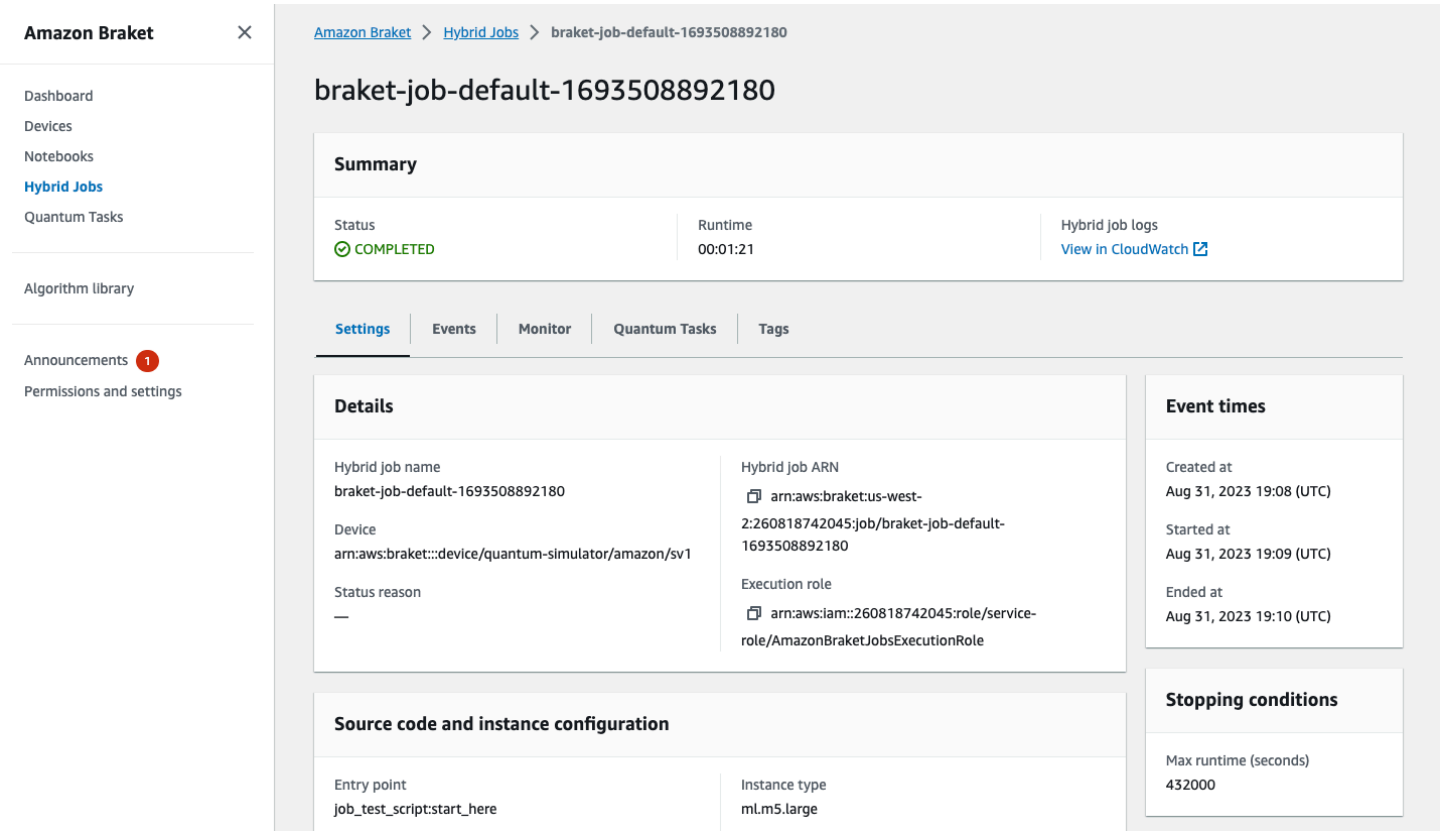
> **Note**
>
> Você também pode usar seu módulo personalizado com o AwsQuantumJobmétodo.create passando sua localização (o caminho para um diretório ou arquivo local ou um URI do S3 de um arquivo tar.gz). Para um exemplo prático, consulte o arquivo Parallelize_training_for_QML.ipynb na pasta hybrid jobs do repositório de exemplos do Amazon Braket no Github.

Monitorar resultados

Como alternativa, você pode acessar a saída do log da Amazon CloudWatch. Para fazer isso, acesse a guia Grupos de logs no menu esquerdo da página de detalhes do trabalho, selecione o grupo de logs `aws/braket/jobs` e escolha o fluxo de logs que contém o nome do trabalho. No exemplo acima, ele é `braket-job-default-1631915042705/algo-1-1631915190`.



Você também pode visualizar o status do trabalho híbrido no console selecionando a página Trabalhos híbridos e, em seguida, escolhendo Configurações.



Seu trabalho híbrido produz alguns artefatos no Amazon S3 enquanto é executado. O nome padrão do bucket do S3 é `amazon-braket-<region>-<accountid>` e o conteúdo está no diretório `jobs/<jobname>/<timestamp>`. Você pode configurar os locais do S3 em que esses artefatos são armazenados especificando um `code_location` diferente quando a tarefa híbrida é criada com o SDK Braket Python.

> **Note**
>
> Esse bucket do S3 deve estar localizado da Região da AWS mesma forma que seu script de trabalho.

O diretório `jobs/<jobname>/<timestamp>` contém uma subpasta com a saída do script do ponto de entrada em um arquivo `model.tar.gz`. Também há um diretório chamado `script` que contém os artefatos do script do algoritmo em um `source.tar.gz` arquivo. Os resultados de suas tarefas quânticas reais estão no diretório nomeado `jobs/<jobname>/tasks`.

Salve seus resultados

Você pode salvar os resultados gerados pelo script do algoritmo para que estejam disponíveis no objeto de trabalho híbrido no script de trabalho híbrido, bem como na pasta de saída no Amazon S3 (em um arquivo compactado em tar chamado model.tar.gz).

A saída deve ser salva em um arquivo usando o formato JavaScript Object Notation (JSON). Se os dados não puderem ser prontamente serializados em texto, como no caso de uma matriz numérica, você poderá passar uma opção para serializar usando um formato de dados em conserva. Consulte o módulo braket.jobs.data_persistence para obter mais detalhes.

Para salvar os resultados das tarefas híbridas, adicione as seguintes linhas comentadas com #ADD ao arquivo algorithm_script.py.

```
import os

from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.jobs import save_job_result  # ADD


def start_here():

    print("Test job started!")

    device = AwsDevice(os.environ['AMZN_BRAKET_DEVICE_ARN'])

    results = []  # ADD

    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)
        results.append(task.result().measurement_counts)  # ADD

        save_job_result({"measurement_counts": results})  # ADD

    print("Test job completed!")
```

Em seguida, você pode exibir os resultados do trabalho a partir do seu script de trabalho anexando a linha **`print(job.result())`** comentada com #ADD.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
)

print(job.arn)
while job.state() not in AwsQuantumJob.TERMINAL_STATES:
    print(job.state())
    time.sleep(10)

print(job.state())
print(job.result())   # ADD
```

Neste exemplo, removemos `wait_until_complete=True` para suprimir a saída detalhada. Você pode adicioná-lo novamente para depuração. Quando você executa essa tarefa híbrida, ela gera o identificador e o `job-arn`, seguidos pelo estado da tarefa híbrida a cada 10 segundos até que a tarefa híbrida chegue a `COMPLETED`, após o qual mostra os resultados do circuito da campainha. Veja o exemplo a seguir.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-123456789012
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Usando pontos de verificação

Você pode salvar iterações intermediárias de seus trabalhos híbridos usando pontos de verificação. No exemplo de script de algoritmo da seção anterior, você adicionaria as seguintes linhas comentadas com #ADD para criar arquivos de ponto de verificação.

```
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!")
```

Ao executar o trabalho híbrido, o arquivo <jobname>-checkpoint-1.json é criado nos artefatos do trabalho híbrido, no diretório de checkpoints, com um caminho padrão `/opt/jobs/checkpoints`. O script de trabalho híbrido permanece inalterado, a menos que você queira alterar esse caminho padrão.

Se você quiser carregar uma tarefa híbrida a partir de um ponto de verificação gerado por uma tarefa híbrida anterior, o script do algoritmo usa `from braket.jobs import load_job_checkpoint`. A lógica a ser carregada em seu script de algoritmo é a seguinte.

```
from braket.jobs import load_job_checkpoint

checkpoint_1 = load_job_checkpoint(
```

```
    "previous_job_name",
    checkpoint_file_suffix="checkpoint-1",
)
```

Depois de carregar esse ponto de verificação, você pode continuar sua lógica com base no conteúdo carregado no `checkpoint-1`.

> **Note**
>
> O checkpoint_file_suffix deve corresponder ao sufixo especificado anteriormente ao criar o ponto de verificação.

Seu script de orquestração precisa especificar o `job-arn` do trabalho híbrido anterior com a linha comentada com #ADD.

```
from braket.aws import AwsQuantumJob

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

Execute seu código local como um trabalho híbrido

O Amazon Braket Hybrid Jobs fornece uma orquestração totalmente gerenciada de algoritmos clássicos quânticos híbridos, combinando recursos computacionais do Amazon EC2 com o acesso à Amazon Braket Quantum Processing Unit (QPU). As tarefas quânticas criadas em um trabalho híbrido têm prioridade na fila em relação às tarefas quânticas individuais, para que seus algoritmos não sejam interrompidos por flutuações na fila de tarefas quânticas. Cada QPU mantém uma fila de trabalhos híbridos separada, garantindo que somente um trabalho híbrido possa ser executado a qualquer momento.

Nesta seção:

- Crie um trabalho híbrido a partir do código Python local
- Instale pacotes e código-fonte adicionais do Python
- Salvar e carregar dados em uma instância de trabalho híbrida

- Práticas recomendadas para decoradores de trabalhos híbridos

Crie um trabalho híbrido a partir do código Python local

Você pode executar seu código Python local como um Amazon Braket Hybrid Job. Você pode fazer isso anotando seu código com um decorador `@hybrid_job`, como mostrado no exemplo de código a seguir. Para ambientes personalizados, você pode optar por usar um contêiner personalizado do Amazon Elastic Container Registry (ECR).

> **Note**
>
> Somente o Python 3.12 é suportado por padrão.

Você pode usar o decorador `@hybrid_job` para anotar uma função. O Braket transforma o código dentro do decorador em um script de algoritmo de trabalho híbrido do Braket. O trabalho híbrido então invoca a função dentro do decorador em uma instância do Amazon EC2. É possível monitorar o progresso do trabalho com `job.state()` ou com o console Braket. O exemplo de código a seguir mostra como executar uma sequência de cinco estados no 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}
```

Você cria o trabalho híbrido invocando a função como faria com as funções normais do Python. No entanto, a função decoradora retorna o identificador de trabalho híbrido em vez do resultado da função. Para recuperar os resultados após a conclusão, use `job.result()`.

```
job = run_hybrid_job(num_tasks=1)
result = job.result()
```

O argumento do dispositivo no `@hybrid_job` decorador especifica o dispositivo ao qual a tarefa híbrida tem acesso prioritário - nesse caso, o simulador SV1. Para obter prioridade de QPU, você deve garantir que o ARN do dispositivo usado na função corresponda ao especificado no decorador. Por conveniência, você pode usar a função auxiliar `get_job_device_arn()` para capturar o ARN do dispositivo declarado em `@hybrid_job`.

> **Note**
>
> Cada trabalho híbrido tem um tempo de inicialização de pelo menos um minuto, pois cria um ambiente em contêineres no Amazon EC2. Portanto, para workloads muito curtas, como um único circuito ou um lote de circuitos, pode ser suficiente usar tarefas quânticas.

Hiperparâmetros

A função `run_hybrid_job()` usa o argumento `num_tasks` para controlar o número de tarefas quânticas criadas. A tarefa híbrida captura isso automaticamente como um hiperparâmetro.

> **Note**
>
> Os hiperparâmetros são exibidos no console do Braket como sequências de caracteres, limitadas a 2500 caracteres.

Métricas e log

Dentro da função `run_hybrid_job()`, métricas de algoritmos iterativos são registradas com `log_metrics`. As métricas são plotadas automaticamente na página do console do Braket, na guia de tarefas híbridas. Você pode usar métricas para rastrear os custos quânticos da tarefa quase em tempo real durante a execução do trabalho híbrido com o rastreador de custos Braket. O exemplo acima usa o nome da métrica “probabilidade” que registra a primeira probabilidade do tipo de resultado.

Recuperando resultados

Depois que a tarefa híbrida for concluída, você usará `job.result()` para recuperar os resultados da tarefa híbrida. Todos os objetos na declaração de retorno são automaticamente capturados pelo Braket. Observe que os objetos retornados pela função devem ser uma tupla com cada elemento sendo serializável. Por exemplo, o código a seguir mostra um exemplo de funcionamento e um de falha.

```
import numpy as np


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

# # Failing example
# @hybrid_job(device=Devices.Amazon.SV1)
# def failing():
#     return MyObject() # Not serializable
```

Nome do trabalho

Por padrão, o nome desse trabalho híbrido é inferido do nome da função. Você também pode especificar um nome personalizado com até 50 caracteres. Por exemplo, no código a seguir, o nome do trabalho é "my-job-name”.

```
@hybrid_job(device=Devices.Amazon.SV1, job_name="my-job-name")
def function():
    pass
```

Modo local

Os trabalhos locais são criados adicionando o argumento `local=True` ao decorador. Isso executa a tarefa híbrida em um ambiente conteinerizado em seu ambiente de computação local, como seu laptop. Os trabalhos locais não têm fila prioritária para tarefas quânticas. Para casos avançados, como vários nós ou MPI, as tarefas locais podem ter acesso às variáveis de ambiente Braket necessárias. O código a seguir cria uma tarefa híbrida local com o dispositivo como SV1 simulador.

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

Todas as outras opções de trabalhos híbrida são compatíveis. Para obter uma lista de opções, consulte o módulo braket.jobs.quantum_job_creation.

Instale pacotes e código-fonte adicionais do Python

Você pode personalizar seu ambiente de runtime para usar seus pacotes Python preferidos. Você pode usar um arquivo `requirements.txt`, uma lista de nomes de pacotes ou trazer seu próprio contêiner (BYOC). Por exemplo, o arquivo `requirements.txt` pode incluir outros pacotes para instalação.

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

Para personalizar um ambiente de runtime usando um arquivo `requirements.txt`, consulte o exemplo de código a seguir.

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

Como alternativa, você pode fornecer os nomes dos pacotes como uma lista do Python da seguinte maneira.

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies=["qiskit", "pennylane>=0.31",
 "mitiq==0.29"])
def run_hybrid_job(num_tasks=1):
```

```
    return ...
```

O código-fonte adicional pode ser especificado como uma lista de módulos ou como um único módulo, como no exemplo de código a seguir.

```
@hybrid_job(device=Devices.Amazon.SV1, include_modules=["my_module1", "my_module2"])
def run_hybrid_job(num_tasks=1):
    return ...
```

Salvar e carregar dados em uma instância de trabalho híbrida

Especificando dados de treinamento de entrada

Ao criar um trabalho híbrido, será possível fornecer uma entrada de conjuntos de dados de treinamento especificando um bucket do Amazon Simple Storage Service (Amazon S3). Você também pode especificar um caminho local e, em seguida, o Braket carrega automaticamente os dados para o Amazon S3 em `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>`. Se você especificar um caminho local, o nome do canal será padronizado como “entrada”. O código a seguir mostra um arquivo numpy do caminho `data/file.npy` local.

```
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1, input_data="data/file.npy")
def run_hybrid_job(num_tasks=1):
    data = np.load("data/file.npy")
    return ...
```

Para o S3, você deve usar a função auxiliar `get_input_data_dir()`.

```
import numpy as np
from braket.jobs import get_input_data_dir

s3_path = "s3://amazon-braket-us-east-1-123456789012/job-data/file.npy"


@hybrid_job(device=None, input_data=s3_path)
def job_s3_input():
    np.load(get_input_data_dir() + "/file.npy")
```

```
@hybrid_job(device=None, input_data={"channel": s3_path})
def job_s3_input_channel():
    np.load(get_input_data_dir("channel") + "/file.npy")
```

Você pode especificar várias fontes de dados de entrada fornecendo um dicionário de valores de canais e caminhos S3 URIs ou locais.

```
import numpy as np
from braket.jobs import get_input_data_dir

input_data = {
    "input": "data/file.npy",
    "input_2": "s3://amzn-s3-demo-bucket/data.json"
}


@hybrid_job(device=None, input_data=input_data)
def multiple_input_job():
    np.load(get_input_data_dir("input") + "/file.npy")
    np.load(get_input_data_dir("input_2") + "/data.json")
```

> **Note**
>
> Quando os dados de entrada são grandes (>1 GB), há um longo tempo de espera até que a tarefa seja criada. Isso se deve aos dados de entrada locais quando eles são carregados pela primeira vez em um bucket do S3 e, em seguida, o caminho do S3 é adicionado à solicitação de trabalho. Finalmente, a solicitação de trabalho é enviada ao serviço Braket.

Salvando resultados no S3

Para salvar resultados não incluídos na instrução de retorno da função decorada, você deve acrescentar o diretório correto a todas as operações de gravação de arquivos. O exemplo a seguir mostra como salvar uma matriz numérica e uma figura matplotlib.

```
import matplotlib.pyplot as plt
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1)
```

```
def run_hybrid_job(num_tasks=1):
    result = np.random.rand(5)

    # Save a numpy array
    np.save("result.npy", result)

    # Save a matplotlib figure
    plt.plot(result)
    plt.savefig("fig.png")
    return ...
```

Todos os resultados são compactados em um arquivo chamado `model.tar.gz`. Você pode baixar os resultados com a função `job.result()` do Python ou navegando até a pasta de resultados na página de trabalhos híbridos no console de gerenciamento do Braket.

Salvando e retomando a partir dos pontos de verificação

Para trabalhos híbridos de longa duração, é recomendável salvar periodicamente o estado intermediário do algoritmo. Você pode usar a função auxiliar `save_job_checkpoint()` integrada ou salvar arquivos no caminho `AMZN_BRAKET_JOB_RESULTS_DIR`. O último está disponível com a função auxiliar `get_job_results_dir()`.

Veja a seguir um exemplo prático mínimo para salvar e carregar pontos de verificação com um decorador de tarefas 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()
```

No primeiro trabalho híbrido, `save_job_checkpoint()` é chamado com um dicionário contendo os dados que queremos salvar. Por padrão, cada valor deve ser serializável como texto. Para verificar objetos Python mais complexos, como matrizes numpy, você pode definir `data_format = PersistedJobDataFormat.PICKLED_V4`. Esse código cria e sobrescreve um arquivo de ponto de verificação com nome padrão `<jobname>.json` em seus artefatos de trabalho híbridos em uma subpasta chamada “pontos de verificação”.

Para criar um novo trabalho híbrido para continuar a partir do posto de controle, precisamos informar `copy_checkpoints_from_job=job_arn` onde `job_arn` é o ARN do trabalho híbrido do trabalho anterior. Em seguida, usamos `load_job_checkpoint(job_name)` para carregar a partir do ponto de verificação.

Práticas recomendadas para decoradores de trabalhos híbridos

Adote a assincronicidade

As tarefas híbridas criadas com a anotação do decorador são assíncronas — elas são executadas quando os recursos clássicos e quânticos estão disponíveis. Você monitora o progresso do algoritmo usando o Braket Management Console ou Amazon CloudWatch. Quando você envia seu algoritmo para execução, o Braket executa seu algoritmo em um ambiente escalável em contêineres e os resultados são recuperados quando o algoritmo é concluído.

Execute algoritmos variacionais iterativos

Os trabalhos híbridos oferecem as ferramentas para executar algoritmos clássicos quânticos iterativos. Para problemas puramente quânticos, use tarefas quânticas ou um lote de tarefas quânticas. O acesso prioritário a determinados QPUs é mais benéfico para algoritmos variacionais de longa execução que exigem várias chamadas iterativas para o QPUs com processamento clássico intermediário.

Depurar usando o modo local

Antes de executar uma tarefa híbrida em uma QPU, é recomendável executá-la primeiro no simulador SV1 para confirmar se ela é executada conforme o esperado. Para testes de pequena escala, você pode executar com o modo local para iteração e depuração rápidas.

Melhore a reprodutibilidade com Bring your own container (BYOC)

Crie um experimento reproduzível encapsulando seu software e suas dependências em um ambiente em contêineres. Ao empacotar todo o seu código, dependências e configurações em um contêiner, você evita possíveis conflitos e problemas de versão.

Simuladores distribuídos de várias instâncias

Para executar um grande número de circuitos, considere usar o suporte MPI integrado para executar simuladores locais em várias instâncias em uma única tarefa híbrida. Para obter mais informações, consulte simuladores incorporados.

Use circuitos paramétricos

Os circuitos paramétricos que você envia a partir de um trabalho híbrido são compilados automaticamente em determinados, QPUs usando a compilação paramétrica para melhorar os tempos de execução de seus algoritmos.

Ponto de verificação periódico

Para trabalhos híbridos de longa duração, é recomendável salvar periodicamente o estado intermediário do algoritmo.

Para mais exemplos, casos de uso e melhores práticas, consulte exemplos do Amazon Braket. GitHub

Usando a API com trabalhos híbridos

Você pode acessar e interagir com o Amazon Braket Hybrid Jobs diretamente usando o API. No entanto, os padrões e os métodos de conveniência não estão disponíveis ao usar o API diretamente.

> **Note**
>
> É altamente recomendável que você interaja com o Amazon Braket Hybrid Jobs usando o SDK Amazon Braket Python. Ele oferece padrões e proteções convenientes que ajudam seus trabalhos híbridos a serem executados com êxito.

Este tópico aborda os conceitos básicos do uso do API. Se você optar por usar a API, lembre-se de que essa abordagem pode ser mais complexa e estar preparada para várias iterações para que seu trabalho híbrido seja executado.

Para usar a API, sua conta deve ter um papel na política gerenciada `AmazonBraketFullAccess`.

> **Note**
>
> Para obter mais informações sobre como obter uma função com a política gerenciada `AmazonBraketFullAccess`, consulte a página Habilitar o Amazon Braket.

Além disso, você precisa de um perfil de execução. Esse perfil será passado para o serviço. Você pode criar o perfil usando o console do Amazon Braket. Use a guia Perfis de execução na página Permissões e configurações para criar um perfil padrão para trabalhos híbridos.

O `CreateJob` API exige que você especifique todos os parâmetros necessários para a tarefa híbrida. Para usar o Python, compacte seus arquivos de script de algoritmo em um pacote tar, como um arquivo input.tar.gz, e execute o script a seguir. Atualize as partes do código entre colchetes angulares (<>) para corresponder às informações da sua conta e ao ponto de entrada que especificam o caminho, o arquivo e o método em que seu trabalho híbrido começa.

```
from braket.aws import AwsDevice, AwsSession
import boto3
from datetime import datetime

s3_client = boto3.client("s3")
client = boto3.client("braket")

project_name = "job-test"
job_name = project_name + "-" + datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")
bucket = "amazon-braket-<your_bucket>"
s3_prefix = job_name

job_script = "input.tar.gz"
job_object = f"{s3_prefix}/script/{job_script}"
s3_client.upload_file(job_script, bucket, job_object)

input_data = "inputdata.csv"
input_object = f"{s3_prefix}/input/{input_data}"
s3_client.upload_file(input_data, bucket, input_object)

job = client.create_job(
    jobName=job_name,
    roleArn="arn:aws:iam::<your_account>:role/service-role/
AmazonBraketJobsExecutionRole",  # https://docs.aws.amazon.com/braket/latest/
developerguide/braket-manage-access.html#about-amazonbraketjobsexecution
```

```
    algorithmSpecification={
        "scriptModeConfig": {
            "entryPoint": "<your_execution_module>:<your_execution_method>",
            "containerImage": {"uri": "292282985366.dkr.ecr.us-west-1.amazonaws.com/
amazon-braket-base-jobs:1.0-cpu-py37-ubuntu18.04"},   # Change to the specific region
 you are using
            "s3Uri": f"s3://{bucket}/{job_object}",
            "compressionType": "GZIP"
        }
    },
    inputDataConfig=[
        {
            "channelName": "hellothere",
            "compressionType": "NONE",
            "dataSource": {
                "s3DataSource": {
                    "s3Uri": f"s3://{bucket}/{s3_prefix}/input",
                    "s3DataType": "S3_PREFIX"
                }
            }
        }
    ],
    outputDataConfig={
        "s3Path": f"s3://{bucket}/{s3_prefix}/output"
    },
    instanceConfig={
        "instanceType": "ml.m5.large",
        "instanceCount": 1,
        "volumeSizeInGb": 1
    },
    checkpointConfig={
        "s3Uri":  f"s3://{bucket}/{s3_prefix}/checkpoints",
        "localPath": "/opt/omega/checkpoints"
    },
    deviceConfig={
        "priorityAccess": {
            "devices": [
                "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3"
            ]
        }
    },
    hyperParameters={
        "hyperparameter key you wish to pass": "<hyperparameter value you wish to
 pass>",
```

```
    },
    stoppingCondition={
        "maxRuntimeInSeconds": 1200,
        "maximumTaskLimit": 10
    },
)
```

Depois de criar seu trabalho híbrido, você pode acessar os detalhes do trabalho híbrido por meio do `GetJob` API ou do console. Para obter os detalhes do trabalho híbrido da sessão do Python na qual você executou o código `createJob`, como no exemplo anterior, use o comando Python a seguir.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

Para cancelar um trabalho híbrido, chame o `CancelJob` API com o Amazon Resource Name do trabalho ('JobArn').

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

Você pode especificar pontos de verificação como parte do `createJob` API usando o `checkpointConfig` parâmetro.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> O localPath de `checkpointConfig` não pode começar com nenhum dos seguintes caminhos reservados: `/opt/ml`, `/opt/braket`, `/tmp`, ou `/usr/local/nvidia`.

Crie e depure uma tarefa híbrida com o modo local

Quando você está criando um novo algoritmo híbrido, o modo local ajuda você a depurar e testar seu script de algoritmo. O modo local é um recurso que permite executar o código que você planeja usar no Amazon Braket Hybrid Jobs, mas sem precisar que o Braket gerencie a infraestrutura para executar o trabalho híbrido. Em vez disso, execute trabalhos híbridos localmente em sua instância do Amazon Braket Notebook ou em um cliente preferencial, como um laptop ou computador desktop.

No modo local, você ainda pode enviar tarefas quânticas para dispositivos reais, mas não obtém os benefícios de desempenho ao executar em uma unidade de processamento quântico (QPU) real no modo local.

Para usar o modo local, modifique `AwsQuantumJob` para `LocalQuantumJob` onde quer que ele ocorra dentro do seu programa. Por exemplo, para executar o exemplo de Criar seu primeiro trabalho híbrido, edite o script de trabalho híbrido no código da seguinte forma.

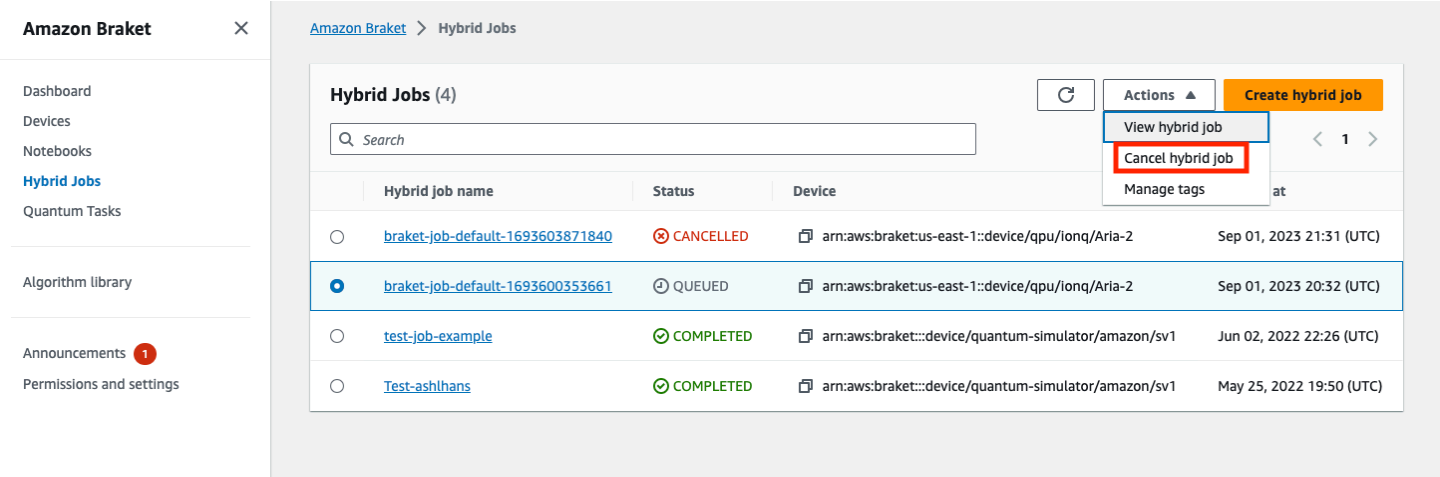
```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

> **Note**
>
> O Docker, que já vem pré-instalado nos cadernos Amazon Braket, precisa ser instalado em seu ambiente local para usar esse recurso. As instruções para instalar o Docker podem ser encontradas na página Obter Docker. Além disso, nem todos os parâmetros são compatíveis com o modo local.

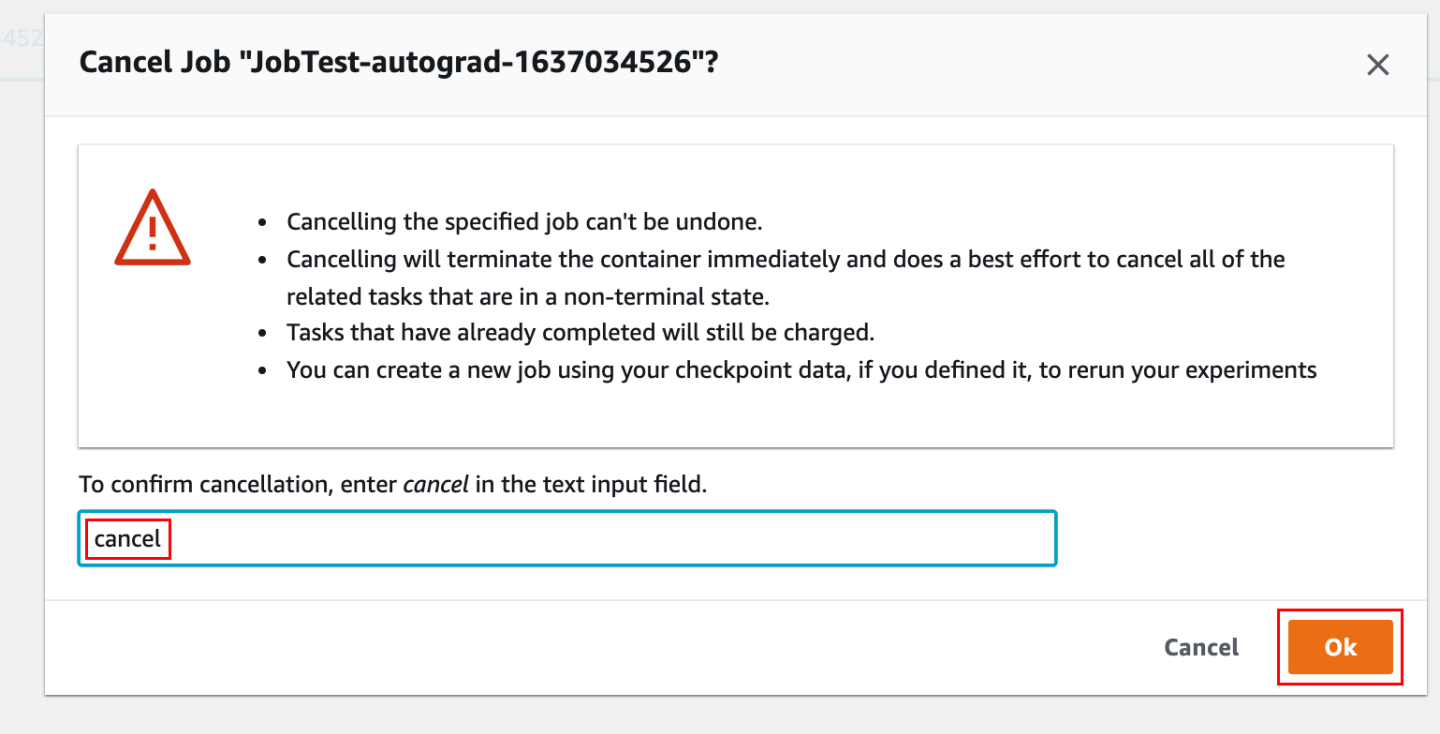
Cancelar um Hybrid Job

Talvez seja necessário cancelar um trabalho híbrido em um estado não terminal. Isso pode ser feito no console ou com código.

Para cancelar seu trabalho híbrido no console, selecione o trabalho híbrido a ser cancelado na página Trabalhos híbridos e, em seguida, selecione Cancelar trabalho híbrido no menu suspenso Ações.



Para confirmar o cancelamento, insira cancelar no campo de entrada quando solicitado e selecione OK.



Para cancelar seu trabalho híbrido usando o código do SDK do Braket Python, use o `job_arn` para identificar o trabalho híbrido e, em seguida, chame o comando `cancel` nele, conforme mostrado no código a seguir.

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

O comando `cancel` encerra imediatamente o contêiner de trabalho híbrido clássico e faz o possível para cancelar todas as tarefas quânticas relacionadas que ainda estão em um estado não terminal.

Personalizar seu Hybrid Job

O Amazon Braket fornece várias maneiras de personalizar a execução de seus trabalhos híbridos, permitindo que você adapte o ambiente às suas necessidades específicas. Esta seção explora as opções para personalizar trabalhos híbridos, desde a definição do ambiente de script de algoritmo até a criação de seu próprio contêiner. Você aprenderá a otimizar seu fluxo de trabalho usando hiperparâmetros, configurar instâncias de trabalho e aproveitar a compilação paramétrica para melhorar o desempenho. Essas técnicas de personalização ajudam você a maximizar o potencial de suas computações quânticas híbridas no Amazon Braket.

Nesta seção:

- Defina o ambiente para seu script de algoritmo
- Usando hiperparâmetros
- Configure sua instância de trabalho híbrida
- Usar compilação paramétrica para acelerar trabalhos híbridos

Defina o ambiente para seu script de algoritmo

O Amazon Braket oferece suporte a ambientes definidos por contêineres para seu script de algoritmo:

- Um contêiner base (o padrão, se nenhum `image_uri` for especificado)
- Um contêiner com CUDA-Q
- Um contêiner com Tensorflow e PennyLane
- Um contêiner com PyTorch PennyLane, e CUDA-Q

A tabela a seguir fornece detalhes sobre os contêineres e as bibliotecas que eles incluem.

Contêineres Amazon Braket

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
URI da imagem	292282985 366.dkr. ecr.us-	292282985 366.dkr. ecr.us-	292282985 366.dkr. ecr.us-	292282985 366.dkr. ecr.us-

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
	west-2.amaz onaws.com /:latest amazon-braket-base-jobs	west-2.amaz onaws.com /:latest amazon-braket-cudaq-jobs	east-1.amaz onaws.com /:latest amazon-braket-tensorflow-jobs	west-2.amaz onaws.com /:latest amazon-braket-pytorch-jobs

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
Bibliotecas herdadas		• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • botocore • boto3 • dask • matplotlib • numpy • pandas • PennyLane • PennyLane-Relâmpago • qiskit-braket-provider • API • sagemaker-training • scikit-learn • scipy	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy

Tipo	Base	CUDA-Q	TensorFlow	PyTorch
Bibliotecas adicionais	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • boto3 • ipykernel • matplotlib • networkx • numpy • openbabel • pandas • PennyLane • protobuf • psi4 • rsa • scipy	• cudaq • cudaq-qec • cudaq-solvers	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras • matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU Lightning • cuQuantum	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras • matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU Lightning • cuQuantum • cudaq • cudaq-qec • cudaq-solvers

Você pode visualizar e acessar as definições de contêiner de código aberto em aws/ amazon-braket-containers. Escolha o contêiner que corresponde ao seu caso de uso. Você pode usar qualquer uma das AWS regiões disponíveis no Braket (us-east-1, us-west-1, us-west-2, eu-north-1, eu-west-2), mas a região do contêiner deve corresponder à região do seu trabalho híbrido. Especifique a imagem do

contêiner ao criar um trabalho híbrido adicionando um dos três argumentos a seguir à sua chamada `create(…)` no script de trabalho híbrido. Você pode instalar dependências adicionais no contêiner escolhido em runtime (ao custo da inicialização ou do runtime) porque os contêineres do Amazon Braket têm conectividade com a internet. O exemplo a seguir refere-se à região us-west-2.

- Imagem base: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest” amazon-braket-base-jobs
- Imagem CUDA-Q: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest” amazon-braket-cudaq-jobs
- Imagem do Tensorflow: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest” amazon-braket-tensorflow-jobs
- PyTorch imagem: image_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /:latest” amazon-braket-pytorch-jobs

A `image-uris` pode ser recuperada usando a função `retrieve_image()` no SDK do Amazon Braket. O exemplo a seguir mostra como recuperá-los do Região da AWS us-west-2.

```
from braket.jobs.image_uris import retrieve_image, Framework

image_uri_base = retrieve_image(Framework.BASE, "us-west-2")
image_uri_cudaq = retrieve_image(Framework.CUDAQ, "us-west-2")
image_uri_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Traga seu próprio contêiner (BYOC)

O Amazon Braket Hybrid Jobs fornece três contêineres pré-criados para execução de código em ambientes diferentes. Se um desses contêineres oferecer suporte ao seu caso de uso, você só precisará fornecer seu script de algoritmo ao criar um trabalho híbrido. Pequenas dependências ausentes podem ser adicionadas a partir do seu script de algoritmo ou de um arquivo `requirements.txt` usando `pip`.

Se nenhum desses contêineres oferecer suporte ao seu caso de uso, ou se você quiser expandi-los, o Braket Hybrid Jobs suporta a execução de trabalhos híbridos com sua própria imagem de Docker contêiner personalizada ou traga seu próprio contêiner (BYOC). Certifique-se que esse é o recurso certo para o seu caso de uso.

Nesta seção:

- Quando trazer meu próprio contêiner é a decisão certa?
- Como trazer seu próprio contêiner
- Executando trabalhos híbridos do Braket em seu próprio contêiner

Quando trazer meu próprio contêiner é a decisão certa?

Trazer seu próprio contêiner (BYOC) para o Braket Hybrid Jobs oferece a flexibilidade de usar seu próprio software instalando-o em um ambiente empacotado. Dependendo de suas necessidades específicas, pode haver maneiras de obter a mesma flexibilidade sem precisar passar pelo ciclo completo de criação do BYOC Docker - upload do Amazon ECR - URI de imagem personalizada.

> **Note**
>
> O BYOC pode não ser a escolha certa se você quiser adicionar um pequeno número de pacotes Python adicionais (geralmente menos de 10) que estão disponíveis publicamente. Por exemplo, se você estiver usando PyPi.

Nesse caso, você pode usar uma das imagens pré-criadas do Braket e, em seguida, incluir um arquivo `requirements.txt` no diretório de origem no envio do trabalho. O arquivo é lido automaticamente e o `pip` instalará os pacotes com as versões especificadas normalmente. Se você estiver instalando um grande número de pacotes, o runtime de seus trabalhos poderá ser substancialmente aumentado. Verifique a versão Python e, se aplicável, CUDA do contêiner pré-construído que você deseja usar para testar se seu software funcionará.

O BYOC é necessário quando você deseja usar uma linguagem não Python (como C++ ou Rust) para seu script de trabalho ou se quiser usar uma versão em Python não disponível nos contêineres pré-construídos do Braket. Essa também é uma boa escolha se:

- Você está usando um software com uma chave de licença e precisa autenticar essa chave em um servidor de licenciamento para executar o software. Com o BYOC, você pode incorporar a chave de licença em sua imagem Docker e incluir um código para autenticá-la.
- Você está usando um software que não está disponível ao público. Por exemplo, o software está hospedado em um repositório privado GitLab ou GitHub repositório que você precisa de uma chave SSH específica para acessar.

- Você precisa instalar um grande pacote de software que não esteja empacotado nos contêineres fornecidos pelo Braket. O BYOC permitirá que você elimine longos tempos de inicialização de seus contêineres de trabalhos híbridos devido à instalação do software.

O BYOC também permite que você disponibilize seu SDK ou algoritmo personalizado aos clientes criando um contêiner Docker com seu software e disponibilizando-o para seus usuários. Você pode fazer isso definindo as permissões apropriadas no Amazon ECR.

> **Note**
>
> Você deve estar em conformidade com todas as licenças de software aplicáveis.

Como trazer seu próprio contêiner

Nesta seção, fornecemos um step-by-step guia do que você precisa fazer bring your own container (BYOC) para Braket Hybrid Jobs — os scripts, arquivos e etapas para combiná-los para começar a trabalhar com suas imagens personalizadasDocker. As receitas para dois casos comuns:

1. Instale software adicional em uma imagem Docker e use somente scripts de algoritmo Python em seus trabalhos.
2. Use scripts de algoritmo escritos em uma linguagem não Python com trabalhos híbridos ou uma arquitetura de CPU além de x86.

Definir o script de entrada do contêiner é mais complexo para o caso 2.

Quando o Braket executa seu Hybrid Job, ele inicia o número e o tipo solicitados de instâncias do Amazon EC2 e, em seguida, Docker executa a imagem especificada pela entrada do URI da imagem para a criação do trabalho nelas. Ao usar o recurso BYOC, você especifica um URI de imagem hospedado em um repositório privado do Amazon ECR ao qual você tem acesso de leitura. O Braket Hybrid Jobs usa essa imagem personalizada para executar o trabalho.

Os componentes específicos necessários para criar uma imagem Docker que possa ser usada com trabalhos híbridos. Se você não estiver familiarizado com escrever e criar`Dockerfiles`, consulte a documentação do Dockerfile e a documentação Amazon ECR CLI.

Requisitos:

- Uma imagem base para seu Dockerfile

- (Opcional) Um script de ponto de entrada de contêiner modificado
- Instale o software e o script de contêiner necessários com Dockerfile

Uma imagem base para seu Dockerfile

Se você estiver usando Python e quiser instalar software além do que é fornecido nos contêineres fornecidos pelo Braket, uma opção para uma imagem base é uma das imagens do contêiner do Braket, hospedada em nosso GitHub repositório e no Amazon ECR. Você precisará se autenticar no Amazon ECR para extrair a imagem e criar em cima dela. Por exemplo, a primeira linha do seu arquivo Docker BYOC pode ser: `FROM [IMAGE_URI_HERE]`

Em seguida, preencha o restante do Dockerfile para instalar e configurar o software que você deseja adicionar ao contêiner. As imagens pré-criadas do Braket já conterão o script de ponto de entrada do contêiner apropriado, então você não precisa se preocupar em incluí-lo.

Se você quiser usar uma linguagem não Python, como C++, Rust ou Julia, ou se quiser criar uma imagem para uma arquitetura de CPU não x86, como ARM, talvez seja necessário criar com base em uma imagem pública básica. Você pode encontrar muitas dessas imagens na Galeria Pública do Amazon Elastic Container Registry. Certifique-se de escolher uma que seja apropriada para a arquitetura da CPU e, se necessário, para a GPU que você deseja usar.

(Opcional) Um script de ponto de entrada de contêiner modificado

> **Note**
>
> Se você só estiver adicionando mais software a uma imagem de Braket pré-criada, ignore esta seção.

Para executar código não Python como parte de seu trabalho híbrido, modifique o script Python que define o ponto de entrada do contêiner. Por exemplo, o script `braket_container.py` python no Amazon Braket Github. Esse é o script que as imagens pré-construídas pelo Braket usam para iniciar seu script de algoritmo e definir as variáveis de ambiente apropriadas. O script de ponto de entrada do contêiner em si deve estar em Python, mas pode iniciar scripts que não sejam Python. No exemplo pré-construído, você pode ver que os scripts do algoritmo Python são lançados como um subprocesso do Python ou como um processo totalmente novo. Ao modificar essa lógica, você pode habilitar o script de ponto de entrada para iniciar scripts de algoritmo não Python. Por exemplo,

você pode modificar a função `thekick_off_customer_script()` para iniciar processos Rust dependendo do final da extensão do arquivo.

Você também pode escrever um `braket_container.py` completamente novo. Ele deve copiar dados de entrada, arquivos de origem e outros arquivos necessários do Amazon S3 para o contêiner e definir as variáveis de ambiente apropriadas.

Instale o software e o script de contêiner necessários com **`Dockerfile`**

> **Note**
>
> Se você usar uma imagem Braket pré-criada como imagem Docker base, o script do contêiner já estará presente.

Se você criou um script de contêiner modificado na etapa anterior, precisará copiá-lo para o contêiner e definir a variável de ambiente de `SAGEMAKER_PROGRAM` para `braket_container.py`, ou para o nome que você deu ao seu novo script de ponto de entrada do contêiner.

Veja a seguir um exemplo de um `Dockerfile` que permite que você use Julia em instâncias de Jobs aceleradas por GPU:

```
FROM nvidia/cuda:12.2.0-devel-ubuntu22.04


 ARG DEBIAN_FRONTEND=noninteractive
 ARG JULIA_RELEASE=1.8
 ARG JULIA_VERSION=1.8.3


 ARG PYTHON=python3.11
 ARG PYTHON_PIP=python3-pip
 ARG PIP=pip


 ARG JULIA_URL = https://julialang-s3.julialang.org/bin/linux/x64/${JULIA_RELEASE}/
 ARG TAR_NAME = julia-${JULIA_VERSION}-linux-x86_64.tar.gz


 ARG PYTHON_PKGS = # list your Python packages and versions here
```

```
RUN curl -s -L ${JULIA_URL}/${TAR_NAME} | tar -C /usr/local -x -z --strip-components=1
-f -


RUN apt-get update \

   && apt-get install -y --no-install-recommends \

   build-essential \

   tzdata \

   openssh-client \

   openssh-server \

   ca-certificates \

   curl \

   git \

   libtemplate-perl \

   libssl1.1 \

   openssl \

   unzip \

   wget \

   zlib1g-dev \

   ${PYTHON_PIP} \

   ${PYTHON}-dev \




RUN ${PIP} install --no-cache --upgrade ${PYTHON_PKGS}
```

```
 RUN ${PIP} install --no-cache --upgrade sagemaker-training==4.1.3


 # Add EFA and SMDDP to LD library path
 ENV LD_LIBRARY_PATH="/opt/conda/lib/python${PYTHON_SHORT_VERSION}/site-packages/
smdistributed/dataparallel/lib:$LD_LIBRARY_PATH"
 ENV LD_LIBRARY_PATH=/opt/amazon/efa/lib/:$LD_LIBRARY_PATH


 # Julia specific installation instructions
 COPY Project.toml /usr/local/share/julia/environments/v${JULIA_RELEASE}/
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using Pkg; Pkg.instantiate(); Pkg.API.precompile()'
 # generate the device runtime library for all known and supported devices
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using CUDA; CUDA.precompile_runtime()'


 # Open source compliance scripts
 RUN HOME_DIR=/root \

 && curl -o ${HOME_DIR}/oss_compliance.zip https://aws-dlinfra-
utilities.s3.amazonaws.com/oss_compliance.zip \

 && unzip ${HOME_DIR}/oss_compliance.zip -d ${HOME_DIR}/ \

 && cp ${HOME_DIR}/oss_compliance/test/testOSSCompliance /usr/local/bin/
testOSSCompliance \

 && chmod +x /usr/local/bin/testOSSCompliance \

 && chmod +x ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh \

 && ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh ${HOME_DIR} ${PYTHON} \

 && rm -rf ${HOME_DIR}/oss_compliance*


 # Copying the container entry point script
 COPY braket_container.py /opt/ml/code/braket_container.py
 ENV SAGEMAKER_PROGRAM braket_container.py
```

Este exemplo baixa e executa scripts fornecidos por AWS para garantir a conformidade com todas as licenças de código aberto relevantes. Por exemplo, atribuindo adequadamente qualquer código instalado governado por um MIT license.

Se você precisar incluir código não público, por exemplo, código hospedado em um GitLab repositório GitHub ou privado, não incorpore chaves SSH na Docker imagem para acessá-la. Em vez disso, use o Docker Compose ao criar para permitir ao Docker o acesso ao SSH na máquina host em que ele foi criado. Para obter mais informações, consulte o guia Como usar chaves SSH com segurança no Docker para acessar repositórios privados do Github.

Criando e enviando sua imagem Docker

Com um repositório Amazon ECR devidamente definido`Dockerfile`, agora você está pronto para seguir as etapas para criar um repositório privado do Amazon ECR, caso ainda não exista um. Você também pode criar, marcar e carregar sua imagem de contêiner no repositório.

Você está pronto para criar, marcar e enviar a imagem. Consulte a documentação de compilação do Docker para obter uma explicação completa das opções para o `docker build` e alguns exemplos.

Para o arquivo de amostra definido acima, você pode executar:

```
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
```

Atribuição de permissões apropriadas do Amazon ECR

As imagens do Braket Hybrid Jobs Docker devem ser hospedadas em repositórios privados do Amazon ECR. Por padrão, um repositório privado do Amazon ECR não fornece acesso de leitura ao Braket Hybrid Jobs IAM role ou a nenhum outro usuário que queira usar sua imagem, como um colaborador ou estudante. Você precisa definir uma política de repositório para conceder as permissões apropriadas. Em geral, dê permissão apenas aos usuários e perfis do IAM específicos aos quais você deseja acessar suas imagens, em vez de permitir que qualquer pessoa com a image URI as extraia.

Executando trabalhos híbridos do Braket em seu próprio contêiner

Para criar um trabalho híbrido com seu próprio contêiner, chame `AwsQuantumJob.create()` com o argumento `image_uri` especificado. Você pode usar uma QPU, um simulador sob demanda ou executar seu código localmente no processador clássico disponível com o Braket Hybrid Jobs. Recomendamos testar seu código em um simulador como SV1 DM1, ou TN1 antes de executar em uma QPU real.

Para executar seu código no processador clássico, especifique o `instanceType` e o `instanceCount` que você usa atualizando o `InstanceConfig`. Observe que, se você especificar um `instance_count` > 1, precisará garantir que seu código possa ser executado em vários hosts. O limite superior para o número de instâncias que você pode escolher é 5. Por exemplo:

```
job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    image_uri="111122223333.dkr.ecr.us-west-2.amazonaws.com/my-byoc-container:latest",
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

> **Note**
>
> Use o ARN do dispositivo para rastrear o simulador que você usou como metadados do trabalho híbrido. Os valores aceitáveis devem seguir o formato `device = "local:<provider>/<simulator_name>"`. Lembre-se que `<provider>` e `<simulator_name>` deve consistir apenas em letras, números, `_`, `-`, e `.`. O tamanho máximo da string é 256 caracteres.
> Se você planeja usar o BYOC e não está usando o SDK do Braket para criar tarefas quânticas, deve passar o valor da variável ambiental `AMZN_BRAKET_JOB_TOKEN` para o parâmetro `jobToken` na solicitação `CreateQuantumTask`. Caso contrário, as tarefas quânticas não têm prioridade e são cobradas como tarefas quânticas autônomas regulares.

Usando hiperparâmetros

Você pode definir os hiperparâmetros necessários ao seu algoritmo, como a taxa de aprendizado ou o tamanho da etapa, ao criar um trabalho híbrido. Os valores dos hiperparâmetros são normalmente

usados para controlar vários aspectos do algoritmo e geralmente podem ser ajustados para otimizar o desempenho do algoritmo. Para usar hiperparâmetros em uma tarefa híbrida do Braket, você precisa especificar seus nomes e valores explicitamente como um dicionário. Especifique os valores dos hiperparâmetros a serem testados ao pesquisar o conjunto ideal de valores. A primeira etapa para usar hiperparâmetros é configurar e definir os hiperparâmetros como um dicionário, que pode ser visto no código a seguir.

```
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

hyperparameters = {"shots": 1_000}
```

Em seguida, passe os hiperparâmetros definidos no trecho de código fornecido acima para serem usados no algoritmo de sua escolha. Para executar o exemplo de código a seguir, crie um diretório chamado “src” no mesmo caminho do seu arquivo de hiperparâmetros. Dentro do diretório "src", adicione os arquivos de código 0_Getting_started_papermill.ipynb, notebook_runner.py, e requirements.txt

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="src",
    entry_point="src.notebook_runner:run_notebook",
    input_data="src/0_Getting_started_papermill.ipynb",
    hyperparameters=hyperparameters,
    job_name=f"papermill-job-demo-{int(time.time())}",
)

# Print job to record the ARN
print(job)
```

Para acessar seus hiperparâmetros de dentro do seu script de trabalho híbrido, consulte a função `load_jobs_hyperparams()` no arquivo notebook_runner.py. Para acessar seus hiperparâmetros fora do script de trabalho híbrido, execute o código a seguir.

```
from braket.aws import AwsQuantumJob
```

```
# Get the job using the ARN
job_arn = "arn:aws:braket:us-east-1:111122223333:job/5eabb790-d3ff-47cc-98ed-
b4025e9e296f"  # Replace with your job ARN
job = AwsQuantumJob(arn=job_arn)

# Access the hyperparameters
job_metadata = job.metadata()
hyperparameters = job_metadata.get("hyperParameters", {})
print(hyperparameters)
```

Para obter mais informações sobre como aprender a usar hiperparâmetros, consulte os tutoriais QAOA com Amazon Braket Hybrid Jobs e Quantum PennyLane machine learning nos Amazon Braket Hybrid Jobs.

Configure sua instância de trabalho híbrida

Dependendo do seu algoritmo, você pode ter requisitos diferentes. Por padrão, o Amazon Braket executa seu script de algoritmo em uma instância `ml.m5.large`. No entanto, você pode personalizar esse tipo de instância ao criar um trabalho híbrido usando o seguinte argumento de importação e configuração.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"), # Use NVIDIA T4
 instance with 4 GPUs.
    ...
    ),
```

Se você estiver executando uma simulação incorporada e tiver especificado um dispositivo local na configuração do dispositivo, também poderá solicitar mais de uma instância no `InstanceConfig` especificando `instanceCount` e definindo que seja maior que um. O limite superior é 5. Por exemplo, você pode escolher 3 instâncias da seguinte maneira.

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3), #
 Use 3 NVIDIA T4 instances
    ...
```

```
    ),
```

Ao usar várias instâncias, considere distribuir seu trabalho híbrido usando o recurso de data parallel. Consulte o exemplo de caderno a seguir para obter mais detalhes sobre como ver esse exemplo de treinamento do Parallelize para QML.

As três tabelas a seguir listam os tipos e especificações de instância disponíveis para instâncias padrão, de alto desempenho e aceleradas por GPU.

> **Note**
>
> Para ver as cotas padrão de instância de computação clássica para trabalhos híbridos, consulte a página Cotas do Amazon Braket.

Instâncias padrão	vCPU	Memória (GiB)
ml.m5.large (padrão)	4	16
ml.m5.xlarge	4	16
ml.m5.2xlarge	8	32
ml.m5.4xlarge	16	64
ml.m5.12xlarge	48	192
ml.m5.24xlarge	96	384

Instâncias de alta performance	vCPU	Memória (GiB)
ml.c5.xlarge	4	8
ml.c5.2xlarge	8	16
ml.c5.4xlarge	16	32
ml.c5.9xlarge	36	72

Instâncias de alta performance	vCPU	Memória (GiB)
ml.c5.18xlarge	72	144
ml.c5n.xlarge	4	10.5
ml.c5n.2xlarge	8	21
ml.c5n.4xlarge	16	32
ml.c5n.9xlarge	36	72
ml.c5n.18xlarge	72	192

Inferência acelerada por GPU	GPUs	vCPU	Memória (GiB)	Memória de GPU (GiB)
ml.p4d.24xlarge	8	96	1152	320
ml.g4dn.xlarge	1	4	16	16
ml.g4dn.2xlarge	1	8	32	16
ml.g4dn.4xlarge	1	16	64	16
ml.g4dn.8xlarge	1	32	128	16
ml.g4dn.12xlarge	4	48	192	64
ml.g4dn.16xlarge	1	64	256	16

Cada instância usa uma configuração padrão de armazenamento de dados (SSD) de 30 GB. Mas você pode ajustar o armazenamento da mesma forma que configura o `instanceType`. O exemplo a seguir mostra como aumentar o armazenamento total para 50 GB.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
```

```
    ...
    instance_config=InstanceConfig(
        instanceType="ml.g4dn.xlarge",
        volumeSizeInGb=50,
    ),
    ...
    ),
```

Configure o bucket padrão em **`AwsSession`**

A utilização de sua própria instância de `AwsSession` oferece maior flexibilidade, como a capacidade de especificar um local personalizado para seu bucket padrão do Amazon S3. Por padrão, uma `AwsSession` tem uma localização pré-configurada de bucket do Amazon S3 do `"amazon-braket-{id}-{region}"`. No entanto, você tem a opção de substituir a localização padrão do bucket do Amazon S3 ao criar uma `AwsSession`. Opcionalmente, os usuários podem passar um objeto `AwsSession` para o método `AwsQuantumJob.create()`, fornecendo o parâmetro `aws_session` conforme demonstrado no exemplo de código a seguir.

```
aws_session = AwsSession(default_bucket="amazon-braket-s3-demo-bucket")

# Then you can use that AwsSession when creating a hybrid job
job = AwsQuantumJob.create(
    ...
    aws_session=aws_session
)
```

Usar compilação paramétrica para acelerar trabalhos híbridos

O Amazon Braket oferece suporte à compilação paramétrica em alguns. QPUs Isso permite que você reduza a sobrecarga associada à etapa de compilação computacionalmente cara compilando um circuito apenas uma vez e não para cada iteração em seu algoritmo híbrido. Isso pode melhorar drasticamente os runtimes de trabalhos híbridos, já que você evita a necessidade de recompilar seu circuito em cada etapa. Basta enviar circuitos parametrizados para um dos nossos serviços suportados QPUs como Braket Hybrid Job. Para trabalhos híbridos de longa duração, o Braket usa automaticamente os dados de calibração atualizados do fornecedor de hardware ao compilar seu circuito para garantir resultados da mais alta qualidade.

Para criar um circuito paramétrico, primeiro você precisa fornecer parâmetros como entradas em seu script de algoritmo. Neste exemplo, usamos um pequeno circuito paramétrico e ignoramos qualquer

processamento clássico entre cada iteração. Para workloads típicas, você enviaria vários circuitos em lote e executaria o processamento clássico, como atualizar os parâmetros em cada iteração.

```
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.")
```

Você pode enviar o script do algoritmo para ser executado como um Hybrid Job com o script de trabalho a seguir. Ao executar o Hybrid Job em uma QPU compatível com compilação paramétrica, o circuito é compilado somente na primeira execução. Nas execuções seguintes, o circuito compilado é reutilizado, aumentando o desempenho do runtime do Hybrid Job sem nenhuma linha adicional de código.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="algorithm_script.py",
)
```

> **Note**
>
> A compilação paramétrica é suportada em todos os formatos supercondutores baseados em portas, Rigetti Computing com exceção QPUs dos programas de nível de pulso.

Usando PennyLane com o Amazon Braket

Algoritmos híbridos são algoritmos que contêm instruções clássicas e quânticas. As instruções clássicas são executadas em hardware clássico (uma instância EC2 ou seu laptop), e as instruções quânticas são executadas em um simulador ou em um computador quântico. Recomendamos que você execute algoritmos híbridos usando o recurso Hybrid Jobs. Para obter mais informações, consulte Quando usar o Amazon Braket Jobs.

O Amazon Braket permite que você configure e execute algoritmos quânticos híbridos com a ajuda do plug-in Amazon Braket ou com o SDK Amazon PennyLane Braket para Python e exemplos de repositórios de notebooks. Os notebooks de exemplo do Amazon Braket, baseados no SDK, permitem que você configure e execute determinados algoritmos híbridos sem o plug-in. PennyLane No entanto, recomendamos PennyLane porque proporciona uma experiência mais rica.

Sobre algoritmos quânticos híbridos

Os algoritmos quânticos híbridos são importantes para a indústria atual porque os dispositivos de computação quântica contemporâneos geralmente produzem ruído e, portanto, erros. Cada porta quântica adicionada a uma computação aumenta a chance de adicionar ruído; portanto, algoritmos de longa execução podem ser sobrecarregados pelo ruído, o que resulta em cálculos defeituosos.

Algoritmos quânticos puros, como o de Shor (exemplo de estimativa de fase quântica) ou o de Grover (exemplo de Grover), exigem milhares ou milhões de operações. Por esse motivo, eles podem ser impraticáveis para dispositivos quânticos existentes, geralmente chamados de dispositivos quânticos ruidosos de escala intermediária (NISQ).

Em algoritmos quânticos híbridos, as unidades de processamento quântico (QPUs) funcionam como coprocessadores para o clássico CPUs, especificamente para acelerar certos cálculos em um algoritmo clássico. As execuções de circuitos se tornam muito mais curtas, ao alcance dos recursos dos dispositivos atuais.

Nesta seção:

- Amazon Braket com PennyLane
- Algoritmos híbridos em cadernos de exemplo do Amazon Braket
- Algoritmos híbridos com PennyLane simuladores incorporados
- Gradiente adjunto PennyLane com simuladores Amazon Braket
- Usando trabalhos híbridos e PennyLane para executar um algoritmo QAOA
- Execute cargas de trabalho híbridas com PennyLane simuladores incorporados

Amazon Braket com PennyLane

O Amazon Braket fornece suporte PennyLanepara uma estrutura de software de código aberto criada com base no conceito de programação quântica diferenciável. Você pode usar essa estrutura para treinar circuitos quânticos da mesma forma que treinaria uma rede neural para encontrar soluções para problemas computacionais em química quântica, aprendizado de máquina quântica e otimização.

A PennyLane biblioteca fornece interfaces para ferramentas conhecidas de aprendizado de máquina, incluindo PyTorch e TensorFlow, para tornar o treinamento de circuitos quânticos rápido e intuitivo.

- A PennyLane Biblioteca -— PennyLane está pré-instalada nos notebooks Amazon Braket. Para acessar os dispositivos Amazon Braket a partir de PennyLane, abra um notebook e importe a PennyLane biblioteca com o comando a seguir.

```
import pennylane as qml
```

Os cadernos tutoriais ajudam você a começar rapidamente. Como alternativa, você pode usar PennyLane no Amazon Braket a partir de um IDE de sua escolha.

- O PennyLane plug-in Amazon Braket — Para usar seu próprio IDE, você pode instalar o plug-in Amazon PennyLane Braket manualmente. O plug-in se PennyLane conecta ao SDK Amazon Braket Python, para que você possa executar circuitos em dispositivos Braket. PennyLane Amazon Para instalar o PennyLane plug-in, use o comando a seguir.

```
pip install amazon-braket-pennylane-plugin
```

O exemplo a seguir demonstra como configurar o acesso aos dispositivos Amazon Braket em: 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 exemplos de tutoriais e mais informações sobre PennyLane, consulte o repositório de exemplos do Amazon Braket.

O PennyLane plug-in Amazon Braket permite que você alterne entre o Amazon Braket QPU e dispositivos simuladores incorporados PennyLane com uma única linha de código. Ele oferece dois dispositivos quânticos Amazon Braket para trabalhar com: PennyLane

- `braket.aws.qubit`para funcionar com os dispositivos quânticos do serviço Amazon Braket, inclusive simuladores QPUs
- `braket.local.qubit` para execução com o simulador local do SDK do Amazon Braket

O PennyLane plugin Amazon Braket é de código aberto. Você pode instalá-lo a partir do GitHub repositório de PennyLane plug-ins.

Para obter mais informações sobre PennyLane, consulte a documentação no PennyLane site.

Algoritmos híbridos em cadernos de exemplo do Amazon Braket

O Amazon Braket fornece uma variedade de exemplos de notebooks que não dependem do plug-in para executar algoritmos PennyLane híbridos. Você pode começar com qualquer um destes cadernos de exemplo híbridos do Amazon Braket que ilustram métodos variacionais, como o Algoritmo de Otimização Aproximada Quântica (QAOA) ou o Solucionador de Autovalores Quântico Variacional (VQE).

Os cadernos de exemplo do Amazon Braket dependem do SDK Amazon Braket Python. O SDK fornece uma estrutura para interagir com dispositivos de hardware de computação quântica por meio do Amazon Braket. É uma biblioteca de código aberto projetada para ajudá-lo com a parte quântica do seu fluxo de trabalho híbrido.

Você pode explorar ainda mais o Amazon Braket com nossos exemplos de cadernos.

Algoritmos híbridos com PennyLane simuladores incorporados

O Amazon Braket Hybrid Jobs agora vem com simuladores incorporados de alto desempenho baseados em CPU e GPU da. PennyLane Essa família de simuladores integrados pode ser incorporada diretamente no seu contêiner de tarefas híbridas e inclui o simulador `lightning.qubit` de vetor de estado rápido, o simulador `lightning.gpu` acelerado usando a biblioteca cuQuantum da NVIDIA, entre outros. Esses simuladores incorporados são ideais para algoritmos variacionais, como aprendizado de máquina quântico, que podem se beneficiar de métodos avançados, como o método de diferenciação adjunta. Você pode executar esses simuladores incorporados em uma ou várias instâncias de CPU ou GPU.

Com o Hybrid Jobs, agora você pode executar seu código de algoritmo variacional usando uma combinação de um coprocessador clássico e uma QPU, um simulador sob demanda Amazon Braket, comoSV1, ou usando diretamente o simulador incorporado do. PennyLane

O simulador incorporado já está disponível com o contêiner Hybrid Jobs. Você precisa decorar sua função principal do Python com o decorador `@hybrid_job`. Para usar o PennyLane `lightning.gpu` simulador, você também precisa especificar uma instância de GPU no, `InstanceConfig` conforme mostrado no seguinte trecho de código:

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


@hybrid_job(device="local:pennylane/lightning.gpu",
 instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"))
def function(wires):
    dev = qml.device("lightning.gpu", wires=wires)
    ...
```

Consulte o exemplo de caderno para começar a usar um simulador PennyLane incorporado com o Hybrid Jobs.

Gradiente adjunto PennyLane com simuladores Amazon Braket

Com o PennyLane plug-in do Amazon Braket, você pode calcular gradientes usando o método de diferenciação adjunta ao executar no simulador vetorial estadual local ou. SV1

Nota: Para usar o método de diferenciação adjunta, você deve especificar `diff_method='device'` em seu qnode, e não `diff_method='adjoint'`. Veja o exemplo a seguir.

```
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**
>
> Atualmente, PennyLane calculará índices de agrupamento para hamiltonianos da QAOA e os usará para dividir o hamiltoniano em vários valores de expectativa. Se você quiser usar o recurso SV1 de diferenciação adjunta ao executar o QAOA a partir dePennyLane, precisará reconstruir o hamiltoniano de custo removendo os índices de agrupamento, da seguinte forma: `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False)`
> `cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Usando trabalhos híbridos e PennyLane para executar um algoritmo QAOA

Nesta seção, você usará o que aprendeu para escrever um programa híbrido real usando PennyLane a compilação paramétrica. Você usa o script do algoritmo para resolver um problema do Algoritmo de Otimização Aproximada Quântica (QAOA). O programa cria uma função de custo correspondente a um problema clássico de otimização do Max Cut, especifica um circuito quântico parametrizado e usa um método de gradiente descendente para otimizar os parâmetros para que a função de custo seja minimizada. Neste exemplo, geramos o gráfico do problema no script do algoritmo para simplificar, mas para casos de uso mais comuns, a melhor prática é fornecer a especificação do problema por meio de um canal dedicado na configuração dos dados de entrada. O `parametrize_differentiable` padrão do sinalizador é para que você `True` obtenha automaticamente os benefícios do desempenho aprimorado do tempo de execução a partir da compilação paramétrica, quando suportada. QPUs

```
import os
import json
import time

from braket.jobs import save_job_result
from braket.jobs.metrics import log_metric

import networkx as nx
import pennylane as qml
from pennylane import numpy as np
from matplotlib import pyplot as plt

def init_pl_device(device_arn, num_nodes, shots, max_parallel):
    return qml.device(
        "braket.aws.qubit",
        device_arn=device_arn,
        wires=num_nodes,
        shots=shots,
        # Set s3_destination_folder=None to output task results to a default folder
        s3_destination_folder=None,
        parallel=True,
        max_parallel=max_parallel,
        parametrize_differentiable=True, # This flag is True by default.
    )

def start_here():
    input_dir = os.environ["AMZN_BRAKET_INPUT_DIR"]
    output_dir = os.environ["AMZN_BRAKET_JOB_RESULTS_DIR"]
    job_name = os.environ["AMZN_BRAKET_JOB_NAME"]
    checkpoint_dir = os.environ["AMZN_BRAKET_CHECKPOINT_DIR"]
    hp_file = os.environ["AMZN_BRAKET_HP_FILE"]
    device_arn = os.environ["AMZN_BRAKET_DEVICE_ARN"]

    # Read the hyperparameters
    with open(hp_file, "r") as f:
        hyperparams = json.load(f)

    p = int(hyperparams["p"])
    seed = int(hyperparams["seed"])
    max_parallel = int(hyperparams["max_parallel"])
    num_iterations = int(hyperparams["num_iterations"])
    stepsize = float(hyperparams["stepsize"])
    shots = int(hyperparams["shots"])
```

```
# Generate random graph
num_nodes = 6
num_edges = 8
graph_seed = 1967
g = nx.gnm_random_graph(num_nodes, num_edges, seed=graph_seed)

# Output figure to file
positions = nx.spring_layout(g, seed=seed)
nx.draw(g, with_labels=True, pos=positions, node_size=600)
plt.savefig(f"{output_dir}/graph.png")

# Set up the QAOA problem
cost_h, mixer_h = qml.qaoa.maxcut(g)

def qaoa_layer(gamma, alpha):
    qml.qaoa.cost_layer(gamma, cost_h)
    qml.qaoa.mixer_layer(alpha, mixer_h)

def circuit(params, **kwargs):
    for i in range(num_nodes):
        qml.Hadamard(wires=i)
    qml.layer(qaoa_layer, p, params[0], params[1])

dev = init_pl_device(device_arn, num_nodes, shots, max_parallel)

np.random.seed(seed)
cost_function = qml.ExpvalCost(circuit, cost_h, dev, optimize=True)
params = 0.01 * np.random.uniform(size=[2, p])

optimizer = qml.GradientDescentOptimizer(stepsize=stepsize)
print("Optimization start")

for iteration in range(num_iterations):
    t0 = time.time()

    # Evaluates the cost, then does a gradient step to new params
    params, cost_before = optimizer.step_and_cost(cost_function, params)
    # Convert cost_before to a float so it's easier to handle
    cost_before = float(cost_before)

    t1 = time.time()

    if iteration == 0:
```

```
            print("Initial cost:", cost_before)
        else:
            print(f"Cost at step {iteration}:", cost_before)

        # Log the current loss as a metric
        log_metric(
            metric_name="Cost",
            value=cost_before,
            iteration_number=iteration,
        )

        print(f"Completed iteration {iteration + 1}")
        print(f"Time to complete iteration: {t1 - t0} seconds")

    final_cost = float(cost_function(params))
    log_metric(
        metric_name="Cost",
        value=final_cost,
        iteration_number=num_iterations,
    )

    # We're done with the hybrid job, so save the result.
    # This will be returned in job.result()
    save_job_result({"params": params.numpy().tolist(), "cost": final_cost})
```

> **Note**
>
> A compilação paramétrica é suportada em todos os formatos supercondutores baseados em portas, Rigetti Computing com exceção QPUs dos programas de nível de pulso.

Execute cargas de trabalho híbridas com PennyLane simuladores incorporados

Vamos ver como você pode usar simuladores incorporados do PennyLane Amazon Braket Hybrid Jobs para executar cargas de trabalho híbridas. O simulador incorporado baseado em GPU da Pennylane, `lightning.gpu`, usa a biblioteca Nvidia CuQuantum para acelerar as simulações de circuitos. O simulador de GPU incorporado é pré-configurado em todos os contêineres de trabalho do Braket que os usuários podem usar imediatamente. Nesta página, mostraremos como usar `lightning.gpu` para acelerar suas workloads híbrida.

Usando **`lightning.gpu`** para workloads QAOA

Considere os exemplos do Algoritmo de Otimização Aproximada Quântica (QAOA) deste caderno. Para selecionar um simulador incorporado, você especifica o argumento `device` para ser uma string no formato: `"local:<provider>/<simulator_name>"`. Por exemplo, você definiria `"local:pennylane/lightning.gpu"` para `lightning.gpu`. A string do dispositivo que você fornece ao Hybrid Job ao iniciar é passada para o trabalho como a variável de ambiente `"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)
```

Nesta página, compare os dois simuladores de vetores de PennyLane estado incorporados `lightning.qubit` (que são baseados em CPU) e `lightning.gpu` (que são baseados em GPU). Forneça aos simuladores decomposições de portas personalizadas para calcular vários gradientes.

Agora você está pronto para preparar o script híbrido de lançamento de tarefas. Execute o algoritmo QAOA usando dois tipos de instância: `ml.m5.2xlarge` e `ml.g4dn.xlarge`. O tipo `ml.m5.2xlarge` de instância é comparável a um laptop padrão para desenvolvedores. `ml.g4dn.xlarge`É uma instância de computação acelerada que tem uma única GPU NVIDIA T4 com 16 GB de memória.

Para executar a GPU, primeiro precisamos especificar uma imagem compatível e a instância correta (cujo padrão é uma `ml.m5.2xlarge` instância).

```
from braket.aws import AwsSession
from braket.jobs.image_uris import Framework, retrieve_image

image_uri = retrieve_image(Framework.PL_PYTORCH, AwsSession().region)
instance_config = InstanceConfig(instanceType="ml.g4dn.xlarge")
```

Em seguida, precisamos inseri-los no decorador de trabalho híbrido, junto com os parâmetros atualizados do dispositivo nos argumentos do sistema e do trabalho híbrido.

```
@hybrid_job(
        device="local:pennylane/lightning.gpu",
        input_data=input_file_path,
        image_uri=image_uri,
```

```
        instance_config=instance_config)
def run_qaoa_hybrid_job_gpu(p=1, steps=10):
    params = np.random.rand(2, p)

    braket_task_tracker = Tracker()

    graph = nx.read_adjlist(input_file_path, nodetype=int)
    wires = list(graph.nodes)
    cost_h, _mixer_h = qaoa.maxcut(graph)

    device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
    prefix, device_name = device_string.split("/")
    dev= qml.device(simulator_name, wires=len(wires))
    ...
```

> **Note**
>
> Se você especificar o `instance_config` como usando uma instância baseada em GPU, mas `device` escolher o como simulador baseado em CPU incorporado (`lightning.qubit`), a GPU não será usada. Certifique-se de usar o simulador incorporado baseado em GPU se quiser atingir a GPU!

O tempo médio de iteração para a `m5.2xlarge` instância é de cerca de 73 segundos, enquanto para a `ml.g4dn.xlarge` instância é de cerca de 0,6 segundos. Para esse fluxo de trabalho de 21 qubits, a instância da GPU nos dá uma aceleração de 100x. Se você olhar a página de preços do Amazon Braket Hybrid Jobs, verá que o custo por minuto para `m5.2xlarge` uma instância é de 0,00768 USD, enquanto para a instância é de 0,01227 USD. `ml.g4dn.xlarge` Nesse caso, é mais rápido e barato executar na instância da GPU.

Aprendizado de máquina quântico e paralelismo de dados

Se seu tipo de workload for aprendizado de máquina quântico (QML) treinado em conjuntos de dados, você poderá acelerar ainda mais sua workload usando o paralelismo de dados. No QML, o modelo contém um ou mais circuitos quânticos. O modelo também pode ou não conter redes neurais clássicas. Ao treinar o modelo com o conjunto de dados, os parâmetros no modelo são atualizados para minimizar a função de perda. Uma função de perda geralmente é definida para um único ponto de dados e a perda total para a perda média em todo o conjunto de dados. Em QML, as perdas geralmente são calculadas em série antes da média da perda total para cálculos de gradiente. Esse procedimento é demorado, especialmente quando há centenas de pontos de dados.

Como a perda de um ponto de dados não depende de outros pontos de dados, as perdas podem ser avaliadas paralelamente! Perdas e gradientes associados a diferentes pontos de dados podem ser avaliados ao mesmo tempo. Isso é conhecido como paralelismo de dados. Com SageMaker a biblioteca paralela de dados distribuídos, o Amazon Braket Hybrid Jobs facilita o uso do paralelismo de dados para acelerar seu treinamento.

Considere a seguinte workload QML para paralelismo de dados, que usa o conjunto de dados Sonar do conhecido repositório UCI como exemplo de classificação binária. O conjunto de dados Sonar tem 208 pontos de dados, cada um com 60 características que são coletadas de sinais de sonar refletidos em materiais. Cada ponto de dados é rotulado como “M” para minas ou “R” para rochas. Nosso modelo QML consiste em uma camada de entrada, um circuito quântico como camada oculta e uma camada de saída. As camadas de entrada e saída são redes neurais clássicas implementadas em PyTorch. O circuito quântico é integrado às PyTorch redes neurais usando PennyLane o módulo qml.qnn. Veja nossos exemplos de cadernos para obter mais detalhes sobre a workload. Como no exemplo de QAOA acima, você pode aproveitar o poder da GPU usando simuladores incorporados baseados em GPU, como PennyLane os nossos, `lightning.gpu` para melhorar o desempenho em relação aos simuladores baseados em CPU incorporada.

Para criar uma tarefa híbrida, você pode chamar `AwsQuantumJob.create` e especificar o script do algoritmo, o dispositivo e outras configurações por meio de seus argumentos de palavra-chave.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.xlarge')

hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_single",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    ...
)
```

Para usar o paralelismo de dados, você precisa modificar algumas linhas de código no script do algoritmo da biblioteca SageMaker distribuída para paralelizar corretamente o treinamento. Primeiro, você importa o `smdistributed` pacote que faz a maior parte do trabalho pesado para

distribuir suas cargas de trabalho em várias e várias GPUs instâncias. Este pacote é pré-configurado no Braket PyTorch e nos contêineres. TensorFlow O `dist` módulo informa ao nosso script de algoritmo qual é o número total de GPUs para o treinamento (`world_size`), bem como o final `rank` `local_rank` de um núcleo de GPU. `rank`é o índice absoluto de uma GPU em todas as instâncias, enquanto `local_rank` é o índice de uma GPU dentro de uma instância. Por exemplo, se houver quatro instâncias, cada uma com oito GPUs alocadas para o treinamento, `rank` elas variam de 0 a 31 e as `local_rank` 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)
```

Em seguida, você define um `DistributedSampler` de acordo com `world_size` e `rank` e, em seguida, o passa para o carregador de dados. Esse amostrador evita GPUs acessar a mesma fatia de um conjunto de dados.

```
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,
)
```

Em seguida, você usa a classe `DistributedDataParallel` para ativar o paralelismo de dados.

```
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"])
```

As alterações acima são necessárias para usar o paralelismo de dados. Em QML, você geralmente deseja salvar os resultados e imprimir o progresso do treinamento. Se cada GPU executar o comando de salvar e imprimir, o log será inundado com as informações repetidas e os resultados se substituirão. Para evitar isso, você só pode salvar e imprimir a partir da GPU que tenha `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})
```

O Amazon Braket Hybrid Jobs `ml.g4dn.12xlarge` oferece suporte a tipos de instância para a biblioteca paralela de dados SageMaker distribuídos. Você configura o tipo de instância por meio do argumento `InstanceConfig` em Hybrid Jobs. Para que a biblioteca paralela de dados SageMaker distribuídos saiba que o paralelismo de dados está ativado, você precisa adicionar dois hiperparâmetros adicionais, `"sagemaker_distributed_dataparallel_enabled"` configurando `"true"` e `"sagemaker_instance_type"` configurando o tipo de instância que você está usando. Esses dois hiperparâmetros são usados por pacote `smdistributed`. Seu script de algoritmo não precisa usá-los explicitamente. No Amazon Braket SDK, ele fornece um argumento `distribution` de palavra-chave conveniente. Com `distribution="data_parallel"` de empregos híbridos, o Amazon Braket SDK insere automaticamente os dois hiperparâmetros para você. Se você usa a API Amazon Braket, precisa incluir esses dois hiperparâmetros.

Com o paralelismo de instâncias e dados configurados, agora você pode enviar seu trabalho híbrido. Há 4 GPUs em uma `ml.g4dn.12xlarge` instância. Quando você define`instanceCount=1`, a carga de trabalho é distribuída entre os 8 GPUs na instância. Quando você define `instanceCount` mais de um, a carga de trabalho é distribuída entre os GPUs disponíveis em todas as instâncias. Ao usar várias instâncias, cada instância incorre em uma cobrança com base no tempo de uso. Por exemplo, quando você usa quatro instâncias, o tempo faturável é quatro vezes o tempo de execução por instância, pois há quatro instâncias executando suas workloads ao mesmo tempo.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.12xlarge',
                                 instanceCount=1,
)
```

```
hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...,
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_dp",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    distribution="data_parallel",
    ...
)
```

> **Note**
>
> Na criação de empregos híbridos acima, `train_dp.py` é o script de algoritmo modificado para usar o paralelismo de dados. Lembre-se de que o paralelismo de dados só funciona corretamente quando você modifica seu script de algoritmo de acordo com a seção acima. Se a opção de paralelismo de dados for ativada sem um script de algoritmo modificado corretamente, a tarefa híbrida poderá gerar erros ou cada GPU poderá processar repetidamente a mesma fatia de dados, o que é ineficiente.

Se usado corretamente, o uso de várias instâncias pode levar a uma redução de ordens de magnitude no tempo e no custo. Consulte o exemplo de caderno para obter mais detalhes.

Usando o CUDA-Q com o Amazon Braket

NVIDIA's CUDA-Qé uma biblioteca de software projetada para programar algoritmos quânticos híbridos que combinam CPUs GPUs, e unidades de processamento quântico (QPUs). Ele fornece um modelo de programação unificado, permitindo que os desenvolvedores expressem instruções clássicas e quânticas em um único programa, simplificando os fluxos de trabalho. CUDA-Qacelera a simulação e o tempo de execução de programas quânticos com seus simuladores integrados de CPU e GPU. CUDA-Qestá disponível com instâncias nativas do notebook Braket (NBIs) e Amazon Braket Hybrid Jobs.

Nesta seção:

- CUDA-Q em NBIs
- CUDA-Q no Hybrid Jobs

CUDA-Q em NBIs

CUDA-Q é instalado por padrão no ambiente Braket NBI. Você pode abrir um caderno de exemplo CUDA-Q acessando a página inicial do Jupyter e selecionando o bloco CUDA-Q e Braket. Isso abre o caderno `0_Getting_started_with_CUDA-Q.ipynb` de exemplo na janela principal. Para obter mais exemplos CUDA-Q, consulte o painel esquerdo no `nvidia_cuda_q/` diretório.

Você também pode verificar a versão CUDA-Q ou qualquer outro pacote de terceiros instalado em seu NBI. Por exemplo, você pode executar o comando a seguir em uma célula de código do notebook para verificar as versões dos CUDA-Q pacotes Qiskit e Braket que estão instalados no ambiente. PennyLane

```
%pip freeze | grep -i -e cudaq -e qiskit -e pennylane -e braket
```

CUDA-Q no Hybrid Jobs

Usar CUDA-Q no Amazon Braket Hybrid Jobs oferece um ambiente de computação flexível e sob demanda. As instâncias computacionais são executadas somente durante a duração da workload, garantindo que você pague somente pelo que usar. O Amazon Braket Hybrid Jobs também oferece uma experiência escalável. Os usuários podem começar com instâncias menores para prototipagem e teste e, em seguida, escalar para instâncias maiores, capazes de lidar com workloads maiores para experimentos completos.

O Amazon Braket Hybrid Jobs GPUs oferece suporte essencial para CUDA-Q maximizar o potencial da Amazon Braket. GPUs aceleram significativamente as simulações de programas quânticos em comparação com simuladores baseados em CPU, especialmente ao trabalhar com circuitos de alta contagem de qubits. A paralelização se torna simples quando usada no Amazon Braket Hybrid Jobs CUDA-Q. O Hybrid Jobs simplifica a distribuição da amostragem do circuito e das avaliações observáveis em vários nós computacionais. Essa paralelização perfeita das workloads CUDA-Q permite que os usuários se concentrem mais no desenvolvimento de suas workloads em vez de configurar a infraestrutura para experimentos em grande escala.

Para começar, veja o exemplo inicial CUDA-Q no Github de exemplos do Amazon Braket para CUDA-Q usar um contêiner de trabalhos híbrido fornecido pela Braket.

O trecho de código a seguir é um exemplo `hello-world` de execução de um programa CUDA-Q com o Amazon Braket Hybrid Jobs.

```
image_uri = retrieve_image(Framework.CUDAQ, AwsSession().region)

@hybrid_job(device='local:nvidia/qpp-cpu', image_uri=image_uri)
def hello_quantum():
    import cudaq

    # define the backend
    device=get_job_device_arn()
    cudaq.set_target(device.split('/')[-1])

    # define the Bell circuit
    kernel = cudaq.make_kernel()
    qubits = kernel.qalloc(2)
    kernel.h(qubits[0])
    kernel.cx(qubits[0], qubits[1])

    # sample the Bell circuit
    result = cudaq.sample(kernel, shots_count=1000)
    measurement_probabilities = dict(result.items())

    return measurement_probabilities
```

O exemplo acima simula um circuito Bell em um simulador de CPU. Este exemplo é executado localmente em seu laptop ou notebook Braket Jupyter. Por causa da configuração `local=True` , ao executar esse script, um contêiner será iniciado em seu ambiente local para executar o programa CUDA-Q para teste e depuração. Depois de concluir o teste, você pode remover o sinalizador `local=True` e continuar executando seu trabalho AWS. Para saber mais, consulte Trabalhando com o Amazon Braket Hybrid Jobs.

Se suas workloads tiverem uma alta contagem de qubits, um grande número de circuitos ou um grande número de iterações, você poderá usar recursos de computação de CPU mais poderosos especificando a configuração `instance_config`. O trecho de código a seguir mostra como configurar a configuração `instance_config` no decorador `hybrid_job`. Para obter mais informações sobre os tipos de instância compatíveis, consulte Configurar sua instância de trabalho híbrida. Para obter uma lista dos tipos de instância , consulte Tipos de instância do Amazon EC2.

```
@hybrid_job(
    device="local:nvidia/qpp-cpu",
```

```
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.c5.2xlarge"),
)
def my_job_script():
    ...
```

Para workloads mais exigentes, você pode executar suas workloads em um simulador de GPU CUDA-Q. Para habilitar um simulador de GPU, use o nome do back-end `nvidia`. O back-end `nvidia` funciona como um simulador de GPU CUDA-Q. Em seguida, selecione um tipo de instância do Amazon EC2 que ofereça suporte a uma GPU NVIDIA. O seguinte trecho de código mostra o decorador configurado em GPU `hybrid_job` configurado.

```
@hybrid_job(
    device="local:nvidia/nvidia",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"),
)
def my_job_script():
    ...
```

Amazon Braket Hybrid Jobs e NBIs suporte simulações paralelas de GPU com. CUDA-Q Você pode paralelizar a avaliação de vários observáveis ou vários circuitos para aumentar o desempenho de sua workload. Para paralelizar vários observáveis, faça as seguintes alterações em seu script de algoritmo.

Defina a opção `mgpu` do back-end `nvidia`. Isso é necessário para paralelizar os observáveis. A paralelização usa MPI para comunicação entre eles GPUs, portanto, o MPI precisa ser inicializado antes da execução e finalizado depois dela.

Em seguida, especifique o modo de execução por meio da configuração `execution=cudaq.parallel.mpi`. O seguinte trecho de código mostra essas alterações.

```
cudaq.set_target("nvidia", option="mqpu")
cudaq.mpi.initialize()
result = cudaq.observe(
    kernel, hamiltonian, shots_count=n_shots, execution=cudaq.parallel.mpi
)
cudaq.mpi.finalize()
```

No `hybrid_job` decorador, especifique um tipo de instância que hospede várias, GPUs conforme mostrado no trecho de código a seguir.

```
@hybrid_job(
    device="local:nvidia/nvidia-mqpu",
    instance_config=InstanceConfig(instanceType="ml.g4dn.12xlarge", instanceCount=1),
    image_uri=image_uri,
)
def parallel_observables_gpu_job(sagemaker_mpi_enabled=True):
    ...
```

O caderno de simulações paralelas no Amazon Braket Examples end-to-end Github fornece exemplos que demonstram como executar simulações de programas quânticos em back-ends de GPU e realizar simulações paralelas de observáveis e lotes de circuitos.

Executando suas workloads em computadores quânticos

Depois de concluir o teste do simulador, você pode fazer a transição para a execução de experimentos no QPUs. Basta mudar o alvo para uma QPU Amazon Braket, como o IQM, os dispositivos IonQ, ou Rigetti. O trecho de código a seguir ilustra como definir o destino para o dispositivo IQM Garnet. Para obter uma lista dos disponíveis QPUs, consulte o console Amazon Braket.

```
device_arn = "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet"
cudaq.set_target("braket", machine=device_arn)
```

Para obter mais informações sobre trabalhos híbridos, consulte Trabalhando com o Amazon Braket Hybrid Jobs no guia do desenvolvedor. Para saber mais sobre o CUDA-Q, consulte a documentação NVIDIA CUDA-Q.

Solução de problemas do Amazon Braket

Use as informações e as soluções de problemas nesta seção para ajudar a resolver problemas no Amazon Braket.

Nesta seção:

- AccessDeniedException
- Ocorreu um erro (ValidationException) ao chamar a CreateQuantumTask operação
- Um recurso de SDK não funciona
- O trabalho híbrido falha devido a ServiceQuotaExceededException
- Os componentes pararam de funcionar na instância do notebook
- Solução de problemas da atualização do Python 3.12
- Solução de problemas do OpenQASM

AccessDeniedException

Se você receber um AccessDeniedExceptionao ativar ou usar o Braket, provavelmente está tentando habilitar ou usar o Braket em uma região onde sua função restrita não tem acesso.

Nesses casos, entre em contato com seu AWS administrador interno para entender quais das seguintes condições se aplicam:

- Se houver restrições de perfil impedindo o acesso a uma região.
- Se o perfil que você está tentando usar tiver permissão para usar o Braket.

Se seu perfil não tiver acesso a uma determinada região ao usar o Braket, você não poderá usar dispositivos nessa região específica.

Ocorreu um erro (ValidationException) ao chamar a CreateQuantumTask operação

Se você receber um erro semelhante a: `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to`

`amazon-braket-…` , verifique se você está se referindo a uma s3_folder existente. O Braket não cria automaticamente novos buckets e prefixos do Amazon S3 para você.

Se você estiver acessando a API diretamente e recebendo um erro semelhante a: `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET`, verifique se você não está incluindo `s3://` no caminho do bucket do Amazon S3.

Um recurso de SDK não funciona

Sua versão do Python deve ser 3.10 ou superior. Para Amazon Braket Hybrid Jobs, recomendamos o Python 3.12.

Verifique se o SDK e os esquemas estão. up-to-date Para atualizar o SDK a partir do caderno ou do editor de Python, execute o seguinte comando:

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Para atualizar os esquemas, execute o seguinte comando.

```
pip install amazon-braket-schemas --upgrade
```

Se você estiver acessando o Amazon Braket de seu próprio cliente, verifique se a Região AWS está definida como uma região com suporte pelo Amazon Braket.

O trabalho híbrido falha devido a ServiceQuotaExceededException

Um trabalho híbrido executando tarefas quânticas nos simuladores Amazon Braket pode deixar de ser criado se você exceder o limite de tarefas quânticas simultâneas para o dispositivo simulador que você está almejando. Para saber mais sobre limites, consulte o tópico Cotas.

Se você estiver executando tarefas simultâneas em um dispositivo simulador em vários trabalhos híbridos da sua conta, poderá encontrar esse erro.

Para ver o número de tarefas quânticas simultâneas em um dispositivo simulador específico, use a API `search-quantum-tasks`, conforme mostrado no exemplo de código a seguir.

```
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
```

Você também pode visualizar as tarefas quânticas criadas em um dispositivo usando CloudWatch as métricas da Amazon: Braket > Por dispositivo.

Para evitar esses erros:

1. Solicite um aumento de cotas de serviço para o número de tarefas quânticas simultâneas para o dispositivo simulador. Isso só se aplica ao dispositivo SV1.
2. Gerencie as exceções de `ServiceQuotaExceeded` em seu código e tente novamente.

Os componentes pararam de funcionar na instância do notebook

Se alguns componentes do notebook pararem de funcionar, tente o seguinte:

1. Baixe todos os notebooks que você criou ou modificou em uma unidade local.
2. Interrompa a instância do notebook.
3. Exclua a instância de notebook.
4. Crie uma nova instância de notebook com um nome diferente.
5. Faça o upload dos notebooks para a nova instância.

Solução de problemas da atualização do Python 3.12

Data de vigência: 21 de janeiro de 2026

Visão geral do

A partir de 21 de janeiro de 2026, o Amazon Braket atualiza o tempo de execução do Python de 3.10 para 3.12 para todas as instâncias de notebook e imagens de contêiner gerenciadas (Base, CUDA-Qe). TensorFlow PyTorch Este guia fornece soluções para problemas comuns de compatibilidade.

Nesta seção:

- Mensagens de erro comuns
- Notebooks gerenciados com suporte
- Decorador de empregos híbrido
- Bring-Your-Own-Container (BYOC)
- Atualização da instância do Braket Notebook

Mensagens de erro comuns

Erro de incompatibilidade de versão do SDK Python

Erro:

```
RuntimeError: Python version must match between local environment and container. Client
 is running Python 3.10 locally, but container uses Python 3.12.
```

Causa: O SDK do Braket detectou que seu notebook está executando o Python 3.10, mas o contêiner Hybrid Job está executando o Python 3.12.

Solução: atualize seu notebook para o Python 3.12 ou fixe-o nos contêineres do Python 3.10.

Erro de serialização do Cloudpickle

Erro:

```
TypeError: code() argument 13 must be str, not int
```

Causa: se a validação do SDK for ignorada, o cloudpickle não conseguirá serializar o código entre o Python 3.10 e 3.12 devido a uma alteração do construtor no Python 3.12. CodeType

Solução: garanta que seu notebook e contêiner usem a mesma versão do Python.

Notebooks gerenciados com suporte

Se você estiver executando uma instância do Braket Notebook no Python 3.10 e enviando trabalhos híbridos, encontrará erros de incompatibilidade de versão porque os contêineres de trabalhos agora usam o Python 3.12 por padrão.

Você tem duas opções:

1. [Recomendado] Crie uma nova instância de notebook com Python 3.12 - consulte Braket Notebook Instance Upgrade
2. Fixe em contêineres Python 3.10 - consulte Hybrid Job Decorator

Decorador de empregos híbrido

Para usar o `@hybrid_job` decorador, a versão do Python do seu ambiente deve corresponder à versão do Python do contêiner.

Opção 1: usar contêineres Python 3.12 (recomendado)

Se você atualizou seu ambiente para o Python 3.12, ele usa a tag mais recente (comportamento padrão).

Opção 2: usar contêineres Python 3.10

Se você precisar permanecer no Python 3.10, especifique explicitamente o parâmetro no `image_uri` decorador. `@hybrid_job`

Tags de contêiner do Python 3.10:

Nome da imagem	Tag
Base	1.0-cpu-py310-ubuntu22.04
CUDA-Q	0.12.0-cpu-py310-0.12.0
PyTorch	2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow	2.14.1-gpu-py310-cu118-ubuntu20.04

O exemplo a seguir refere-se à região us-west-2.

Imagem completa URIs:

```
Base:       292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-py310-ubuntu22.04
CUDA-Q:     292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-py310-0.12.0
```

```
PyTorch:    292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-
jobs:2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow: 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-
jobs:2.14.1-gpu-py310-cu118-ubuntu20.04
```

Exemplo:

```
from braket.jobs.hybrid_job import hybrid_job
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

@hybrid_job(
    device=device_arn,
    image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-
jobs:1.0-cpu-py310-ubuntu22.04"
)
def my_job():
    pass
```

> **Note**
>
> - Os contêineres do Python 3.10 permanecerão disponíveis, mas não receberão atualizações.
> - Consulte Definir o ambiente para seu script de algoritmo.

Bring-Your-Own-Container (BYOC)

Se seu Dockerfile usa uma imagem gerenciada pelo Braket com a tag mais recente, a reconstrução após 21 de janeiro de 2026 extrairá imagens compatíveis com Python 3.12.

Para permanecer nas imagens gerenciadas do Braket compatíveis com Python 3.10, atualize seu Dockerfile:

Antes (obtém o Python 3.12 após a atualização):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:latest
```

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:latest
```

Depois (permanece no Python 3.10):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-
py310-ubuntu22.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-
py310-0.12.0
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:2.2.0-gpu-
py310-cu121-ubuntu20.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:2.14.1-
gpu-py310-cu118-ubuntu20.04
```

Atualização da instância do Braket Notebook

Siga estas etapas para atualizar para o Python 3.12:

> **Important**
>
> Antes de excluir sua instância do notebook, verifique se você baixou todos os cadernos e arquivos que deseja manter. Esses dados não podem ser recuperados após a exclusão.

1. Baixe todos os notebooks que você criou ou modificou em uma unidade local.
2. Interrompa a instância do notebook.
3. Exclua a instância de notebook.
4. Crie uma nova instância de notebook com um nome diferente.
5. Faça upload de seus cadernos para a nova instância.

Solução de problemas do OpenQASM

Esta seção fornece dicas de solução de problemas que podem ser úteis ao encontrar erros usando o OpenQASM 3.0.

Nesta seção:

- Incluir erro de declaração
- Erro não contíguo do qubits

- Erro de mistura do qubits físico com o qubits virtual
- Erro de solicitação de tipos de resultados que o qubits mede no mesmo programa
- Os limites clássicos e de registro qubit excederam o erro
- Caixa não precedida por um erro de pragma literal
- Erro de caixas textuais sem portas nativas
- Caixas textuais sem erro qubits físico
- O pragma literal não contém o erro “braket”
- Erro único qubits não pode ser indexado
- O qubits físico em um erro qubit de duas portas não está conectado
- Aviso de suporte do simulador local

Incluir erro de declaração

Atualmente, o Braket não tem um arquivo de biblioteca de portas padrão para ser incluído nos programas OpenQASM. Por exemplo, o seguinte exemplo gera um erro de análise.

```
OPENQASM 3;
include "standardlib.inc";
```

Esse código gera a mensagem de erro: `No terminal matches '"' in the current parser context, at line 2 col 17.`

Erro não contíguo do qubits

O uso não contíguo do qubits em dispositivos `requiresContiguousQubitIndices` configurados como `true` na capacidade do dispositivo resulta em um erro.

Ao executar tarefas quânticas em simuladores e IonQ, o programa a seguir aciona o erro.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

Esse código gera a mensagem de erro: `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

Erro de mistura do qubits físico com o qubits virtual

Não é permitido misturar o qubits físico com o qubits virtual no mesmo programa, resulta em um erro. O código a seguir gera o erro.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

Esse código gera a mensagem de erro: `[line 4] mixes physical qubits and qubits registers.`

Erro de solicitação de tipos de resultados que o qubits mede no mesmo programa

Solicitar tipos de resultados que o qubits mede explicitamente no mesmo programa resulta em um erro. O código a seguir gera o erro.

```
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])
```

Esse código gera a mensagem de erro: `Qubits should not be explicitly measured when result types are requested.`

Os limites clássicos e de registro qubit excederam o erro

Somente um registro clássico e um registro qubit são permitidos. O código a seguir gera o erro.

```
OPENQASM 3;
```

```
qubit[2] q0;
qubit[2] q1;
```

Esse código gera a mensagem de erro: `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

Caixa não precedida por um erro de pragma literal

Todas as caixas devem ser precedidas por um pragma literal. O código a seguir gera o erro.

```
box{
rx(0.5) $0;
}
```

Esse código gera a mensagem de erro: `In verbatim boxes, native gates are required. x is not a device native gate.`

Erro de caixas textuais sem portas nativas

As caixas textuais devem ter portas qubits nativas e físicas. O código a seguir gera o erro de portas nativas.

```
#pragma braket verbatim
box{
x $0;
}
```

Esse código gera a mensagem de erro: `In verbatim boxes, native gates are required. x is not a device native gate.`

Caixas textuais sem erro qubits físico

As caixas textuais devem ter qubits físicas. O código a seguir gera o erro qubits físico ausente.

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
```

```
}
```

Esse código gera a mensagem de erro: `Physical qubits are required in verbatim box.`

O pragma literal não contém o erro “braket”

Você deve incluir “braket” no pragma literal. O código a seguir gera o erro.

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

Esse código gera a mensagem de erro: `You must include “braket” in the verbatim pragma`

Erro único qubits não pode ser indexado

O único qubits não pode ser indexado. O código a seguir gera o erro.

```
OPENQASM 3;

qubit q;
h q[0];
```

Esse código gera o erro: `[line 4] single qubit cannot be indexed.`

No entanto, matrizes únicas qubit podem ser indexadas da seguinte forma:

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

O qubits físico em um erro qubit de duas portas não está conectado

Para usar o qubits físico, primeiro confirme se o dispositivo usa o qubits físico verificando `device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits` e, em seguida, verifique o gráfico de conectividade marcando `device.properties.paradigm.connectivity.connectivityGraph` ou `device.properties.paradigm.connectivity.fullyConnected`.

```
OPENQASM 3;

cnot $0, $14;
```

Esse código gera a mensagem de erro: `[line 3] has disconnected qubits 0 and 14`

Aviso de suporte do simulador local

`LocalSimulator`Ele suporta recursos avançados no OpenQASM que podem não estar disponíveis em QPUs simuladores sob demanda. Se o seu programa contiver recursos de linguagem específicos somente para o `LocalSimulator`, conforme mostrado no exemplo a seguir, você receberá um aviso.

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

Esse código gera o aviso: `Este programa usa recursos da linguagem OpenQASM suportados somente no. LocalSimulator Alguns desses recursos podem não ser compatíveis com simuladores sob demanda QPUs ou sob demanda.

Para obter mais informações sobre os recursos com suporte do OpenQASM, explore a página Suporte avançado de recursos para o OpenQASM no Simulador Local.

Segurança no Amazon Braket

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de data centers e arquiteturas de rede criados para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O Modelo de Responsabilidade Compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços no Nuvem AWS. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam ao Amazon Braket, AWS consulte Serviços no escopo do programa de conformidade Serviços no escopo AWS de conformidade.
- Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da sua empresa e as leis e normas aplicáveis.

Esta documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Braket. Os tópicos a seguir mostram como configurar o Braket para atender aos seus objetivos de segurança e conformidade. Você também aprende a usar outros AWS serviços que ajudam a monitorar e proteger seus recursos do Braket.

Nesta seção:

- Responsabilidade compartilhada pela segurança
- Proteção de dados
- Retenção de dados
- Gerenciamento do acesso ao Amazon Braket
- Perfil vinculado ao serviço do Amazon Braket
- Validação de compatibilidade para o Amazon Braket
- Segurança da infraestrutura no Amazon Braket
- Segurança dos fornecedores de hardware Amazon Braket
- Endpoints da Amazon VPC para o Amazon Braket

Responsabilidade compartilhada pela segurança

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem.

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que é executada Serviços da AWS no Nuvem AWS. AWS também fornece serviços que você pode usar com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos Programas de conformidade da AWS. Para conhecer os programas de conformidade que se aplicam ao Amazon Braket, consulte Serviços AWS abrangidos pelo Programa de Conformidade.
- Segurança na nuvem — Você é responsável por manter o controle sobre o conteúdo hospedado nessa AWS infraestrutura. Esse conteúdo inclui as tarefas de configuração e gerenciamento de segurança do Serviços da AWS que você usa.

Proteção de dados

O modelo de responsabilidade AWS compartilhada de se aplica à proteção de dados no Amazon Braket. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para saber mais sobre a privacidade de dados, consulte as Data Privacy FAQ. Para saber mais sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.

- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para saber mais sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com o Amazon Braket ou Serviços da AWS outro usando o console, a API AWS CLI ou. AWS SDKs Quaisquer dados inseridos em tags ou em campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Retenção de dados

Depois de 90 dias, o Amazon Braket remove automaticamente todas as IDs tarefas quânticas e outros metadados associados às suas tarefas quânticas. Como resultado dessa política de retenção de dados, essas tarefas e resultados não podem mais ser recuperados pela pesquisa no console do Amazon Braket, embora permaneçam armazenados em seu bucket do S3.

Se você precisar acessar tarefas e resultados quânticos históricos armazenados em seu bucket do S3 por mais de 90 dias, deverá manter um registro separado do ID da tarefa e de outros metadados associados a esses dados. Certifique-se de salvar as informações antes de 90 dias. Você pode usar essas informações salvas para recuperar os dados históricos.

Gerenciamento do acesso ao Amazon Braket

Este capítulo descreve as permissões necessárias para executar o Amazon Braket ou para restringir o acesso de usuários e perfis específicos. Você pode conceder ou negar as permissões necessárias

para qualquer usuário ou perfil em sua conta. Para fazer isso, anexe a política apropriada do Amazon Braket a esse usuário ou perfil em sua conta, conforme descrito nas seções a seguir.

Como pré-requisito, você deve habilitar o Amazon Braket. Para habilitar o Braket, certifique-se de fazer login como usuário ou função que tenha (1) permissões de administrador ou (2) tenha a AmazonBraketFullAccesspolítica atribuída e tenha permissões para criar buckets do Amazon Simple Storage Service (Amazon S3).

Nesta seção:

- Recursos do Amazon Braket
- Cadernos e perfis
- AWS políticas gerenciadas para o Amazon Braket
- Restringir o acesso do usuário a determinados dispositivos
- Restrinja o acesso do usuário a determinadas instâncias do caderno
- Restringir o acesso do usuário a determinados buckets do S3

Recursos do Amazon Braket

O Braket cria um tipo de recurso: o recurso de tarefa quântica. O nome do AWS recurso (ARN) para esse tipo de recurso é o seguinte:

- Nome do recurso: AWS::Service::Braket
- Regex de ARN: arn: $ {Partition} :braket: $ {Region} :$ {Account} :quantum-task/$ {} RandomId

Cadernos e perfis

Você pode usar o tipo de recurso caderno no Braket. Um notebook é um recurso de SageMaker IA da Amazon que o Braket pode compartilhar. Para usar um caderno com o Braket, você deve especificar um perfil do IAM com um nome que comece com `AmazonBraketServiceSageMakerNotebook`.

Para criar um caderno, você deve usar um perfil com permissões de administrador ou que tenha a seguinte política embutida anexada a ela.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "CreateTheRole",
      "Effect": "Allow",
      "Action": "iam:CreateRole",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
    },
    {
      "Sid": "CreateThePolicy",
      "Effect": "Allow",
      "Action": "iam:CreatePolicy",
      "Resource": [
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
      ]
    },
    {
      "Sid": "AttachTheRolePolicy",
      "Effect": "Allow",
      "Action": "iam:AttachRolePolicy",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
      "Condition": {
        "ArnLike": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/AmazonBraketFullAccess",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
          ]
        }
      }
    }
  ]
```

```
}
```

Para criar o perfil, siga as etapas fornecidas na página Criar um caderno ou peça ao administrador que a crie para você. Certifique-se de que a AmazonBraketFullAccesspolítica esteja anexada.

Depois de criar o perfil, você pode reutilizá-lo em todos os cadernos que você lançar no futuro.

AWS políticas gerenciadas para o Amazon Braket

Uma política AWS gerenciada é uma política autônoma criada e administrada por AWS. AWS as políticas gerenciadas são projetadas para fornecer permissões para muitos casos de uso comuns, para que você possa começar a atribuir permissões a usuários, grupos e funções.

Lembre-se de que as políticas AWS gerenciadas podem não conceder permissões de privilégio mínimo para seus casos de uso específicos porque elas estão disponíveis para uso de todos os AWS clientes. Recomendamos que você reduza ainda mais as permissões definindo as políticas gerenciadas pelo cliente que são específicas para seus casos de uso.

Você não pode alterar as permissões definidas nas políticas AWS gerenciadas. Se AWS atualizar as permissões definidas em uma política AWS gerenciada, a atualização afetará todas as identidades principais (usuários, grupos e funções) às quais a política está anexada. AWS é mais provável que atualize uma política AWS gerenciada quando uma nova AWS service (Serviço da AWS) é lançada ou novas operações de API são disponibilizadas para serviços existentes.

Para saber mais, consulte AWS Políticas gerenciadas pela no Guia do usuário do IAM.

Tópicos

- AWS política gerenciada: AmazonBraketFullAccess
- AWS política gerenciada: AmazonBraketJobsExecutionPolicy
- AWS política gerenciada: AmazonBraketServiceRolePolicy
- Atualizações do Amazon Braket para políticas gerenciadas AWS

AWS política gerenciada: AmazonBraketFullAccess

A AmazonBraketFullAccesspolítica concede permissões para as operações do Amazon Braket, incluindo permissões para essas tarefas:

- Baixe contêineres do Amazon Elastic Container Registry — Para ler e baixar imagens de contêineres que são usadas para o recurso Amazon Braket Hybrid Jobs. Os contêineres devem estar em conformidade com o formato "arn:aws:ecr:::repository/amazon-braket".
- Mantenha AWS CloudTrail registros — para todas as ações de descrever, obter e listar, além de iniciar e interromper consultas, testar filtros de métricas e filtrar eventos de registro. O arquivo de AWS CloudTrail log contém um registro de todas as atividades do Amazon API Braket que ocorrem em sua conta.
- Utilize perfis para controlar recursos — Para criar um perfil vinculado ao serviço em sua conta. A função vinculada ao serviço tem acesso aos AWS recursos em seu nome. Ele pode ser usado somente pelo serviço Amazon Braket. Além disso, passar funções do IAM para o Amazon `CreateJob` API Braket, criar uma função e anexar uma política com escopo AmazonBraketFullAccess definido à função.
- Crie grupos de log, eventos de log e grupos de log de consulta para manter arquivos de log de uso da sua conta — Para criar, armazenar e visualizar informações de registro sobre o uso do Amazon Braket em sua conta. Métricas de consulta em grupos de logs do hybrid jobs. Abrange o caminho correto do Braket e permite colocar dados de log. Insira dados métricos CloudWatch.
- Crie e armazene dados em buckets do Amazon S3 e liste todos os buckets – Para criar buckets do S3, liste os buckets do S3 em sua conta e insira e obtenha objetos de qualquer bucket em sua conta cujo nome comece com amazon-braket-. Essas permissões são necessárias para que o Braket coloque arquivos contendo resultados de tarefas quânticas processadas no bucket e os recupere do bucket.
- Passar os perfis do IAM — Para passar os perfis do IAM para o `CreateJob` da API.
- Amazon SageMaker AI Notebook — Para criar e gerenciar instâncias de SageMaker notebook com o escopo do recurso “arn:aws:sagemaker: ::notebook-instance/amazon-braket-”.
- Valide cotas de serviço — Para criar notebooks de SageMaker IA e trabalhos do Amazon Braket Hybrid, sua contagem de recursos não pode exceder as cotas da sua conta.
- Veja os preços dos produtos — Analise e planeje os custos de hardware quântico antes de enviar suas workloads.

Para ver as permissões dessa política, consulte AmazonBraketFullAccessna AWS Managed Policy Reference.

AWS política gerenciada: AmazonBraketJobsExecutionPolicy

A AmazonBraketJobsExecutionPolicypolítica concede permissões para funções de execução usadas no Amazon Braket Hybrid Jobs da seguinte forma:

- Baixe contêineres do Amazon Elastic Container Registry - Permissões para ler e baixar imagens de contêineres que são usadas para o recurso Amazon Braket Hybrid Jobs. Os contêineres devem estar em conformidade com o formato "arn:aws:ecr:*:*:repository/amazon-braket*".
- Crie grupos de log e eventos de log e consulte grupos de log para manter arquivos de log de uso de sua conta — Crie, armazene e visualize informações de registro sobre o uso do Amazon Braket em sua conta. Métricas de consulta em grupos de logs do hybrid jobs. Abrange o caminho correto do Braket e permite colocar dados de log. Insira dados métricos CloudWatch.
- Armazene dados em buckets do Amazon S3 — Liste os buckets do S3 em sua conta, coloque objetos e obtenha objetos de qualquer bucket em sua conta que comece com amazon-braket- em seu nome. Essas permissões são necessárias para que o Braket coloque arquivos contendo resultados de tarefas quânticas processadas no bucket e os recupere do bucket.
- Passe as funções do IAM — Passando as funções do IAM para CreateJob API o. Os perfis devem estar no formato arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*.

Para ver as permissões dessa política, consulte AmazonBraketJobsExecutionPolicyna AWS Managed Policy Reference.

AWS política gerenciada: AmazonBraketServiceRolePolicy

A AmazonBraketServiceRolePolicypolítica concede permissões para as operações do Amazon Braket, incluindo permissões para essas tarefas:

- Amazon S3 — permissões para listar os buckets em sua conta e colocar objetos e obter objetos de qualquer bucket em sua conta com um nome que comece com `amazon-braket-`.
- Amazon CloudWatch Logs — permissões para listar e criar grupos de log, criar os fluxos de log associados e colocar eventos no grupo de log criado para o Amazon Braket.

Para obter mais informações sobre funções vinculadas a serviços, consulte a página sobre funções vinculadas a serviços do Amazon Braket.

Para ver as permissões dessa política, consulte AmazonBraketServiceRolePolicyna AWS Managed Policy Reference.

Atualizações do Amazon Braket para políticas gerenciadas AWS

A tabela a seguir fornece detalhes sobre as atualizações das políticas AWS gerenciadas do Amazon Braket a partir do momento em que esse serviço começou a rastrear essas alterações.

Alteração	Descrição	Data
AmazonBraketServiceRolePolicy- Política de gerenciamento de recursos	O escopo da condição “aws:ResourceAccount": “${aws:PrincipalAccount}” foi adicionado ao Amazon S3 e CloudWatch registra ações.	11 de julho de 2025
AmazonBraketFullAccess- Política de acesso total para Braket	Foi adicionada a ação “pricing: GetProducts”.	14 de abril de 2025
AmazonBraketFullAccess- Política de acesso total para Braket	O escopo da condição “aws: ResourceAccount “: “$ {aws:PrincipalAccount}” foi adicionado às ações do S3.	07 de março de 2025
AmazonBraketFullAccess- Política de acesso total para Braket	Foram adicionadas as ações servicquotas: GetServiceQuota e cloudwatch:. GetMetricData	24 de março de 2023
AmazonBraketFullAccess- Política de acesso total para Braket	Foram adicionadas as ListAllMy Buckets permissões s3: para visualizar e inspecionar os buckets Amazon S3 usados.	31 de março de 2022
AmazonBraketFullAccess- Política de acesso total para Braket	Objetivo ajustado do Braket: PassRole permissões AmazonBraketFullAccess para incluir o `service-role/` caminho.	29 de novembro de 2021
AmazonBraketJobsExecutionPolicy- Política de execução de trabalhos	Braket atualizou o ARN do perfil de execução de trabalhos	29 de novembro de 2021

Alteração	Descrição	Data
híbridos para Amazon Braket Hybrid Jobs	híbridos para incluir o caminho `service-role/` .	
O Braket começou a monitorar alterações	A Braket começou a monitorar as mudanças em suas políticas AWS gerenciadas.	29 de novembro de 2021

Restringir o acesso do usuário a determinados dispositivos

Para restringir o acesso do usuário a determinados dispositivos Braket, você pode adicionar uma política de negação de permissões a um perfil específico do IAM.

As seguintes ações podem ser restringidas:

- `CreateQuantumTask`- negar a criação de tarefas quânticas em dispositivos específicos.
- `CreateJob`- negar a criação de empregos híbridos em dispositivos específicos.
- `GetDevice`- negar a obtenção de detalhes de dispositivos especificados.

O exemplo a seguir restringe o acesso a todos QPUs para o. Conta da AWS `123456789012`

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ],
    "Condition": {
             "StringEquals": {
```

```
                "aws:PrincipalAccount": "123456789012"
            }
        }
    }
  ]
}
```

> **Note**
>
> Exclua a `braket:GetDevice` Ação da política para permitir que o usuário tenha acesso de leitura às propriedades do dispositivo, como disponibilidade do dispositivo, dados de calibração e preços, por meio do console Braket.

Para adaptar esse código, substitua o número do Amazon recurso (ARN) do dispositivo restrito pela string mostrada no exemplo anterior. Essa string fornece o valor do recurso. No Braket, um dispositivo representa um QPU ou simulador que você pode chamar para executar tarefas quânticas. Os dispositivos disponíveis estão listados na página Dispositivos. Há dois esquemas usados para especificar o acesso a esses dispositivos:

- `arn:aws:braket:<region>:*:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:*:device/quantum-simulator/<provider>/<device_id>`

Aqui estão alguns exemplos de vários tipos de acesso a dispositivos

- Para selecionar tudo QPUs em todas as regiões: `arn:aws:braket:*:*:device/qpu/*`
- Para selecionar tudo QPUs SOMENTE na região us-west-2: `arn:aws:braket:us-west-2:*:device/qpu/*`
- Da mesma forma, para selecionar tudo SOMENTE QPUs na região us-west-2 (já que os dispositivos são um recurso de serviço, não um recurso do cliente): `arn:aws:braket:us-west-2:*:device/qpu/*`
- Para restringir o acesso a todos os dispositivos de simulador sob demanda: `arn:aws:braket:*:*:device/quantum-simulator/*`
- Para restringir o acesso a dispositivos de um determinado provedor (por exemplo, a Rigetti QPU dispositivos): `arn:aws:braket:*:*:device/qpu/rigetti/*`

- Para restringir o acesso ao TN1 dispositivo: `arn:aws:braket:*:*:device/quantum-simulator/amazon/tn1`
- Para restringir o acesso a todas as `Create` ações: `braket:Create*`

Restrinja o acesso do usuário a determinadas instâncias do caderno

Para restringir o acesso de determinados usuários a instâncias específicas do caderno Braket, você pode adicionar uma política de negação de permissões a um perfil, usuário ou grupo específico.

O exemplo a seguir usa variáveis de política para restringir com eficiência as permissões para iniciar, interromper e acessar instâncias específicas do notebook no Conta da AWS `123456789012`, que é nomeado de acordo com o usuário que deveria ter acesso (por exemplo, o usuário `Alice` teria acesso a uma instância do notebook chamada`amazon-braket-Alice`).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCreateDeleteUpdateNotebookInstances",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyDescribeStartStopNotebookInstances",
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance"
      ],
```

```
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Sid": "DenyNotebookInstanceUrl",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

Restringir o acesso do usuário a determinados buckets do S3

Para restringir o acesso de determinados usuários a buckets específicos do Amazon S3, você pode adicionar uma política de negação a um perfil, usuário ou grupo específico.

O exemplo a seguir restringe as permissões para recuperar e colocar objetos em um S3 bucket específico (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) e também restringe a listagem desses objetos.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "s3:ListBucket"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice"
      ]
```

```
    },
    {
      "Effect": "Deny",
      "Action": [
        "s3:GetObject"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice/*"
      ]
    }
  ]
}
```

Para restringir o acesso ao bucket para uma determinada instância do caderno, você pode adicionar a política anterior ao perfil de execução do caderno.

Perfil vinculado ao serviço do Amazon Braket

Ao criar um analisador para ativar o Amazon Braket, um perfil vinculado ao serviço será criado em sua conta.

Um perfil vinculado ao serviço é um tipo exclusivo de perfil do IAM vinculado diretamente ao Amazon Braket. O perfil vinculado ao serviço é predefinido pelo Amazon Braket e inclui todas as permissões que o serviço requer para chamar outros produtos da Serviços da AWS em seu nome.

Um perfil vinculado ao serviço facilita a configuração do Amazon Braket porque você não precisa adicionar as permissões necessárias manualmente. O Amazon Braket define as permissões dos perfis vinculados a serviço. A menos que você altere essas definições, somente o Amazon Braket pode assumir seus perfis. As permissões definidas incluem a política de confiança e a política de permissões. A política de permissões não pode ser anexada a nenhuma outra entidade do IAM.

O perfil vinculado ao serviço que o Amazon Braket configura faz parte da capacidade de perfis vinculados ao serviço AWS Identity and Access Management (IAM). Para obter informações sobre outros serviços Serviços da AWS que oferecem suporte a perfis vinculados ao serviço, consulte Serviços AWS compatíveis com o IAM e procure os serviços que têm Sim na coluna Perfil vinculado ao serviço. Escolha um Sim com um link para visualizar a documentação da função vinculada a esse serviço.

Para obter mais informações sobre a política AWS gerenciada para perfis vinculados a serviços, consulte AmazonBraketServiceRolePolicy.

Validação de compatibilidade para o Amazon Braket

> **Note**
>
> Os relatórios de compatibilidade AWS não abrangem QPUs de fornecedores de hardware terceirizados que podem optar por passar por suas próprias auditorias independentes.

Para saber se um AWS service (Serviço da AWS) está no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo por programa de conformidade e selecione o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de Conformidade da AWS.

É possível baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Serviços da AWS é determinada pela confidencialidade dos seus dados, pelos objetivos de conformidade da sua empresa e pelos regulamentos e leis aplicáveis. Para ter mais informações sobre sua responsabilidade pela conformidade ao usar Serviços da AWS, consulte a documentação da AWS sobre segurança.

Segurança da infraestrutura no Amazon Braket

Como um serviço gerenciado, o Amazon Braket é protegido pela segurança da rede global da AWS. Para obter informações sobre serviços de segurança da AWS e como a AWS protege a infraestrutura, consulte Segurança na Nuvem AWS. Para projetar seu ambiente da AWS usando as práticas recomendadas de segurança da infraestrutura, consulte Proteção de Infraestrutura em Pilar de Segurança: AWS Estrutura bem arquitetada.

Você usa chamadas de API publicadas pela AWS para acessar o Amazon Braket por meio da rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Você chamar essas operações de API de qualquer local da rede, mas o Amazon Braket não é suportado por políticas de acesso baseadas em recursos, que podem incluir restrições com base no endereço IP de origem. Você também pode usar políticas do Amazon Braket para controlar o acesso de Amazon Virtual Private Cloud (Amazon VPC) endpoints ou de VPCs específicas. Efetivamente, isso isola o acesso à rede para um determinado recurso do Amazon Braket apenas da VPC específica dentro da rede da AWS.

Segurança dos fornecedores de hardware Amazon Braket

As QPUs no Amazon Braket são hospedadas por fornecedores terceirizados de hardware. Quando você executa sua tarefa quântica em uma QPU, o Amazon Braket usa o DevicEarn como identificador ao enviar o circuito para a QPU especificada para processamento.

Se você usar o Amazon Braket para acessar o hardware de computação quântica operado por um dos fornecedores de hardware terceirizados, seu circuito e seus dados associados serão processados por fornecedores de hardware fora das instalações operadas pela. AWS Informações sobre a localização física e a região AWS em que cada QPU está disponível podem ser encontradas na seção Detalhes do dispositivo do console Amazon Braket.

Seu conteúdo é anonimizado. Somente o conteúdo necessário para processar o circuito é enviado a terceiros. As informações da Conta da AWS não são transmitidas a terceiros.

Os dados são criptografados em trânsito e em repouso. Os dados são descriptografados somente para processamento. Os fornecedores terceirizados do Amazon Braket não têm permissão para armazenar ou usar seu conteúdo para outros fins que não sejam processar seu circuito. Quando o circuito é concluído, os resultados são devolvidos ao Amazon Braket e armazenados em seu bucket do S3.

A segurança dos fornecedores terceirizados de hardware quântico da Amazon Braket é auditada periodicamente para garantir que os padrões de segurança de rede, controle de acesso, proteção de dados e segurança física sejam atendidos.

Endpoints da Amazon VPC para o Amazon Braket

Você pode estabelecer uma conexão privada entre sua VPC e o Amazon Braket criando uma interface de endpoint da VPC. Os endpoints de interface são alimentados por AWS PrivateLinkuma tecnologia que permite o acesso ao Braket APIs sem um gateway de internet, dispositivo NAT, conexão VPN ou conexão. Direct Connect As instâncias na sua VPC não precisam de endereços IP públicos para se comunicar com o Braket. APIs

Cada endpoint de interface é representado por uma ou mais Interfaces de Rede Elástica nas sub-redes.

Com isso AWS PrivateLink, o tráfego entre sua VPC e o Braket não sai da Amazon rede, o que aumenta a segurança dos dados que você compartilha com aplicativos baseados em nuvem, pois reduz a exposição de seus dados à Internet pública. Para obter mais informações, consulte Acessar um AWS serviço usando uma interface VPC endpoint no Guia do usuário da Amazon VPC.

Nesta seção:

- Considerações sobre endpoints da VPC do Amazon Braket
- Configure o Braket e PrivateLink
- Informações adicionais sobre como criar um endpoint
- Controle o acesso com as políticas de endpoint da VPC Amazon

Considerações sobre endpoints da VPC do Amazon Braket

Antes de configurar uma interface de endpoint da VPC para o Braket, certifique-se de revisar os pré-requisitos de interface de endpoint no Guia do Usuário da Amazon VPC.

O Braket permite fazer chamadas para todas as suas ações de API a partir da sua VPC.

Por padrão, o acesso total ao Braket é permitido através do endpoint da VPC. Você pode controlar o acesso se especificar as políticas de endpoint da VPC. Para obter mais informações, consulte Controlar o acesso aos endpoints da VPC usando políticas de endpoint no Guia do usuário da Amazon VPC.

Configure o Braket e PrivateLink

Para usar AWS PrivateLink com o Amazon Braket, você deve criar um endpoint da Amazon Virtual Private Cloud (Amazon VPC) como uma interface e, em seguida, conectar-se ao endpoint por meio do serviço Amazon Braket. API

Aqui estão as etapas gerais desse processo, que são explicadas em detalhes nas seções posteriores.

- Configure e lance uma Amazon VPC para hospedar seus AWS recursos. Se você já tiver uma VPC, ignore esta etapa.

- Criar uma interface de endpoint da VPC para o Amazon Braket
- Conecte e execute tarefas quânticas do Braket em seu endpoint

Etapa 1: executar uma VPC da Amazon, se necessário

Lembre-se de que você pode pular essa etapa se sua conta já tiver uma VPC em operação.

Uma VPC controla as configurações da sua rede, como o intervalo de endereços IP, sub-redes, tabelas de rotas e gateways de rede. Basicamente, você está lançando seus AWS recursos em uma rede virtual personalizada. Para obter mais informações sobre VPCs, consulte o Guia do usuário da Amazon VPC.

Abra o console da Amazon VPC e crie uma nova VPC com sub-redes, grupos de segurança e gateways de rede.

Etapa 2: criar uma interface de endpoint da VPC para o Braket

Você pode criar um VPC endpoint para o serviço Braket usando o console Amazon VPC ou o (). AWS Command Line Interface AWS CLI Para obter mais informações, consulte Crie um endpoint da VPC no Guia do usuário do Amazon VPC.

Para criar um endpoint da VPC no console, abra o console Amazon VPC, abra a página Endpoints e continue criando o novo endpoint. Anote o ID do endpoint para referência posterior. É necessário como parte da bandeira `–endpoint-url` quando você está fazendo determinadas chamadas para o API Braket.

Crie um endpoint da VPC para o Braket usando o seguinte nome de serviço:

- `com.amazonaws.substitute_your_region.braket`

Para obter mais informações, consulte Acessar um AWS serviço usando uma interface VPC endpoint no Guia do usuário da Amazon VPC.

Etapa 3: Conecte e execute tarefas quânticas do Braket por meio de seu endpoint

Depois de criar um endpoint da VPC, você pode usar os seguintes comandos da CLI de exemplo que usam o parâmetro `endpoint-url` para especificar interfaces de endpoints da API ou o runtime do exemplo:

```
aws braket search-quantum-tasks --endpoint-url
 VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

Se você habilitar nomes de hosts DNS privados para seu endpoint da VPC, não precisará especificar a URL do endpoint. Em vez disso, o nome do host API DNS do Amazon Braket, que a CLI e o SDK do Braket usam por padrão, é resolvido para seu endpoint da VPC. Possui o formato mostrado no exemplo a seguir:

```
https://braket.substituteYourRegionHere.amazonaws.com
```

A postagem do blog chamada Acesso direto aos notebooks Amazon SageMaker AI da Amazon VPC usando AWS PrivateLink um endpoint fornece um exemplo de como configurar um endpoint para fazer conexões seguras com notebooks, que são semelhantes SageMaker aos notebooks Braket. Amazon

Se você estiver seguindo as etapas na postagem do blog, lembre-se de substituir o nome AmazonBraket por Amazon SageMaker AI. Em Nome do serviço, insira `com.amazonaws.us-east-1.braket` ou substitua seu Região da AWS nome correto nessa string, se sua região não for us-east-1.

Informações adicionais sobre como criar um endpoint

- Para obter informações sobre como criar uma VPC com sub-redes privadas, consulte Criar uma VPC com sub-redes privadas.
- Para obter informações sobre como criar e configurar um endpoint usando o console da Amazon VPC ou o AWS CLI, consulte Criar um endpoint de VPC no Guia do usuário da Amazon VPC.
- Para obter informações sobre como criar e configurar um endpoint usando CloudFormation, consulte o VPCEndpoint recurso AWS: :EC2:: no Guia do usuário.CloudFormation

Controle o acesso com as políticas de endpoint da VPC Amazon

Para controlar o acesso de conectividade ao Amazon Braket, você pode anexar uma política de endpoint do AWS Identity and Access Management (IAM) ao criar um endpoint da VPC Amazon. Essa política especifica as seguintes informações:

- A entidade principal (usuário ou perfil) que pode executar ações.
- As ações que podem ser realizadas.

- Os recursos aos quais as ações podem ser aplicadas.

Para obter mais informações, consulte Controlar o acesso aos endpoints da VPC usando políticas de endpoint no Guia do usuário da Amazon VPC.

Exemplo: Política de endpoint VPC para ações Braket

O exemplo a seguir mostra uma política de endpoint para o Braket. Quando associada a um endpoint, esta política concede acesso às ações do Braket listadas para todas as entidades principais em todos os recursos.

```
{
 "Statement":[
 {
   "Principal":"*",
   "Effect":"Allow",
   "Action":[
     “braket:action-1",
     “braket:action-2",
     “braket:action-3”
     ],
   "Resource":"*"
   }
  ]
}
```

É possível criar regras complexas do IAM anexando várias políticas de endpoint. Para ter mais informações e exemplos, consulte:

- Políticas de endpoint da Amazon Virtual Private Cloud para Step Functions
- Criação de permissões granulares do IAM para usuários não administradores
- Controle o acesso aos VPC endpoints usando políticas de endpoint

Registro em log e monitoramento

Depois de enviar uma tarefa quântica por meio do serviço Amazon Braket, você pode monitorar de perto o status e a progressão dessa tarefa por meio do Amazon Braket SDK e do console. Isso fornece uma interface centralizada para rastrear a implementação de suas workloads, identificar possíveis gargalos ou problemas e tomar as medidas apropriadas para otimizar o desempenho e a confiabilidade de seus aplicativos quânticos. Quando a tarefa quântica é concluída, o Braket salva os resultados em sua localização especificada no Amazon S3. O tempo de conclusão das tarefas quânticas pode variar, especialmente para aquelas executadas em dispositivos de unidade de processamento quântico (QPU). Isso se deve em grande parte ao tamanho da fila de execução, pois os recursos de hardware quântico são compartilhados entre vários usuários.

Lista de tipos de status:

- `CREATED`— O Amazon Braket recebeu sua tarefa quântica.
- `QUEUED`— O Amazon Braket processou sua tarefa quântica e agora está esperando para ser executada no dispositivo.
- `RUNNING`— Sua tarefa quântica está sendo executada em um QPU ou simulador sob demanda.
- `COMPLETED`— Sua tarefa quântica terminou de ser executada no QPU ou no simulador sob demanda.
- `FAILED`— Sua tarefa quântica tentou ser executada e falhou. Dependendo do motivo pelo qual sua tarefa quântica falhou, tente enviar sua tarefa quântica novamente.
- `CANCELLED`— Você cancelou a tarefa quântica. A tarefa quântica não foi executada.

Nesta seção:

- Rastreamento de tarefas quânticas a partir do Amazon Braket SDK
- Monitoramento de tarefas quânticas por meio do console Amazon Braket
- Marcação de recursos do Amazon Braket
- Monitorando suas tarefas quânticas com EventBridge
- Monitorando suas métricas com CloudWatch
- Registrando suas tarefas quânticas com CloudTrail
- Registro em log avançado com o Amazon Braket

Rastreamento de tarefas quânticas a partir do Amazon Braket SDK

O comando `device.run(…)` define uma tarefa quântica com um ID de tarefa quântica exclusivo. Você pode consultar e rastrear o status com `task.state()`, conforme mostrado no exemplo a seguir.

Nota: `task = device.run()` é uma operação assíncrona, o que significa que você pode continuar trabalhando enquanto o sistema processa sua tarefa quântica em segundo plano.

Recuperar um resultado

Quando você chama `task.result()`, o SDK começa a pesquisar o Amazon Braket para ver se a tarefa quântica foi concluída. O SDK usa os parâmetros de pesquisa que você definiu em `.run()`. Depois que a tarefa quântica é concluída, o SDK recupera o resultado do bucket S3 e o retorna como um objeto `QuantumTaskResult`.

```
# create a circuit, specify the device and run the circuit
circ = Circuit().rx(0, 0.15).ry(1, 0.2).cnot(0,2)
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
task = device.run(circ, s3_location, shots=1000)

# get ID and status of submitted task
task_id = task.id
status = task.state()
print('ID of task:', task_id)
print('Status of task:', status)
# wait for job to complete
while status != 'COMPLETED':
    status = task.state()
    print('Status:', status)
```

```
ID of task:
arn:aws:braket:us-west-2:123412341234:quantum-task/b68ae94b-1547-4d1d-aa92-1500b82c300d
Status of task: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: RUNNING
```

```
Status: RUNNING
Status: COMPLETED
```

Cancelar uma tarefa quântica

Para cancelar uma tarefa quântica, chame o método `cancel()`, conforme mostrado no exemplo a seguir.

```
# cancel quantum task
task.cancel()
status = task.state()
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

Verifique os metadados

Você pode verificar os metadados da tarefa quântica concluída, conforme mostrado no exemplo a seguir.

```
# 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
```

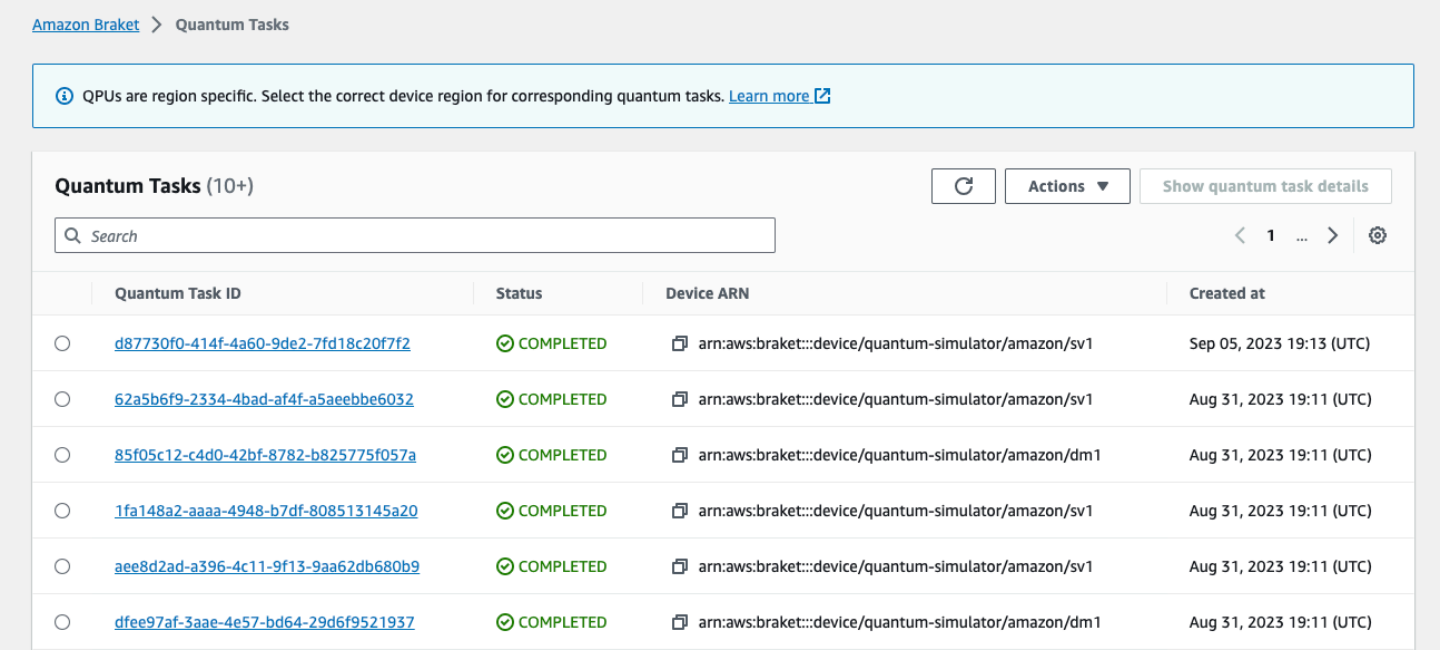
Recupere uma tarefa ou resultado quântico

Se o kernel morrer após o envio da tarefa quântica ou se você fechar o caderno ou o computador, você poderá reconstruir o objeto `task` com seu ARN (ID de tarefa quântica) exclusivo. Em seguida, você pode chamar `task.result()` para obter o resultado do bucket do S3 em que ele está armazenado.

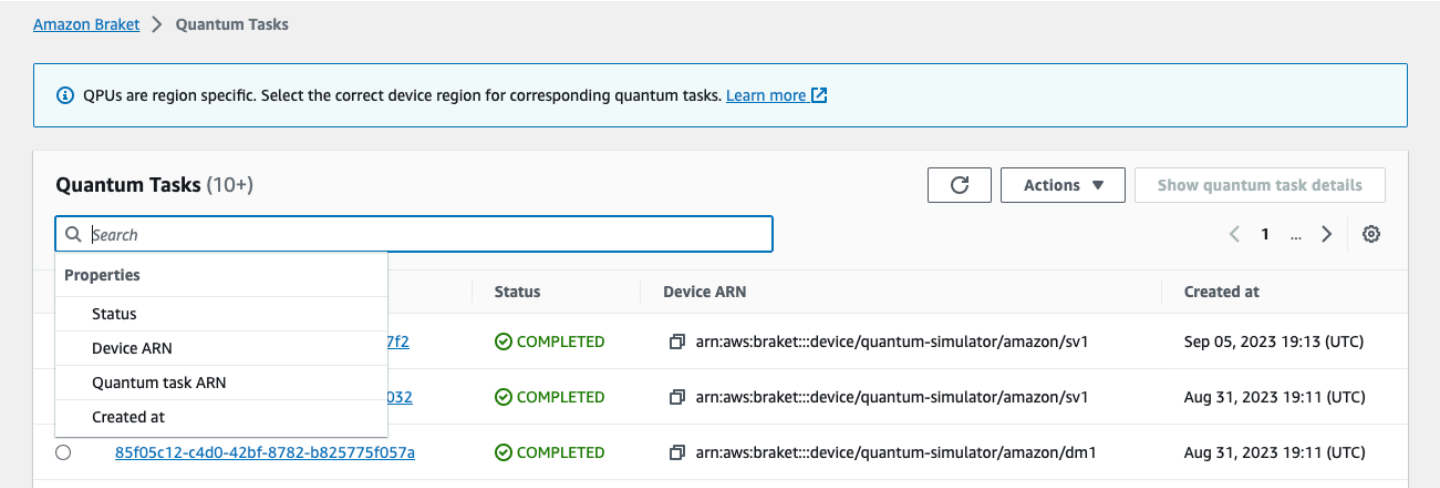
```
from braket.aws import AwsSession, AwsQuantumTask

# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Monitoramento de tarefas quânticas por meio do console Amazon Braket

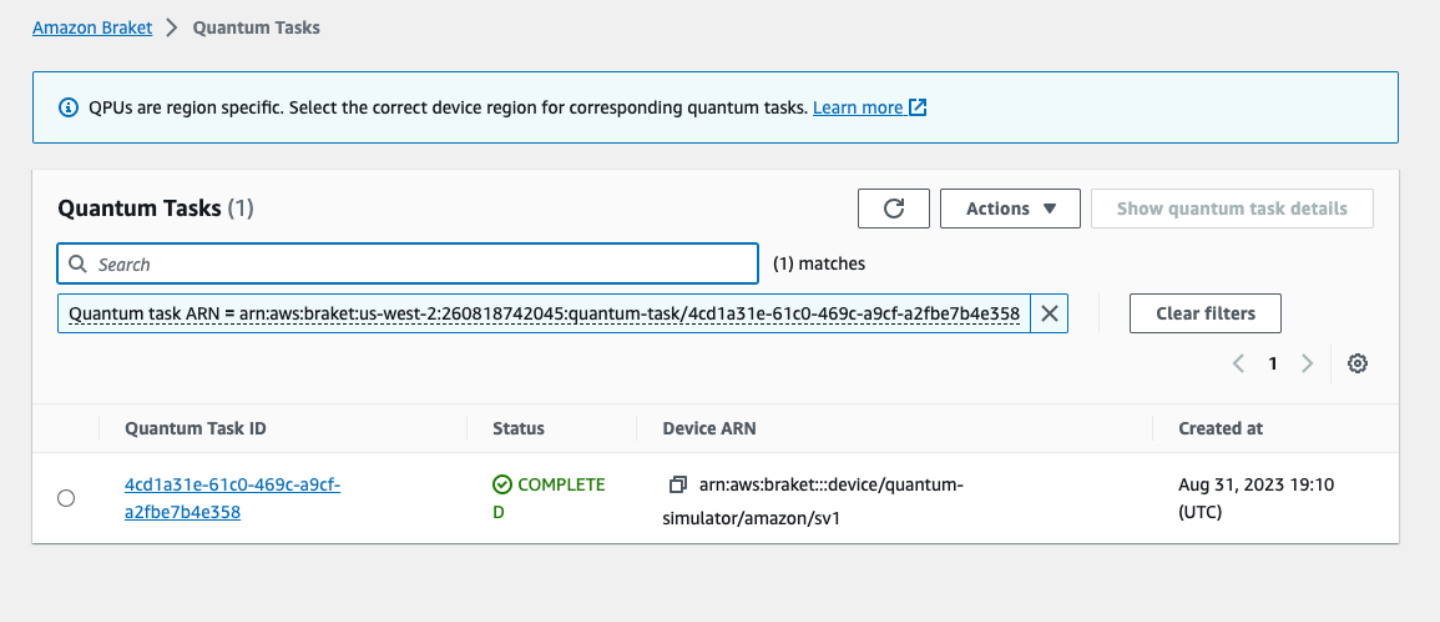
O Amazon Braket oferece uma maneira conveniente de monitorar a tarefa quântica por meio do console Amazon Braket. Todas as tarefas quânticas enviadas são listadas no campo Tarefas quânticas, conforme mostrado na figura a seguir. Esse serviço é específico da região, o que significa que você só pode visualizar as tarefas quânticas criadas na região específica. Região da AWS



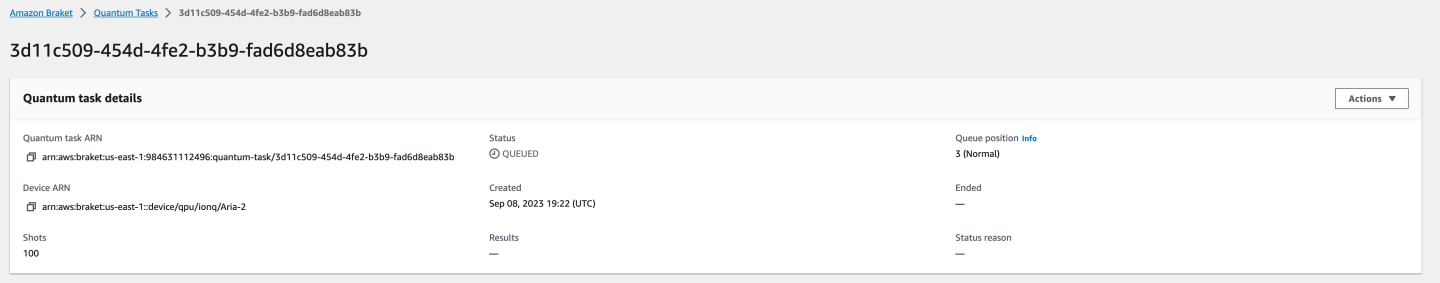
Você pode pesquisar tarefas quânticas específicas por meio da barra de navegação. A pesquisa pode ser baseada no ARN (ID) da Quantum Task, status, dispositivo e hora de criação. As opções aparecem automaticamente quando você seleciona a barra de navegação, conforme mostrado no exemplo a seguir.



A imagem a seguir mostra um exemplo de busca por uma tarefa quântica com base em seu ID de tarefa quântica exclusivo, que pode ser obtido por meio de uma chamada `task.id`.



Além disso, visto na figura abaixo, o status de uma tarefa quântica pode ser monitorado enquanto ela está em um estado QUEUED. Clicar no ID da tarefa quântica mostra a página de detalhes. Esta página exibe a posição dinâmica da fila para sua tarefa quântica em relação ao dispositivo em que ela será processada.



As tarefas quânticas enviadas como parte de um trabalho híbrido terão prioridade quando estiverem na fila. As tarefas quânticas enviadas fora de um trabalho híbrido terão prioridade normal na fila.

Os clientes que desejam consultar o SDK do Braket podem obter programaticamente suas posições na fila de tarefas quânticas e híbridas. Para obter mais informações, consulte a página Quando minha tarefa será executada.

Marcação de recursos do Amazon Braket

Uma tag é um rótulo de atributo personalizado que você atribui ou AWS atribui a um AWS recurso. Uma tag são metadados que informam mais sobre seu recurso. Cada tag consiste em uma chave e um valor. Juntos, são conhecidos como pares de chave/valor. Em tags atribuídas por você, você mesmo define a chave e o valor.

No console do Amazon Braket, você pode navegar até uma tarefa quântica ou um caderno e ver a lista de tags associadas a ela. Você pode adicionar uma tag, remover uma tag ou modificar uma tag. Você pode marcar uma tarefa quântica ou um notebook após a criação e, em seguida, gerenciar as tags associadas por meio do console AWS CLI, ouAPI.

Mais sobre AWS e tags

- Para obter informações gerais sobre marcação, incluindo convenções de nomenclatura e uso, consulte O que é o Tag Editor? no Guia do usuário AWS dos recursos de marcação e do editor de tags.
- Para obter informações sobre restrições à marcação, consulte Limites e requisitos de nomenclatura de tags no Guia do usuário dos AWS recursos de marcação e do editor de tags.
- Para ver as melhores práticas e estratégias de marcação, consulte Melhores práticas para marcar recursos da AWS.
- Para obter uma lista de serviços que oferecem suporte à atribuição de tags, consulte a Referência da API de atribuição de tags a grupos de recursos.

As seções a seguir fornecem mais informações sobre tags para o Amazon Braket.

Nesta seção:

- Usar tags
- Recursos compatíveis para marcação no Amazon Braket
- Marcar com o Amazon Braket API
- Restrições de marcação
- Gerenciamento de tags no Amazon Braket
- Exemplo de AWS CLI marcação no Amazon Braket

Usar tags

As tags podem organizar seus recursos em categorias que são úteis para você. Por exemplo, você pode atribuir uma tag “Department” para especificar o departamento responsável pelo recurso.

Cada tag da tem duas partes:

- Uma chave de tag (por exemplo CostCenter, Ambiente ou Projeto). As chaves de tag diferenciam maiúsculas de minúsculas
- Um campo opcional, conhecido como valor de tag (por exemplo, 111122223333 ou Production). Omitir o valor da tag é o mesmo que usar uma string vazia. Assim como as chaves de tag, os valores de tag diferenciam maiúsculas de minúsculas.

As tags ajudam a:

- Identifique e organize seus AWS recursos. Muitos Serviços da AWS oferecem suporte à marcação, então você pode atribuir a mesma tag a recursos de serviços diferentes para indicar que os recursos estão relacionados.
- Acompanhe seus AWS custos. Você ativa essas tags no Gerenciamento de Faturamento e Custos da AWS painel. AWS usa as tags para categorizar seus custos e entregar um relatório mensal de alocação de custos para você. Para obter mais informações, consulte Usar tags de alocação de custos no Guia do Usuário do Gerenciamento de Faturamento e Custos da AWS.
- Controle o acesso aos seus AWS recursos. Para obter mais informações, consulte Controlando o acesso usando tags.

Recursos compatíveis para marcação no Amazon Braket

Os seguintes recursos do Amazon Braket são compatíveis com a marcação:

- Recurso do `quantum-task`
- Nome do recurso: `AWS::Service::Braket`
- Regex do ARN: `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

Nota: Você pode aplicar e gerenciar etiquetas para seus cadernos Amazon Braket no console Amazon Braket, usando o console para navegar até o recurso de notebook, embora os notebooks sejam, na verdade, recursos de IA da Amazon. SageMaker Para obter mais informações, consulte Metadados da instância do notebook na SageMaker documentação.

Marcar com o Amazon Braket API

- Se você estiver usando a API do Amazon Braket para configurar tags em um recurso, chame o `TagResource`API.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tags {\"city\": \"Seattle\"}
```

- Para remover tags de um recurso, chame `UntagResource`API.

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- Para listar todas as tags anexadas a um recurso específico, chame `ListTagsForResource`API.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city\",\"state\"]"
```

Restrições de marcação

As restrições básicas a seguir aplicam-se às tags nos recursos do Amazon Braket:

- O número máximo de tags que você pode atribuir a um recurso: 50
- Comprimento máximo da chave: 128 caracteres Unicode

- Comprimento máximo de valor: 256 caracteres Unicode
- Caracteres válidos para chave e valor: `a-z, A-Z, 0-9, space`, e esses caracteres: `_ . : / = + - e @`
- As chaves e os valores diferenciam letras maiúsculas de minúsculas.
- Não use `aws` como prefixo para chaves; está reservado para AWS uso.

Gerenciamento de tags no Amazon Braket

Você define as tags como propriedades em um recurso. Você pode visualizar, adicionar, modificar, listar e excluir tags por meio do console Amazon Braket, da API do Amazon Braket ou da AWS CLI. Para mais informações, consulte Referência da API do Amazon Braket.

Nesta seção:

- Adicionar marcações
- Visualizar tags
- Editando tags
- Remover tags

Adicionar marcações

Você pode adicionar tags a recursos que podem ser marcados nos seguintes momentos:

- Ao criar o recurso: use o console ou inclua o parâmetro `Tags` com a operação `Create` na AWS API.
- Depois de criar o recurso: use o console para navegar até a tarefa quântica ou o recurso do caderno, ou chame a operação `TagResource` na AWS API.

Para adicionar tags a um recurso ao criá-lo, você também precisa de permissão para criar um recurso do tipo especificado.

Visualizar tags

Você pode visualizar as tags em qualquer um dos recursos marcáveis no Amazon Braket usando o console para navegar até o recurso da tarefa ou do notebook, ou chamando a operação. AWS `ListTagsForResource` API

Você pode usar o AWS API comando a seguir para visualizar as tags em um recurso:

- AWS API: `ListTagsForResource`

Editando tags

Você pode editar tags usando o console para navegar até a tarefa quântica ou o recurso do caderno ou pode usar o comando a seguir para modificar o valor de uma tag anexada a um recurso que pode ser marcado. Quando você especifica uma chave de tag que já existe, o valor dessa chave é sobrescrito:

- AWS API: `TagResource`

Remover tags

Você pode remover tags de um recurso especificando as chaves a serem removidas, usando o console para navegar até a tarefa quântica ou o recurso do caderno, ou ao chamar a operação `UntagResource`.

- AWS API: `UntagResource`

Exemplo de AWS CLI marcação no Amazon Braket

Quando você está trabalhando com o AWS Command Line Interface (AWS CLI) para interagir com o Amazon Braket, o código a seguir é um exemplo de comando que demonstra como criar uma tag que se aplica a uma tarefa quântica que você cria. Neste exemplo, a tarefa está sendo executada no simulador SV1 quântico com configurações de parâmetros especificadas para a unidade de processamento Rigetti quântico (QPU). É importante que, dentro do comando de exemplo, a tag seja especificada no final, depois de todos os outros parâmetros necessários. Nesse caso, a tag tem uma Chave de `state` e um Valor de`Washington`. Essas marcações podem ser usadas para ajudar a categorizar ou identificar essa tarefa quântica específica.

```
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\"}
```

Este exemplo demonstra como você pode aplicar tags às suas tarefas quânticas ao executá-las na AWS CLI, o que é útil para organizar e rastrear seus recursos do Braket.

Monitorando suas tarefas quânticas com EventBridge

A Amazon EventBridge monitora eventos de mudança de status nas tarefas quânticas do Amazon Braket. Os eventos do Amazon Braket são entregues quase em EventBridge tempo real. Você pode escrever regras que indiquem quais eventos lhe interessam, incluindo ações automatizadas a serem tomadas quando um evento corresponder a uma regra. As ações automáticas que podem ser acionadas incluem as seguintes:

- Invocando uma função AWS Lambda
- Ativando uma máquina de AWS Step Functions estado
- Notificar um tópico do Amazon SNS

EventBridge monitora esses eventos de mudança de status do Amazon Braket:

- O estado das mudanças na tarefa quântica

O Amazon Braket garante a entrega de eventos quânticos de mudança de status de tarefas. Esses eventos são entregues pelo menos uma vez, mas possivelmente fora de ordem.

Para obter mais informações, consulte os Eventos na Amazon EventBridge.

Nesta seção:

- Monitore o status da tarefa quântica com EventBridge
- Exemplo de evento Amazon Braket EventBridge

Monitore o status da tarefa quântica com EventBridge

Com EventBridge, você pode criar regras que definem ações a serem tomadas quando o Amazon Braket envia uma notificação de uma mudança de status em relação a uma tarefa quântica do Braket. Por exemplo, você pode criar uma regra que lhe envie uma mensagem de e-mail sempre que o status de uma tarefa quântica for alterado.

1. Faça login AWS usando uma conta que tenha permissões de uso EventBridge e Amazon Braket.
2. Abra o EventBridge console da Amazon.
3. Usando os valores a seguir, crie uma EventBridge regra:
   - Em Tipo de regra, escolha Regra com um padrão de evento.
   - Em Origem do evento, escolha Outra.
   - Em Padrão de evento, escolha Padrões personalizados (editor JSON) e cole um dos seguintes exemplos de padrão de evento na área de texto:

     ```
     {
       "source": [
         "aws.braket"
       ],
       "detail-type": [
         "Braket Task State Change"
       ]
     }
     ```

     Para capturar todos os eventos do Amazon Braket, exclua a seção `detail-type` conforme mostrado no código a seguir:

     ```
     {
       "source": [
         "aws.braket"
       ]
     }
     ```

   - Para Tipos de destino, escolha e AWS service (Serviço da AWS), em Selecionar um destino, escolha um destino, como um tópico ou AWS Lambda função do Amazon SNS. O alvo é

acionado quando um evento de mudança de estado de tarefa quântica é recebido do Amazon Braket.

Por exemplo, use um tópico do Amazon Simple Notification Service (SNS) para enviar um e-mail ou mensagem de texto quando ocorrer um evento. Para isso, primeiro crie um tópico do Amazon SNS usando o console do Amazon SNS. Para saber mais, consulte Usar o Amazon SNS para notificações de usuários.

Para obter detalhes sobre a criação de regras, consulte Criação de EventBridge regras da Amazon que reagem a eventos.

Exemplo de evento Amazon Braket EventBridge

Para obter informações sobre os campos de um evento de mudança de status de tarefas do Amazon Braket Quantum, consulte Eventos na Amazon. EventBridge

Os atributos a seguir aparecem no campo “detalhe” do JSON.

- **`quantumTaskArn`** (str): A tarefa quântica para a qual esse evento foi gerado.
- **`status`** (Opcional [str]): o status para o qual a tarefa quântica foi transferida.
- **`deviceArn`** (str): O dispositivo especificado pelo usuário para o qual essa tarefa quântica foi criada.
- shots (int): O número de solicitações de shots do usuário.
- **`outputS3Bucket`** (str): o bucket de saída especificado pelo usuário.
- **`outputS3Directory`** (str): o prefixo da chave de saída especificado pelo usuário.
- **`createdAt`** (str): O tempo de criação da tarefa quântica como uma string ISO-8601.
- **`endedAt`** (Opcional [str]): o momento em que a tarefa quântica atingiu um estado terminal. Esse campo está presente somente quando a tarefa quântica passou para um estado terminal.

O código JSON a seguir mostra um exemplo de alteração do status da tarefa do Amazon Braket Quantum.

```
{
    "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"
      }
}
```


Monitorando suas métricas com CloudWatch

Você pode monitorar o Amazon Braket usando a CloudWatch Amazon, que coleta dados brutos e os processa em métricas legíveis, quase em tempo real. Você visualiza informações históricas geradas até 15 meses atrás ou pesquisa métricas que foram atualizadas nas últimas duas semanas no CloudWatch console da Amazon para ter uma melhor perspectiva do desempenho do Amazon Braket. Para saber mais, consulte Como usar CloudWatch métricas.

> **Note**
>
> Você pode visualizar os fluxos de CloudWatch log dos notebooks Amazon Braket navegando até a página de detalhes do caderno no console Amazon AI. SageMaker Configurações adicionais do notebook Amazon Braket estão disponíveis no console. SageMaker

Nesta seção:

- Métricas e dimensões do Amazon Braket

Métricas e dimensões do Amazon Braket

As métricas são o conceito fundamental em CloudWatch. Uma métrica representa um conjunto ordenado por tempo de pontos de dados publicados em. CloudWatch Cada métrica é caracterizada por um conjunto de dimensões. Para saber mais sobre as dimensões métricas em CloudWatch, consulte CloudWatch dimensões.

O Amazon Braket envia os seguintes dados métricos, específicos do Amazon Braket, para as métricas da Amazon: CloudWatch

Métricas de tarefas quânticas

As métricas estão disponíveis se existirem tarefas quânticas. Eles são exibidos em AWS/Braket/Por dispositivo no console. CloudWatch

Métrica	Description
Contagem	Número de tarefas quânticas.
Latência	Essa métrica é emitida quando uma tarefa quântica é concluída. Ele representa o tempo total desde a inicialização da tarefa quântica até a conclusão.

Dimensões para métricas de tarefas quânticas

As métricas da tarefa quântica são publicadas com uma dimensão baseada no parâmetro `deviceArn`, que tem o formato arn:aws:braket:::device/xxx.

Registrando suas tarefas quânticas com CloudTrail

O Amazon Braket é integrado AWS CloudTrail com, um serviço que fornece um registro das ações realizadas por um usuário, uma função ou um no AWS service (Serviço da AWS) Amazon Braket. CloudTrail captura todas as API chamadas para o Amazon Braket como eventos. As chamadas capturadas incluem chamadas do console do Amazon Braket e chamadas de código para as operações do Amazon Braket. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo eventos para o Amazon Braket. Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console

no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita ao Amazon Braket, o endereço IP a partir do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais CloudTrail, consulte o Guia AWS CloudTrail do usuário.

Nesta seção:

- Informações sobre o Amazon Braket em CloudTrail
- Noções básicas sobre entradas de arquivos de log do Amazon Braket

Informações sobre o Amazon Braket em CloudTrail

CloudTrail é ativado no seu Conta da AWS quando você cria a conta. Quando a atividade ocorre no Amazon Braket, essa atividade é registrada em CloudTrail um evento junto com AWS service (Serviço da AWS) outros eventos no histórico de eventos. Você pode visualizar, pesquisar e baixar eventos recentes no seu Conta da AWS. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

Para obter um registro contínuo dos eventos em seu Conta da AWS, incluindo eventos do Amazon Braket, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as Regiões da AWS. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros Serviços da AWS para analisar e agir com base nos dados do evento coletados nos CloudTrail registros. Para saber mais, consulte:

- Visão Geral para Criar uma Trilha
- CloudTrail Serviços e integrações compatíveis
- Configurando notificações do Amazon SNS para CloudTrail
- Recebendo arquivos de CloudTrail log de várias regiões e recebendo arquivos de CloudTrail log de várias contas

Todas as ações do Amazon Braket são registradas por. CloudTrail Por exemplo, chamadas para as `GetDevice` ações `GetQuantumTask` ou geram entradas nos arquivos de CloudTrail log.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS service (Serviço da AWS).

Para obter mais informações, consulte Elemento userIdentity do CloudTrail .

Noções básicas sobre entradas de arquivos de log do Amazon Braket

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte e inclui informações sobre a ação solicitada, a data e a hora da ação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das API chamadas públicas, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir é uma entrada de log para a ação `GetQuantumTask`, que obtém os detalhes de uma tarefa quântica.

```
{
  "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 seguir, é apresentada uma entrada de registro da ação `GetDevice`, que retorna os detalhes de um evento do dispositivo.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
        "userName": "foobar"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:46:29Z"
      }
    }
```

```
  },
  "eventTime": "2020-08-07T00:46:32Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetDevice",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "Boto3/1.14.33 Python/3.7.6 Linux/4.14.158-129.185.amzn2.x86_64 exec-
env/AWS_ECS_FARGATE Botocore/1.17.33",
  "errorCode": "404",
  "requestParameters": {
    "deviceArn": "foobar"
  },
  "responseElements": null,
  "requestID": "c614858b-4dcf-43bd-83c9-bcf9f17f522e",
  "eventID": "9642512a-478b-4e7b-9f34-75ba5a3408eb",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```

Registro em log avançado com o Amazon Braket

Você pode gravar todo o processo de processamento de tarefas usando um registrador. Essas técnicas avançadas de registro permitem que você veja a pesquisa em segundo plano e crie um registro para depuração posterior.

Para usar o registrador, recomendamos alterar os parâmetros `poll_interval_seconds` e `poll_timeout_seconds`, para que uma tarefa quântica possa ser demorada e o status da tarefa quântica seja registrado continuamente, com os resultados salvos em um arquivo. Você pode transferir esse código para um script Python em vez de um caderno Jupyter, para que o script possa ser executado como um processo em segundo plano.

Configurar o registrador

Primeiro, configure o registrador para que todos os registros sejam gravados automaticamente em um arquivo de texto, conforme mostrado nas linhas de exemplo a seguir.

```
# 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
```

Crie e execute o circuito

Agora você pode criar um circuito, enviá-lo a um dispositivo para execução e ver o que acontece conforme mostrado neste exemplo.

```
# 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
)
```

Verifique o arquivo de log

Você pode verificar o que está escrito no arquivo inserindo um dos comandos a seguir.

```
# 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})
```

Obtenha o ARN do arquivo de log

Da saída do arquivo de log retornada, conforme mostrado no exemplo anterior, você pode obter as informações do ARN. Com o ARN ID, você pode recuperar o resultado da tarefa quântica concluída.

```
# 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()
```

Cotas do Amazon Braket

A tabela a seguir lista as cotas de serviço do Amazon Braket. As cotas de serviço, também chamadas de limites, correspondem ao número máximo de recursos ou operações de serviço para sua Conta da AWS.

Nem todas as cotas podem ser aumentadas. Para obter mais informações, consulte as AWS service (Serviço da AWS) cotas.

- Essas cotas de expansão não podem ser aumentadas.
- O aumento máximo da taxa para cotas ajustáveis (exceto a taxa de expansão, que não pode ser ajustada) é 2x o limite de taxa padrão especificado. Por exemplo, uma cota padrão de 60 pode ser ajustada para um máximo de 120.
- A cota ajustável para tarefas quânticas simultâneas SV1 (DM1) permite um máximo de 60 por Região da AWS.
- O número máximo de instâncias computacionais permitidas para uma tarefa híbrida é 1 e as cotas são ajustáveis.

Recurso	Description	Limites	Ajustável
Taxa de solicitações API	O número máximo de solicitações por segundo que é possível enviar nesta conta na atual região.	140	Sim
Taxa de expansão de solicitações API	O número máximo de solicitações adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	600	Não

Recurso	Description	Limites	Ajustável
Taxa de solicitações `CreateQuantumTask`	O número máximo de solicitações `CreateQuantumTask` que é possível enviar por segundo nessa conta por região.	20 por segundo	Sim
Taxa de expansão de solicitações `CreateQuantumTask`	O número máximo de solicitações `CreateQuantumTask` adicionais por segundo (RPS) que é possível enviar em uma intermitência nesta conta na região atual.	40	Não
Taxa de solicitações `SearchQuantumTasks`	O número máximo de solicitações `SearchQuantumTasks` que é possível enviar por segundo nessa conta por região.	5 por segundo	Sim
Taxa de expansão de solicitações `SearchQuantumTasks`	O número máximo de solicitações `SearchQuantumTasks` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	50	Não

Recurso	Description	Limites	Ajustável
Taxa de solicitações `GetQuantumTask`	O número máximo de solicitações `GetQuantumTask` que é possível enviar por segundo nessa conta por região.	100 por segundo	Sim
Taxa de expansão de solicitações `GetQuantumTask`	O número máximo de solicitações `GetQuantumTask` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	500	Não
Taxa de solicitações `CancelQuantumTask`	O número máximo de solicitações `CancelQuantumTask` que é possível enviar por segundo nessa conta por região.	2 por segundo	Sim
Taxa de expansão de solicitações `CancelQuantumTask`	O número máximo de solicitações `CancelQuantumTask` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	20	Não

Recurso	Description	Limites	Ajustável
Taxa de solicitações `GetDevice`	O número máximo de solicitações `GetDevice` que é possível enviar por segundo nessa conta por região.	5 por segundo	Sim
Taxa de expansão de solicitações `GetDevice`	O número máximo de solicitações `GetDevice` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	50	Não
Taxa de solicitações `SearchDevices`	O número máximo de solicitações `SearchDevices` que é possível enviar por segundo nessa conta por região.	5 por segundo	Sim
Taxa de expansão de solicitações `SearchDevices`	O número máximo de solicitações `SearchDevices` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	50	Não

Recurso	Description	Limites	Ajustável
Taxa de solicitações `CreateJob`	O número máximo de solicitações `CreateJob` que é possível enviar por segundo nessa conta por região.	1 por segundo	Sim
Taxa de expansão de solicitações `CreateJob`	O número máximo de solicitações `CreateJob` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	5	Não
Taxa de solicitações `SearchJobs`	O número máximo de solicitações `SearchJob` que é possível enviar por segundo nessa conta por região.	5 por segundo	Sim
Taxa de expansão de solicitações `SearchJobs`	O número máximo de solicitações `SearchJob` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	50	Não

Recurso	Description	Limites	Ajustável
Taxa de solicitações `GetJob`	O número máximo de solicitações `GetJob` que é possível enviar por segundo nessa conta por região.	5 por segundo	Sim
Taxa de expansão de solicitações `GetJob`	O número máximo de solicitações `GetJob` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	25	Não
Taxa de solicitações `CancelJob`	O número máximo de solicitações `CancelJob` que é possível enviar por segundo nessa conta por região.	2 por segundo	Sim
Taxa de expansão de solicitações `CancelJob`	O número máximo de solicitações `CancelJob` adicionais por segundo (RPS) que é possível enviar em uma expansão nesta conta na região atual.	5	Não

Recurso	Description	Limites	Ajustável
Número de tarefas SV1 quânticas simultâneas	O número máximo de tarefas simultâneas em execução no State Vector Simulator (SV1) na região atual.	100 us-east-1, 50 us-west-1, 100 us-west-2, 50 eu-west-2	Não
Número de tarefas DM1 quânticas simultâneas	O número máximo de tarefas quânticas simultâneas em execução no simulador de matriz de densidade (DM1) na região atual.	100 us-east-1, 50 us-west-1, 100 us-west-2, 50 eu-west-2	Não
Número de tarefas TN1 quânticas simultâneas	O número máximo de tarefas quânticas simultâneas em execução no simulador de rede tensorial (TN1) na região atual.	10 us-east-1, 10 us-west-2, 5 eu-west-2,	Sim
Número de tarefas híbridas simultâneas	O número máximo de tarefas híbridas simultâneas em execução na região atual.	3	Sim
Limite de runtime do Hybrid jobs	A quantidade máxima de tempo em dias em que uma tarefa híbrida pode ser executada.	5	Não

A seguir estão os limites padrão de instâncias computacionais clássicas para tarefas híbridas. Para aumentar essas cotas, entre em contato com Suporte. Além disso, as regiões disponíveis são especificadas para cada instância.

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c4.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c4.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Não
Número máximo de instâncias de ml.c4.2xlarge para	O número máximo de instâncias do tipo ml.c4.2xlarge	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
tarefas híbridas	permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.							
Número máximo de instâncias de ml.c4.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c4.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c4.8xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c4.8xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5.2xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	1	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5.9xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5.9xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	1	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5.18xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5.18xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5n.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5n.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5n.2xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5n.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5n.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5n.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5n.9xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5n.9xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c5n.18xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.c5n.18xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.2xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.8xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.8xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.12xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.12xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.g4dn.16xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.g4dn.16xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m4.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m4.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m4.2xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m4.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m4.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m4.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	2	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m4.10› large para tarefas híbridas	O número máximo de instâncias do tipo ml.m4.10› large permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m4.16› large para tarefas híbridas	O número máximo de instâncias do tipo ml.m4.16› large permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.large para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.large permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.2xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.4xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.4xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.12› large para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.12› large permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.m5.24› large para tarefas híbridas	O número máximo de instâncias do tipo ml.m5.24› large permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p2.xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.p2.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p2.8xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.p2.8xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Recurso	Description	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p2.16xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.p2.16xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Recurso	Descripti on	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p4d.24xlarge para tarefas híbridas	O número máximo de instâncias do tipo ml.p4d.24xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Solicitando atualizações de limite

Se você receber uma ServiceQuotaExceeded exceção para um tipo de instância e não tiver instâncias suficientes disponíveis para ela, você pode solicitar um aumento de limite na página Service Quotas no AWS console e pesquisar Amazon Braket em Serviços.AWS

> **Note**
>
> Se seu trabalho híbrido não conseguir provisionar a capacidade computacional de ML solicitada, use outra região. Além disso, se você não vê uma instância na tabela, ela não está disponível para trabalhos híbridos.

Cotas e limites adicionais

- A ação da tarefa quântica do Amazon Braket está limitada a 3 MB de tamanho.
- Pois SV1, a duração máxima de execução é de 3 horas para circuitos de até 31 qubits e 11 horas para circuitos acima de 31 qubits.
- O número máximo de disparos por tarefa permitido para dispositivos SV1, DM1, e Rigetti é 50.000.
- O número máximo de disparos por tarefa permitido para o TN1 é 1000.
- Para AQT o IBEX-Q1 dispositivo, o máximo é de 2000 fotos por tarefa.
- Para todos IonQ os dispositivos: o número mínimo de disparos por tarefa é 100. Ao usar um modelo sob demanda, há um limite de 1 milhão de disparos e um mínimo de 2.500 disparos para tarefas de mitigação de erros. Para uma reserva direta, não há limite de disparos e há um mínimo de 500 disparos para tarefas de mitigação de erros.
- Para o dispositivo QuEra Aquila, o máximo é de 1.000 disparos por tarefa.
- Para IQM Emerald dispositivos Garnet e dispositivos, o máximo é de 20.000 fotos por tarefa.
- Para dispositivos QPU TN1 e para os dispositivos QPU, as fotos por tarefa devem ser > 0.

Histórico de documentos do Guia do Desenvolvedor do Amazon Braket

A tabela a seguir descreve as versões da documentação do Amazon Braket.

- Última atualização API de referência: 20 de novembro de 2025
- Última atualização da documentação: 2 de março de 2026

Alteração	Descrição	Data
Desativação do dispositivo IonQ Aria-1	Suporte removido para o dispositivo IonQ Aria-1.	2 de março de 2026
Atualize as páginas “Trabalhando com reservas”	Maior clareza das páginas “Trabalhando com reservas”	3 de fevereiro de 2026
Support Python versão 3.12 para notebooks Braket e contêineres gerenciados	Foi adicionado suporte ao Python versão 3.12 para notebooks Amazon Braket e contêineres gerenciados (Base, CUDA-Q e Tensorflow). PennyLane Inclui guia de solução de problemas para a atualização do Python 3.12.	21 de janeiro de 2026
Remover exemplos de P3	SageMaker está aposentando sua família de `ml.p3` instância. Exemplos substituídos pela família de `ml.g4dn` instâncias recomendada.	19 de dezembro de 2025
Novo recurso de limite de gastos	Foi adicionado suporte ao recurso de limite de gastos do Amazon Braket, que permite definir limites orçamentários opcionais para QPUs	20 de novembro de 2025

	indivíduos que validam e rejeitam automaticamente tarefas que excedem o limite de gastos configurado.	
Novo dispositivo Braket AQT IBEX-Q1	Foi adicionado suporte para o AQT IBEX-Q1dispositivo. Este dispositivo é baseado em um cristal de ^{40} íons Ca^{+} em uma armadilha macroscópica de radiofrequência localizada em uma câmara de vácuo ultra-alto.	18 de novembro de 2025
Novo suporte nativo para o CUDA-Q Amazon Braket NBIs	Foi adicionado suporte nativo para instâncias CUDA-Q de caderno do Amazon Braket. Para obter mais informações, consulte CUDA-Q em. NBIs	10 de novembro de 2025
Retirada do dispositivo IonQ Aria-2	Suporte removido para o dispositivo IonQ Aria-2.	27 de outubro de 2025
Documentação consolidada de trabalhos híbridos	As seções de tarefas híbridas foram consolidadas para aparecer em Trabalhando com o Amazon Braket Hybrid Jobs.	21 de outubro de 2025
Novo contêiner CUDA-Q fornecido pelo Braket	Foi adicionado suporte para um contêiner CUDA-Q de tarefas híbridas fornecido. Para obter mais informações, consulte Definir o ambiente para seu script de algoritmo.	2 de setembro de 2025

Novo recurso de emulador de dispositivo local	Foi adicionado suporte para uma ferramenta local de emulador de dispositivos quânticos para emular seus programas textuais antes de enviá-los para dispositivos quânticos.	25 de agosto de 2025
As páginas Pennylane e CUDA-Q foram movidas para a seção Construir	As páginas Pennylane e CUDA-Q foram movidas para aparecerem na seção Construir no sumário.	15 de agosto de 2025
Novo recurso `ProgramSet`	Foi adicionado suporte para conjuntos de programas, uma operação para executar vários circuitos quânticos em uma única tarefa quântica.	14 de agosto de 2025
Novo dispositivo IQM Emerald	Foi adicionado suporte para o dispositivo IQM Emerald. Um dispositivo de 54 qubits com uma topologia de rede quadrada (Crystal).	21 de julho de 2025
Atualizou a AmazonBraketServiceRolePolicypolítica	AmazonBraketServiceRolePolicyagora fornece apenas ações s3: * e logs: * para o aws: PrincipalAccount. Isso restringe o acesso somente aos buckets e grupos de registros do solicitante.	11 de julho de 2025

Novo recurso de capacidade experimental: circuitos dinâmicos	A medição do meio do circuito e as operações de avanço estão disponíveis como recursos experimentais, consulte Acesso a circuitos dinâmicos em dispositivos IQM.	26 de junho de 2025
Atualizou a AmazonBraketFullAccesspolítica	AmazonBraketFullAccessagora inclui preços: GetProducts para exibir os custos de hardware no console.	14 de abril de 2025
Novo dispositivo IonQ Forte-Enterprise-1	Foi adicionado suporte para o dispositivo IonQ Forte-Enterprise-1. Um dispositivo de 36 qubits que utiliza tecnologia de íons presos.	17 de março de 2025
Permissões aprimoradas de condições do S3	Para melhorar a segurança, `AmazonBraketFullAccess` agora só fornece ações `s3:*` para o `aws:PrincipalAccount`. Isso restringe o acesso somente aos buckets do próprio solicitante.	07 de março de 2025
Novo dispositivo Rigetti Ankaa-3	Foi adicionado suporte para o dispositivo Rigetti Ankaa-3. Um dispositivo de 84 qubits que utiliza tecnologia escalável de vários chips.	14 de janeiro de 2025
Retirada do dispositivo Rigetti Ankaa-2	Suporte removido para o dispositivo Rigetti Ankaa-2.	14 de janeiro de 2025

Support para IPv6 tráfego	O Amazon Braket agora IPv6 oferece suporte ao tráfego usando o endpoint dualstack `.braket.{region}.api.aws`	12 de dezembro de 2024
Suporte para NVIDIA's CUDA-Q no Amazon Braket	Agora, os clientes podem executar programas quânticos usando a estrutura NVIDIA's CUDA-Q do desenvolvedor no Amazon Braket.	06 de dezembro de 2024
O dispositivo IonQ Forte-1 está prontamente disponível	O dispositivo IonQ Forte-1 não é mais apenas para reservas e agora está prontamente disponível para nossos clientes.	22 de novembro de 2024
Retirada do dispositivo Rigetti Aspen-M-3	Suporte removido para o dispositivo Rigetti Aspen-M-3.	27 de setembro de 2024
Retirada do dispositivo IonQ Harmony	Suporte removido para o dispositivo IonQ Harmony.	29 de agosto de 2024
Novo dispositivo Rigetti Ankaa-2	Foi adicionado suporte para o dispositivo Rigetti Ankaa-2. Um dispositivo de 84 qubits que utiliza tecnologia escalável de vários chips.	26 de agosto de 2024
Reorganização do guia do desenvolvedor	O novo guia do desenvolvedor aborda a jornada existente do cliente de criar, testar e executar e orienta os usuários nesse caminho com o Amazon Braket.	23 de agosto de 2024

Retirada do dispositivo OQC Lucy	Suporte removido para o dispositivo OQC Lucy.	28 de junho de 2024
Novo dispositivo IQM Garnet e região Europe North 1	Foi adicionado suporte para o dispositivo IQM Garnet. Um dispositivo de 20 qubits com uma topologia de rede quadrada. Expandiu as regiões suportadas pelo Braket na Europa Norte 1 (Estocolmo).	22 de maio de 2024
Lançado o desajuste local	Os recursos experimentais agora incluem o recurso de desajuste local QuEra do Aquila QPU.	11 de abril de 2024
Lançado o gerenciador de inatividade do caderno	Ao criar uma instância do caderno, ative o gerenciador de inatividade e defina um tempo de inatividade para redefinir automaticamente a instância do caderno do Braket.	27 de março de 2024
Sumário reformulado	Reorganizou o índice do Amazon Braket para atender aos requisitos do guia de AWS estilo e melhorar o fluxo de conteúdo para a experiência do cliente.	12 de dezembro de 2023

Braket lançado diretamente	Foi adicionado suporte para recursos diretos do Braket, incluindo: • Trabalhando com reservas • Obter aconselhamento especializado • Explorar capacidades experimentais	27 de novembro de 2023
Atualização do Criar uma instância de caderno do Amazon Braket	A documentação foi atualizada para adicionar informações para criar uma instância de caderno para clientes novos e existentes do Amazon Braket.	27 de novembro de 2023
Atualização do Traga seu próprio contêiner (BYOC)	A documentação foi atualizada para adicionar informações sobre quando usar o BYOC, a receita para o BYOC e executar trabalhos híbridos do Braket no contêiner.	18 de outubro de 2023

Lançado o decorador de empregos híbrido	Página Execute seu código local como um trabalho híbrido adicionada. Contém exemplos: • Criar um trabalho híbrido a partir do código Python local • Instalar pacotes e código-fonte adicionais do Python • Salvar e carregar dados em uma instância de trabalho híbrida • Práticas recomendadas para decoradores de trabalhos híbridos	16 de outubro de 2023
Visibilidade de fila adicionada	A documentação do Guia do Desenvolvedor foi atualizada para incluir queue depth e queue position. A documentação da API foi atualizada para refletir as novas alterações da API para visibilidade da fila.	25 de setembro de 2023
Padronizar a nomenclatura na documentação	A documentação foi atualizada para alterar qualquer instância de “trabalho” para “trabalho híbrido” e “tarefa” para “tarefa quântica”	11 de setembro de 2023
Novo dispositivo IonQ Aria 2	Suporte adicionado para o dispositivo IonQ Aria 2	8 de setembro de 2023

Portões nativos atualizados	A documentação foi atualizada para adicionar informações sobre o acesso programático aos portões nativos da Rigetti.	16 de agosto de 2023
Partida Xanadu	Atualizou a documentação para remover todos os dispositivos Xanadu	2 de junho de 2023
Novo dispositivo IonQ Aria	Suporte adicionado para o dispositivo IonQ Aria	16 de maio de 2023
RigettiDispositivo retirado	Descontinuando o suporte para Rigetti Aspen-M-2	2 de maio de 2023
Informações atualizadas sobre a AmazonBraketFullAccesspolítica	Atualizou o script que define o conteúdo da AmazonBraketFullAccesspolítica para incluir as GetMetricData ações servicequotas: GetServiceQuota e cloudwatch:, bem como informações sobre limitações com relação às cotas.	19 de abril de 2023
Lançamento do Guided Journeys	A documentação foi alterada para refletir o método mais atualizado e simplificado de integração do Braket.	5 de abril de 2023
Novo dispositivo Rigetti Aspen-M-3	Suporte adicionado para o dispositivo Rigetti Aspen-M-3	17 de janeiro de 2023
Novo recurso de gradiente adjunto	Foram adicionadas informações sobre o recurso de gradiente adjunto oferecido pelo SV1	7 de dezembro de 2022

Novo recurso de biblioteca de algoritmos	Foram adicionadas informaçõ es sobre a biblioteca de algoritmos Braket, que fornece um catálogo de algoritmos quânticos pré-construídos	28 de novembro de 2022
Partida D-Wave	A documentação foi atualizada para acomodar a remoção de todos os dispositivos D-Wave	17 de novembro de 2022
Novo dispositivo QuEra Aquila	Suporte adicionado para o dispositivo QuEra Aquila	31 de outubro de 2022
Suporte para Braket Pulse	Foi adicionado suporte para Braket Pulse, que permite que o controle de pulso seja usado em dispositivos Rigetti e OQC	20 de outubro de 2022
Suporte para portas nativas IonQ	Foi adicionado suporte para o conjunto de portas nativo oferecido pelo dispositivo IonQ	13 de setembro de 2022
Novas cotas de instâncias	As cotas padrão de instância de computação clássica associadas a trabalhos híbridos foram atualizadas	22 de agosto de 2022
Novo painel de serviço	Capturas de tela do console atualizadas para incluir o painel de serviços	17 de agosto de 2022
Novo dispositivo Rigetti Aspen-M-2	Suporte adicionado para o dispositivo Rigetti Aspen-M-2	12 de agosto de 2022
Novos recursos do OpenQASM	Adicionado suporte aos recursos do OpenQASM para os simuladores locais (braket_sv e braket_dm)	4 de agosto de 2022

Novos procedimentos de monitoramento de custos	Foi adicionado como obter estimativas de custo máximo quase em tempo real para simuladores e workloads de hardware	18 de julho de 2022
Novo dispositivo Xanadu Borealis	Suporte adicionado para o dispositivo Xanadu Borealis	2 de junho de 2022
Novos procedimentos de simplificação da integração	Informações adicionadas sobre como os novos e simplificados procedimentos de integração funcionam	16 de maio de 2022
Novo dispositivo D-Wave Advantage_system6.1	Suporte adicionado para o dispositivo D-Wave Advantage _system6.1	12 de maio de 2022
Suporte para simuladores embarcados	Foi adicionado como executar simulações incorporadas com trabalhos híbridos e como usar o simulador de PennyLane raios	4 de maio de 2022
AmazonBraketFullAccess - Política de acesso total para o Amazon Braket	S3 adicionadas: ListAllMy Buckets permissões para permitir que os usuários visualizem e inspecionem os buckets criados e usados para o Amazon Braket	31 de março de 2022
Suporte para OpenQASM	Foi adicionado suporte ao OpenQASM 3.0 para dispositi vos e simuladores quânticos baseados em portas	7 de março de 2022

Novo fornecedor de hardware quântico e nova região Oxford Quantum Circuits, eu-west-2	Encerramento de compatibilidade com OQC e eu-west-2	28 de fevereiro de 2022
Novo dispositivo Rigetti	Adicionado o suporte para Rigetti Aspen M-1	15 de fevereiro de 2022
Novos limites de recursos	Aumentamos o número máximo de tarefas SV1 e tarefas DM1simultâneas de 55 para 100	5 de janeiro de 2022
Novo dispositivo Rigetti	Adicionado o suporte para Rigetti Aspen-11	20 de dezembro de 2021
Dispositivo Rigetti retirado	Encerramento de compatibilidade com dispositivo Rigetti Aspen-10	20 de dezembro de 2021
Novo tipo de resultado	Tipo de resultado de matriz de densidade reduzida suportado por dispositivos e DM1 simuladores de matriz de densidade locais	20 de dezembro de 2021
Descrição da política atualizada	O Amazon Braket atualizou o ARN da função para incluir o caminho. `servicerole/` Para obter informações sobre atualizações de políticas, consulte a tabela de atualizações do Amazon Braket AWS para políticas gerenciadas.	29 de novembro de 2021
Trabalhos do Amazon Braket	Guia do usuário para Amazon Braket Hybrid Jobs e adicionado API	29 de novembro de 2021

Novo dispositivo Rigetti	Adicionado o suporte para Rigetti Aspen-10	20 de novembro de 2021
D-WaveDispositivo retirado	Suporte descontinuado para D-Wave QPU, Advantage _system1	4 de novembro de 2021
Novo dispositivo D-Wave	Suporte adicional para uma D-Wave QPU adicional, Advantage_system4	5 de outubro de 2021
Novos simuladores de ruído	Foi adicionado suporte para um simulador de matriz de densidade (DM1), que pode simular circuitos de até 17 qubits e um simulador de ruído local braket_dm	25 de maio de 2021
PennyLane apoio	Suporte adicionado para PennyLane no Amazon Braket	8 de dezembro de 2020
Novo simulador	Foi adicionado suporte para um simulador de rede Tensor (TN1), que permite circuitos maiores	8 de dezembro de 2020
Agrupamento de tarefas	O Braket suporta o agrupamento de tarefas do cliente	24 de novembro de 2020
Alocação manual de qubit	O Braket é compatível com alocação manual de qubit no dispositivo Rigetti	24 de novembro de 2020
Cotas ajustáveis	O Braket suporta cotas ajustáveis de autoatendimento para seus recursos de tarefas	30 de outubro de 2020

Support for PrivateLink	Você pode configurar endpoints de VPC privados para suas tarefas do Braket.	30 de outubro de 2020
Suporte para tags	O Braket suporta tags API baseadas para o recurso de tarefas quânticas	30 de outubro de 2020
Novo dispositivo D-Wave	Suporte adicional para uma D-Wave QPU adicional, Advantage_system1	29 de setembro de 2020
Lançamento inicial	Versão inicial da documentação do Amazon Braket	12 de agosto de 2020

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.