aws

Guia do Desenvolvedor

Amazon Braket

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Braket: Guia do Desenvolvedor

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

As marcas comerciais e imagens comerciais 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 são propriedade da Amazon pertencem aos respectivos proprietários, os quais podem ou não ser afiliados, estar conectados ou ser patrocinados pela Amazon.

As marcas comerciais e imagens comerciais 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 são propriedade da Amazon pertencem aos respectivos proprietários, os quais podem ou não ser afiliados, estar conectados ou ser patrocinados pela Amazon.

Table of Contents

O que é o Amazon Braket? 1
Termos e conceitos do Amazon Braket 3
AWS terminologia e dicas para o Amazon Braket 7
Definição de preço 8
Acompanhamento de custos quase em tempo real 9
Melhores práticas para redução de custos 11
Como funciona 13
Fluxo de tarefas quânticas do Amazon Braket 14
Processamento de dados de terceiros 15
Repositórios e plug-ins principais para Braket 15
Repositórios principais 15
Plug-ins 15
Dispositivos compatíveis 16
IonQ 21
IQM 21
Rigetti 22
Oxford Quantum Circuits (OQC) 22
QuEra 23
Simulador vetorial estadual local () `braket_sv` 24
Simulador de matriz de densidade local () `braket_dm` 24
Simulador AHS local () `braket_ahs` 25
Simulador de vetores de estado () SV1 25
Simulador de matriz de densidade () DM1 26
Simulador de rede tensora () TN1 27
Simuladores incorporados 28
Compare simuladores 29
Regiões e endpoints 33
Quando minha tarefa quântica será executada? 34
Notificações de alteração de status por e-mail ou SMS 35
Janelas e status de disponibilidade da QPU 35
Visibilidade da fila 35
Conceitos básicos 38
Ativar o Amazon Braket 38
Pré-requisitos 38

Etapas para habilitar o Amazon Braket ... 39
Crie uma instância do notebook Amazon Braket ... 40
Execute seu primeiro circuito usando o Amazon Braket Python SDK ... 42
Execute seus primeiros algoritmos quânticos ... 47
Trabalhe com o Amazon Braket ... 49
Olá AHS: Execute sua primeira simulação hamiltoniana analógica ... 50
TEM ... 50
Cadeia de rotação interativa ... 51
Arranjo ... 52
Interação ... 54
Campo de condução ... 55
Programa AHS ... 57
Executando no simulador local ... 58
Analisando os resultados do simulador ... 58
Executando no QuEra Aquila QPU ... 61
Analisando os resultados da QPU ... 63
Próximo ... 64
Construa circuitos no SDK ... 64
Portões e circuitos ... 65
Medição parcial ... 71
qubitAlocação manual ... 72
Compilação literal ... 73
Simulação de ruído ... 74
Inspecionando o circuito ... 75
Tipos de resultados ... 77
Envio de tarefas quânticas para QPUs e simuladores ... 82
Exemplo de tarefas quânticas no Amazon Braket ... 83
Envio de tarefas quânticas para uma QPU ... 89
Executando uma tarefa quântica com o simulador local ... 91
Tratamento em lotes quânticos de tarefas ... 93
Configurar notificações do SNS (opcional) ... 95
Inspecionando circuitos compilados ... 96
Execute seus circuitos com o OpenQASM 3.0 ... 96
O que é o OpenQASM 3.0? ... 97
Quando usar o OpenQASM 3.0 ... 97
Como funciona o OpenQASM 3.0 ... 98

Pré-requisitos 98
Quais recursos do OpenQASM o Braket suporta? 98
Crie e envie um exemplo de tarefa quântica do OpenQASM 3.0 104
Support para OpenQASM em diferentes dispositivos Braket 107
Simule ruídos com o OpenQASM 3.0 119
Qubitreligando com o OpenQASM 3.0 121
Compilação literal com o OpenQASM 3.0 121
O console Braket 122
Mais atributos 122
Gradientes de computação com o OpenQASM 3.0 122
Medindo qubits específicos com o OpenQASM 3.0 123
Envie um programa analógico usando o QuEra Aquila 124
hamiltoniano 124
Esquema do programa Braket AHS 126
Esquema de resultados de tarefas do Braket AHS 131
QuEra esquema de propriedades do dispositivo 138
Trabalhando com o Boto3 143
Ativar o cliente Amazon Braket Boto3 144
Configurar AWS CLI perfis para o Boto3 e o Amazon Braket SDK 147
Controle de pulso no Amazon Braket 150
Pulso de suporte 150
Frames (Quadros) 150
Portas 151
Formas de onda 151
Funções de estruturas e portas 152
Rigetti 152
OQC 154
Olá Pulse 155
Hello Pulse usando OpenPulse 160
Acessando portões nativos usando pulsos 167
Empregos na Amazon Braket Hybrid 169
O que é um Hybrid Job? 170
Quando usar o Amazon Braket Hybrid Jobs 170
Execute seu código local como um trabalho híbrido 171
Crie um trabalho híbrido a partir do código Python local 171
Instale pacotes e código-fonte adicionais do Python 175

Salve e carregue dados em uma instância de trabalho híbrida 176
Melhores práticas para decoradores de trabalho híbridos 11
Execute um trabalho híbrido com o Amazon Braket Hybrid Jobs 179
Crie seu primeiro Hybrid Job 181
Definir permissões 182
Crie e execute 185
Resultados do monitoramento 189
Entradas, saídas, variáveis ambientais e funções auxiliares 191
Entradas 191
Outputs 192
Variáveis de ambiente 193
Funções auxiliares 194
Salve os resultados do trabalho 194
Salve e reinicie trabalhos híbridos usando pontos de verificação 196
Defina o ambiente para seu script de algoritmo 198
Usar hiperparâmetros 200
Configure a instância de trabalho híbrida para executar seu script de algoritmo 202
Cancelar um Hybrid Job 205
Usando compilação paramétrica para acelerar trabalhos híbridos 207
Use PennyLane com o Amazon Braket 208
Amazon Braket com PennyLane 209
Algoritmos híbridos em notebooks de exemplo do Amazon Braket 211
Algoritmos híbridos com PennyLane simuladores incorporados 211
Gradiente adjunto PennyLane com simuladores Amazon Braket 212
Use o Amazon Braket Hybrid Jobs PennyLane e execute um algoritmo QAOA 213
Acelere suas cargas de trabalho híbridas com simuladores incorporados da PennyLane 216
Usando `lightning.gpu` para cargas de trabalho do algoritmo de otimização aproximada quântica 216
Aprendizado de máquina quântico e paralelismo de dados 219
Crie e depure uma tarefa híbrida com o modo local 223
Traga seu próprio contêiner (BYOC) 224
Quando levar meu próprio contêiner é a decisão certa? 225
Receita para trazer seu próprio recipiente 226
Executando trabalhos híbridos do Braket em seu próprio contêiner 232
Configure o bucket padrão em `AwsSession` 232
Interaja com trabalhos híbridos diretamente usando o API 233

Mitigação de erros 237
Mitigação de erros em IonQ Aria 237
Nitidez 238
Braket Direct 239
Reservas 239
Crie uma reserva 240
Execute sua carga de trabalho com uma reserva 241
Cancelar ou reagendar uma reserva existente 245
Conselhos de especialistas 245
Capacidades experimentais 246
Acesso somente para reservas ao IonQ Forte 247
Acesso ao desvio local em Aquila QuEra 247
Acesso a geometrias altas em Aquila QuEra 248
Acesso a geometrias estreitas em Aquila QuEra 248
Registro e Monitoramento 250
Rastreamento de tarefas quânticas a partir do Amazon Braket SDK 250
Monitoramento de tarefas quânticas por meio do console Amazon Braket 253
Marcar recursos 255
Usar tags 255
Mais sobre AWS e tags 256
Recursos suportados no Amazon Braket 256
Restrições de tags 257
Gerenciamento de tags no Amazon Braket 257
Exemplo de marcação de CLI no Amazon Braket 258
Marcar com o Amazon Braket API 259
Eventos Amazon Braket com EventBridge 259
Monitore o status da tarefa quântica com EventBridge 260
Exemplo de evento Amazon Braket EventBridge 261
Monitor com CloudWatch 263
Métricas e dimensões do Amazon Braket 263
Dispositivos compatíveis 264
Fazendo login com CloudTrail 264
Informações sobre o Amazon Braket em CloudTrail 264
Entendendo as entradas do arquivo de log do Amazon Braket 265
Crie um caderno Braket usando CloudFormation 268
Etapa 1: criar um script de configuração do SageMaker ciclo de vida da Amazon 268

Etapa 2: criar a função do IAM assumida pela Amazon SageMaker 269
Etapa 3: criar uma instância de SageMaker notebook da Amazon com o prefixo `amazon-braket-` 270
Registro avançado 271
Segurança 274
Responsabilidade compartilhada pela segurança 274
Proteção de dados 274
Retenção de dados 275
Gerenciando o acesso ao Amazon Braket 276
Recursos do Amazon Braket 276
Cadernos e funções 277
Sobre a AmazonBraketFullAccess política 278
Sobre a AmazonBraketJobsExecutionPolicy política 283
Restringir o acesso do usuário a determinados dispositivos 286
Atualizações do Amazon Braket para políticas gerenciadas AWS 287
Restrinja o acesso do usuário a determinadas instâncias do notebook 288
Restringir o acesso do usuário a determinados buckets do S3 289
Função vinculada ao serviço 290
Permissões de função vinculada ao serviço para o Amazon Braket 291
Resiliência 292
Validação de conformidade 293
Segurança da infraestrutura 293
Segurança de terceiros 294
Enpoints da VPC (PrivateLink) 294
Considerações sobre os endpoints Amazon Braket VPC 295
Configure o Braket e PrivateLink 295
Saiba mais sobre como criar um endpoint 297
Controle o acesso com as políticas de endpoint da Amazon VPC 297
Solução de problemas 299
AccessDeniedException 299
Ocorreu um erro (ValidationException) ao chamar a CreateQuantumTask operação 299
Um recurso do SDK não funciona 300
O trabalho híbrido falha devido a ServiceQuotaExceededException 300
Os componentes pararam de funcionar na instância do notebook 301
Cotas 301
Cotas e limites adicionais 348

Solucionar problemas do OpenQASM 348
Incluir erro de declaração 349
Erro não contíguo qubits 349
Misturando erro físico qubits com qubits erro virtual 349
Solicitando tipos de resultados e medindo qubits no mesmo erro de programa 350
Os limites clássicos e de qubit registro excederam o erro 350
Caixa não precedida por um erro de pragma literal 350
Erro de caixas textuais sem portas nativas 351
Caixas textuais sem erro físico qubits 351
O pragma literal não contém o erro “braket” 351
Erro único qubits não pode ser indexado 352
O físico qubits em um erro de duas qubit portas não está conectado 352
GetDevice não retorna o erro de resultados do OpenQASM 352
Aviso de suporte do simulador local 354
Referência de API e SDK 355
Histórico do documento 356
AWS Glossário 365
..... ccclxvi

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.

AmazonO Braket é totalmente gerenciado AWS service (Serviço da AWS) 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:

- Explore e projete algoritmos quânticos e híbridos.
- Algoritmos de teste em diferentes simuladores de circuitos quânticos.
- Execute algoritmos em diferentes tipos de computadores quânticos.
- Crie 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 notebooks.

O desenvolvimento do Braket tem três estágios: construir, testar e executar:

O Build - Braket fornece ambientes de notebook Jupyter totalmente gerenciados que facilitam o início. Os notebooks Braket são pré-instalados com exemplos de algoritmos, recursos e ferramentas para desenvolvedores, incluindo o SDK do Braket. Amazon Com o SDK do Amazon Braket, 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 alto desempenho (HPC).

Run - Braket fornece acesso seguro e sob demanda a diferentes tipos de computadores quânticos. Você tem acesso a computadores quânticos baseados em portas deIonQ,, e OQCRigetti, 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 não existe nenhum computador quântico universal e tolerante a falhas no momento. 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. A 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, unidades de processamento quântico (QPUs) são usadas como coprocessadores para 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 notebooks que já podem ser aplicados a muitos problemas de prova de conceito, além de certos problemas práticos.

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 termos e conceitos 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. No 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.

Suporte

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.

Trabalho híbrido Braket

AmazonO 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 back-end 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.

Computação quântica baseada em Gate

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.

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 $2^{N \text{ por } 2^{N}}$ 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 pessoas a qubits serem simuladas. O Braket fornece acesso a simuladores de circuitos quânticos gerenciados (acessados por meio do BraketAPI) e locais (parte do SDK do Amazon Braket).

Computador quântico

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. As QPUs podem ser baseadas 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.

Portões compatíveis com QPU

As portas compatíveis com QPU são as portas aceitas pelo dispositivo QPU. Talvez essas portas não possam ser executadas 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 deshots) e outros metadados de solicitação. Você pode criar tarefas quânticas por meio do SDK do Amazon Braket ou usando a CreateQuantumTask API operação 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 Quantum Tasks do console Amazon Braket ou usando as operações GetQuantumTask ou SearchQuantumTasksAPI.

Qubit

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

Queue depth

Queue depthrefere-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) ouAmazon 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 Amazon Braket Hybrid Jobs execução. 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 tasksenviados como parte de um trabalho híbrido têm prioridade e são agregados noPriority Task Queue.

Queue position

Queue positionrefere-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) ouAmazon 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 Serviços da AWS e recursos. 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 menor privilégio 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 intervalo específico, especifique o intervalo na política do IAM em vez de conceder ao funcionário acesso a todos os intervalos do seu. Conta da AWS

Perfis do IAM

Uma função do IAM é uma identidade que você pode assumir para obter acesso temporário às permissões. Antes que um usuário, aplicativo ou serviço possa assumir uma função do IAM, ele deve receber permissões para mudar para a função. Quando alguém assume uma função do IAM, abandona todas as permissões anteriores que tinha em uma função anterior e assume as permissões da nova função.

- Prática recomendada: as funções 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 Amazon S3

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 e políticas de usuário na documentação do Amazon S3.

Preços 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.

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 preços, visite nossa página de preços.

Acompanhamento de custos quase em tempo real

O Braket SDK oferece a opção de adicionar um rastreamento de custos quase em tempo real às suas cargas de trabalho 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 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 levam em consideração nenhum desconto ou crédito, e você pode ter 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 simulador SV1. 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)
```

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.

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

tracker.simulator_tasks_cost()
$0.00375
```

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 0,37 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/Aspen-M-3")
with Tracker() as tracker:
    while tracker.qpu_tasks_cost() < 1:
        result = device.run(circ, shots=200).result()
print(tracker.quantum_tasks_statistics())
print(tracker.qpu_tasks_cost(), "USD")
```

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

> **Note**
>
> O rastreador de custos não rastreará a duração de tarefas TN1 quânticas que falharam. Durante uma TN1 simulação, 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 seguintes melhores práticas para usar o Amazon Braket. Economize tempo, minimize os custos e evite erros comuns.

Verifique 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.
- Recomendamos que você não use sua conta de administrador como forma de conceder ou restringir o acesso do usuário aos dispositivos Amazon Braket.

Definir alarmes de cobrança

- Você pode definir um alarme de cobrança para notificá-lo 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 em AWS Budgets.

Teste tarefas TN1 quânticas com baixa contagem de disparos

- Os simuladores custam menos do que os QHPs, mas certos simuladores podem ser caros se as tarefas quânticas forem executadas com altas contagens de disparos. Recomendamos que você teste suas TN1 tarefas com uma shot contagem baixa. Shota contagem não afeta o custo 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.

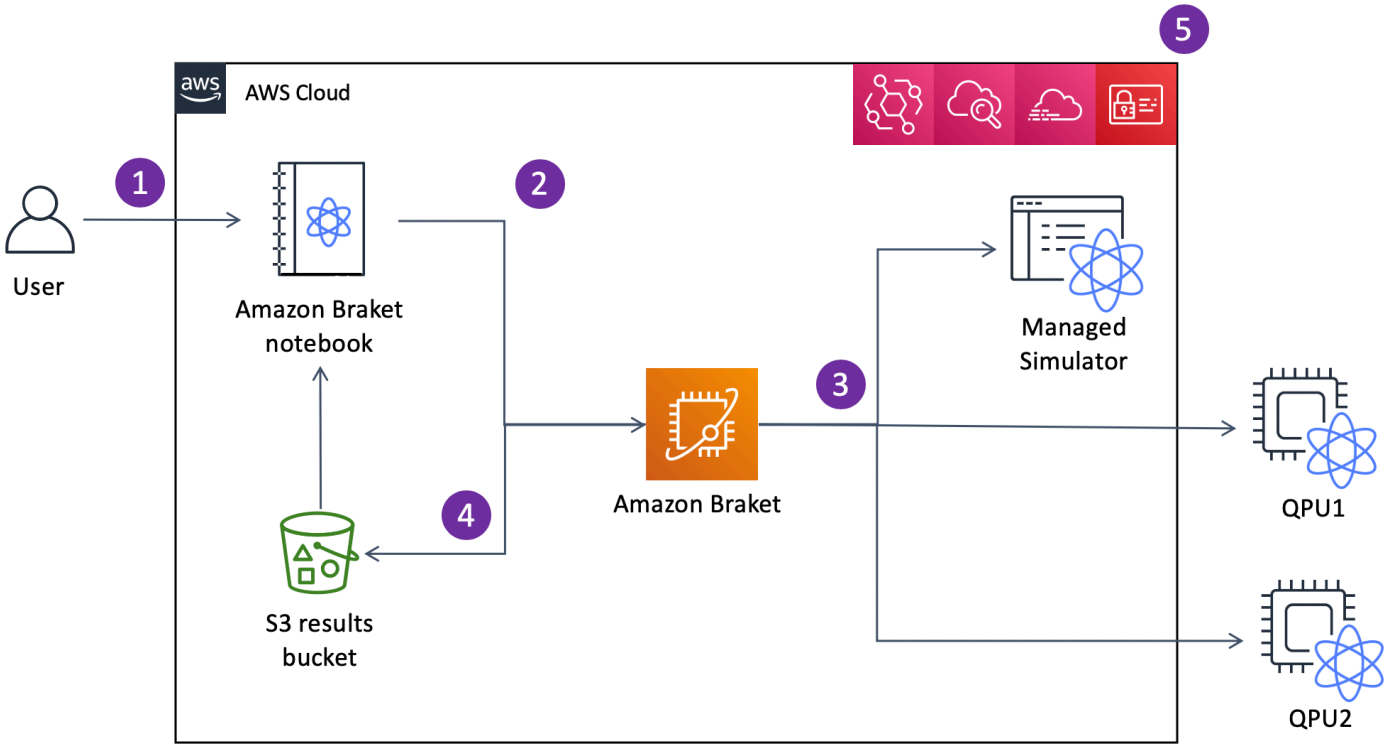
Como funciona o Amazon Braket

> **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 fornece acesso sob demanda a dispositivos de computação quântica, incluindo simuladores de circuito sob demanda e diferentes tipos de QPUs. No Amazon Braket, a solicitação atômica para um dispositivo é uma tarefa quântica. Para dispositivos de controle de qualidade 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.

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

Fluxo de tarefas quânticas do Amazon Braket



Com Jupyter notebooks, você pode definir, enviar e monitorar convenientemente 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. 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 diferentes tipos de QPUs IonQ (Oxford Quantum Circuits (OQC),,,)QuEra, Rigetti três simuladores sob demanda SV1 (,,) DM1TN1, dois simuladores locais e um simulador incorporado. Para saber mais, consulte Dispositivos compatíveis com o Amazon Braket.

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

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

Processamento de dados de 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.

Repositórios e plug-ins principais para 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.

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 notebooks na linguagem de programação Python. Jupyter Depois que seus Jupyter notebooks 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).

Plug-ins

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 oferece 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

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 QPU ou simulador que você pode chamar para executar tarefas quânticas. O Amazon Braket fornece acesso a dispositivos de QPU a IonQ partir deIQM,,, Rigetti e Oxford Quantum CircuitsQuEra, três simuladores sob demanda, três simuladores locais e um simulador incorporado. 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 por meio da API. `GetDevice` Ao construir um circuito com os simuladores, o Amazon Braket atualmente exige que você use qubits ou índices contíguos. Se você estiver trabalhando com o Amazon Braket SDK, terá acesso às propriedades do dispositivo, conforme mostrado no exemplo de código a seguir.

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

device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/sv1')
 #SV1
```

```
# device = LocalSimulator()
 #Local State Vector Simulator
# device = LocalSimulator("default")
 #Local State Vector Simulator
# device = LocalSimulator(backend="default")
 #Local State Vector Simulator
# device = LocalSimulator(backend="braket_sv")
 #Local State Vector Simulator
# device = LocalSimulator(backend="braket_dm")
 #Local Density Matrix Simulator
# device = LocalSimulator(backend="braket_ahs")
 #Local Analog Hamiltonian Simulation
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/tn1')
 #TN1
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/dm1')
 #DM1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Harmony')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet')
 #IQM Garnet
# device = AwsDevice('arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy')
 #OQC Lucy
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/quera/Aquila')
 #QuEra Aquila
# device = AwsDevice('arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3')
 #Rigetti Aspen-M-3

# get device properties
device.properties
```

Fornecedores de hardware quântico compatíveis

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

- Rigetti

Simuladores compatíveis

- Simulador vetorial de estado local (braket_sv) ('Simulador padrão')
- Simulador de matriz de densidade local () `braket_dm`
- Simulador AHS local
- Simulador de vetores de estado () SV1
- Simulador de matriz de densidade () DM1
- Simulador de rede tensora () TN1
- PennyLane's Lightning Simulators

Escolha o melhor simulador para sua tarefa quântica

- Compare simuladores

> **Note**
>
> Para ver o disponível Regiões da AWS para cada dispositivo, role para a direita na tabela a seguir.

Dispositivos Amazon Braket

Provedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
IonQ	Aria 1	baseado em portões	QPU	arn:aws:braket:us-east-1: :dispositivo/QPU/ionq/aria-1	us-east-1
IonQ	Aria 2	baseado em portões	QPU	arn:aws:braket:us-east-1: :dispositivo/QPU/ionq/aria-2	us-east-1

Provedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
IonQ	Forte 1	baseado em portões	QPU (somente reserva)	arn:aws:braket:us-east-1: :dispositivo/QPU/ionq/forte-1	us-east-1
IonQ	Harmony	baseado em portões	QPU	arn:aws:braket:us-east-1: :dispositivo/QPU/ionq/Harmony	us-east-1
IQM	Garnet	baseado em portões	QPU	arn:aws:braket:eu-north-1: :dispositivo/QPU/IQM/Garnet	eu-north-1
Oxford Quantum Circuits	Lucy	baseado em portões	QPU	arn: aws:braket:eu-west-2: :dispositivo/qpu/oqc/Lucy	eu-west-2
QuEra	Aquila	Simulação hamiltoniana analógica	QPU	arn:aws:braket:us-east-1: :dispositivo/qpu/quera/aquila	us-east-1
Rigetti	Aspen M-3	baseado em portões	QPU	arn:aws:braket:us-west-1: :dispositivo/QPU/rigetti/Aspen-M-3	us-west-1
AWS	braket_sv	baseado em portões	Simulador local	N/A (simulador local no Braket SDK)	N/D
AWS	braket_dm	baseado em portões	Simulador local	N/A (simulador local no Braket SDK)	N/D

Provedor	Nome do dispositivo	Paradigma	Tipo	ARN do dispositivo	Região
AWS	SV1	baseado em portões	Simulador sob demanda	arn:aws:braket: ::device/simulador quântico/amazon/sv1	Todas as regiões em que o Amazon Braket está disponível.
AWS	DM1	baseado em portões	Simulador sob demanda	arn:aws:braket: ::device/simulador quântico/amazon/dm1	Todas as regiões em que o Amazon Braket está disponível.
AWS	TN1	baseado em portões	Simulador sob demanda	arn:aws:braket: ::device/simulador quântico/amazon/tn1	us-west-2, us-east-1 e eu-west-2

> **Note**
>
> Certas QPUs só podem ser acessadas usando reservas via Braket Direct, consulte Reservas.

Para ver detalhes adicionais sobre as QPUs que você pode usar com o Amazon Braket, consulte Amazon Braket Hardware Providers.

IonQ

IonQoferece QPUs baseadas em portas baseadas na tecnologia de captura de íons. IonQ'sQPUs de íons aprisionados 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.

IonQos dispositivos suportam 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, as 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

IQMos processadores quânticos são dispositivos universais e do tipo porta baseados em qubits transmon supercondutores. O IQM Garnet dispositivo é um dispositivo de 20 qubits com uma topologia de rede quadrada.

Os IQM dispositivos 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 IQM dispositivos suportam as seguintes portas nativas.

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

Rigetti

Rigettios processadores quânticos são máquinas universais, do tipo porta, baseadas em supercondutores totalmente ajustáveis. qubits O Aspen-M-3 dispositivo de 79 qubits aproveita sua tecnologia proprietária de vários chips e é montado a partir de 2 processadores de 40 qubits.

O Rigetti dispositivo 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, os dispositivos Rigetti suportam as seguintes portas nativas.

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

Rigettiprocessadores 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 Rigetti dispositivos, que suportam um conjunto de quadros predefinidos dos seguintes tipos:

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

Para obter mais informações sobre esses quadros, consulte Funções de quadros e portas.

Oxford Quantum Circuits (OQC)

OQCos processadores quânticos são máquinas universais, modelo de porta, construídas usando a tecnologia Coaxmon escalável. O OQC Lucy sistema é um 8-qubit dispositivo com a topologia de um anel no qual cada um qubit está conectado aos dois vizinhos mais próximos.

O Lucy dispositivo suporta as seguintes portas quânticas.

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

Com a compilação literal, o dispositivo OQC suporta as seguintes portas nativas.

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

O controle de nível de pulso está disponível nos OQC dispositivos. Os OQC dispositivos suportam um conjunto de quadros predefinidos dos seguintes tipos:

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

OQCos dispositivos oferecem suporte à declaração dinâmica de quadros, desde que você forneça um identificador de porta válido. Para obter mais informações sobre esses quadros e portas, consulte Funções de quadros e portas.

> **Note**
>
> Ao usar o controle de pulso comOQC, a duração dos seus programas não pode exceder no máximo 90 microssegundos. A duração máxima é de aproximadamente 50 nanossegundos para portas de qubit único e 1 microssegundo para portas de dois qubit. Esses números podem variar dependendo dos qubits usados, da calibração atual do dispositivo e da compilação do circuito.

QuEra

QuEraoferece 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

Simulador vetorial estadual local () `braket_sv`

O simulador vetorial de estado 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 25qubits), dependendo das especificações de hardware da instância do notebook 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 suportadas 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 notebook 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 12qubits), dependendo das especificações de hardware da instância do notebook 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 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 deshots:

- 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 notebook OpenQASM Local Simulator.

Para saber mais sobre o simulador de matriz de densidade local, consulte o 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 notebook 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

SV1é um simulador vetorial de estado universal, de alto desempenho e sob demanda. Ele pode simular circuitos de até 34qubits. Você pode esperar que um 34-qubit circuito 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.

SV1tem um tempo de execução 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.

SV1resultados

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: = 0 Shots

Para saber mais sobre os resultados, consulte Tipos de resultados.

SV1está sempre disponível, ele opera seus circuitos sob demanda e pode executar vários circuitos em paralelo. O tempo de execução é dimensionado linearmente com o número de operações e exponencialmente com o número de. qubits O número de shots tem um pequeno impacto no tempo de execução. 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 suportadas 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.

DM1tem um tempo de execução máximo de 6 horas, um padrão de 35 tarefas quânticas simultâneas e um máximo de 50 tarefas quânticas simultâneas.

DM1resultados

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 resultados, consulte Tipos de resultados.

DM1está sempre disponível, ele opera seus circuitos sob demanda e pode executar vários circuitos em paralelo. O tempo de execução é dimensionado linearmente com o número de operações e exponencialmente com o número de. qubits O número de shots tem um pequeno impacto no tempo de execução. 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 1.000 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 TN1 poder estimar o tempo de execução do próximo estágio, que é chamado de fase de contração. Se o tempo estimado de contração exceder o limite de tempo de execução da TN1 simulação, TN1 não tente contrair.

TN1tem um limite de tempo de execução de 6 horas. É limitado a um máximo de 10 (5 em eu-west-2) tarefas quânticas simultâneas.

TN1resultados

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 os resultados, consulte Tipos de resultados.

TN1está sempre disponível, ele opera 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 suportadas de um dispositivo nas propriedades do dispositivo.

Visite o GitHub repositório Amazon Braket para obter um exemplo de caderno TN1 para ajudar você a começar a usar. TN1

Práticas recomendadas para trabalhar com TN1

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

Simuladores incorporados

Os simuladores incorporados funcionam incorporando a simulação com o código do algoritmo no mesmo contêiner e executando a simulação diretamente na instância de trabalho híbrida. Isso pode

ser útil para remover gargalos associados à comunicação da simulação com um dispositivo remoto. Isso pode resultar em um uso de memória significativamente menor, número reduzido de execuções de circuitos para alcançar o resultado desejado e um desempenho aprimorado de dez vezes ou mais. Para obter mais informações sobre simuladores incorporados, consulte a página 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 aproveitar métodos avançados de computação de gradientes, como diferenciação adjunta, para avaliar gradientes mais rapidamente. O simulador lightning.qubit está disponível como um dispositivo via 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 notebook Braket Hybrid Jobs para ver um exemplo do uso de lightning.gpu.

Compare simuladores

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 simuladores sob demanda (SV1,,) TN1 DM1

O desempenho dos simuladores locais depende do hardware que hospeda o ambiente local, como uma instância do notebook 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 18qubits, use um simulador local.
- Ao empregar de 18 a 24 anosqubits, escolha um simulador com base na carga de trabalho.
- Ao empregar mais de 24qubits, use um simulador sob demanda.

Para simulações de ruído:

- Ao empregar menos de 9qubits, use um simulador local.
- Ao empregar 9—12qubits, escolha um simulador com base na carga de trabalho.
- Ao empregar mais de 12qubits, useDM1.

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?

DM1simula 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 tempo de execução geralmente é escalonado linearmente com o número de operações e exponencialmente com o número de. qubits

O que é um simulador de rede tensorial?

TN1codifica circuitos quânticos em um gráfico estruturado.

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

Como resultado dessa estrutura, 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, 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 ele 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 deshots. SV1geralmente é mais rápido do que 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, é TN1 possível avaliar circuitos muito maiores em tempo real do que SV1 porque 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, TN1 é muito bom para avaliar circuitos (gráficos) nos quais as portas são de curto alcance (ou seja, cada uma qubit é conectada por portas somente ao vizinho mais próximoqubits) 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 facilmente.

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 tempo de execução 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 100shots, que TN1 decide ser incontratável, mas se você reenviar com apenas 10, a contração prossegue. Nessa situação,

para obter 100 amostras, você pode 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 circuitoTN1, antes de prosseguir com um número maior deshots.

> **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 Amazon Braket não tenta contrações extremamente complexas.

Simultaneidade

Todos os simuladores Braket oferecem a capacidade de executar vários circuitos simultaneamente. Os limites de simultaneidade variam de acordo com o simulador e a região. Para obter mais informações sobre limites de simultaneidade, consulte a página Cotas.

Regiões e endpoints do Amazon Braket

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

Disponibilidade regional do Amazon Braket

Nome da região	Região	Endpoint de suporte	QPU
Leste dos EUA (Norte da Virgínia)	us-east-1	braket.us-east-1.amazonaws.com	Íon Q
Leste dos EUA (Norte da Virgínia)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
Oeste dos EUA (Norte da Califórnia)	us-west-1	braket.us-west-1.amazonaws.com	Rigetti

Nome da região	Região	Endpoint de suporte	QPU
Norte da UE 1 (Estocolmo)	eu-north-1	braket.eu-north-1.amazonaws.com	IQM
EU West 2 (Londres)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Você pode executar o Amazon Braket em qualquer região em que ele esteja disponível, mas cada QPU está disponível somente em uma única região. As tarefas quânticas executadas em um dispositivo QPU podem ser visualizadas no console Amazon Braket na região desse dispositivo. Se você estiver usando o Amazon Braket SDK, poderá enviar tarefas quânticas para qualquer dispositivo QPU, independentemente da região em que estiver trabalhando. O SDK cria automaticamente uma sessão na região para a QPU especificada.

Para obter informações gerais sobre como AWS funciona com regiões e endpoints, consulte AWS service (Serviço da AWS) endpoints na Referência AWS geral.

Quando minha tarefa quântica será executada?

> **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.

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.

Notificações de alteração de status 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 da 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. Crie 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

Alertas de conclusão de tarefa quântica

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, ao enviar uma tarefa grande ou ao enviar uma tarefa fora da janela de disponibilidade de um dispositivo. Se você não quiser esperar a conclusão da tarefa, você pode configurar uma notificação do SNS.

Um caderno Amazon Braket orienta você nas etapas de configuração. Para obter mais informações, consulte o exemplo de caderno Amazon Braket para configurar notificações.

Janelas e status de disponibilidade da QPU

A disponibilidade da QPU varia de dispositivo para dispositivo.

Na página Dispositivos do console do 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 off-line se não estiver disponível para os clientes, independentemente da janela de disponibilidade. Por exemplo, ele pode estar off-line 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 depthrefere-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) ouAmazon 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 Amazon Braket Hybrid Jobs execução. 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 tasksenviados como parte de um trabalho híbrido têm prioridade e são agregados noPriority 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/Harmony")

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


# returns the number of hybrid jobs queued on the device
print(device.queue_depth().jobs)
'3'
```

Enviar uma tarefa quântica ou um trabalho híbrido para uma QPU pode fazer com que sua carga de trabalho fique em um 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 positionrefere-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) ouAmazon 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:us-east-1::device/qpu/ionq/Harmony")

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

# retrieve the queue position information
print(task.queue_position().queue_position)

# Returns the number of Quantum Tasks queued ahead of you
'2'


from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:us-east-1::device/qpu/ionq/Harmony",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=False
)

# retrieve the queue position information
print(job.queue_position().queue_position)
'3' # returns the number of hybrid jobs queued ahead of you
```

Comece a usar 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.

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

As etapas para começar incluem:

- Ativar o Amazon Braket
- Crie uma instância do notebook Amazon Braket
- Execute seu primeiro circuito usando o Amazon Braket Python SDK
- Execute seus primeiros algoritmos quânticos

Ativar 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

Pré-requisitos

Para habilitar e executar o Amazon Braket, você deve ter um usuário ou função 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 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. Criação de funções vinculadas a serviços (SLR)
   b. Habilitando o acesso a computadores quânticos de terceiros
   c. Criação de uma nova instância do notebook 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 Enable 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.

Crie uma instância do notebook 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 notebooks Jupyter totalmente gerenciados para você começar. As instâncias do notebook Amazon Braket são baseadas nas instâncias do notebook Amazon SageMaker . As instruções a seguir descrevem as etapas necessárias para criar uma nova instância de notebook para clientes novos e existentes.

Novos clientes do Amazon Braket

1. Abra o console do Amazon Braket e navegue até a página do painel no painel esquerdo.
2. Clique em Get Started no modal Bem-vindo ao Amazon Braket, localizado no centro da página do seu painel, para fornecer um nome de caderno. Isso criará um notebook Jupyter padrão.
3. Pode levar alguns minutos para criar seu caderno. Seu caderno será listado na página Cadernos com o status Pendente. Quando a instância do notebook estiver pronta para uso, o status mudará para InService. Talvez seja necessário atualizar a página para exibir o status atualizado do notebook.

Clientes existentes do Amazon Braket

1. Abra o console Amazon Braket, selecione Notebooks no painel esquerdo e escolha Create notebook instance. Se você não tiver cadernos, selecione a configuração Padrão para criar um caderno Jupyter padrão e insira o nome de uma instância do Notebook usando somente caracteres alfanuméricos e hífen e selecione seu modo visual preferido. Em seguida, ative ou desative o gerenciador de inatividade do seu notebook.
   a. Se ativado, selecione o tempo de inatividade desejado antes que o notebook seja reiniciado. Quando um notebook é reiniciado, as cobranças de computação param de ser cobradas, mas as cobranças de armazenamento continuarão.
   b. Para visualizar o tempo ocioso restante na instância do notebook, navegue até a barra de comando e selecione a guia Braket, seguida pela guia Inactivity Manager.

> **Note**
>
> Para evitar que seu trabalho seja perdido, considere integrar sua instância do SageMaker notebook a um repositório git. Como alternativa, mover seu trabalho para fora das `/Braket Examples` pastas `/Braket Algorithms` e evitará que o arquivo seja sobrescrito pela reinicialização da instância do notebook.

2. (Opcional) Com a Configuração avançada, você pode criar um notebook com permissões de acesso, configurações adicionais e configurações de acesso à rede:
   a. Na configuração do Notebook, escolha seu tipo de instância. O tipo de instância padrão e econômico, ml.t3.medium, é escolhido por padrão. Para saber mais sobre os preços das instâncias, consulte os SageMaker preços da Amazon. Se você quiser associar um repositório público do Github à sua instância do notebook, clique na lista suspensa do repositório Git e selecione Clonar um repositório git público a partir do URL no menu suspenso Repositório. Insira a URL do repositório na barra de texto da URL do repositório Git.
   b. Em Permissões, configure todas as funções opcionais do IAM, acesso root e chaves de criptografia.
   c. Em Rede, defina configurações personalizadas de rede e acesso para sua Jupyter Notebook instância.
3. Revise suas configurações, defina quaisquer tags para identificar a instância do seu notebook e clique em Launch.

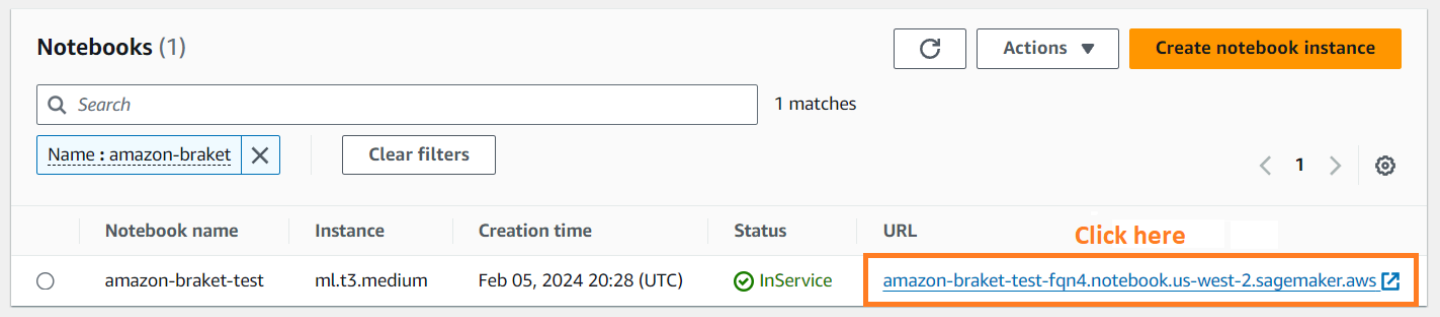
> **Note**
>
> Você pode visualizar e gerenciar suas instâncias de notebook Amazon Braket nos consoles Amazon Braket e Amazon. SageMaker Configurações adicionais do notebook Amazon Braket estão disponíveis no console. SageMaker

Se você estiver trabalhando no console do Amazon Braket, dentro do SDK do AWS Amazon Braket, os plug-ins estão pré-carregados nos cadernos que você criou. Se quiser executar em sua própria máquina, você pode instalar 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 uso com PennyLane plug-ins.

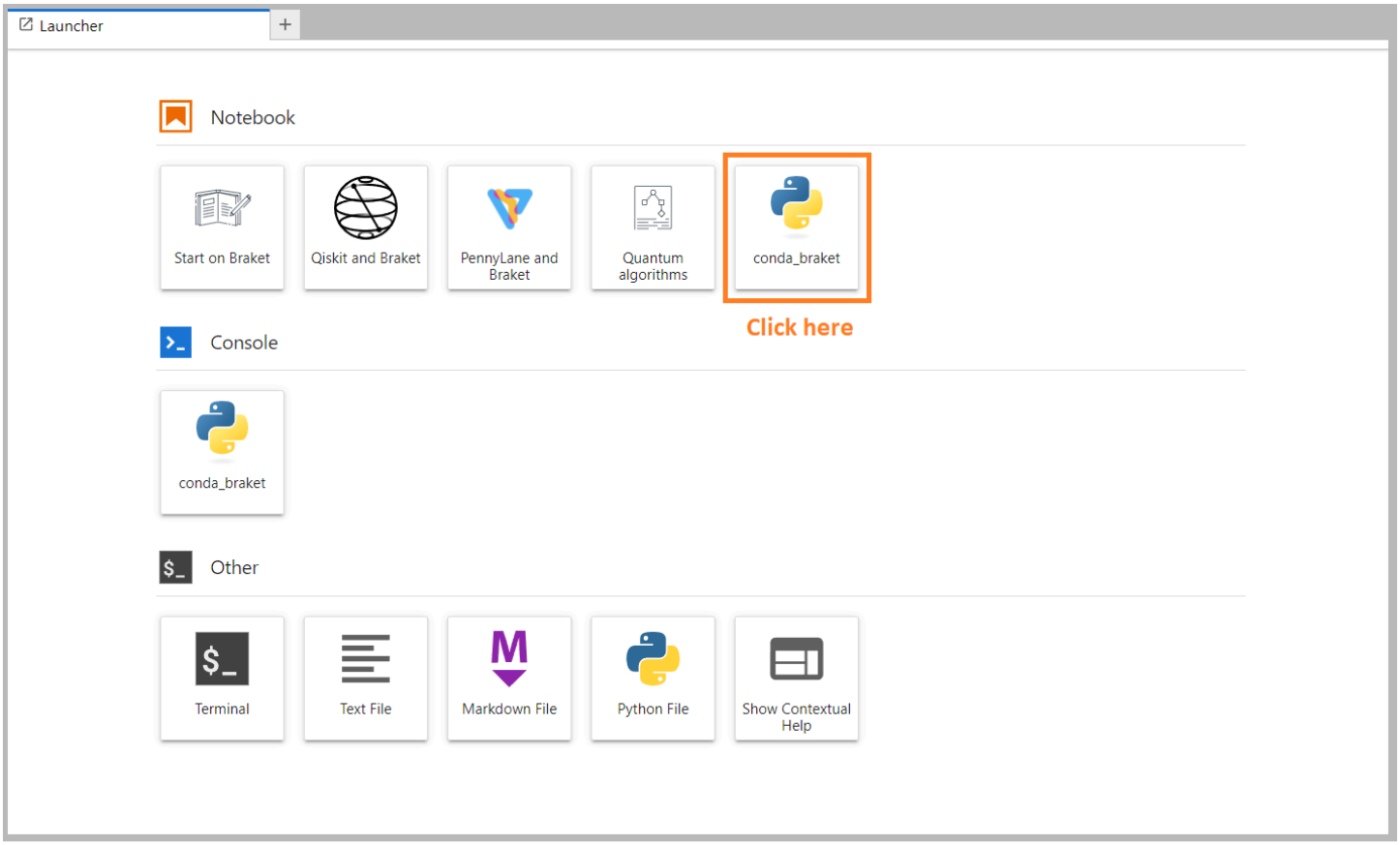
Execute seu primeiro circuito usando o Amazon Braket Python SDK

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



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



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 os módulos do SDK do Amazon Braket e definindo um circuito Bell State simples.

```
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)
```

Você pode visualizar o circuito com este comando:

```
print(bell)
```

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).

Veja como instanciar o simulador local:

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

e execute o circuito:

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

Você deve ver um resultado mais ou menos assim:

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

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

Execute seu circuito em um simulador sob demanda

AmazonO Braket também fornece acesso a um simulador de alto desempenho sob demanda para operar circuitos maioresSV1. 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 SDK do Braket 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 `example-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 Introdução ao Amazon S3.

```
# get the account ID
aws_account_id = boto3.client("sts").get_caller_identity()["Account"]
# the name of the bucket
my_bucket = "example-bucket"
# the name of the folder in the bucket
my_prefix = "simulation-output"
s3_folder = (my_bucket, my_prefix)
```

Para executar um circuitoSV1, você deve fornecer a localização do bucket S3 que você selecionou anteriormente como argumento posicional na `.run()` chamada.

```
# 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**
>
> Após 90 dias, o Amazon Braket remove automaticamente todos os IDs de 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 QPU a dispositivos IonQ deOxford Quantum Circuits,, e. QuEra Rigetti 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 Rigetti dispositivo.

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

Escolha um IonQ dispositivo com este código:

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

Depois de selecionar um dispositivo e antes de executar sua carga de trabalho, 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)
'2' # returns the number of hybrid jobs queued on the device
```

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 `poll_timeout_seconds` parâmetro no `.run()` comando, 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 `task.result()` função.

```
# 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)
```

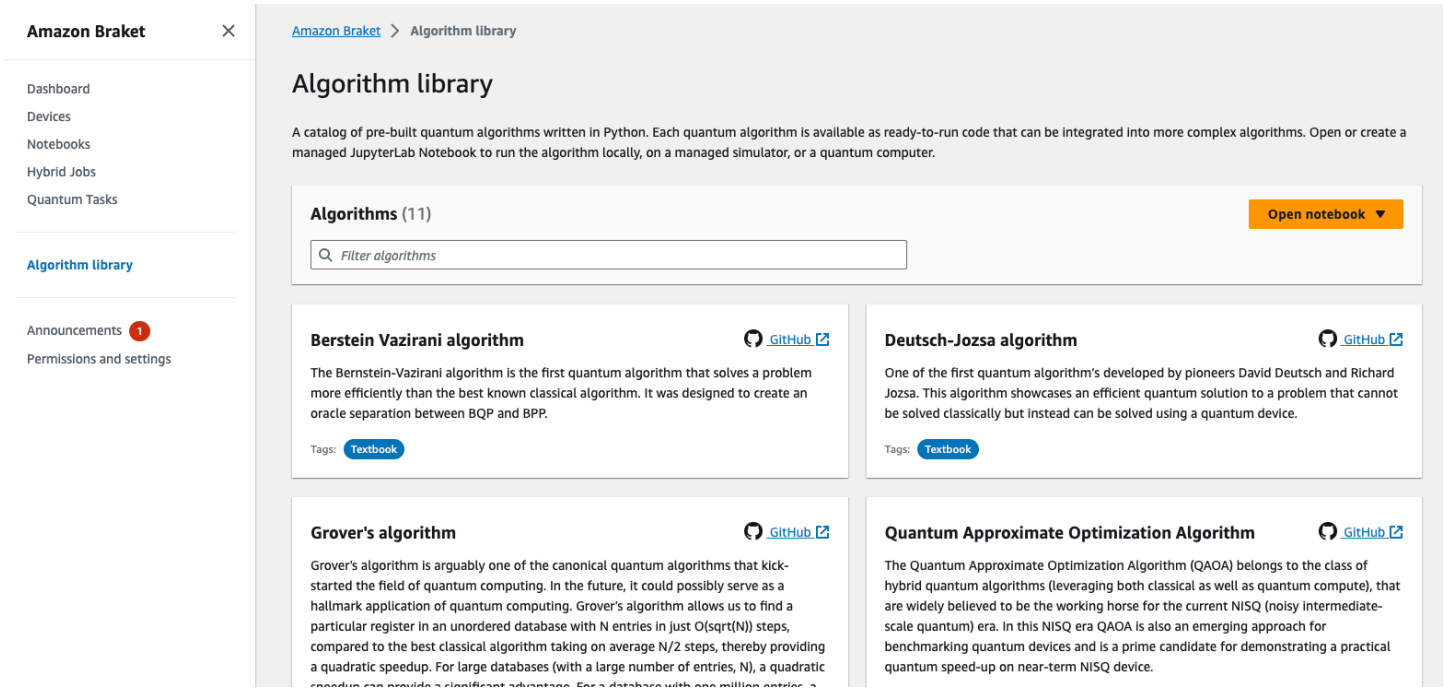
Além disso, depois de enviar sua tarefa quântica ou trabalho híbrido, você pode chamar a `queue_position()` função para verificar a posição da fila.

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

Execute seus primeiros algoritmos quânticos

> **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.

A biblioteca de algoritmos Amazon Braket é um catálogo de algoritmos quânticos pré-criados escritos em Python. Você pode executar esses algoritmos como estão ou usá-los como ponto de partida para criar algoritmos mais complexos. Você pode acessar a biblioteca de algoritmos no console do Braket. Você também pode acessar a biblioteca de algoritmos Braket no Github: https://github.com/aws-samples/. amazon-braket-algorithm-library



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 notebook, poderá encontrar a biblioteca de algoritmos Braket na pasta raiz do seu notebook.

Trabalhe com o Amazon Braket

Esta seção mostra como projetar circuitos quânticos, enviar esses problemas como tarefas quânticas para dispositivos e monitorar as tarefas quânticas com o Amazon Braket SDK.

A seguir estão os principais meios de interagir com recursos no Amazon Braket.

- 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 e do console. O SDK pode ser acessado por meio de notebooks Amazon Braket pré-configurados.
- A API Amazon Braket pode ser acessada por meio do SDK e dos notebooks Amazon Braket Python. Você pode fazer chamadas diretamente para o API se estiver criando aplicativos que funcionam com a computação quântica de forma programática.

Os exemplos desta seção demonstram como você pode trabalhar com o Amazon Braket API diretamente usando o SDK do Braket Python junto com o SDK do Python para Amazon Braket (AWS Boto3).

Mais sobre o SDK do Amazon Braket Python

Para trabalhar com o SDK do Amazon Braket Python, primeiro instale o SDK do AWS Python para Braket (Boto3) para que você possa se comunicar com o. AWS API Você pode pensar no SDK Amazon Braket Python como um complemento conveniente do 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 SDK Amazon Braket Python 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 notebooks, que são pré-carregados com todos os módulos e dependências de que você precisa para executar tarefas quânticas.
- Você pode importar módulos do SDK do Amazon Braket Python para qualquer script Python se não quiser trabalhar com notebooks.

Depois de instalar o Boto3, uma visão geral das etapas para criar uma tarefa quântica por meio do SDK 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:

- Olá AHS: Execute sua primeira simulação hamiltoniana analógica
- Construa circuitos no SDK
- Envio de tarefas quânticas para QPUs e simuladores
- Execute seus circuitos com o OpenQASM 3.0
- Envie um programa analógico usando o QuEra Aquila
- Trabalhando com o Boto3

Olá AHS: Execute sua primeira simulação hamiltoniana analógica

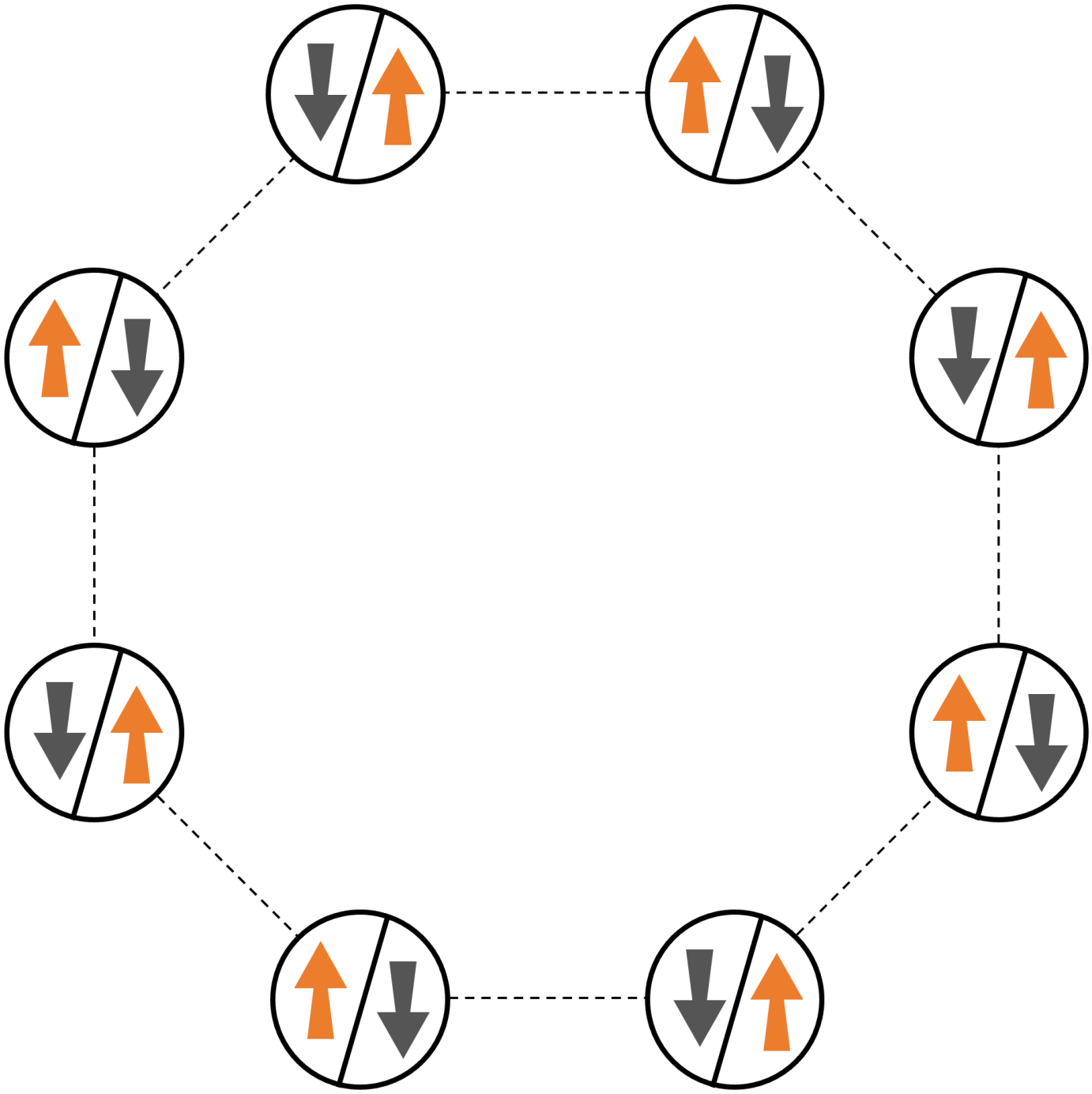
TEM

A Simulação Hamiltoniana Analógica (AHS) é um paradigma de computação quântica diferente dos circuitos quânticos: em vez de uma sequência de portas, cada uma atuando 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.

Dispositivos quânticos capazes de realizar AHS ajustarão seus parâmetros (por exemplo, amplitude e desajuste de um campo de condução coerente) para aproximar de perto a evolução temporal do sistema quântico sob o hamiltoniano personalizado. O paradigma AHS é adequado para simular propriedades estáticas e dinâmicas de sistemas quânticos de muitas partículas em interação. QPUs criadas especificamente, como o dispositivo Aquila da, QuEra podem simular a evolução temporal de sistemas com tamanhos que, de outra forma, seriam inviáveis em hardware clássico.

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

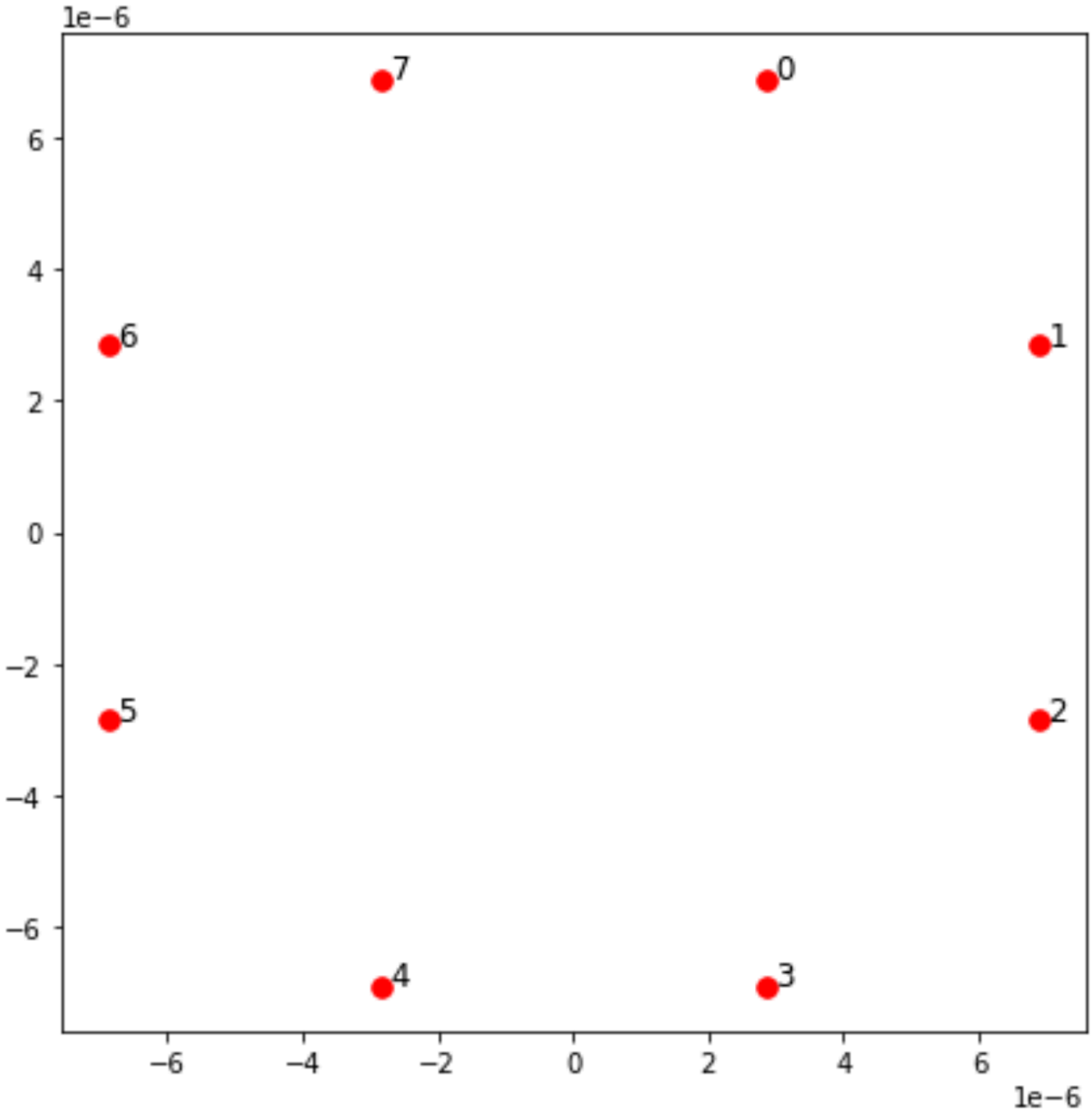
Usaremos um átomo neutro para representar cada spin, e os estados de spin “para cima” e “para baixo” serão codificados no estado excitado de Rydberg e no estado fundamental dos átomos, respectivamente. Primeiro, criamos o arranjo 2D. Podemos programar o anel de giros acima com o código a seguir.

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

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

from braket.ahs.atom_arrangement import AtomArrangement

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

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

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 Aquila dispositivo deQuEra). 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 n =↑ #↑) é o mesmo operador de antes. k k k 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.

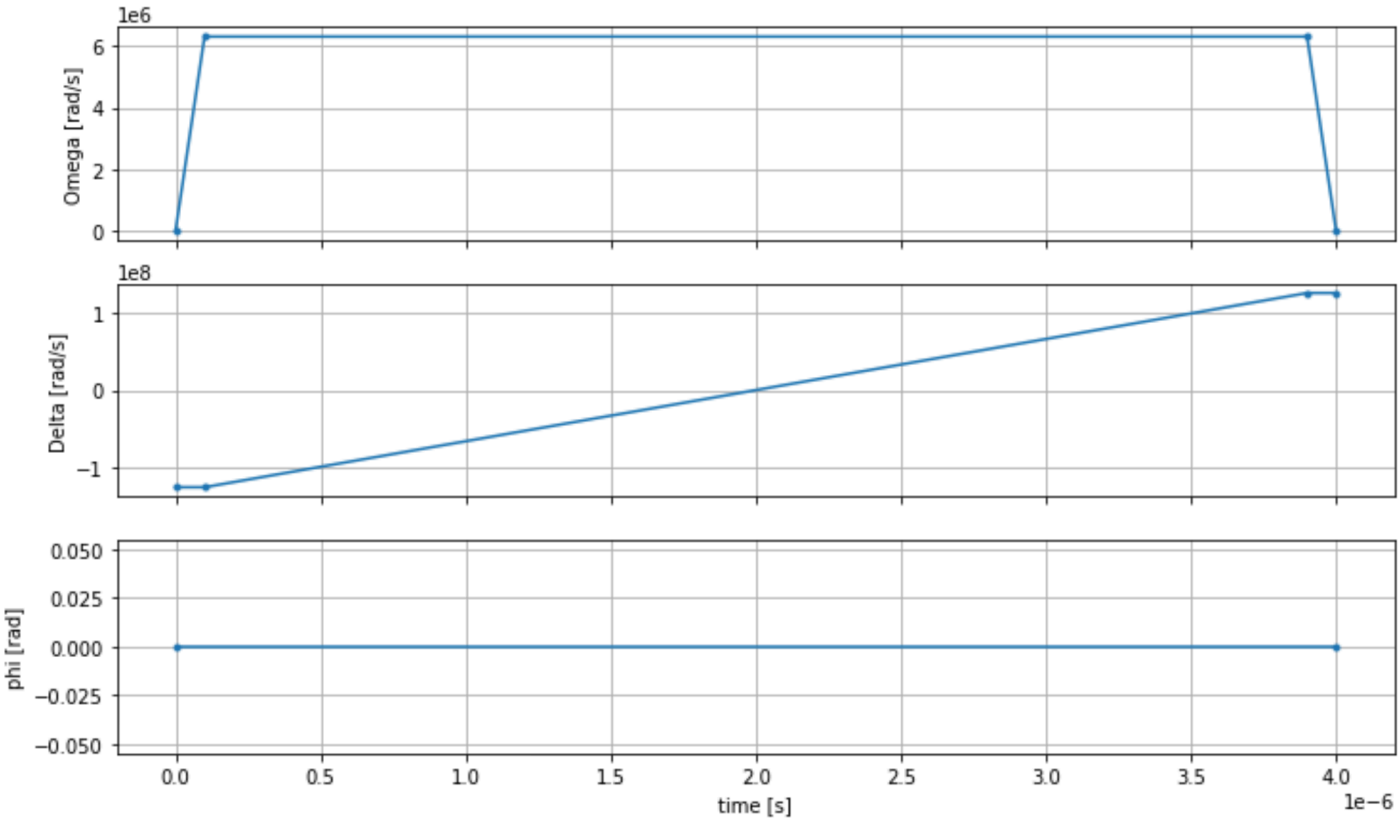
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField
```

```
# 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 tempo de execução 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
```

Analisando os resultados do simulador

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
```

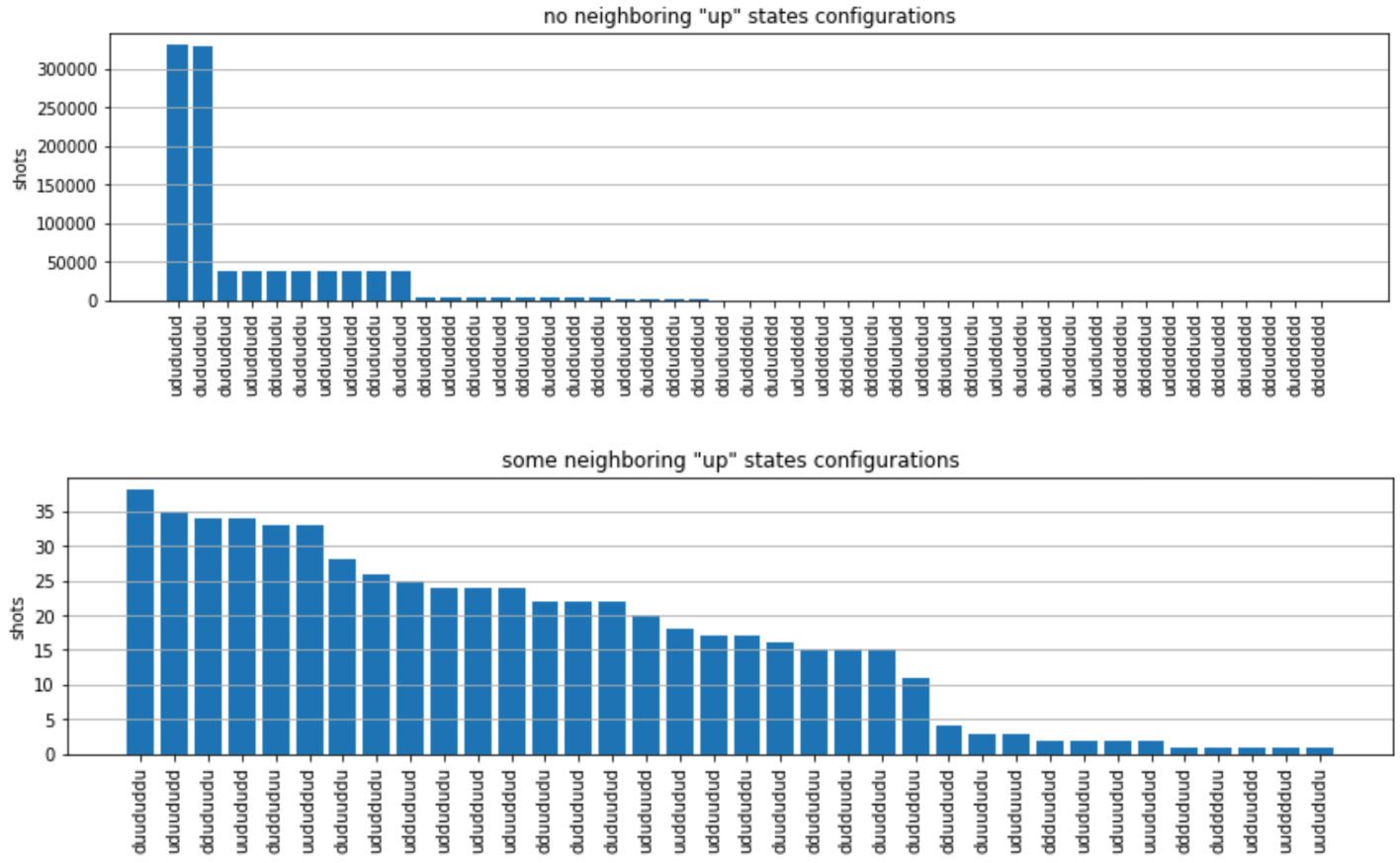
```
    Returns
        dict: number of times each state configuration is measured

    """
    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)
```

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

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 e. `"dudududu"` `"udududud"`
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 Get Started.

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

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

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

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

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

> **Note**
>
> A execução desse programa no Aquila processador 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}")
```

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

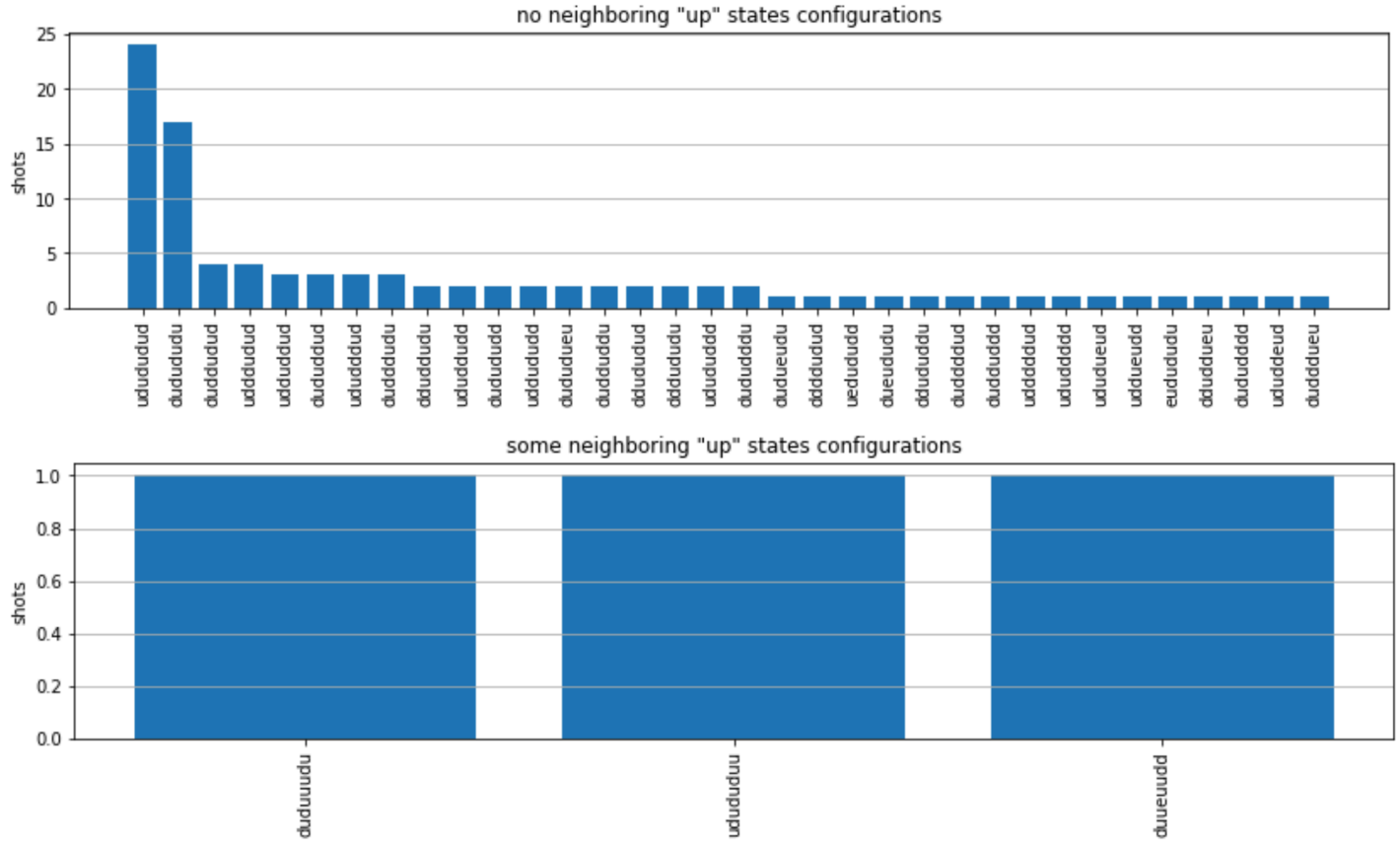
Usando as mesmas `get_counts` funções de antes, podemos calcular as contagens:

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

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

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 Aquila QPU. Além disso, os resultados coincidem com a simulação dentro da flutuação estatística esperada devido ao pequeno número de disparos.

Próximo

Parabéns, agora você executou sua primeira carga de trabalho 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 Aquila dispositivo, consulte nossos exemplos de notebooks.

Construa circuitos no SDK

Esta seção fornece exemplos de definição de um circuito, visualização das portas disponíveis, extensão de um circuito e visualização das portas suportadas por cada dispositivo. Ele também

contém instruções sobre como alocar manualmentequbits, 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 determinadas QPUs. Para obter mais informações, consulte Pulse Control no Amazon Braket.

Nesta seção:

- Portões e circuitos
- Medição parcial
- qubitAlocação manual
- Compilação literal
- Simulação de ruído
- Inspecionando o circuito
- Tipos de resultados

Portões e circuitos

As portas e circuitos quânticos são definidos na `braket.circuits`classe do SDK Amazon Braket Python. 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 (rotuladoq`0`,, `q1q2`, `eq3`) consistindo em portas Hadamard padrão de um qubit e portas CNOT de dois qubits. Você pode visualizar esse circuito chamando a `print` função 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-C---
        |
q1 : -H-|-C-
```

```
        | |
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)
```

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 novos portões, como o `Ry` portão controlado.

```
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 os portões disponíveis

O exemplo a seguir mostra como ver 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', 'GPi', 'GPi2', 'H', 'I', 'ISwap', 'MS', 'PSwap',
 'PhaseShift', 'PulseGate', 'Rx', 'Ry', 'Rz', 'S', 'Si', 'Swap', 'T', 'Ti', 'Unitary',
 'V', 'Vi', 'X', 'XX', 'XY', 'Y', 'YY', 'Z', 'ZZ']
```

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

> **Note**
>
> As portas são anexadas no local, 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. An `Instruction` é uma diretiva quântica que descreve a tarefa quântica a ser executada em um dispositivo quântico. `Instruction`os operadores incluem `Gate` somente objetos do tipo.

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

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

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

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

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

# instructions can be copied
new_instr = instr.copy()
# appoint the instruction to target
new_instr = instr.copy(target=[5])
new_instr = instr.copy(target_mapping={0: 5})
```

Exemplo: veja os portões 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 suportadas de um dispositivo nas propriedades do dispositivo. O seguinte mostra um exemplo com um dispositivo IonQ:

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

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

# get device name
device_name = device.name
# show supportedQuantumOperations (supported gates for a device)
device_operations = device.properties.dict()['action']['braket.ir.openqasm.program']
['supportedOperations']
```

```
print('Quantum Gates supported by {}:\n {}'.format(device_name, device_operations))
```

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

As portas suportadas podem precisar 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 suportadas 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 qubit portas 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/Aspen-M-3")

#specify the qubits
a=10
b=113
print(f"Fidelity of the XY gate between qubits {a} and {b}: ",
 device.properties.provider.specs["2Q"][f"{a}-{b}"]["fXY"])
```

Medição parcial

Seguindo os exemplos anteriores, medimos todos os qubits no circuito quântico. No entanto, é possível medir qubits individuais ou um subconjunto de qubits.

Exemplo: medir um subconjunto de qubits

Neste exemplo, demonstramos uma medição parcial adicionando uma `measure` instrução com os qubits de destino ao final do circuito.

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

# Define an example bell circuit and measure qubit 0
```

```
circuit = Circuit().h(0).cnot(0, 1).measure(0)

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

# Get the results
result = task.result()

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

qubitAlocação manual

Ao executar um circuito quântico em computadores quânticos a partir deRigetti, você pode, opcionalmente, usar a qubit alocação manual 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. qubits

A qubit alocação manual permite que você execute circuitos com maior precisão e investigue qubit propriedades 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 explicitamentequbits.

```
circ = Circuit().h(0).cnot(0, 7)  # Indices of actual qubits in the QPU
my_task = device.run(circ, s3_location, shots=100, disable_qubit_rewiring=True)
```

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

> **Note**
>
> O OQC compilador não oferece suporte à configuraçãodisable_qubit_rewiring=True. Definir esse sinalizador como True gera o seguinte erro:An error occurred (ValidationException) when calling the CreateQuantumTask operation: Device arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy does not support disabled qubit rewiring.

Compilação literal

Quando você executa um circuito quântico em computadores quânticos a partir de RigettiIonQ,, ou Oxford Quantum Circuits (OQC), 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 (suportado porRigetti,IonQ, eOQC) conforme especificado ou que somente partes específicas dele sejam preservadas (suportadas somente porRigetti). Ao desenvolver algoritmos para benchmarking de hardware ou protocolos de mitigação de erros, você precisa ter a opção de especificar exatamente os layouts de portas e circuitos que você está executando 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.

Atualmente, a compilação literal é suportada em dispositivos RigettiIonQ, e Oxford Quantum Circuits (OQC) 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 conectadas qubits e que o circuito use as portas nativas suportadas no hardware. O exemplo a seguir mostra como acessar programaticamente a lista de portas nativas suportadas por um dispositivo.

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

PoisRigetti, a qubit reconexão 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 que não a suporta, um erro é gerado indicando que uma operação não suportada 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/Aspen-M-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 notebook e simuladores sob

demanda com base nas taxas atuais, conforme especificado na página de preços do Amazon Braket. Para obter mais informações, consulte o caderno de exemplo de compilação Verbatim.

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

Simulação de ruído

Para instanciar o simulador de ruído local, você pode alterar o back-end da seguinte maneira.

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

Você pode criar 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 simples com ruído despolarizador e um canal Kraus personalizado.

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

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

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

```
# Inject noise approach
# define phase damping noise
```

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

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

Exemplo 1

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

Ou

Exemplo 2

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

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

Inspecionando o circuito

Os circuitos quânticos em Amazon Braket têm um conceito de pseudo-tempo chamado. `Moments` Cada um qubit pode experimentar um único portão por`Moment`. O objetivo `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 os portões são executados em uma QPU.

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

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

```
print('Total circuit depth:', circ.depth)
```

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

q0 : -Rx(0.15)-C----------X-
                |
q1 : -Ry(0.2)--|-ZZ(0.15)---
               | |
q2 : ----------X-|----------
                 |
q3 : ------------ZZ(0.15)---

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

A profundidade total do circuito acima é 3 (mostrada como momentos 0, 1, e 2). Você pode verificar a operação do portão a cada momento.

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

- A chave é`MomentsKey()`, que contém pseudo-tempo e qubit informações.
- 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)]))
Instruction('operator': Rx('angle': 0.15, 'qubit_count': 1), 'target':
 QubitSet([Qubit(0)]))

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

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

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

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

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

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

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

q0 : -S-Z---
        |
q1 : ---C-H-

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

Tipos de resultados

AmazonO Braket pode retornar diferentes tipos de resultados quando um circuito é medido usando. `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 `Observable` classe 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 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 especificados serão retornadas. qubits
- `Reduced density matrix`retorna uma matriz de densidade para um subsistema de destino especificado qubits de um sistema dequbits. Para limitar o tamanho desse tipo de resultado, 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 especificado qubit definido e observável. Se nenhum alvo for especificado, o observável deve operar somente em 1 qubit e é aplicado a todos qubits em paralelo. Se os alvos forem especificados, o número de alvos especificados deverá ser igual ao número qubits sobre os quais o observável atua.
- `Variance`retorna a variância (`mean([x-mean(x)]`^{2}`)`) do qubit conjunto 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 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 suportados para diferentes dispositivos:

	Sim local	SV1	DM1	TN1	Rigetti	IonQ	OQC
Gradiente adjunto	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
Expectativa	Y	Y	Y	Y	Y	Y	Y
Probabilidade	Y	Y	Y	N	Y*	Y	Y

Matriz de densidade reduzida	Y	N	Y	N	N	N	N
Vetor de estado	Y	N	N	N	N	N	N
Amostra	Y	Y	Y	Y	Y	Y	Y
Variação	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> * suporta Rigetti apenas tipos de resultados de probabilidade de até 40qubits.

Você pode verificar os tipos de resultados suportados examinando as propriedades do dispositivo, conforme mostrado no exemplo a seguir.

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

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

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

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

```
from braket.circuits import Observable

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

```
circ.state_vector()
circ.variance(observable=Observable.Z(), target=0)

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

> **Note**
>
> Alguns dispositivos fornecem medições (por exemploRigetti) como resultados e outros fornecem probabilidades como resultados (por exemplo IonQ eOQC). O SDK fornece uma propriedade de medição nos resultados, mas para os dispositivos que retornam probabilidades, ela é pós-computada. Assim, dispositivos como os fornecidos por IonQ e OQC têm resultados de medição determinados pela probabilidade, uma vez que as medições por disparo não são retornadas. Você pode verificar se um resultado foi pós-computado visualizando o `measurements_copied_from_device` no objeto resultante, conforme mostrado neste arquivo.

Observáveis

AmazonBraket inclui uma `Observable` classe, que pode ser usada para especificar um observável a ser medido.

Você pode aplicar no máximo uma única não identidade observável a cada um. qubit Se você especificar dois ou mais observáveis sem identidade diferentes para o mesmoqubit, verá um erro. Para tanto, cada fator de um produto tensorial conta como um indivíduo observável, portanto, é permitido ter vários produtos tensoriais atuando sobre o mesmoqubit, desde que o fator que atua sobre ele seja o mesmo. qubit

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

A `Observable` classe inclui os seguintes observáveis.

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

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

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

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

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

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

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

Parâmetros

Os circuitos podem incluir parâmetros livres, que você pode usar de forma “construir uma vez - executar várias vezes” e para calcular gradientes. Os parâmetros livres têm um nome codificado por string que você pode usar para especificar seus valores ou determinar se deve ser diferenciado em relação a eles.

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

Para os parâmetros que você deseja diferenciar, especifique-os usando seu nome (como uma string) ou por referência direta. Observe que o cálculo do gradiente usando o tipo de `AdjointGradient` resultado é feito com relação ao valor esperado do observável.

Nota: Se você fixou os valores dos parâmetros livres passando-os como argumentos para o circuito parametrizado, executar um circuito com o tipo de `AdjointGradient` resultado e os parâmetros especificados produzirá um erro. Isso ocorre porque os parâmetros que estamos usando para diferenciar não estão mais presentes. Veja o exemplo a seguir.

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

Envio de tarefas quânticas para QPUs e simuladores

AmazonO Braket fornece acesso a vários dispositivos que podem executar tarefas quânticas. Você pode enviar tarefas quânticas individualmente ou configurar o agrupamento de tarefas quânticas.

QPUs

Você pode enviar tarefas quânticas para QPUs a qualquer 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.

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

- Rigetti Aspen-M-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3`

Simuladores

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

Nesta seção:

- Exemplo de tarefas quânticas no Amazon Braket
- Envio de tarefas quânticas para uma QPU
- Executando uma tarefa quântica com o simulador local
- Tratamento em lotes quânticos de tarefas
- Configurar notificações do SNS (opcional)
- Inspecionando circuitos compilados

Exemplo de tarefas quânticas no Amazon Braket

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

Nesta seção:

- Especifique o dispositivo
- Envie um exemplo de tarefa quântica
- Envie uma tarefa parametrizada
- Especifique shots
- Enquete para obter resultados
- Veja os resultados de exemplo

Especifique o dispositivo

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

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

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', <DeviceActionType.JAQCD: 'braket.ir.jaqcd.program'>)
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
 'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', 'h', 'i', 'iswap', 'pswap',
 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'unitary', 'v', 'vi',
 'x', 'xx', 'xy', 'y', 'yy', 'z', 'zz'])
('supportedResultTypes', [ResultType(name='Sample', observables=['x', 'y', 'z', 'h',
 'i', 'hermitian'], minShots=1, maxShots=100000), ResultType(name='Expectation',
 observables=['x', 'y', 'z', 'h', 'i', 'hermitian'], minShots=0, maxShots=100000),
 ResultType(name='Variance', observables=['x', 'y', 'z', 'h', 'i', 'hermitian'],
 minShots=0, maxShots=100000), ResultType(name='Probability', observables=None,
 minShots=1, maxShots=100000), ResultType(name='Amplitude', observables=None,
 minShots=0, maxShots=0)])
```

Envie um exemplo de tarefa quântica

Envie um exemplo de tarefa quântica para ser executada no simulador sob demanda.

```
# create a circuit with a result type
circ = Circuit().rx(0, 1).ry(1, 0.2).cnot(0,2).variance(observable=Observable.Z(),
 target=0)
# add another result type
circ.probability(target=[0, 2])

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

# submit the quantum task to run
my_task = device.run(circ, s3_location, shots=1000, poll_timeout_seconds = 100,
 poll_interval_seconds = 10)
# the positional argument for the S3 bucket is optional if you want to specify a bucket
 other than the default

# get results of the quantum task
result = my_task.result()
```

O `device.run()` comando cria uma tarefa quântica por meio da `CreateQuantumTask` 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. Nesse caso, o dispositivo éSV1. 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.

Envie uma tarefa parametrizada

Os simuladores e QPUs locais e sob demanda do Amazon Braket também oferecem suporte à especificação de valores 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, onde as chaves são os nomes dos parâmetros.

A compilação paramétrica pode melhorar o desempenho da execução de circuitos paramétricos em determinadas 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 tempos de execução 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 todas as QPUs supercondutoras baseadas em portas de Rigetti Computing e Oxford Quantum Circuits com exceção 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})
```

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 emTN1.)

- Para valores diferentes de zeroshots, o simulador extrai amostras da distribuição de saída para emular o shot ruído de QPUs reais. Os dispositivos QPU permitem somente shots > 0.

Para obter informações sobre o número máximo de disparos por tarefa quântica, consulte Braket Quotas.

Enquete para obter resultados

Durante a execuçãomy_task.result(), o SDK começa a pesquisar um resultado com os parâmetros que você define na criação da tarefa quântica:

- poll_timeout_secondsé 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, ou seja, 5 dias.
- Nota: Para dispositivos QPU como Rigetti eIonQ, 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_secondsé a frequência com que a tarefa quântica é pesquisada. Ele especifica com que frequência você liga para o Braket API para obter o status quando a tarefa quântica é executada no simulador sob demanda e em dispositivos QPU. O valor padrão 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]
 [1 0 1]
```

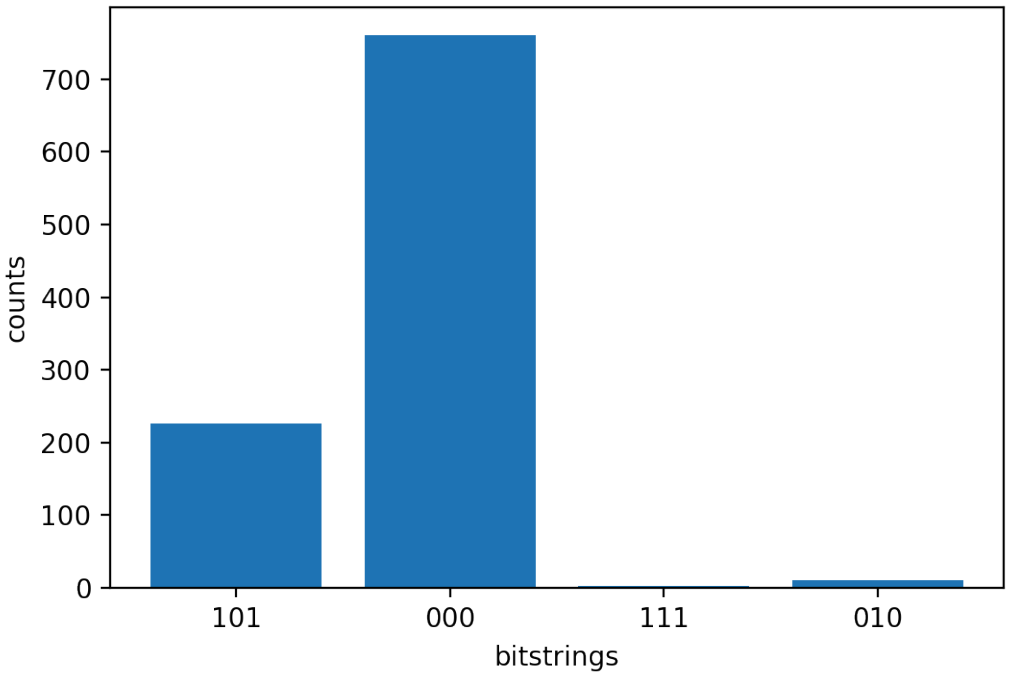
```
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Counts for collapsed states:
 Counter({'000': 761, '101': 226, '010': 10, '111': 3})
Probabilities for collapsed states:
 {'101': 0.226, '000': 0.761, '111': 0.003, '010': 0.01}
```

Veja os resultados de exemplo

Como você também especificou o`ResultType`, você pode ver os resultados retornados. Os tipos de resultados 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])

# you can plot the result and do some analysis
import matplotlib.pyplot as plt
plt.bar(result.measurement_counts.keys(), result.measurement_counts.values());
plt.xlabel('bitstrings');
plt.ylabel('counts');
```

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



Envio de tarefas quânticas para uma QPU

AmazonO Braket permite que você execute um circuito quântico em um dispositivo QPU. O exemplo a seguir mostra como enviar uma tarefa quântica para Rigetti nossos IonQ dispositivos.

Escolha o Rigetti Aspen-M-3 dispositivo 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/Aspen-M-3")

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

```
{'fullyConnected': False,
 'connectivityGraph': {'0': ['1', '7'],
  '1': ['0', '16'],
  '2': ['3', '15'],
  '3': ['2', '4'],
  '4': ['3', '5'],
  '5': ['4', '6'],
```

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

O dicionário anterior `connectivityGraph` contém informações sobre a conectividade do Rigetti dispositivo atual.

Escolha o IonQ Harmony dispositivo

Para o IonQ Harmony dispositivo, o `connectivityGraph` está vazio, conforme mostrado no exemplo a seguir, porque o dispositivo oferece all-to-allconectividade. Portanto, um detalhado não `connectivityGraph` é necessário.

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

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

```
{'fullyConnected': True, 'connectivityGraph': {}}
```

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 Rigetti dispositivos IonQ e compilam automaticamente o circuito fornecido em seus respectivos conjuntos de portas nativas e mapeiam os qubit índices abstratos para físicos qubits na respectiva QPU.

> **Note**
>
> Os dispositivos QPU têm capacidade limitada. Você pode esperar tempos de espera mais longos quando a capacidade for atingida.

AmazonO 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` seu ID exclusivo de tarefa quântica.

Executando uma tarefa quântica com o simulador local

Você pode enviar tarefas quânticas diretamente para um simulador local para prototipagem e testes rápidos. Esse simulador é executado em seu ambiente local, então 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 shots parâmetro.

> **Note**
>
> A velocidade de execução e o número máximo que qubits o simulador local pode processar dependem do tipo de instância do notebook 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 `measure` instrução 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)
```

Tratamento em lotes quânticos de tarefas

O agrupamento de tarefas quânticas está disponível em todos os dispositivos Amazon Braket, exceto no simulador local. O agrupamento em lotes é especialmente útil para tarefas quânticas que você executa nos simuladores sob demanda (TN1ouSV1) porque eles podem processar várias tarefas quânticas em paralelo. Para ajudá-lo a configurar várias tarefas quânticas, o Amazon Braket fornece exemplos de notebooks.

O processamento em lotes permite que você inicie tarefas quânticas em paralelo. Por exemplo, se você quiser fazer um cálculo que exija 10 tarefas quânticas e os circuitos dessas tarefas quânticas sejam independentes uns dos outros, é uma boa ideia usar lotes. Dessa forma, você não precisa esperar que uma tarefa quântica seja concluída antes que outra tarefa comece.

O exemplo a seguir mostra como executar um lote de tarefas quânticas:

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

Para obter mais informações, consulte os exemplos do Amazon Braket sobre o agrupamento de tarefas do Quantum, que tem informações mais específicas GitHub sobre o agrupamento em lotes.

Sobre o agrupamento e os 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 `run_batch` atribuição antes de iniciar um lote de tarefas quânticas. Não recomendamos o uso TN1 com`run_batch`.
- TN1podem incorrer em custos por falhas nas tarefas da fase de ensaio (consulte a descrição do TN1 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 o uso TN1 (consulte Quantum Task Batching, Linha 186).

Tratamento por lotes de tarefas quânticas e PennyLane

Aproveite o agrupamento em lotes ao usar PennyLane no Amazon Braket configurando `parallel = True` quando você instancia um dispositivo Amazon Braket, conforme mostrado no exemplo a seguir.

```
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 `inputs` dicionário, que é usado para todas as tarefas quânticas do lote, ou um dicionário `list` de entrada. Nesse caso, o -ésimo dicionário é emparelhado com a `i` -ésima tarefa, conforme mostrado no exemplo a `i` seguir.

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

# create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')

# create two circuits
circ_a = Circuit().rx(0, alpha).ry(1, alpha).cnot(0,2).xx(0, 2, beta)
circ_a.variance(observable=Observable.Z(), target=0)

circ_b = Circuit().rx(0, alpha).rz(1, alpha).cnot(0,2).zz(0, 2, beta)
```

```
circ_b.expectation(observable=Observable.Z(), target=2)

# use the same inputs for both circuits in one batch

tasks = device.run_batch([circ_a, circ_b], inputs={'alpha': 0.1, 'beta':0.2})

# or provide each task its own set of inputs

inputs_list = [{'alpha': 0.3,'beta':0.1}, {'alpha': 0.1,'beta':0.4}]

tasks = device.run_batch([circ_a, circ_b], inputs=inputs_list)
```

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)
```

Configurar notificações do SNS (opcional)

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 notebook 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.

Inspecionando circuitos compilados

Quando um circuito é executado em um dispositivo de hardware, ele deve ser compilado em um formato aceitável, como transpilar o circuito até as portas nativas suportadas pela QPU. Inspecionar a saída real compilada pode ser muito útil para fins de depuração. Você pode visualizar esse circuito para ambos Rigetti e OQC dispositivos usando o código abaixo.

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

> **Note**
>
> Hoje, você não pode visualizar seu circuito compilado para IonQ dispositivos.

Execute seus circuitos com o OpenQASM 3.0

AmazonO Braket agora suporta o 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 fornecer diretamente strings do OpenQASM 3.0 a todos os dispositivos baseados em portas com a API Amazon Braket e o SDK Amazon Braket Python.

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 suportadas 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 funciona o OpenQASM 3.0
- Pré-requisitos
- Quais recursos do OpenQASM o Braket suporta?
- Crie e envie um exemplo de tarefa quântica do OpenQASM 3.0
- Support para OpenQASM em diferentes dispositivos Braket
- Simule ruídos com o OpenQASM 3.0
- Qubitreligando com o OpenQASM 3.0
- Compilação literal com o OpenQASM 3.0
- O console Braket
- Mais atributos
- 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 simples.

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 funciona o OpenQASM 3.0

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 Braket. API 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 deve ter a versão v1.8.0 dos esquemas do Amazon Braket Python e a versão 1.17.0 ou superior do SDK do Amazon Braket Python.

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 OpenQASM suportados
- Declarações OpenQASM suportadas
- Suporte os pragmas do OpenQASM
- Suporte de recursos avançados para o OpenQASM no simulador local

- Operações e gramática suportadas com OpenPulse

Tipos de dados OpenQASM suportados

Os seguintes tipos de dados do OpenQASM são suportados pelo Braket. Amazon

- 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 do portão:
  - `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 `im` notação 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 OpenQASM suportadas

As seguintes declarações do OpenQASM são apoiadas pelo Braket. Amazon

- `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 físicoqubits: `$0`
- Portões e operações compatíveis em um dispositivo:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> As portas suportadas 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. Portões nativos e físicos qubits são obrigatórios em caixas textuais.

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

- Medição e atribuição de medições em qubits ou em um qubit registro completo.
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`

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

Suporte os pragmas do OpenQASM

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

- 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 do tipo de resultado
  - 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])`
    - Variação: `#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 vs e vs. `qreg` `creg` `qubit` `bit` Também há diferenças na sintaxe de medição, e elas precisam ser suportadas com a sintaxe correta.

Suporte de recursos avançados para o OpenQASM no simulador local

`LocalSimulator`Ele 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 suportada no`LocalSimulator`:

- Modificadores de portão
- Portões embutidos OpenQASM
- Variáveis clássicas
- Operações clássicas
- Portões personalizados
- Controle clássico
- Arquivos QASM
- 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 SDK Amazon Braket Python, o Boto3 ou o para enviar tarefas quânticas do OpenQASM 3.0 AWS CLI para um dispositivo Braket. Amazon

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 simples (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 Braket com o código Amazon a seguir.

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

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

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

# You can also specify an optional s3 bucket location and number of shots,
 # if you so choose, when running the program
```

```
s3_location = ("amazon-braket-my-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
)
```

Use o Boto3 para criar tarefas quânticas do OpenQASM 3.0

Você também pode usar o AWS Python SDK for 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 = "amazon-braket-my-bucket"
s3_prefix = "openqasm-tasks"

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

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

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

```
    outputS3Bucket=my_bucket,
    outputS3KeyPrefix=s3_prefix,
)
```

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

Support para OpenQASM em diferentes dispositivos Braket

Para dispositivos que suportam o OpenQASM 3.0, o `action` campo suporta uma nova ação por meio da `GetDevice` resposta, conforme mostrado no exemplo a seguir para os Rigetti dispositivos 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 `pulse` campo é exibido na `GetDevice` resposta. Os exemplos a seguir mostram esse `pulse` campo para Rigetti os OQC dispositivos e.

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

```
        },
        {
          "name": "iq",
          "type": "complex",
          "optional": false
        }
      ]
    },
    ...
  },
  "ports": {
    "q0_ff": {
      "portId": "q0_ff",
      "direction": "tx",
      "portType": "ff",
      "dt": 1e-9,
      "centerFrequencies": [
        375000000
      ]
    },
    ...
  },
  "supportedFunctions": {
    "shift_phase": {
      "functionName": "shift_phase",
      "arguments": [
        {
          "name": "frame",
          "type": "frame",
          "optional": false
        },
        {
          "name": "phase",
          "type": "float",
          "optional": false
        }
      ]
    },
   ...
  },
  "frames": {
    "q0_q1_cphase_frame": {
      "frameId": "q0_q1_cphase_frame",
      "portId": "q0_ff",
```

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

// OQC

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

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

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

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 `OpenQASM 3.0` programa 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 do porto. 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 (QuBitMappings)
  - Os qubits associados à porta especificada.
- 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 `OpenQASM 3.0` programa 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 (frequência central)
  - 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.
- Portão associado (Portão associado)
  - Os portões associados ao quadro fornecido.
- Mapeamentos de Qubit (QuBitMappings)
  - Os qubits associados ao quadro 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 ou não ser declarado `cal` ou `defcal` bloqueado por meio da OpenPulse `newframe` função. Se isso for falso, somente os quadros listados na estrutura do quadro poderão ser usados no programa.

SupportedFunctions:

Descreve as OpenPulse funções suportadas 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
- 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 Gaussian

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

`τ`é o comprimento da forma de onda, σ é a largura da gaussiana, β é um parâmetro livre e `A` é a amplitude. Se configurada como`True`, `ZaE` a Gaussiana de Remoção Derivada por Porta Adiabática (DRAG) é deslocada e redimensionada de forma que seja igual a zero no início e no final da forma de onda, e a parte real atinja o máximo. `A` Para obter mais informações sobre a forma de onda DRAG, consulte o artigo Simple Pulses for Elimination of Leakage in Weakly Nonlinear Qubits.

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

SupportsLocalPulseElements:

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

SupportsNonNativeGatesWithPulses:

Descreve se podemos ou não usar portas não nativas em combinação com programas de pulso. Por exemplo, não podemos usar uma porta não nativa como uma `H` porta em um programa sem primeiro definir a porta de entrada `defcal` para o qubit usado. Você pode encontrar a lista de `nativeGateSet` chaves 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
- Comprimento/duração mínima do pulso em segundos
- Comprimento/duração máxima do pulso 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 `braket.ir.openqasm.program` chave no `action` campo na saída de recursos do dispositivo. Por exemplo, a seguir estão as operações suportadas e os tipos de resultados disponíveis para o simulador Braket State Vector. SV1

```
...
  "action": {
    "braket.ir.jaqcd.program": {
      ...
    },
 "braket.ir.openqasm.program": {
      "version": [
        "1.0"
      ],
```

```
    "actionType": "braket.ir.openqasm.program",
    "supportedOperations": [
      "ccnot",
      "cnot",
      "cphaseshift",
      "cphaseshift00",
      "cphaseshift01",
      "cphaseshift10",
      "cswap",
      "cy",
      "cz",
      "h",
      "i",
      "iswap",
      "pswap",
      "phaseshift",
      "rx",
      "ry",
      "rz",
      "s",
      "si",
      "swap",
      "t",
      "ti",
      "v",
      "vi",
      "x",
      "xx",
      "xy",
      "y",
      "yy",
      "z",
      "zz"
    ],
    "supportedPragmas": [
      "braket_unitary_matrix"
    ],
    "forbiddenPragmas": [],
    "maximumQubitArrays": 1,
    "maximumClassicalArrays": 1,
    "forbiddenArrayOperations": [
      "concatenation",
      "negativeIndex",
      "range",
```

```
        "rangeWithStep",
        "slicing",
        "selection"
      ],
      "requiresAllQubitsMeasurement": true,
      "supportsPhysicalQubits": false,
      "requiresContiguousQubitIndices": true,
      "disabledQubitRewiringSupported": false,
      "supportedResultTypes": [
        {
          "name": "Sample",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
          ],
          "minShots": 1,
          "maxShots": 100000
        },
        {
          "name": "Expectation",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
          ],
          "minShots": 0,
          "maxShots": 100000
        },
        {
          "name": "Variance",
          "observables": [
            "x",
            "y",
            "z",
            "h",
            "i",
            "hermitian"
```

```
          ],
          "minShots": 0,
          "maxShots": 100000
        },
        {
          "name": "Probability",
          "minShots": 1,
          "maxShots": 100000
        },
        {
          "name": "Amplitude",
          "minShots": 0,
          "maxShots": 0
        }
        {
          "name": "AdjointGradient",
          "minShots": 0,
          "maxShots": 0
        }
      ]
    }
  },
...
```

Simule ruídos com o OpenQASM 3.0

Para simular o ruído com o OpenQASM3, você usa instruções pragma 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 para todos os operadores de ruído pragmática suportados 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 Kraus, lembre-se do seguinte:

- O número de não qubits 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 2 matrizes $^{2*num\_qubits}$. 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.

Qubitreligando com o OpenQASM 3.0

AmazonO Braket suporta a qubit notação física no OpenQASM em Rigetti dispositivos (para saber mais, consulte esta página). Ao usar o físico qubits com a estratégia de religação ingênua, certifique-se de que qubits eles estejam conectados ao dispositivo selecionado. Como alternativa, se qubit os registros 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 a partir de RigettiOQC,IonQ, e, 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, IonQ e OQC só suportam 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 um pragma literal em torno de uma caixa de código que é intocada e não otimizada pela rotina de compilação de baixo nível do hardware. O exemplo de código a seguir mostra como usar `#pragma braket verbatim` o.

```
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 mais informações sobre compilação literal, consulte o caderno de amostra de compilação Verbatim.

O console Braket

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

Mais atributos

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

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

Gradientes de computação com o OpenQASM 3.0

O Amazon Braket oferece suporte a gradientes de computação em simuladores sob demanda e locais no modo (exato) usando o método de `shots=0` diferenciação adjunta. Você pode fornecer o pragma apropriado para especificar o gradiente que deseja calcular, conforme mostrado 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 todos os parâmetros individualmente, você também pode especificar `all` no pragma. Isso calcula o gradiente em relação a todos os `input` parâmetros listados. Isso pode ser conveniente quando o número de parâmetros é muito grande. Nesse caso, o pragma será semelhante ao exemplo a seguir.

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

Todos os tipos observáveis são suportados, incluindo operadores individuais, produtos tensores, observáveis hermitianos e. `Sum` O operador que você deseja usar para calcular o gradiente deve ser encapsulado no `expectation()` encapsulador e os qubits nos quais cada termo atua devem ser especificados.

Medindo qubits específicos com o OpenQASM 3.0

O simulador vetorial de estado local e o simulador de matriz de densidade local oferecem suporte ao envio de OpenQASM programas nos quais um subconjunto dos qubits do circuito pode ser medido. Isso 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.

```
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];
"""
```

Existem dois qubits, `q[0]` `q[1]` mas estamos medindo apenas o qubit 0 aqui:. `b[0] = measure q[0]` Agora, execute o seguinte no simulador vetorial estadual local.

```
from braket.devices import LocalSimulator

local_sim = LocalSimulator()
```

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

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

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

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

Envie um programa analógico usando o QuEra Aquila

Esta página fornece uma documentação abrangente sobre os recursos da Aquila máquina a partir deQuEra. Os detalhes abordados aqui são os seguintes: 1) O hamiltoniano parametrizado simulado porAquila, 2) parâmetros do programa AHS, 3) conteúdo do resultado do AHS, 4) parâmetro de capacidades. Aquila Sugerimos usar a pesquisa de texto Ctrl+F para encontrar parâmetros relevantes para suas perguntas.

hamiltoniano

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

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

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

where

- $H_{\text{drive},k}(t) = \left({}^{1}/_{2}\Omega(t)\, e^{i\phi(t)}\, S_{-,k} + {}^{1}/_{2}\Omega(t)\, e^{-i\phi(t)}\, S_{+,k} \right) n_{k}$

  - 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 elevação do spin do átomo k (na base |↓# =|g#, |↑ ⁵ =|r#, eles são S =|g#r|, S =(S) † =|rcb#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 ($_k$coordenadas x_k e y de cada átomo k, em unidades de um), que controla as distâncias atômicas 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, os operadores S_-, S_+, n são fixos) nem a força do coeficiente de interação de Rydberg-Rydberg (C). $_6$

Esquema do programa Braket AHS

objeto braket.ir.ahs.Program_v1.Program (exemplo)

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

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

JSON (exemplo)

```
{
    "braketSchemaHeader": {
        "name": "braket.ir.ahs.program",
        "version": "1"
    },
    "setup": {
        "ahs_register": {
            "sites": [
                [0E-7, 0E-7],
                [0E-7, 4E-6],
                [4E-6, 0E-7],
            ],
            "filling": [1, 1, 1]
        }
    },
    "hamiltonian": {
        "drivingFields": [
            {
                "amplitude": {
                    "time_series": {
                        "values": [0.0, 15700000.0, 15700000.0, 0.0],
```

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

Campos principais

Campo do programa	tipo	description
setup.ahs_register.sites	Lista [Lista [Decimal]]	Lista de coordenadas 2-d em que as pinças capturam átomos

Campo do programa	tipo	description
setup.ahs_register.filling	Lista [int]	Marca os átomos que ocupam os locais da armadilha com 1 e os locais vazios com 0
hamiltonian.drivingFields [] .amplitude.time_series.times	Lista [Decimal]	pontos de tempo da amplitude de condução, Omega (t)
hamiltonian.drivingFields [] .amplitude.time_series.values	Lista [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	Lista [Decimal]	pontos temporais da fase de condução, phi (t)
hamiltonian.drivingFields [] .phase.time_series.values	Lista [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	tipo	description
hamiltonian.drivingFields [] .detuning.time_series.times	Lista [Decimal]	pontos de tempo do desajuste de direção, delta_global (t)
hamiltonian.drivingFields [] .detuning.time_series.values	Lista [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	Lista [Decimal]	pontos temporais do fator dependente do tempo da magnitude de desajuste local, delta_local (t)
hamiltonian.localDetuning [] .magnitude.time_series.values	Lista [Decimal]	valores do fator dependente do tempo da magnitude de desajuste local, delta_local (t)

Campo do programa	tipo	description
hamiltonian.localDetuning [] .magnitude.pattern	Lista [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	tipo	description
braketSchemaHeader.nome	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
        }
    }
}
```

Campos principais

Campo de resultado da tarefa	tipo	description
medições [] .shotResult.Presequence	Lista [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
medições [] .shotResult.postSequence	Lista [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	tipo	description
		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	tipo	description
braketSchemaHeader.nome	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
ID de metadados da tarefa	str	A identific ação da tarefa quântica. Para tarefas AWS

Campo de resultado da tarefa	tipo	description
		quânticas, essa é a tarefa quântica ARN.
Metadados da tarefa. 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.Criado em	str	O timestamp da criação; o formato deve estar no formato de string ISO-8601/ RFC3339 YYYY-MM-DDTHH:mm:ss.sssz. O padrão é Nenhum.

Campo de resultado da tarefa	tipo	description
taskmetadata.shots.Endedat	str	O timestamp de quando a tarefa quântica terminou; o formato deve estar no formato de string ISO-8601/ RFC3339 YYYY-MM-DDTHH:mm:ss.sssz. O padrão é Nenhum.
TaskMetadata.shots.status	str	O status da tarefa quântica (CRIADA, ENFILEIRADA, EM EXECUÇÃO, CONCLUÍDA, FALHA). O padrão é Nenhum.
taskmetadata.shots.Motivo da falha	str	O motivo da falha da tarefa quântica. O padrão é Nenhum.

Campo de resultado da tarefa	tipo	description
Metadados adicionais. Ação	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
medições [] .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 do serviço

Campo de propriedades do serviço	tipo	description
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	tipo	description
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	float	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

Campo de metadados	tipo	description
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

Campo de metadados	tipo	description
service.deviceDocumentation.ImageURL	str	URL para a imagem do dispositivo
Service.DeviceDocumentation.Resumo	str	breve descrição no dispositivo
Documentação do serviço.dispositivo. externalD ocumentationUrl	str	URL de documenta ção externa
Serviço. Localização do dispositivo	str	localização geográfica do dispositivo
Serviço. Atualizado em	datetime	hora em que as propriedades do dispositivo foram atualizadas pela última vez

Trabalhando com o 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 fornece acesso orientado a objetos e API de baixo nível ao Braket. Amazon

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 você quiser usar métodos auxiliares adicionais, também precisará instalar o SDK do Amazon Braket.

Por exemplo, quando você liga`CreateQuantumTask`, o SDK do Amazon Braket 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 Amazon Braket SDK

Ativar o cliente Amazon Braket Boto3

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

> **Note**
>
> Para compatibilidade com versões anteriores do BraketSchemas, as informações do OpenQASM são omitidas das chamadas. `GetDevice` API Para obter essas informações, o agente de usuário precisa apresentar uma versão recente do BraketSchemas (1.8.0 ou posterior). O SDK do Braket relata isso automaticamente para você. Se você não vê resultados do OpenQASM na `GetDevice` resposta ao usar um SDK do Braket, talvez seja necessário definir a variável de `AWS_EXECUTION_ENV` ambiente para configurar o agente de usuário. Consulte os exemplos de código fornecidos no tópico de erro GetDevice não retorna resultados do OpenQASM para saber como fazer isso com os AWS CLI SDKs Boto3 e Go, Java e//. JavaScript TypeScript

```
import boto3
import botocore

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

Agora que você tem um `braket` cliente 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.

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

- Cancelar tarefa quântica

Pesquise 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']}")
```

Cancelar 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 Amazon Braket SDK

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

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 essa CLI `profile` e como incorporar esse perfil ao Amazon Braket para que as API chamadas sejam feitas com as permissões desse perfil.

Nesta seção:

- Etapa 1: configurar um local AWS CLIprofile
- Etapa 2: Estabelecer um objeto de sessão do Boto3
- Etapa 3: incorporar a sessão de Boto3 ao Braket AwsSession

Etapa 1: configurar um local AWS CLI`profile`

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

- Conceitos básicos
- Configurando o AWS CLI para usar AWS IAM Identity Center

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

Etapa 2: Estabelecer um objeto de sessão do 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 API chamadas 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 do 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 `AwsDevice` objeto instanciado (chamando o `device.run(…)` comando, por exemplo). Todas as API chamadas feitas por esse dispositivo podem aproveitar as credenciais do IAM associadas ao perfil da CLI que você designou anteriormente. `profile`

Controle de pulso no Amazon Braket

Esta seção explica como usar o controle de pulso em várias QPUs no Amazon Braket.

Nesta seção:

- Pulso de suporte
- Funções de estruturas e portas
- Olá Pulse
- Acessando portões nativos usando pulsos

Pulso de suporte

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 por meio das APIs do Braket. Primeiro, vamos apresentar alguns conceitos-chave para controle de pulso no Braket.

Frames (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.pulse import Frame
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")
drive_frame = device.frames["q0_rf_frame"]

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

Portas

Uma porta é uma abstração de software que representa qualquer componente de hardware de entrada/saída 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)
```

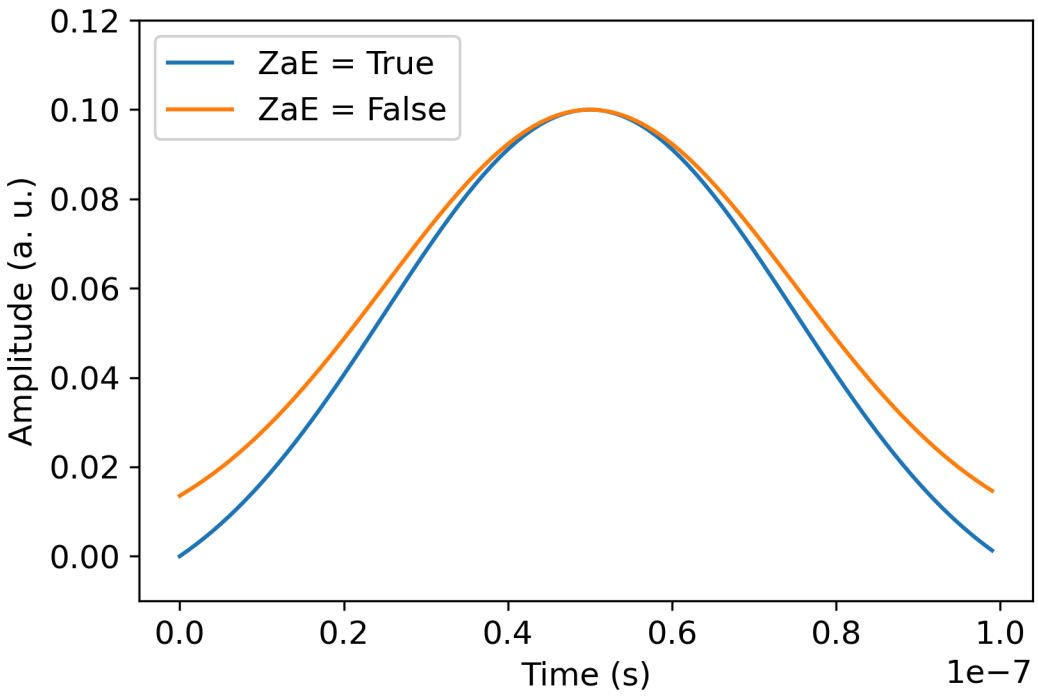
Formas de onda

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
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 `sample` função para desenhar a forma da onda, conforme mostrado no exemplo a seguir.

```
gaussian_waveform = GaussianWaveform(1e-7, 25e-9, 0.1)
x = np.arange(0, gaussian_waveform.length, drive_frame.port.dt)
plt.plot(x, gaussian_waveform.sample(drive_frame.port.dt))
```



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`

Agora que abordamos os conceitos básicos para acesso em nível de pulso, a seguir veremos como construir um circuito usando portas e pulsos.

Funções de estruturas e portas

Esta seção descreve os quadros e portas predefinidos disponíveis para cada dispositivo. Também discutiremos brevemente os mecanismos envolvidos quando os pulsos são reproduzidos em determinados quadros.

Rigetti

Frames (Quadros)

Rigettios dispositivos suportam quadros predefinidos que têm sua frequência e fase calibradas para estarem em ressonância com o qubit associado. A convenção de nomenclatura é `q{i}[_q{j}]_{role}_frame` onde `{i}` se refere ao primeiro número de qubit, `{j}` se refere ao segundo número de qubit, caso o quadro sirva para ativar uma interação de dois qubits, e `{role}` se refere à função do quadro. As funções são as seguintes:

- `rf`é o quadro para conduzir a transição 0-1 do qubit. Os pulsos são transmitidos como sinais transitórios de micro-ondas de frequência e fase fornecidos anteriormente por meio das funções `set` e. `shift` A amplitude do sinal dependente do tempo é dada pela forma de onda reproduzida no quadro. O quadro conecta uma interação de um único qubit, fora da diagonal. Para obter mais informações, consulte Krantz et al. e Rahamim et al. .
- `rf_f12`é semelhante `rf` e seus parâmetros visam a transição 1-2.
- `ro_rx`é usado para obter uma leitura dispersiva do qubit por meio de um guia de ondas coplanar acoplado. A frequência, a fase e o conjunto completo de parâmetros para a forma de onda de leitura são pré-calibrados. Atualmente, é usado por meio do`capture_v0`, que não requer nenhum argumento além do identificador do quadro.
- `ro_tx`é para transmitir sinais do ressonador. Atualmente, não está sendo usado.
- `cz`é uma estrutura calibrada para ativar a porta de dois qubits`cz`. Como acontece com todos os quadros associados a uma `ff` porta, ele ativa uma interação entrelaçada através da linha de fluxo modulando o qubit ajustável do par em ressonância com seu vizinho. Para obter mais informações sobre o mecanismo de emaranhamento, consulte Reagor et al. , Caldwell et al. , e Didier et al. .
- `cphase`é uma estrutura calibrada para ativar a porta de dois qubits e está conectada a uma `cphaseshift` porta. `ff` Para obter mais informações sobre o mecanismo de emaranhamento, consulte a descrição da moldura. `cz`
- `xy`é uma estrutura calibrada para ativar as portas XY (θ) de dois qubits e está conectada a uma porta. `ff` Para obter mais informações sobre o mecanismo de emaranhamento e como obter portas XY, consulte a descrição da `cz` estrutura e Abrams et al. .

À medida que os quadros baseados na `ff` porta alteram a frequência do qubit ajustável, todos os outros quadros de acionamento relacionados ao qubit serão reduzidos por uma quantidade relacionada à amplitude e à duração da mudança de frequência. Consequentemente, você deve compensar esse efeito adicionando uma mudança de fase correspondente aos quadros dos qubits vizinhos.

Portas

Os Rigetti dispositivos fornecem uma lista de portas que você pode inspecionar por meio dos recursos do dispositivo. Os nomes das portas seguem a convenção `q{i}_{type}` em que `{i}` se refere ao número do qubit e `{type}` ao tipo da porta. Observe que nem todos os qubits têm um conjunto completo de portas. Os tipos de portas são os seguintes:

- `rf`representa a interface principal para conduzir a transição de um único qubit. Está associado aos `rf_f12` quadros `rf` e. Ele é acoplado capacitivamente ao qubit, permitindo a condução por micro-ondas na faixa de gigahertz.
- `ro_tx`serve para transmitir sinais para o ressonador de leitura acoplado capacitivamente ao qubit. A entrega do sinal de leitura é multiplexada oito vezes por octógono.
- `ro_rx`serve para receber sinais do ressonador de leitura acoplado ao qubit.
- `ff`representa a linha de fluxo rápido acoplada indutivamente ao qubit. Podemos usar isso para ajustar a frequência do transmon. Somente qubits projetados para serem altamente ajustáveis têm uma `ff` porta. Essa porta serve para ativar a interação qubit-qubit, pois há um acoplamento capacitivo estático entre cada par de transmons vizinhos.

Para obter mais informações sobre a arquitetura, consulte Valery et al. .

OQC

Frames (Quadros)

OQCos dispositivos suportam quadros predefinidos que têm sua frequência e fase calibradas para estarem em ressonância com o qubit associado. A convenção de nomenclatura para esses quadros é a seguinte:

- quadro de condução: `q{i}[_q{j}]_{role}` onde `{i}` se refere ao primeiro número de qubit, `{j}` refere-se ao segundo número de qubit, caso o quadro sirva para ativar uma interação de dois qubits, e `{role}` se refere à função do quadro conforme descrito abaixo.
- quadro de leitura de qubit: `r{i}_{role}` onde `{i}` se refere ao número do qubit e `{role}` se refere à função do quadro conforme descrito abaixo.

Recomendamos usar cada quadro para sua função projetada da seguinte forma:

- `drive`é usado como estrutura principal para conduzir a transição 0-1 do qubit. Os pulsos são transmitidos como sinais transitórios de micro-ondas de frequência e fase fornecidos anteriormente por meio das funções `set` e. `shift` A amplitude do sinal dependente do tempo é dada pela forma

de onda reproduzida no quadro. O quadro conecta uma interação de um único qubit, fora da diagonal. Para obter mais informações, consulte Krantz et al. e Rahamim et al. .

- `second_state`é equivalente ao `drive` quadro, mas sua frequência é ajustada em ressonância com a transição 1-2.
- `measure`é para leitura. A frequência, a fase e o conjunto completo de parâmetros para a forma de onda de leitura são pré-calibrados. Atualmente, é usado por meio do`capture_v0`, que não requer nenhum argumento além do identificador do quadro.
- `acquire`é para capturar sinais do ressonador. Atualmente, não está sendo usado.
- `cross_resonance`ativa a interação de ressonância cruzada entre os qubits `i` e aciona o qubit `j` de controle `i` na frequência de transição do qubit alvo. `j` Consequentemente, a frequência do quadro é definida usando a frequência do qubit de destino. A interação ocorre com uma taxa proporcional à amplitude desse drive de ressonância cruzada. Diferentes tipos de diafonia induzem efeitos indesejados que requerem correções. Veja Patterson et al. para obter mais informações sobre a interação de ressonância cruzada com qubits transmon de formato coaxial ('coaxmons').
- `cross_resonance_cancellation`ajuda a adicionar correções para suprimir os efeitos deletérios induzidos pela interferência quando a interação de ressonância cruzada é ativada. A frequência inicial do quadro é definida como a frequência de transição do qubit `i` de controle. Para obter mais informações sobre o método de cancelamento, consulte Patterson et al. .

Portas

Os OQC dispositivos fornecem uma lista de portas que você pode inspecionar por meio dos recursos do dispositivo. Os quadros descritos anteriormente estão associados a portas que são identificadas por sua identificação, `channel_{N}` onde `{N}` é um número inteiro. As portas são a interface para controlar as linhas (direção`tx`) e os ressonadores de leitura (direção`rx`) conectados aos coaxiais. Cada qubit está associado a uma linha de controle e a um ressonador de leitura. A porta de transmissão é a interface para manipulação de um qubit e dois qubits. A porta de recepção serve para leitura de qubit.

Olá Pulse

Aqui, você aprenderá como construir um simples par de Bell diretamente com pulsos e executar esse programa de pulso no Rigetti dispositivo. Um par de Bell é um circuito de dois qubits que consiste em uma porta Hadamard no primeiro qubit seguida por uma `cnot` porta entre o primeiro e o segundo qubits. A criação de estados emaranhados com pulsos requer mecanismos específicos

que dependem do tipo de hardware e da arquitetura do dispositivo. Não usaremos um mecanismo nativo para criar o `cnot` portão. Em vez disso, usaremos formas de onda e quadros específicos que habilitam a `cz` porta de forma nativa. Neste exemplo, criaremos uma porta Hadamard usando as portas nativas de um único qubit `rx` `rz` e expressaremos a `cz` porta usando pulsos.

Primeiro, vamos importar as bibliotecas necessárias. Além da `Circuit` classe, agora você também precisará importar a `PulseSequence` classe.

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

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

Em seguida, instancie um novo dispositivo Braket usando o Amazon Resource Name (ARN) do dispositivo. Rigetti Aspen-M-3 Consulte a página Dispositivos no console do Amazon Braket para ver o layout do dispositivo. Rigetti Aspen-M-3

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

Como a porta Hadamard não é uma porta nativa do Rigetti dispositivo, ela não pode ser usada em combinação com pulsos. Portanto, você precisa decompô-lo em uma sequência dos `rx` portões `rz` nativos.

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

Para a `cz` porta, usaremos uma forma de onda arbitrária com parâmetros (amplitude, tempo de subida/queda e duração) que foram predeterminados pelo fornecedor do hardware durante um

estágio de calibração. Essa forma de onda será aplicada no. `q10_q113_cz_frame` Para uma versão mais recente da forma de onda arbitrária usada aqui, consulte QCS, no site. Rigetti Talvez seja necessário criar uma conta QCS.

```
a_b_cz_wfm = ArbitraryWaveform([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00017888439538396808, 0.00046751103636033026, 0.0011372942989106456,
 0.002577059611929697, 0.00544394194463236 6, 0.010731922770068104, 0.01976701723583167,
 0.03406712171899736, 0.05503285980691202, 0.08350670755829034, 0.11932853352131022,
 0.16107456696238298, 0.20614055551722368, 0.2512065440720643, 0.292952577513137,
 0.328774403476157, 0.3572482512275353, 0.3782139893154499, 0.3925140937986156,
 0.40154918826437913, 0.4068371690898149, 0.4097040514225177, 0.41114381673553674,
 0.411813599998087, 0.4121022266390633, 0.4122174383870584, 0.41226003881132406,
 0.4122746298554775, 0.4122792591252675, 0.4122806196003006, 0.41228098995582513,
 0.41228108334474756, 0.4122811051578895, 0.4122811098772742, 0.4122811108230642,
 0.4122811109986316, 0.41228111102881937, 0.41228111103362725, 0.4122811110343365,
 0.41228111103443343, 0.4122811110344457, 0.4122811110344471, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.4122811110344471, 0.4122811110344457, 0.41228111103443343, 0.4122811110343365,
 0.41228111103362725, 0.41228111102881937, 0.4122811109986316, 0.4122811108230642,
 0.4122811098772742, 0.4122811051578895, 0.41228108334474756, 0.41228098995582513,
 0.4122806196003006, 0.4122792591252675, 0.4122746298554775, 0.41226003881132406,
 0.4122174383870584, 0.4121022266390633, 0.411813599998087, 0.41114381673553674,
 0.4097040514225176, 0.4068371690898149, 0.40154918826437913, 0.3925140937986155,
 0.37821398931544986, 0.3572482512275351, 0.32877440347615655, 0.2929525775131368,
 0.2512065440720641, 0.20614055551722307, 0.16107456696238268, 0.11932853352131002,
 0.08350670755829034, 0.05503285980691184, 0.03406712171899729, 0.01976701723583167,
 0.010731922770068058, 0.005443941944632366, 0.002577059611929697,
 0.0011372942989106229, 0.00046751103636033026, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0])
a_b_cz_frame = device.frames[f'q{a}_q{b}_cz_frame']

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

Isso deve retornar:

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

Agora podemos construir a `cz` porta usando a forma de onda que acabamos de definir. Lembre-se de que a `cz` porta consiste em uma inversão de fase do qubit de destino se o qubit de controle estiver no estado. |1>

```
phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

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

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

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

A `a_b_cz_wfm` forma de onda é reproduzida em um quadro associado a uma porta de fluxo rápido. Sua função é mudar a frequência qubit para ativar uma interação qubit-qubit. Para obter mais informações, consulte Funções de quadros e portas. Conforme a frequência varia, os quadros de qubit giram em taxas diferentes dos `rf` quadros de um único qubit que são mantidos intocados: os últimos estão sendo desfasados. Essas mudanças de fase foram calibradas previamente por meio de Ramsey sequências e são fornecidas aqui como informações codificadas por meio `phase_shift_a` de e (período completo). `phase_shift_b` Corrigimos essa defasagem usando `shift_phase` instruções nos quadros. `rf` Observe que essa sequência só funcionará em programas em que nenhum XY quadro está relacionado ao qubit a e b é usada, pois não compensamos a mudança de fase que ocorre nesses quadros. Esse é o caso desse programa de par único de Bell, que usa apenas `cz` quadros `rf` e. Para obter mais informações, consulte Caldwell et al. .

Agora estamos prontos para criar um par Bell com pulsos.

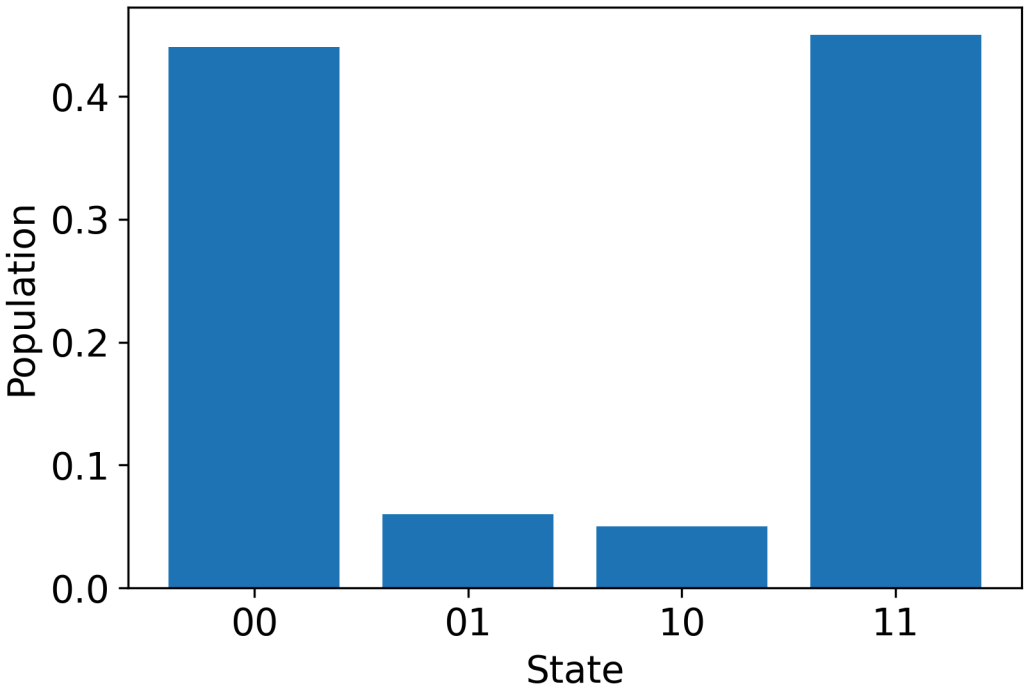
```
bell_circuit_pulse = (
    Circuit()
    .rigetti_native_h(a)
    .rigetti_native_h(b)
```

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

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

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

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

Vamos executar esse par Bell no Rigetti dispositivo. Observe que a execução desse bloco de código incorrerá em uma cobrança. Para obter mais informações sobre esses custos, consulte a página de preços do Amazon Braket. Recomendamos que você teste seus circuitos usando uma pequena quantidade de disparos para garantir que eles funcionem no dispositivo antes de aumentar a contagem de disparos.

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

counts = task.result().measurement_counts

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



Hello Pulse usando OpenPulse

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 OpenPulse suporta a programação direta de pulsos usando a representação OpenQASM 3.0.

Braket usa OpenPulse como representação intermediária subjacente para expressar pulsos em instruções nativas. OpenPulsesuporta 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.

Neste exemplo, construiremos um circuito Bell usando o OpenQASM 3.0 e OpenPulse em um dispositivo usando transmons ajustáveis em frequência. Lembre-se de que um circuito Bell é um circuito de dois qubits que consiste em uma porta Hadamard no primeiro qubit seguida por uma `cnot` porta entre os dois qubits. Como as `cnot` portas diferem das `cz` portas apenas por meio de uma transformação básica, aqui definiremos um par de Bell usando Hadamard e `cz` portas, pois o dispositivo fornece uma maneira mais simples de criar `cz` portas para esta demonstração.

Vamos começar definindo a porta Hadamard usando portas nativas do dispositivo.

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

Para a porta CZ, usaremos uma forma de onda arbitrária com parâmetros (amplitude, tempo de subida/queda e duração) que foram predeterminados previamente. Essa forma de onda será aplicada no. `q10_q113_cz_frame`

```
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.484432196472809 6,
```

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

A duração da `q10_q113_cz_wfm` forma de onda é de 124 amostras, o que corresponde a 124 ns, pois o incremento mínimo de tempo `dt` é de 1 ns.

A `q10_q113_cz_wfm` forma de onda é reproduzida em um quadro vinculado a uma porta de fluxo rápido. Sua função é mudar a frequência qubit para ativar uma interação qubit-qubit. Para obter mais informações, consulte Funções de quadros e portas. Conforme a frequência varia, os quadros de qubit giram em taxas diferentes em comparação com os `rf` quadros de um único qubit que são mantidos intocados: os últimos estão sendo desfasados. Essa defasagem pode ser medida com Ramsey sequências durante um estágio de calibração e compensada com `shift_phase` instruções e quadros. `rf xy` Para obter mais informações, consulte Caldwell et al. .

Agora podemos executar o circuito de pares de Bell, onde decompusemos a `cnot` porta usando alguns Hadamard e portas. `cz`

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

A representação completa do OpenQASM 3.0 para o circuito Bell construído usando uma combinação de portas e pulsos nativos é a seguinte.

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

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

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

Agora você pode usar o SDK do Braket para executar esse programa OpenQASM 3.0 no dispositivo usando o código a Rigetti seguir.

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

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

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

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

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

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

```
)
```

Acessando portões nativos usando pulsos

Os pesquisadores geralmente precisam saber exatamente como as portas nativas suportadas por 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 aos portões nativos 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/Aspen-M-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 `Gate` objeto e os qubits de interesse. Por exemplo, você pode inspecionar a implementação de pulso do RX (π/2) aplicado em 0. qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

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

```
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 circuito a seguir.

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

Você pode executá-lo com uma calibração de porta personalizada para o `rx` portão ligado `qubit 0` passando um dicionário de `PulseSequence` objetos para o argumento da `gate_definitions` palavra-chave. Você pode criar um dicionário a partir `pulse_sequences` do atributo do `GateCalibrations` objeto. 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)
```

Guia do usuário do Amazon Braket Hybrid Jobs

Esta seção fornece instruções sobre como configurar e gerenciar trabalhos híbridos no Amazon Braket.

Você pode acessar trabalhos híbridos no Braket usando:

- O SDK Amazon Braket Python.
- O console Amazon Braket.
- O Amazon BraketAPI.

Nesta seção:

- O que é um Hybrid Job?
- Quando usar o Amazon Braket Hybrid Jobs
- Execute seu código local como um trabalho híbrido
- Execute um trabalho híbrido com o Amazon Braket Hybrid Jobs
- Crie seu primeiro Hybrid Job
- Entradas, saídas, variáveis ambientais e funções auxiliares
- Salve os resultados do trabalho
- Salve e reinicie trabalhos híbridos usando pontos de verificação
- Defina o ambiente para seu script de algoritmo
- Usar hiperparâmetros
- Configure a instância de trabalho híbrida para executar seu script de algoritmo
- Cancelar um Hybrid Job
- Usando compilação paramétrica para acelerar trabalhos híbridos
- Use PennyLane com o Amazon Braket
- Use o Amazon Braket Hybrid Jobs PennyLane e execute um algoritmo QAOA
- Acelere suas cargas de trabalho híbridas com simuladores incorporados da PennyLane
- Crie e depure uma tarefa híbrida com o modo local
- Traga seu próprio contêiner (BYOC)

- Configure o bucket padrão em AwsSession
- Interaja com trabalhos híbridos diretamente usando o API

O que é um Hybrid Job?

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 envolvendo recursos clássicos e quânticos. Você envia seu algoritmo para execução, o Braket o executa em um ambiente escalável em contêineres e você recupera os resultados quando o algoritmo é concluído.

Além disso, as tarefas quânticas criadas a partir de um trabalho híbrido se beneficiam do enfileiramento de maior prioridade para uma QPU de destino. Isso garante que suas tarefas quânticas sejam processadas e executadas antes das outras na fila. Isso é particularmente benéfico para algoritmos híbridos iterativos em que as tarefas subsequentes dependem dos resultados das tarefas quânticas anteriores. Exemplos desses algoritmos incluem o Algoritmo de Otimização Aproximada Quântica (QAOA), o autosolucionadorquântico variacional ou o aprendizado de máquina 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.

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 tempos de execução 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 tempos de execução 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 carga de trabalho 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.

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
- Salve e carregue dados em uma instância de trabalho híbrida
- Melhores práticas para decoradores de trabalho 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 `@hybrid_job` decorador, conforme 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 3.10 é compatível por padrão.

Você pode usar o `@hybrid_job` decorador 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. Você pode 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 noState 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 SV1 simulador. 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 cargas de trabalho muito curtas, como um único circuito ou um lote de circuitos, pode ser suficiente usar tarefas quânticas.

Hiperparâmetros

A `run_hybrid_job()` função 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 registro

Dentro da `run_hybrid_job()` função, 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.

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

@hybrid_job(device=Devices.Amazon.SV1)
def failing():
    return MyObject() # not serializable
```

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 simulador SV1.

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

Todas as outras opções de trabalho híbridas são suportadas. 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 execução para usar seus pacotes Python preferidos. Você pode usar um `requirements.txt` arquivo, uma lista de nomes de pacotes ou trazer seu próprio contêiner (BYOC). Para personalizar um ambiente de tempo de execução usando um `requirements.txt` arquivo, consulte o exemplo de código a seguir.

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

Por exemplo, o `requirements.txt` arquivo pode incluir outros pacotes para instalação.

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

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 ...
```

Salve e carregue dados em uma instância de trabalho híbrida

Especificando dados de treinamento de entrada

Ao criar um trabalho híbrido, você pode fornecer um conjunto de dados de treinamento de entrada 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.

```
@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 `get_input_data_dir()` auxiliar.

```
s3_path = "s3://amazon-braket-us-west-1-961591465522/job-data/file.npy"

@hybrid_job(device=None, input_data=s3_path)
def job_s3_input():
    np.load(get_input_data_dir() + "/file.npy")

@hybrid_job(device=None, input_data={"channel": s3_path})
def job_s3_input_channel():
    np.load(get_input_data_dir("channel") + "/file.npy")
```

Você pode especificar várias fontes de dados de entrada fornecendo um dicionário de valores de canais e URIs do S3 ou caminhos locais.

```
input_data = {
    "input": "data/file.npy",
    "input_2": "s3://my-bucket/data.json"
}


@hybrid_job(device=None, input_data=input_data)
def multiple_input_job():
    np.load(get_input_data_dir("input") + "/file.npy")
    np.load(get_input_data_dir("input_2") + "/data.json")
```

> **Note**
>
> 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.

```
@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()` 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 `save_job_checkpoint()` auxiliar integrada ou salvar arquivos no `AMZN_BRAKET_JOB_RESULTS_DIR` caminho. O último está disponível com a função `get_job_results_dir()` auxiliar.

Veja a seguir um exemplo mínimo de trabalho 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 substitui 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` está o ARN do trabalho híbrido do trabalho anterior. Em seguida, usamos `load_job_checkpoint(job_name)` para carregar a partir do ponto de verificação.

Melhores práticas para decoradores de trabalho 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 determinadas QPUs é mais benéfico para algoritmos variacionais de longa execução que exigem várias chamadas iterativas para as 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 de um trabalho híbrido são compilados automaticamente em determinadas 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 obter mais exemplos, casos de uso e melhores práticas, consulte exemplos do Amazon Braket. GitHub

Execute 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 Traga seu próprio contêiner (BYOC).

Em ambos os casos, em seguida, você cria um trabalho híbrido usando o Amazon API Braket, 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 ouTN1) 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 API chamada), executa seu algoritmo, grava os resultados no Amazon S3 e libera seus recursos. Essa liberação de recursos garante que você pague somente 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. Depois que seu trabalho híbrido for concluído, os recursos serão liberados, o que significa que você paga apenas pelo que usar.

> **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, ou`braket:default-simulator`)`lightning.qubit`, a GPU não será usada e você poderá incorrer em custos desnecessários.

Crie seu primeiro Hybrid Job

Esta seção mostra como criar um Hybrid Job usando um script Python. Como alternativa, para criar uma tarefa híbrida a partir do código Python local, como seu ambiente de desenvolvimento integrado (IDE) preferido ou um notebook Braket, consulte. Execute seu código local como um trabalho híbrido

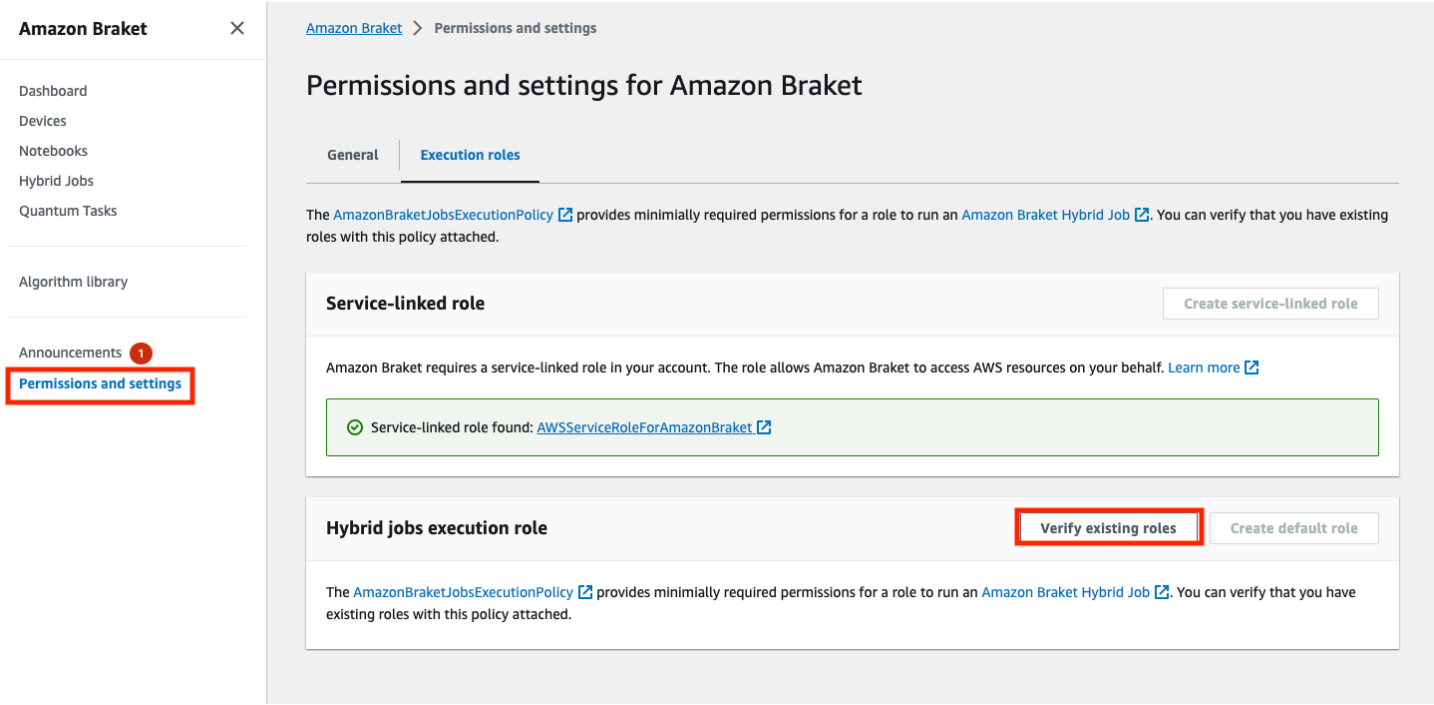
Nesta seção:

- Definir permissões

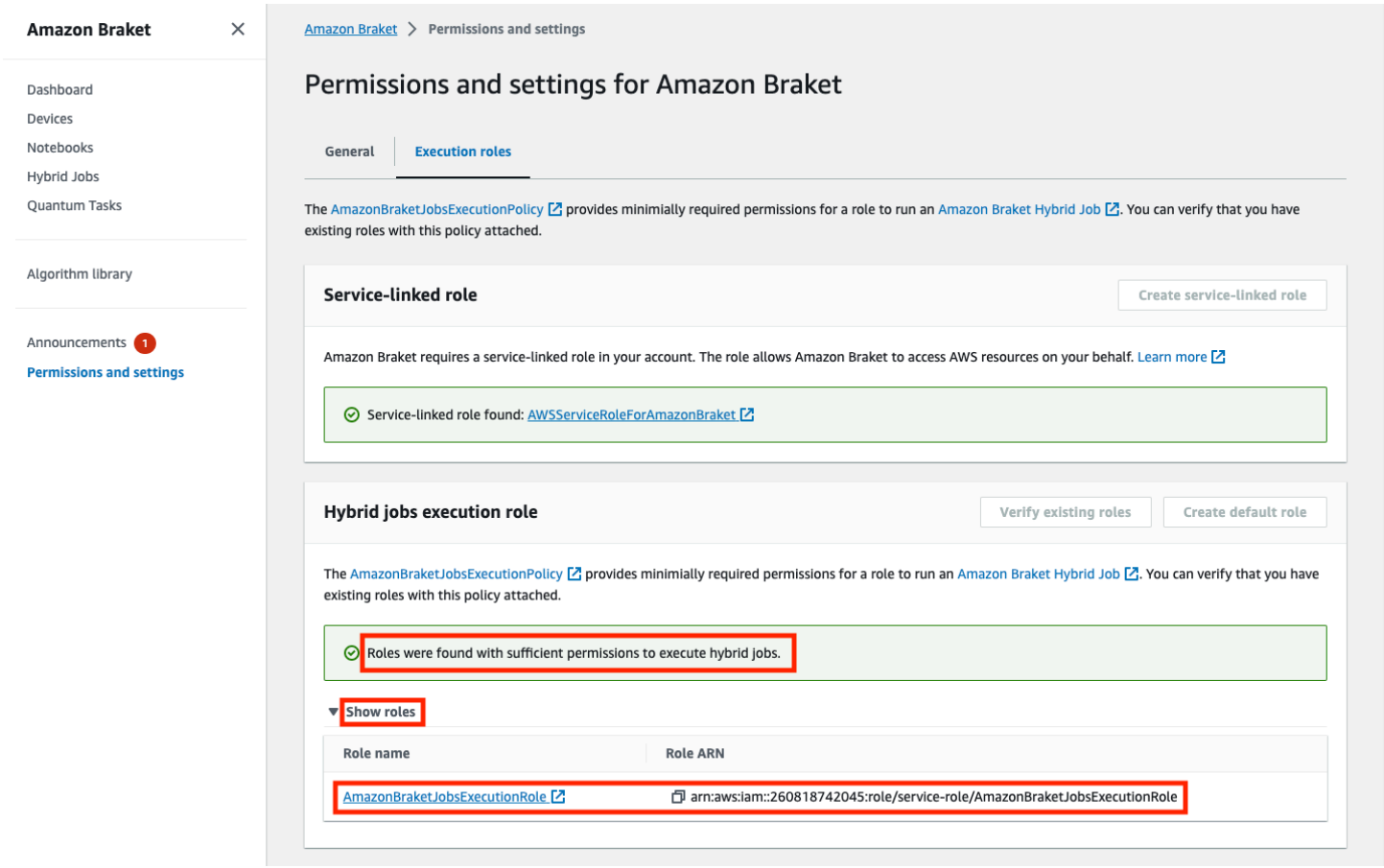
- Crie e execute
- Resultados do monitoramento

Definir permissões

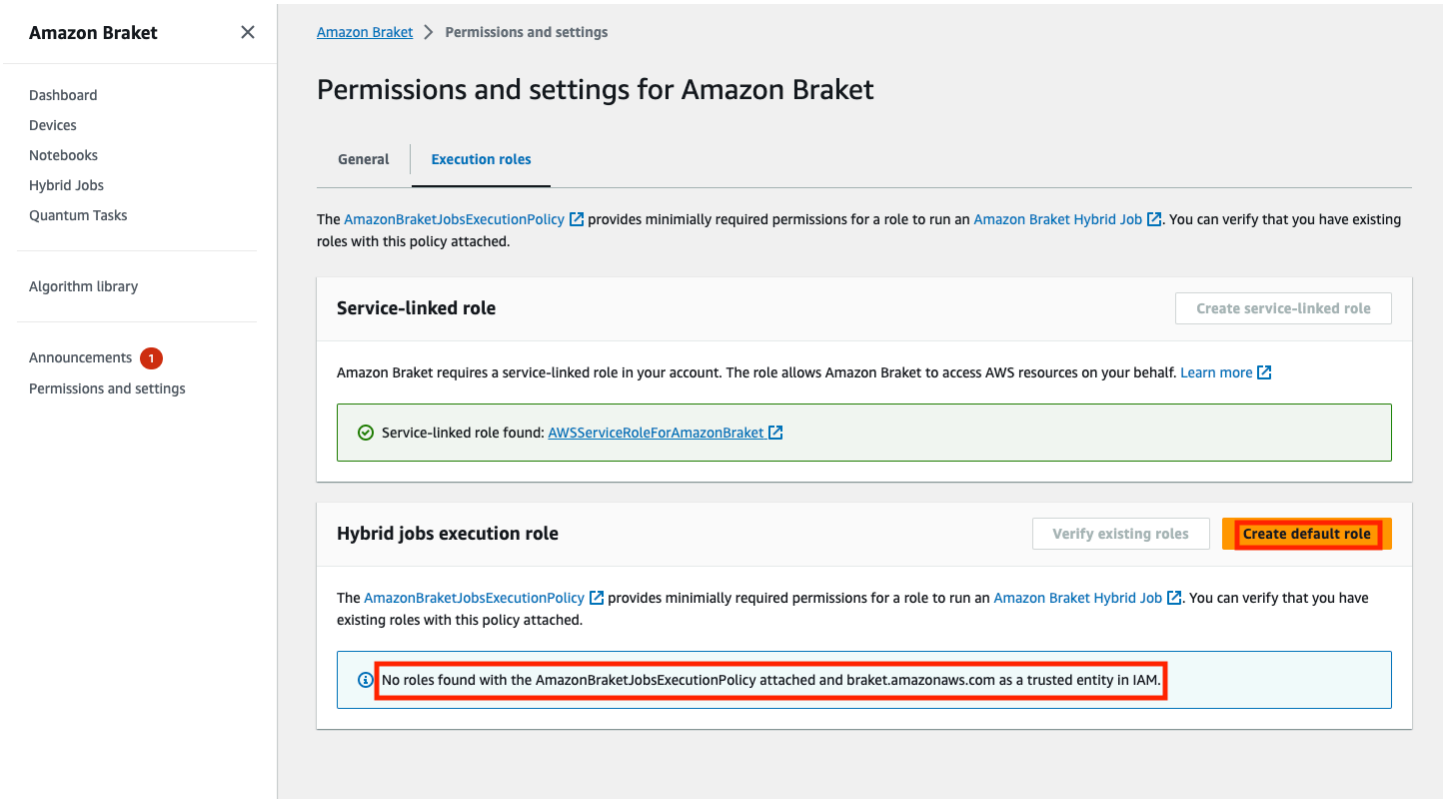
Antes de executar seu primeiro trabalho híbrido, você deve garantir que tenha permissões suficientes para continuar com essa tarefa. Para determinar se você tem as permissões corretas, selecione Permissões no menu à esquerda do console Braket. A página Gerenciamento de permissões para Amazon Braket ajuda 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.



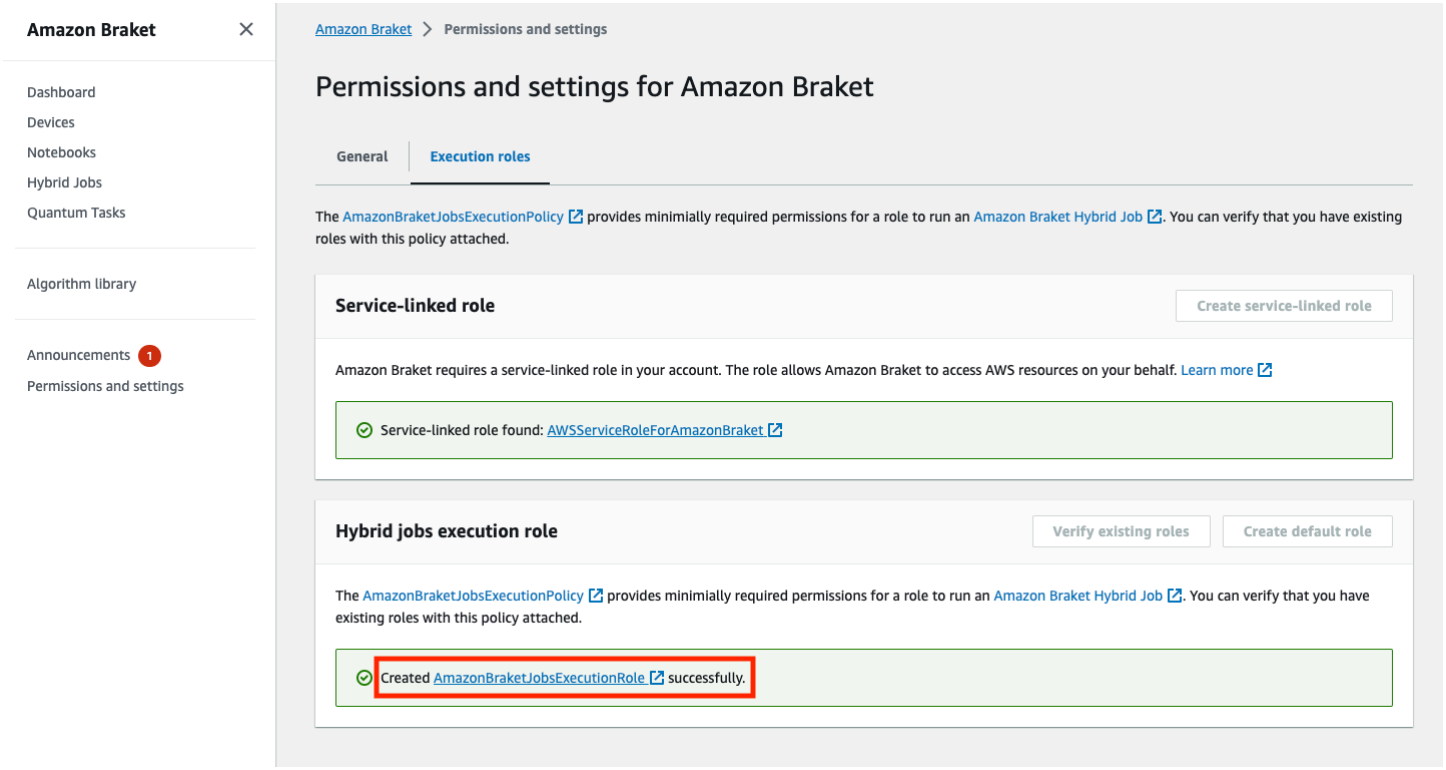
Para verificar se você tem funções com permissões suficientes para executar um trabalho híbrido, selecione o botão Verificar função existente. Se você fizer isso, receberá uma mensagem informando que as funções foram encontradas. Para ver os nomes das funções e seus ARNs de função, selecione o botão Mostrar funções.



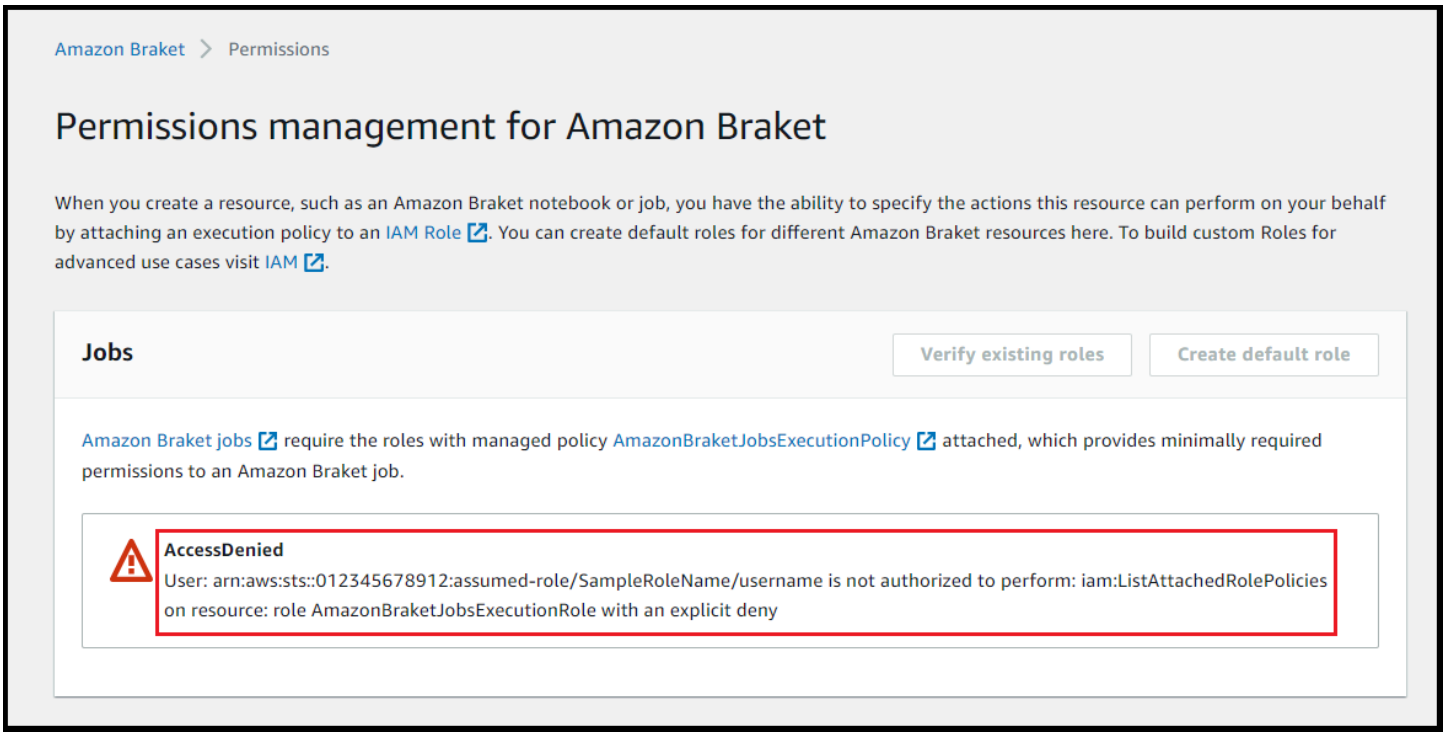
Se você não tiver uma função com permissões suficientes para executar uma tarefa híbrida, você receberá uma mensagem informando que essa função não foi encontrada. Selecione o botão Criar função padrão para obter uma função com permissões suficientes.



Se a função foi criada com sucesso, você receberá uma mensagem confirmando isso.

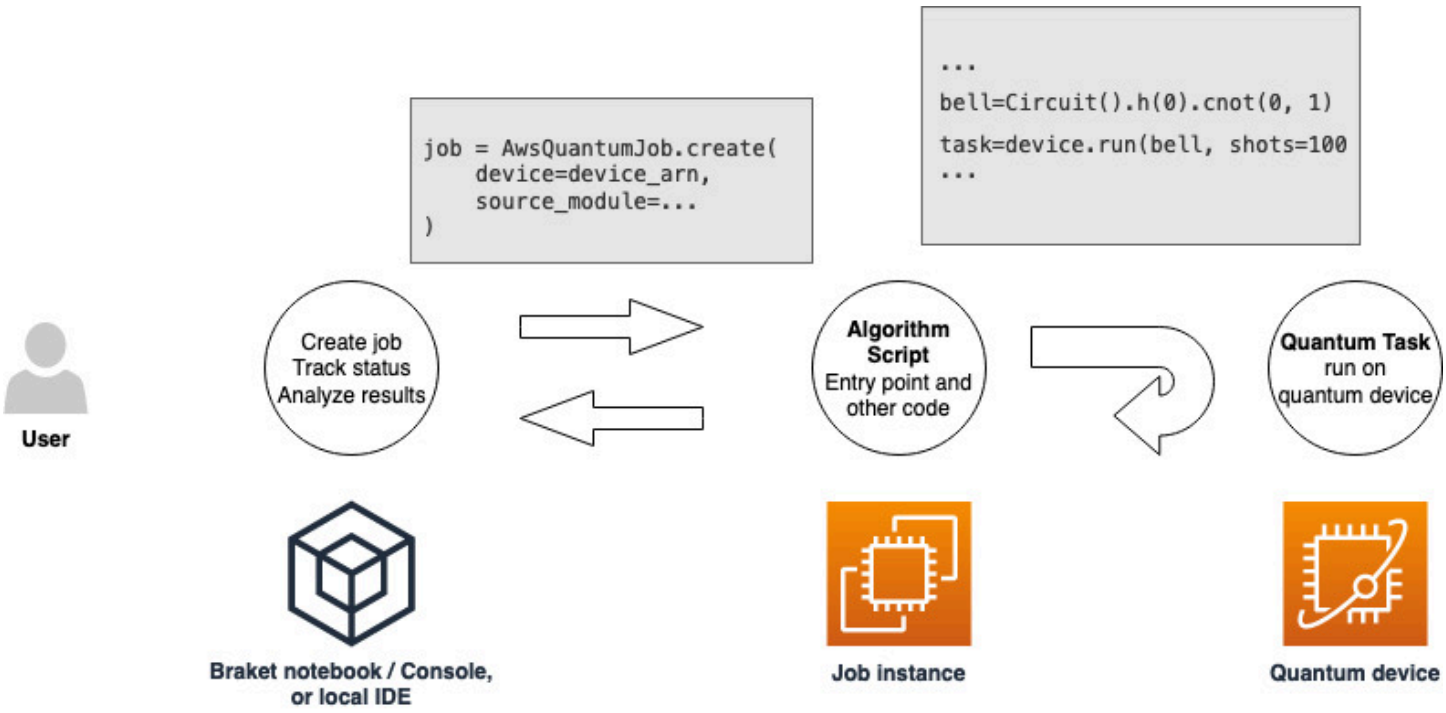


Se você não tiver permissão para fazer essa consulta, seu acesso será negado. Nesse caso, entre em contato com seu AWS administrador interno.



Crie e execute

Depois de ter uma função 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 notebook 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 notebook Braket ou fazer upload de um arquivo, como o arquivo algorithm_script.py, no mesmo diretório dos notebooks, 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 `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 isso automaticamente. Região da AWS Por exemplo, uma tarefa híbrida em us-east-1 pode IonQ usar dispositivosSV1,,, TN1 eDM1, mas não dispositivos. 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
```

```
)
```

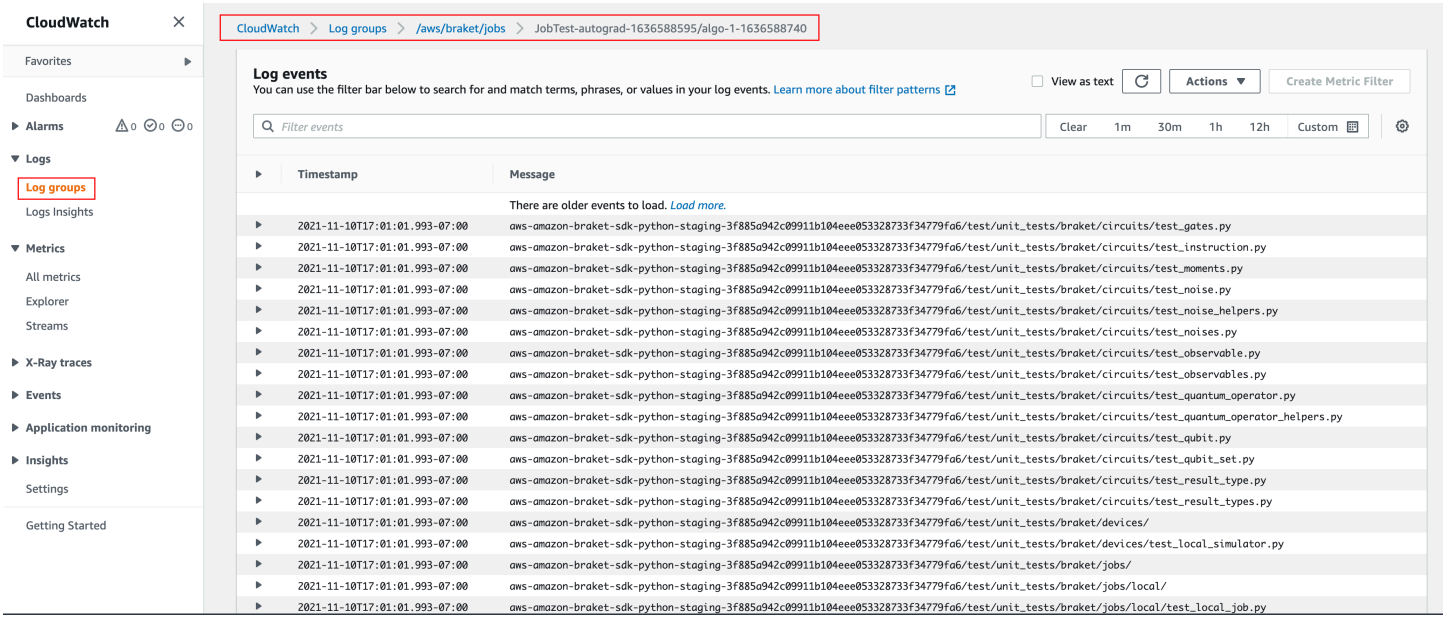
O parâmetro `wait_until_complete=True` define um modo detalhado para que seu trabalho imprima a saída do trabalho real enquanto ele está sendo executado. Você deve ver uma saída semelhante ao exemplo a seguir.

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

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

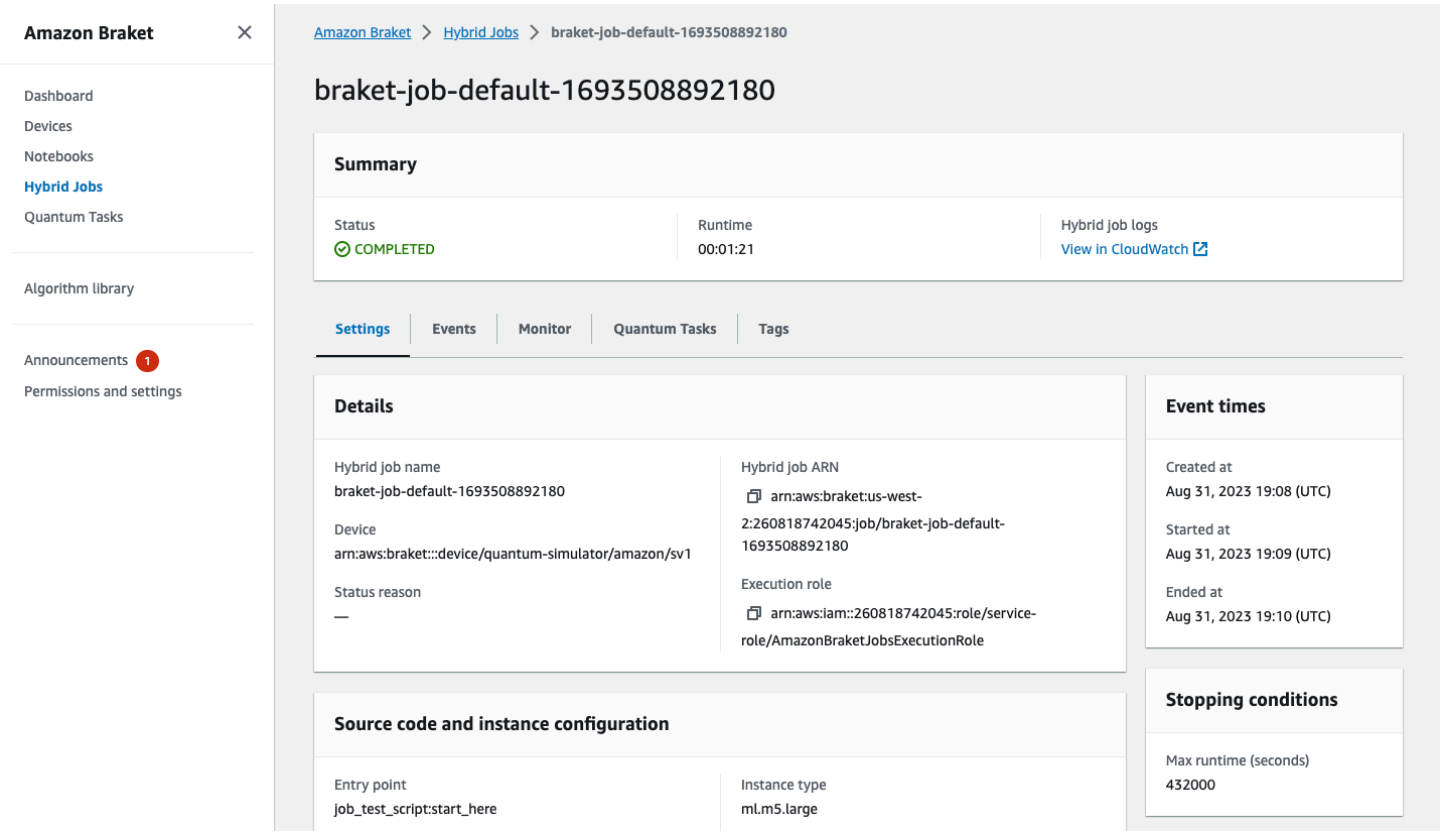
> **Note**
>
> Você também pode usar seu módulo personalizado com o AwsQuantumJobmétodo.create passando sua localização (o caminho para um diretório ou arquivo local ou um URI do S3 de um arquivo tar.gz). Para ver um exemplo prático, consulte o arquivo Parallelize_training_for_qml.ipynb na pasta de trabalhos híbridos no repositório Github de exemplos do Amazon Braket.

Resultados do monitoramento

Como alternativa, você pode acessar a saída do log da Amazon CloudWatch. Para fazer isso, acesse a guia Grupos de registros no menu esquerdo da página de detalhes do trabalho, selecione o grupo `aws/braket/jobs` de registros e escolha o fluxo de registros 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 `jobs/<jobname>/<timestamp>` diretório. Você pode configurar os locais do S3 em que esses artefatos são armazenados especificando um diferente `code_location` 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 `jobs/<jobname>/<timestamp>` diretório contém uma subpasta com a saída do script do ponto de entrada em um `model.tar.gz` arquivo. 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`.

Entradas, saídas, variáveis ambientais e funções auxiliares

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.

AmazonO Braket também fornece várias variáveis de ambiente e funções auxiliares para simplificar as interações com entradas e saídas do contêiner.

Esta seção explica os principais conceitos da `AwsQuantumJob.create` função fornecida pelo SDK do Amazon Braket Python e seu mapeamento para a estrutura do arquivo de contêiner.

Nesta seção:

- Entradas
- Outputs
- 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 `input_data` argumento. O usuário define o `input_data` argumento dentro da `AwsQuantumJob.create` função 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`hyperparameters`, eles estarão disponíveis na variável `"AMZN_BRAKET_HP_FILE"` de ambiente.

> **Note**
>
> Para obter mais informações sobre como criar hiperparâmetros e inserir dados e depois 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 `copy_checkpoints_from_job` comando. Esse comando copia os dados do ponto `checkpoint_configs3Uri` de verificação 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.

Outputs

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 você não especificar esse valor, 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"` Em vez disso, você também pode usar a função `save_job_checkpoint` auxiliar do SDK.

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 Use Amazon Braket Hybrid Jobs para executar um algoritmo QAOA.

Variáveis de ambiente

AmazonO Braket fornece várias variáveis de ambiente para simplificar as interações com as entradas e saídas do contêiner. O código a seguir lista as variáveis ambientais que Braket usa.

```
# the input data directory opt/braket/input/data
os.environ["AMZN_BRAKET_INPUT_DIR"]
# the output directory opt/braket/model to write job results to
os.environ["AMZN_BRAKET_JOB_RESULTS_DIR"]
# the name of the job
os.environ["AMZN_BRAKET_JOB_NAME"]
# the checkpoint directory
os.environ["AMZN_BRAKET_CHECKPOINT_DIR"]
# the file containing the hyperparameters
os.environ["AMZN_BRAKET_HP_FILE"]
# the device ARN (AWS Resource Name)
os.environ["AMZN_BRAKET_DEVICE_ARN"]
# the output S3 bucket, as specified in the CreateJob request’s OutputDataConfig
os.environ["AMZN_BRAKET_OUT_S3_BUCKET"]
# the entry point as specified in the CreateJob request’s ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_ENTRY_POINT"]
# the compression type as specified in the CreateJob request’s ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE"]
# the S3 location of the user’s script as specified in the CreateJob request’s
 ScriptModeConfig
os.environ["AMZN_BRAKET_SCRIPT_S3_URI"]
# the S3 location where the SDK would store the quantum task results by default for the
 job
os.environ["AMZN_BRAKET_TASK_RESULTS_S3_URI"]
# the S3 location where the job results would be stored, as specified in CreateJob
 request’s OutputDataConfig
os.environ["AMZN_BRAKET_JOB_RESULTS_S3_PATH"]
```

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

Funções auxiliares

AmazonO Braket fornece várias funções auxiliares para simplificar as interações com as entradas e saídas do contêiner. Essas funções auxiliares seriam chamadas de dentro do script de algoritmo usado para executar seu Hybrid Job. O exemplo a seguir demonstra como usá-los.

```
get_checkpoint_dir() # get the checkpoint directory
get_hyperparameters() # get the hyperparameters as strings
get_input_data_dir() # get the input data directory
get_job_device_arn() # get the device specified by the hybrid job
get_job_name() # get the name of the hybrid job.
get_results_dir() # get the path to a results directory
save_job_result() # save hybrid job results
save_job_checkpoint() # save a checkpoint
load_job_checkpoint() # load a previously saved checkpoint
```

Salve os resultados do trabalho

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 script do algoritmo.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.jobs import save_job_result #ADD

def start_here():
```

```
    print("Test job started!!!!!")

    device = AwsDevice(os.environ['AMZN_BRAKET_DEVICE_ARN'])

    results = []  #ADD

    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)
        results.append(task.result().measurement_counts)  #ADD

        save_job_result({ "measurement_counts": results })  #ADD

    print("Test job completed!!!!!")
```

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="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`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-1234567890123
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Salve e reinicie trabalhos híbridos 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!!!!!")
```

Quando você executa o trabalho híbrido, ele cria o arquivo -checkpoint-1.json <jobname>nos artefatos do trabalho híbrido no diretório de pontos de verificação 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.

```
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. `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.

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

Defina o ambiente para seu script de algoritmo

AmazonO Braket oferece suporte a três ambientes definidos por contêineres para seu script de algoritmo:

- Um contêiner base (o padrão, se nenhum `image_uri` for especificado)
- Um contêiner com Tensorflow e PennyLane
- Um recipiente com PyTorch e PennyLane

A tabela a seguir fornece detalhes sobre os contêineres e as bibliotecas que eles incluem.

Contêineres Amazon Braket

Tipo	PennyLane com TensorFlow	PennyLane com PyTorch	Pennylane
Base	292282985366.dkr.ecr.us-east-1.amazonaws.com /:latest amazon-braket-tensorflow-jobs	292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest amazon-braket-pytorch-jobs	292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest amazon-braket-base-jobs
Bibliotecas herdadas	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
Bibliotecas adicionais	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • kernel ipy • keras	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • kernel ipy • keras	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • boto3

Tipo	PennyLane com TensorFlow	PennyLane com PyTorch	Pennylane
	• matplotlib • redes • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU Lightning • CuQuantum	• matplotlib • redes • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU Lightning • CuQuantum	• kernel ipy • matplotlib • redes • numpy • openbabel • pandas • PennyLane • protobuf • psi4 • rsa • scipy

Você pode visualizar e acessar as definições de contêiner de código aberto em aws/ amazon-braket-containers. Escolha o contêiner que melhor se adequa ao seu caso de uso. O contêiner deve estar no Região da AWS qual você invoca sua tarefa híbrida. Você especifica a imagem do contêiner ao criar um trabalho híbrido adicionando um dos três argumentos a seguir à sua `create(…)` chamada no script de trabalho híbrido. Você pode instalar dependências adicionais no contêiner escolhido em tempo de execução (ao custo da inicialização ou do tempo de execução) porque os contêineres Amazon Braket têm conectividade com a Internet. O exemplo a seguir é para a região us-west-2.

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

Eles também `image-uris` podem ser recuperados usando a `retrieve_image()` função 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_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Usar 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. Observe que os valores devem ser do tipo de dados string. Você especifica os valores de hiperparâmetros que deseja testar 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:

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

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

Em seguida, você passaria os hiperparâmetros definidos no trecho de código fornecido acima para serem usados no algoritmo de sua escolha com algo parecido com o seguinte:

```
import time
from braket.aws import AwsQuantumJob

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

job = AwsQuantumJob.create(
    #Run this hybrid job on the SV1 simulator
```

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

> **Note**
>
> Para saber mais sobre os dados de entrada, consulte a seção Entradas.

Os hiperparâmetros seriam então carregados no script de trabalho híbrido usando o seguinte código:

```
import json
import os

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

> **Note**
>
> Para obter mais informações sobre como passar informações como os dados de entrada e o arn do dispositivo para o script de trabalho híbrido, consulte esta página do github.

Alguns guias que são muito úteis para aprender sobre como usar hiperparâmetros são fornecidos pelo QAOA com os tutoriais Amazon Braket Hybrid Jobs e Quantum PennyLane Machine Learning nos Amazon Braket Hybrid Jobs.

Configure a instância de trabalho híbrida para executar seu script de algoritmo

Dependendo do seu algoritmo, você pode ter requisitos diferentes. Por padrão, o Amazon Braket executa seu script de algoritmo em uma `ml.m5.large` instância. 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.p3.8xlarge"), # Use NVIDIA Tesla
 V100 instance with 4 GPUs.
    ...
    ),
```

Se você estiver executando uma simulação incorporada e tiver especificado um dispositivo local na configuração do dispositivo, poderá solicitar adicionalmente mais de uma instância no InstanceConfig especificando o InstanceCount e definindo-o como 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.p3.8xlarge", instanceCount=3), #
 Use 3 NVIDIA Tesla V100
    ...
    ),
```

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 Braket.

As três tabelas a seguir listam os tipos e especificações de instância disponíveis para instâncias de computação padrão, otimizadas para computação e aceleradas.

> **Note**
>
> Para ver as cotas padrão de instância de computação clássica para trabalhos híbridos, consulte esta página.

Instâncias padrão	vCPU	Memória
ml.m5.large (padrão)	2	8 GiB
ml.m5.xlarge	4	16 GiB
ml.m5.2xlarge	8	32 GiB
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB
ml.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

Instâncias otimizadas para computação	vCPU	Memória
ml.c4.xlarge	4	7,5 GiB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB

Instâncias otimizadas para computação	vCPU	Memória
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.xlarge	4	10,5 GiB
ml.c5n.2xlarge	8	21 GiB
ml.c5n.4xlarge	16	42 GiB
ml.c5n.9xlarge	36	96 GiB
ml.c5n.18xlarge	72	192 GiB

Instâncias de computação acelerada	vCPU	Memória
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB
ml.g4dn.xlarge	4	16 GiB

Instâncias de computação acelerada	vCPU	Memória
ml.g4dn.2xlarge	8	32 GiB
ml.g4dn.4xlarge	16	64 GiB
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

> **Note**
>
> As instâncias p3 não estão disponíveis em us-west-1. Se seu trabalho híbrido não conseguir provisionar a capacidade computacional de ML solicitada, use outra região.

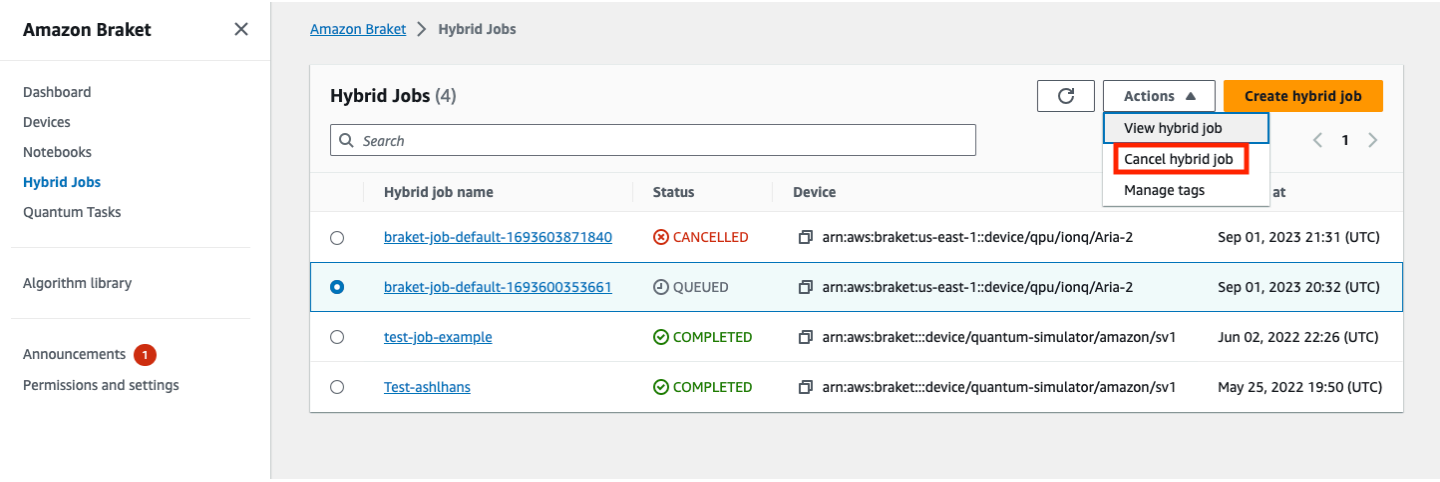
Cada instância usa uma configuração padrão de armazenamento de dados (SSD) de 30 GB. Mas você pode ajustar o armazenamento da mesma forma que configura `instanceType` o. O exemplo a seguir mostra como aumentar o armazenamento total para 50 GB.

```
from braket.jobs.config import InstanceConfig

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

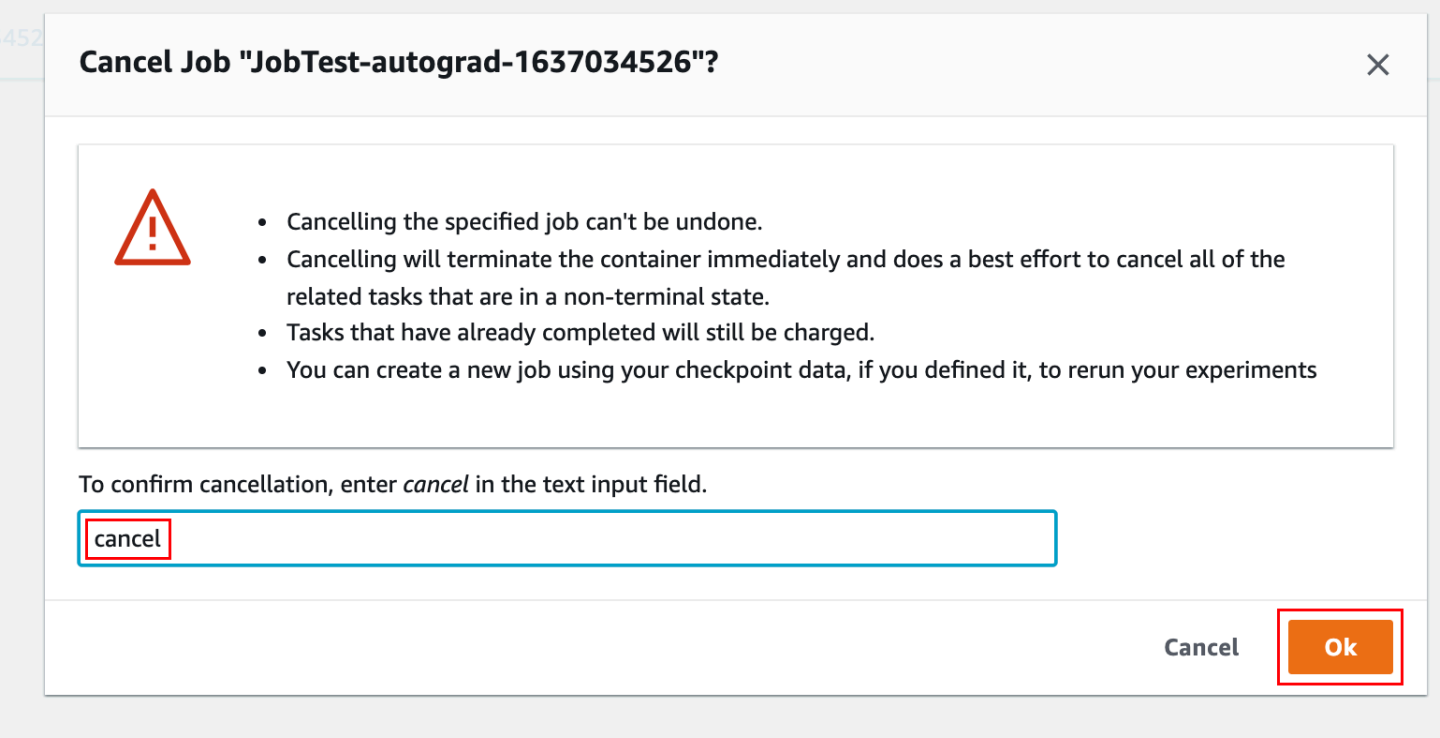
Cancelar um Hybrid Job

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 `cancel` comando nele, conforme mostrado no código a seguir.

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

O `cancel` comando 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.

Usando compilação paramétrica para acelerar trabalhos híbridos

O Amazon Braket oferece suporte à compilação paramétrica em determinadas 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 tempos de execução de trabalhos híbridos, já que você evita a necessidade de recompilar seu circuito em cada etapa. Basta enviar circuitos parametrizados para uma de nossas QPUs suportadas 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 cargas de trabalho 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 job a seguir. Ao executar o Hybrid Job em uma QPU que suporta compilação paramétrica, o circuito é compilado somente na primeira execução. Nas execuções seguintes, o circuito compilado é reutilizado, aumentando o desempenho do tempo de execução 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 todas as QPUs supercondutoras baseadas em portas de Rigetti Computing e Oxford Quantum Circuits com exceção dos programas de nível de pulso.

Use 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.

AmazonO Braket permite que você configure e execute algoritmos quânticos híbridos com a ajuda do PennyLane plug-in Braket ou com o Amazon SDK Braket Amazon Python e exemplos de repositórios de notebooks. Amazon Os notebooks de exemplo do 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 CPUs clássicas, 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.

Amazon Braket com PennyLane

AmazonO Braket fornece suporte para PennyLaneuma estrutura de software de código aberto construída em torno do 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 isso 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, incluindo QPUs e simuladores
- `braket.local.qubit`para execução com o simulador Amazon local do Braket SDK

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 notebooks de exemplo do Amazon Braket

AmazonO Braket fornece uma variedade de exemplos de notebooks que não dependem do PennyLane plug-in para executar algoritmos híbridos. Você pode começar com qualquer um desses notebooks híbridos Amazon Braket que ilustram métodos variacionais, como o Quantum Approximate Optimization Algorithm (QAOA) ou o Variational Quantum Eigensolver (VQE).

Os notebooks 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

AmazonO Braket Hybrid Jobs agora vem com simuladores incorporados de alto desempenho baseados em CPU e GPU da. PennyLane Essa família de simuladores incorporados pode ser incorporada diretamente em seu contêiner de tarefas híbrido e inclui o rápido simulador vetorial de estado, o `lightning.qubit` `lightning.gpu` simulador acelerado usando a biblioteca CuQuantum da NVIDIA e 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 só precisa decorar sua função principal do Python com `@hybrid_job` o decorador. 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 hybird_job
from braket.jobs.config import InstanceConfig
```

```
@hybrid_job(device="local:pennylane/lightning.gpu",
 instance_config=InstanceConfig(instanceType="ml.p3.8xlarge"))
def function(wires):
    dev = qml.device("lightning.gpu", wires=wires)
    ...
```

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 no 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 de diferenciação adjunta do SV1 ao executar o QAOA a partir dePennyLane, você 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)`

Use o Amazon Braket Hybrid Jobs PennyLane e execute 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 simples 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 `True` que você obtenha automaticamente os benefícios do desempenho aprimorado do tempo de execução a partir da compilação paramétrica em QPUs compatíveis.

```
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 todas as QPUs supercondutoras baseadas em portas de Rigetti Computing e Oxford Quantum Circuits com exceção dos programas de nível de pulso.

Acelere suas cargas de trabalho híbridas com simuladores incorporados da PennyLane

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, mostramos como usar `lightning.gpu` para acelerar suas cargas de trabalho híbridas.

Usando **`lightning.gpu`** para cargas de trabalho do algoritmo de otimização aproximada quântica

Considere os exemplos do Algoritmo de Otimização Aproximada Quântica (QAOA) deste notebook. Para selecionar um simulador incorporado, você especifica o `device` argumento 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, vamos comparar os dois simuladores de vetores de PennyLane estado incorporados `lightning.qubit` (que são baseados em CPU) e `lightning.gpu` (que são baseados em GPU). Você precisará fornecer aos simuladores algumas decomposições de portas personalizadas para calcular vários gradientes.

Agora você está pronto para preparar o script híbrido de lançamento de trabalhos. Você executará o algoritmo QAOA usando dois tipos de instância: e. `m5.2xlarge` `p3.2xlarge` O tipo de `m5.2xlarge` instância é comparável a um laptop padrão para desenvolvedores. `p3.2xlarge`É uma instância de computação acelerada que tem uma única GPU NVIDIA Volta com 16 GB de memória.

O `hyperparameters` para todos os seus trabalhos híbridos será o mesmo. Tudo o que você precisa fazer para experimentar diferentes instâncias e simuladores é alterar duas linhas da seguinte forma.

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

ou:

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

> **Note**
>
> Se você especificar o **`instance_config`** como usando uma instância baseada em GPU, mas escolher o **`device`** para ser o 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!

Primeiro, você pode criar duas tarefas híbridas e resolver Max-Cut com QAOA em um gráfico com 18 vértices. Isso se traduz em um circuito de 18 qubits, relativamente pequeno e viável de ser executado rapidamente em seu laptop ou na instância. `m5.2xlarge`

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

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

# And similarly for the p3 job
```

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

O tempo médio de iteração para a `m5.2xlarge` instância é de cerca de 25 segundos, enquanto para a `p3.2xlarge` instância é de cerca de 12 segundos. Para esse fluxo de trabalho de 18 qubits, a instância da GPU nos dá uma aceleração de 2x. 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 0,06375 USD. `p3.2xlarge` Executar um total de 5 iterações, como você fez aqui, custaria 0,016 USD usando a instância de CPU ou 0,06375 USD usando a instância de GPU — ambas bem baratas!

Agora vamos dificultar o problema e tentar resolver um problema de Max-Cut em um gráfico de 24 vértices, que se traduzirá em 24 qubits. Execute as tarefas híbridas novamente nas mesmas duas instâncias e compare o custo.

> **Note**
>
> Você verá que o tempo para executar esse trabalho híbrido na instância da CPU pode ser de cerca de cinco horas!

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

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

# And similarly for the p3 job
m5_big_job = AwsQuantumJob.create(
```

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

O tempo médio de iteração para a `m5.2xlarge` instância é de aproximadamente uma hora, enquanto para a `p3.2xlarge` instância é de aproximadamente dois minutos. Para esse problema maior, a instância da GPU é uma ordem de magnitude mais rápida! Tudo o que você precisava fazer para se beneficiar dessa aceleração era alterar duas linhas de código, trocando o tipo de instância e o simulador local usado. Executar um total de 5 iterações, como foi feito aqui, custaria cerca de 2,27072 USD usando a instância de CPU ou cerca de 0,775625 USD usando a instância de GPU. O uso da CPU não é apenas mais caro, mas também leva mais tempo para ser executado. Acelerar esse fluxo de trabalho com uma instância de GPU disponível AWS, usando o simulador incorporado PennyLane da NVIDIA CuQuantum, permite que você execute fluxos de trabalho com contagens intermediárias de qubits (entre 20 e 30) por menos custo total e em menos tempo. Isso significa que você pode experimentar a computação quântica até mesmo para problemas grandes demais para serem executados rapidamente em seu laptop ou em uma instância de tamanho similar.

Aprendizado de máquina quântico e paralelismo de dados

Se seu tipo de carga de trabalho for aprendizado de máquina quântico (QML) treinado em conjuntos de dados, você poderá acelerar ainda mais sua carga de trabalho 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 a utilização do paralelismo de dados para acelerar seu treinamento.

Considere a seguinte carga de trabalho 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 notebooks para obter mais detalhes sobre a carga de trabalho. 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.p3.2xlarge')

hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_single",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    ...
)
```

Para 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 de distribuir suas cargas de trabalho em várias GPUs e várias 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 `rank` final `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, isso `rank` varia de 0 a 31 e `local_rank` varia 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 de `DistributedSampler` acordo com o `world_size` `rank` e, em seguida, o passa para o carregador de dados. Esse amostrador evita que as GPUs acessem 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 `DistributedDataParallel` classe 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 registro 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.p3.16xlarge` oferece suporte a tipos de instância para a biblioteca paralela de dados SageMaker distribuídos. Você configura o tipo de instância por meio do `InstanceConfig` argumento 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 `smdistributed` pacote. Seu script de algoritmo não precisa usá-los explicitamente. No Amazon Braket SDK, ele fornece um argumento de palavra-chave conveniente. `distribution` Com a criação `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 8 GPUs em uma `ml.p3.16xlarge` instância. Quando você define`instanceCount=1`, a carga de trabalho é distribuída pelas 8 GPUs na instância. Quando você define `instanceCount` mais de um, a carga de trabalho é distribuída entre as 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 cargas de trabalho ao mesmo tempo.

```
instance_config = InstanceConfig(instanceType='ml.p3.16xlarge',
                                 instanceCount=1,
)
```

```
hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...,
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_dp",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    distribution="data_parallel",
    ...
)
```

> **Note**
>
> Na criação de empregos híbridos acima, `train_dp.py` está 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.

Vamos comparar o tempo de execução e o custo em um exemplo em que ao treinar um modelo com um circuito quântico de 26 qubits para o problema de classificação binária mencionado acima. A `ml.p3.16xlarge` instância usada neste exemplo custa 0,4692 USD por minuto. Sem paralelismo de dados, o simulador leva cerca de 45 minutos para treinar o modelo por 1 época (ou seja, mais de 208 pontos de dados) e custa cerca de $20. Com o paralelismo de dados em 1 instância e 4 instâncias, são necessários apenas 6 minutos e 1,5 minutos, respectivamente, o que significa aproximadamente 2,8 USD para ambas. Ao usar o paralelismo de dados em 4 instâncias, você não apenas melhora o tempo de execução em 30 vezes, mas também reduz os custos em uma ordem de magnitude!

Crie e depure uma tarefa híbrida com o modo local

Se você estiver criando um novo algoritmo híbrido, o modo local o ajudará a depurar e testar seu script de algoritmo. O modo local é um recurso que permite executar o código que você planeja

usar nos trabalhos híbridos do Amazon Braket, mas sem precisar do Braket para gerenciar a infraestrutura para executar o trabalho híbrido. Em vez disso, você executa trabalhos híbridos localmente em sua instância do 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 QPU real no modo local.

Para usar o modo local, modifique `AwsQuantumJob` para `LocalQuantumJob` onde quer que ele ocorra. Por exemplo, para executar o exemplo em Criar seu primeiro trabalho híbrido, edite o script de trabalho híbrido 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 notebooks Amazon Braket, precisa ser instalado em seu ambiente local para usar esse recurso. As instruções para instalar o Docker podem ser encontradas aqui. Além disso, nem todos os parâmetros são suportados no modo local.

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 `requirements.txt` arquivo 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). Mas antes de nos aprofundarmos, vamos ter certeza de que é realmente o recurso certo para seu caso de uso.

Quando levar 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 BYOC Docker criação - 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 `requirements.txt` arquivo no diretório de origem no envio do trabalho. O arquivo é lido automaticamente e `pip` instalará os pacotes com as versões especificadas normalmente. Se você estiver instalando um grande número de pacotes, o tempo de execução 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. 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 Docker imagem e incluir um código para autenticá-la.
- Você está usando um software que não está disponível publicamente. 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 Docker contêiner 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.

Receita para trazer seu próprio recipiente

Nesta seção, fornecemos um step-by-step guia do que você precisará bring your own container (BYOC) fazer no Braket Hybrid Jobs: os scripts, os arquivos e as etapas para combiná-los para começar a trabalhar com suas imagens personalizadasDocker. Fornecemos receitas para dois casos comuns:

1. Instale software adicional em uma Docker imagem 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 de que você precisa para criar uma Docker imagem que possa ser usada com trabalhos híbridos. Se você não estiver familiarizado com escrever e criar`Dockerfiles`, sugerimos que consulte a documentação do Dockerfile e a Amazon ECR CLIdocumentação conforme necessário enquanto lê essas instruções.

Aqui está uma visão geral do que você precisará:

- Uma imagem base para seu Dockerfile
- (Opcional) Um script de ponto de entrada de contêiner modificado

- Um Dockerfile que instala qualquer software necessário e inclui o script do contêiner

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 Docker arquivo BYOC pode ser: `FROM [IMAGE_URI_HERE]`

Em seguida, preencha o restante 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 estiver adicionando apenas software adicional a uma imagem pré-criada do Braket, você pode pular esta seção.

Para executar código não Python como parte de seu trabalho híbrido, você precisará modificar 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

thekick_off_customer_script()função para iniciar processos Rust dependendo do final da extensão do arquivo.

Você também pode optar por escrever um texto completamente novobraket_container.py. 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.

Um **Dockerfile** que instala qualquer software necessário e inclui o script do contêiner

> **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 SAGEMAKER_PROGRAM a braket_container.py variável de ambiente ou como você chamou 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 Docker Compose ao criar para permitir 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 Docker imagem

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 `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

Braket Hybrid Jobs Dockeras imagens 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 deve definir uma política de repositório para conceder as permissões apropriadas. Em geral, dê permissão apenas aos usuários e IAM funções específicos aos quais você deseja acessar suas imagens, em vez de permitir que qualquer pessoa com elas 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 `InstanceConfig` o. 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.p3.8xlarge", 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 de que `<provider>` e `<simulator_name>` deve consistir apenas em letras, números `_-`,, `.` e. A string está limitada a 256 caracteres.
>
> Se você planeja usar o BYOC e não está usando o SDK do Braket para criar tarefas quânticas, você deve passar o valor da variável ambiental `AMZN_BRAKET_JOB_TOKEN` para o `jobToken` parâmetro na solicitação. `CreateQuantumTask` Caso contrário, as tarefas quânticas não têm prioridade e são cobradas como tarefas quânticas autônomas regulares.

Configure o bucket padrão em **`AwsSession`**

Fornecer o seu próprio `AwsSession` oferece maior flexibilidade, por exemplo, na localização do seu bucket padrão. Por padrão, an `AwsSession` tem uma localização de bucket padrão `def"amazon-`

`braket-{id}-{region}"`. Mas você pode substituir esse padrão ao criar um`AwsSession`. Opcionalmente, os usuários podem passar um `AwsSession` objeto para `AwsQuantumJob.create` com o nome do parâmetro, `aws_session` conforme mostrado no exemplo de código a seguir.

```
aws_session = AwsSession(default_bucket="other-default-bucket")

# then you can use that AwsSession when creating a hybrid job
job = AwsQuantumJob.create(
    ...
    aws_session=aws_session
)
```

Interaja com trabalhos híbridos diretamente usando o API

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 doAPI. 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 `AmazonBraketFullAccess` gerenciada.

> **Note**
>
> Para obter mais informações sobre como obter uma função com a política `AmazonBraketFullAccess` gerenciada, consulte a página Habilitar o Amazon Braket.

Além disso, você precisa de uma função de execução. Essa função será passada para o serviço. Você pode criar a função usando o console Amazon Braket. Use a guia Funções de execução na página Permissões e configurações para criar uma função padrão para trabalhos híbridos.

Isso `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/Aspen-M-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 console `GetJob` API ou do console. Para obter os detalhes do trabalho híbrido da sessão do Python na qual você executou o `createJob` código, 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 uso do `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`

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.

Mitigação de erros em IonQ Aria

A mitigação de erros envolve a execução de vários circuitos físicos e a combinação de suas medições para obter um resultado aprimorado. O IonQ Aria dispositivo apresenta 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 de computadores quânticos por meio de simetrização.

> **Note**
>
> Usar a eliminação de polarização requer um mínimo de 2500 disparos.

Você pode executar uma tarefa quântica com despolarização em um IoQ Aria dispositivo usando o seguinte código:

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

```
from braket.circuits import Circuit
from braket.error_mitigation import Debias

device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1")
circuit = Circuit().h(0).cnot(0, 1)

task = device.run(circuit, shots=2500, device_parameters={"errorMitigation": Debias()})

result = task.result()
print(result.measurement_counts)
>>> {"00": 1245, "01": 5, "10": 10 "11": 1240} # result from debiasing
```

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 `additional_metadata` campo do SDK Braket `GateModelTaskResult` Python. Observe que a nitidez não retorna as contagens de medição, mas, em vez disso, 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
```

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 carga de trabalho e obter acesso antecipado aos recursos da próxima geração, como novos dispositivos quânticos com disponibilidade limitada.

Nesta seção:

- Reservas
- Conselhos de especialistas
- Capacidades experimentais

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 carga de trabalho começa e termina a execução. As reservas estão disponíveis em incrementos de 1 hora e podem ser canceladas com até 48 horas de antecedência, sem custo adicional. Você pode optar por enfileirar tarefas quânticas e trabalhos híbridos para uma reserva futura com antecedência ou enviar 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).

Os seguintes computadores quânticos estão disponíveis para reservas:

- Ária de IonQ
- Granada do IQM
- QuEraé Aquila
- Aspen-M-3 de Rigetti

Quando usar uma reserva

Aproveitar o acesso dedicado a dispositivos com reservas 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 cargas de trabalho 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:

- Crie uma reserva
- Execute sua carga de trabalho com uma reserva
- Cancelar ou reagendar uma reserva existente

Crie uma reserva

Para criar uma reserva, entre em contato com a equipe Braket seguindo estas etapas:

1. Abra o console 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 carga de trabalho, forneça todos os detalhes sobre a carga de trabalho a ser executada usando sua reserva. Por exemplo, duração desejada da reserva, restrições relevantes ou cronograma desejado.
6. Se você estiver interessado em entrar em contato com um especialista da Braket para uma sessão de preparação de reservas após a confirmação da reserva, opcionalmente, selecione Estou interessado em uma sessão preparatória.

Você também pode entrar em contato conosco para criar uma reserva seguindo estas etapas:

1. Abra o console Amazon Braket.

2. Escolha Dispositivos no painel esquerdo e escolha o dispositivo que você gostaria de reservar.
3. Na seção Resumo, escolha Reservar dispositivo.
4. Siga as etapas de 4 a 6 no procedimento anterior.

Depois de enviar o formulário, você receberá um e-mail da equipe Braket com as próximas etapas para criar sua reserva. Depois que sua reserva for confirmada, você receberá o ARN da reserva por e-mail.

> **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.

Se você indicou interesse em uma sessão de preparação de reservas, a equipe da Braket entra em contato com você por e-mail para agendar uma sessão de 30 minutos com um especialista da Braket.

Execute sua carga de trabalho com uma reserva

Durante uma reserva, somente suas cargas de trabalho são executadas no dispositivo. Para designar as tarefas quânticas e as tarefas híbridas a serem executadas durante uma reserva de dispositivo, você deve usar um ARN de reserva válido.

> **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. Além disso, o ARN da reserva só é válido no dispositivo reservado nos horários de início e término escolhidos.

Para aproveitar ao máximo seu tempo reservado, você pode optar por colocar tarefas e trabalhos na fila antes da reserva. Essas cargas de trabalho permanecem no `QUEUED` status até o início da reserva. Quando a reserva começa, todas as cargas de trabalho em fila são executadas no pedido enviado. As tarefas de Job são priorizadas antes das tarefas quânticas autônomas.

> **Note**
>
> Como somente suas cargas de trabalho são executadas durante sua reserva, não há visibilidade da fila para tarefas e trabalhos enviados com um ARN de reserva.

Exemplos de código para criar uma tarefa quântica para uma reserva:

1. Defina um circuito para preparar o estado GHZ no formato OpenQASM.

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

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

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

c = measure q;
```

2. Crie uma tarefa quântica usando seu circuito e o ARN de reserva.

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

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

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

program = Program(source=ghz_qasm_string)

# Reservation ARN will be of the form arn:aws:braket:us-
east-1:<AccountId>:reservation/<ReservationId>
# Example: arn:aws:braket:us-east-1:123456789012:reservation/f17cc20b-1ba4-461f-8854-
de4bb2aa64c1
###########################################################################
```

```
# IMPORTANT: If the reservation ARN is not specified, the created task
# queues and runs outside of the reservation.
# (The only exception is when the task is created by the script of a hybrid
#  job that had the reservation ARN passed at the time of its creation.
#  See “Code example for creating a hybrid job for a Braket Direct reservation:”
#  in the following section.)
###############################################################################
my_task = device.run(
    program,
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)

# You can also specify a particular Amazon S3 bucket location
# and the desired number of shots, when running the program.
# If no S3 location is specified, a default Amazon S3 bucket is chosen at amazon-
braket-{region}-{account_id}
# If no shot count is specified, 1000 shots are applied by default.
s3_location = ("amazon-braket-my-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)
```

Exemplo de código para criar um trabalho híbrido para uma reserva do Braket Direct:

1. Defina seu script de algoritmo.

```
//algorithm_script.py

from braket.aws import AwsDevice
from braket.circuits import Circuit

def start_here():

    print("Test job started!!!!!")

    # Use the device declared in the job script
    device = AwsDevice(os.environ["AMZN_BRAKET_DEVICE_ARN"])
```

```
    bell = Circuit().h(0).cnot(0, 1)
    for count in range(5):
        task = device.run(bell, shots=100)
        print(task.result().measurement_counts)

    print("Test job completed!!!!!")
```

2. Crie o trabalho híbrido usando seu script de algoritmo e o ARN da reserva.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)
```

3. Crie o trabalho híbrido usando o decorador remoto.

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.devices import Devices
from braket.jobs import hybrid_job, get_job_device_arn

@hybrid_job(device=Devices.IonQ.Aria1, reservation_arn="arn:aws:braket:us-
east-1:<AccountId>:reservation/<ReservationId>")
def sample_job():
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)
    task = device.run(bell, shots=10)
    measurements = task.result().measurements
    return measurements
```

O que acontece no final da sua reserva

Depois que sua reserva terminar, você não terá mais acesso dedicado ao dispositivo. Todas as cargas de trabalho restantes que estão na fila com essa reserva são automaticamente canceladas.

> **Note**
>
> Qualquer trabalho que estava em `RUNNING` status 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.

Conselhos de especialistas

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

Para explorar as opções de aconselhamento especializado via Braket Direct, abra o console 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 da Braket: O horário de atendimento da 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 aproveitar o Braket para 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 carga de trabalho e os tópicos de discussão desejados.
- Você receberá um convite de calendário para a próxima vaga disponível por e-mail.

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

- Ofertas de fornecedores de hardware quântico: IonQ, Oxford Quantum Circuits QuEra, e Rigetti fornecem ofertas de serviços profissionais via. 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.
- AmazonQuantum Solutions Lab (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 Connect 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.

Capacidades experimentais

Para aprimorar suas cargas de trabalho de pesquisa, é importante ter acesso rápido a novos recursos inovadores. Com o Braket Direct, você pode solicitar acesso aos recursos experimentais disponíveis, como novos dispositivos quânticos com disponibilidade limitada, diretamente no console do Braket.

Alguns recursos experimentais operam fora das especificações padrão do dispositivo e precisam de orientação prática adaptada ao seu caso de uso. Para garantir que suas cargas de trabalho estejam

configuradas para o sucesso, o acesso está disponível mediante solicitação por meio do Braket Direct.

Acesso somente para reservas ao IonQ Forte

Com o Braket Direct, você obtém acesso somente para reservas ao IonQ Forte QPU. Devido à sua disponibilidade limitada, este dispositivo só está disponível por meio do Braket Direct.

Para saber mais e solicitar acesso ao IonQ Forte, siga estas etapas:

1. Abra o console Amazon Braket.
2. Selecione Braket Direct no menu à esquerda e, em Recursos experimentais, navegue até IonQ Forte. Escolha Exibir dispositivo.
3. Na página de detalhes do dispositivo Forte, em Resumo, escolha Reservar dispositivo.
4. Forneça suas informações de contato, incluindo nome e e-mail. Forneça um endereço de e-mail válido que você verifique regularmente.
5. Em Conte-nos sobre sua carga de trabalho, forneça detalhes sobre a carga de trabalho a ser executada usando sua reserva, como a duração da reserva desejada, as restrições relevantes ou o cronograma desejado.
6. (Opcional) Se você estiver interessado em entrar em contato com um especialista da Braket para uma sessão de preparação de reservas após a confirmação da reserva, selecione Estou interessado em uma sessão preparatória.

Depois que o formulário for enviado, a equipe do Braket entrará em contato com você com as próximas etapas.

> **Note**
>
> Devido à disponibilidade limitada do dispositivo, o acesso ao Forte é limitado. Entre em contato conosco para saber mais.

Acesso ao desvio local em Aquila QuEra

Com o Braket Direct, você pode solicitar acesso para controlar o desajuste local ao programar na QPU. QuEra Aquila Com esse recurso, você pode ajustar o quanto o campo de condução afeta cada qubit específico.

Para saber mais e solicitar acesso a esse recurso, siga estas etapas:

1. Abra o console Amazon Braket.
2. Selecione Braket Direct no menu à esquerda e, em Recursos experimentais, navegue até QuEra Aquila - desajuste local. Escolha Obter acesso.
3. Forneça suas informações de contato, incluindo nome e e-mail. Forneça um endereço de e-mail válido que você verifique regularmente.
4. Em Conte-nos sobre sua carga de trabalho, forneça detalhes sobre a carga de trabalho e onde você planeja usar esse recurso.

Acesso a geometrias altas em Aquila QuEra

Com o Braket Direct, você pode solicitar acesso a geometrias expandidas ao programar na QPU. QuEra Aquila Com esse recurso, você pode experimentar além dos recursos padrão do dispositivo e especificar geometrias com maior altura de rede.

Para saber mais e solicitar acesso a esse recurso, siga estas etapas:

1. Abra o console Amazon Braket.
2. Selecione Braket Direct no menu à esquerda e, em Recursos experimentais, navegue até QuEra Aquila - geometrias altas. Escolha Obter acesso.
3. Forneça suas informações de contato, incluindo nome e e-mail. Forneça um endereço de e-mail válido que você verifique regularmente.
4. Em Conte-nos sobre sua carga de trabalho, forneça detalhes sobre a carga de trabalho e onde você planeja usar esse recurso.

Acesso a geometrias estreitas em Aquila QuEra

Com o Braket Direct, você pode solicitar acesso a geometrias expandidas ao programar na QPU. QuEra Aquila Com esse recurso, você pode experimentar além dos recursos padrão do dispositivo e organizar linhas de treliça com espaçamento vertical mais apertado.

Para saber mais e solicitar acesso a esse recurso, siga estas etapas:

1. Abra o console Amazon Braket.
2. Selecione Braket Direct no menu à esquerda e, em Recursos experimentais, navegue até QuEra Aquila - geometrias altas. Escolha Obter acesso.

3. Forneça suas informações de contato, incluindo nome e e-mail. Forneça um endereço de e-mail válido que você verifique regularmente.
4. Em Conte-nos sobre sua carga de trabalho, forneça detalhes sobre a carga de trabalho e onde você planeja usar esse recurso.

Registro e Monitoramento

Depois de enviar uma tarefa quântica, você pode acompanhar seu status por meio do SDK e do console do Amazon Braket. Quando a tarefa quântica é concluída, o Braket salva os resultados em sua localização especificada no Amazon S3. A conclusão pode levar algum tempo, especialmente para dispositivos QPU, dependendo do tamanho da fila. Os tipos de status incluem:

- `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
- Eventos e ações automatizadas para o Amazon Braket com a Amazon EventBridge
- Monitorando o Amazon Braket com a Amazon CloudWatch
- Registro da API Amazon Braket com CloudTrail
- Crie uma instância de notebook Amazon Braket usando AWS CloudFormation
- Registro avançado

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 `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 liga`task.result()`, o SDK começa a pesquisar 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 `cancel()` método, 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
```

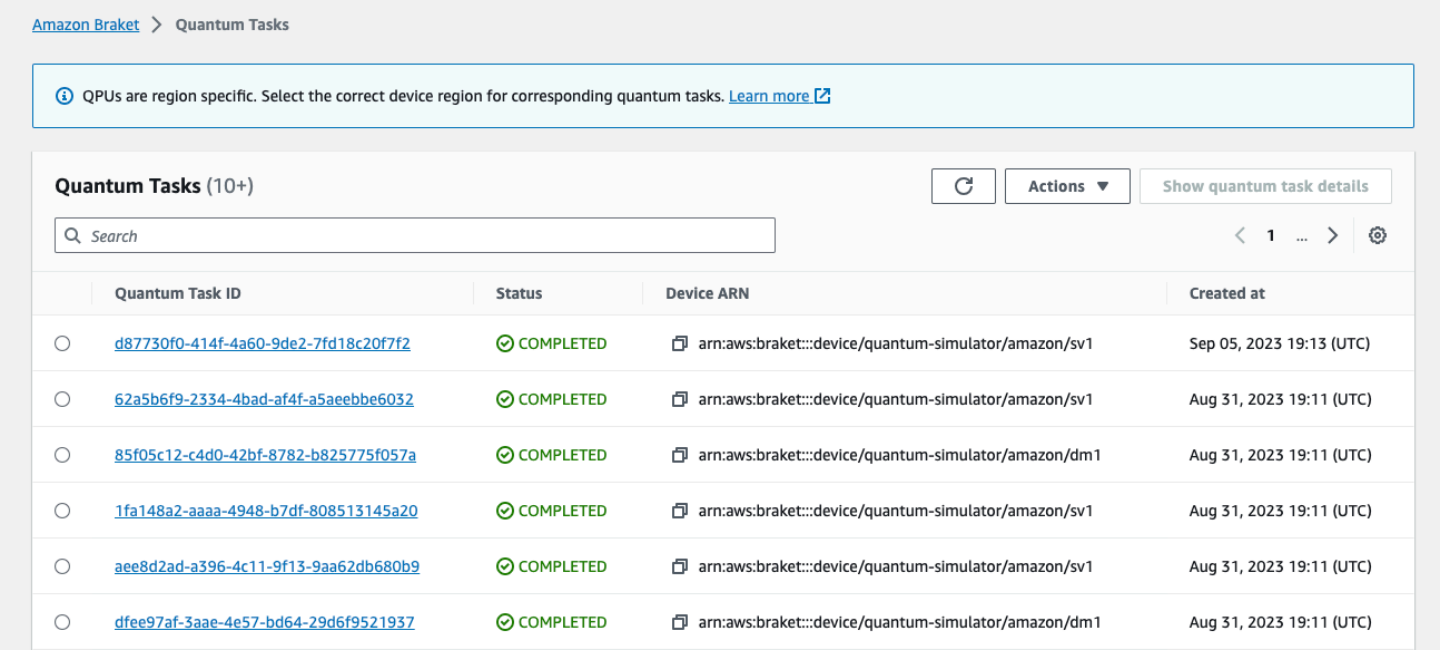
Recupere uma tarefa ou resultado quântico

Se o kernel morrer após o envio da tarefa quântica ou se você fechar o notebook ou o computador, você poderá reconstruir o `task` objeto com seu ARN (ID de tarefa quântica) exclusivo. Em seguida, você pode ligar `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()
```

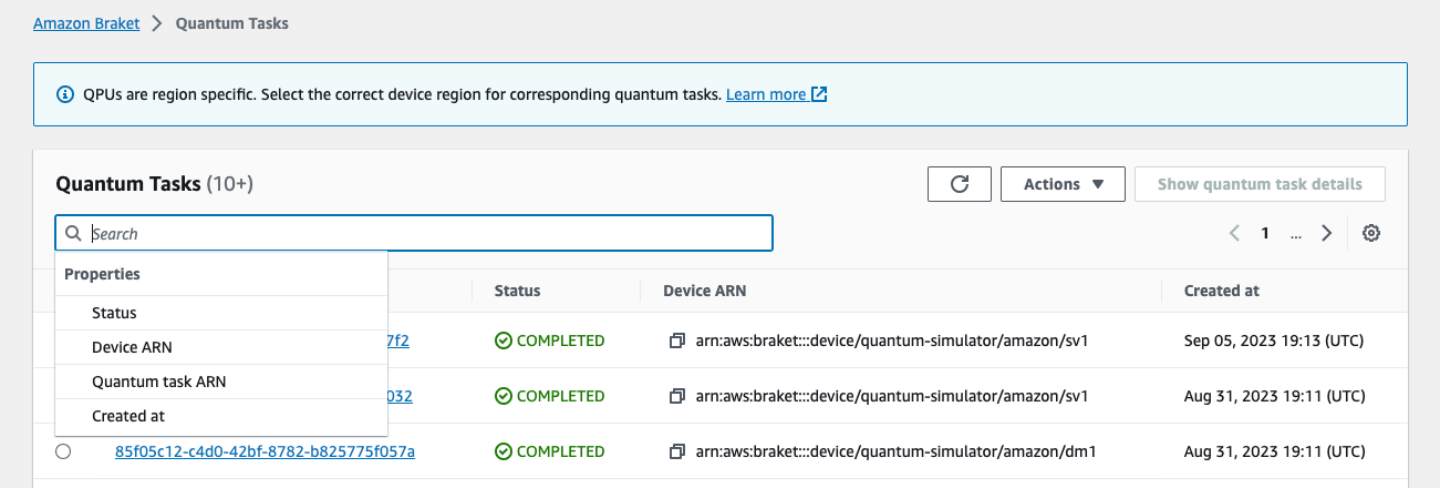
Monitoramento de tarefas quânticas por meio do console Amazon Braket

AmazonO 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

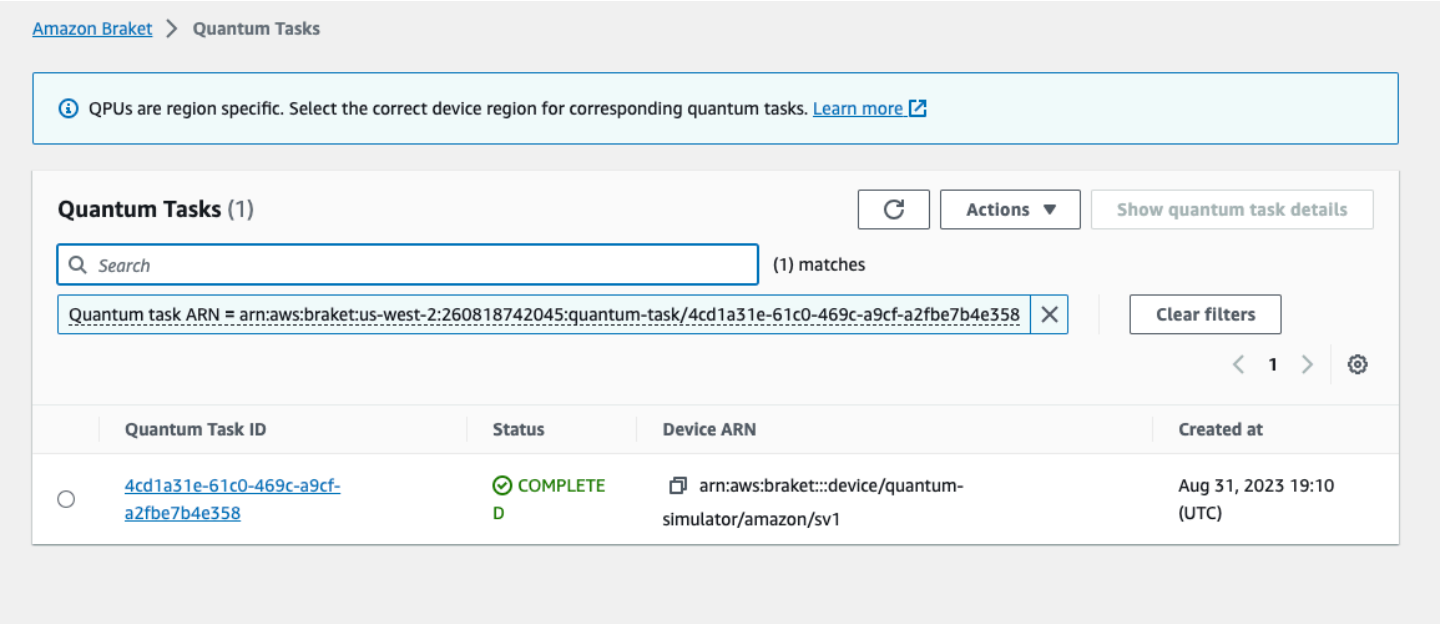


Você pode pesquisar tarefas quânticas específicas por meio da barra de navegação. A pesquisa pode ser baseada no ARN (ID) da Quantum Task, status, dispositivo e hora de criação. As opções

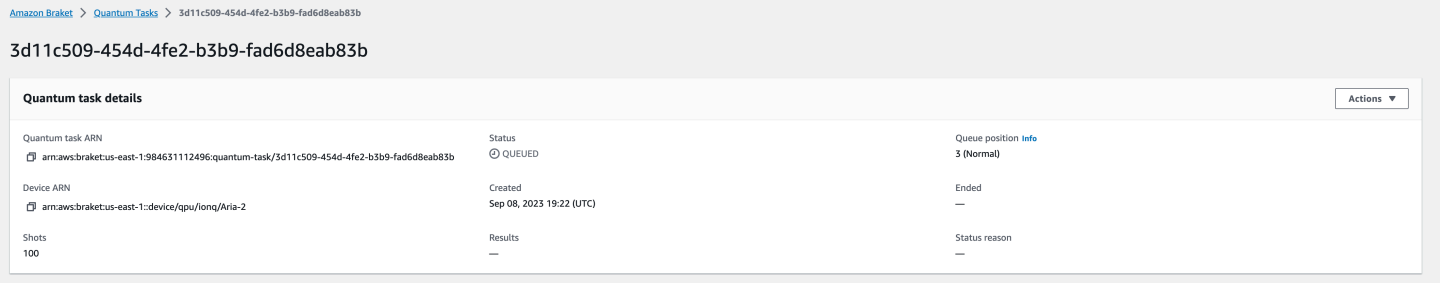
aparecem automaticamente quando você seleciona a barra de navegação, conforme mostrado no exemplo a seguir.



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 QUEUED estado. 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 ou a AWS atribui a um recurso da AWS. 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 notebook 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, AWS CLI gerenciar as tags associadas por meio do console ouAPI.

Usar tags

As tags podem organizar seus recursos em categorias que são úteis para você. Por exemplo, você pode atribuir uma tag “Departamento” para especificar o departamento que possui esse recurso.

Cada tag tem duas partes:

- Uma chave de tag (por exemplo CostCenter, Ambiente ou Projeto). Chaves de tag fazem distinção entre maiúsculas e minúsculas.
- Um campo opcional conhecido como valor de tag (por exemplo, 111122223333 ou Produção). Omitir o valor da tag é o mesmo que usar uma string vazia. Como chaves de tag, os valores das tags diferenciam maiúsculas de minúsculas.

As tags ajudam você a fazer o seguinte:

- Identifique e organize seus AWS recursos. Muitos Serviços da AWS são compatíveis com marcação e permitem que você atribua a mesma tag a atributos de diferentes serviços, para indicar que estes atributos estão relacionados.
- Acompanhe seus AWS custos. Você pode ativar essas tags no painel AWS Billing and Cost Management. AWS usa tags para categorizar seus custos e entregar um relatório mensal de alocação de custos a você. Para obter mais informações, consulte Usar etiquetas de alocação de custos no Guia do UsuárioAWS Billing and Cost Management.
- Controle o acesso aos seus AWS recursos. Para obter mais informações, consulte Controle de acesso usando tags.

Mais sobre AWS e tags

- Para obter informações gerais sobre marcação, incluindo convenções de nomenclatura e uso, consulte AWSRecursos de marcação na Referência geral. AWS
- Para obter informações sobre restrições à marcação, consulte Limites e requisitos de nomenclatura de tags na Referência AWSgeral.
- Para ver as melhores práticas e estratégias de marcação, consulte Práticas recomendadas de marcação e Estratégias de AWSmarcação.
- 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 informações mais específicas sobre tags para Amazon Braket.

Recursos suportados no Amazon Braket

O tipo de recurso a seguir no Amazon Braket oferece suporte à 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 da Amazon. SageMaker Para obter mais informações, consulte Metadados da instância do notebook na SageMaker documentação.

Restrições de tags

As seguintes restrições básicas se aplicam 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 tags como propriedades em um recurso. Você pode visualizar, adicionar, modificar, listar e excluir tags por meio do console Amazon Braket, do Amazon Braket API ou do. AWS CLI Para obter mais informações, consulte a referência da API Amazon Braket.

Adicionar tags

Você pode adicionar tags aos recursos marcáveis nos seguintes momentos:

- Ao criar o recurso: use o console ou inclua o `Tags` parâmetro com a `Create` operação na AWSAPI.
- Depois de criar o recurso: use o console para navegar até a tarefa quântica ou o recurso do notebook, ou chame a `TagResource` operação na AWSAPI.

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`

Editar tags

Você pode editar tags usando o console para navegar até a tarefa quântica ou o recurso do notebook ou pode usar o comando a seguir para modificar o valor de uma tag anexada a um recurso etiquetável. Quando você especifica uma chave de tag que já existe, o valor dessa chave é sobrescrito:

- AWS API: `TagResource`

Remover etiquetas

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 notebook ou ao chamar a `UntagResource` operação.

- AWS API: `UntagResource`

Exemplo de marcação de CLI no Amazon Braket

Se você estiver trabalhando com a AWS CLI, aqui está um exemplo de comando que mostra como criar uma tag que se aplica a uma tarefa quântica para a qual você cria SV1 com as configurações de parâmetros da Rigetti QPU. Observe que a tag está especificada no final do comando de exemplo. Nesse caso, Key recebe o valor **`state`** e Value recebe o valor`Washington`.

```
aws braket create-quantum-task --action /
"{\"braketSchemaHeader\": {\"name\": \"braket.ir.jaqcd.program\", /
    \"version\": \"1\"}, /
    \"instructions\": [{\"angle\": 0.15, \"target\": 0, \"type\": \"rz\"}], /
```

```
    \"results\": null, /
    \"basis_rotation_instructions\": null}" /
  --device-arn "arn:aws:braket:::device/quantum-simulator/amazon/sv1" /
  --output-s3-bucket  "my-example-braket-bucket-name" /
  --output-s3-key-prefix "my-example-username"  /
  --shots 100   /
  --device-parameters /
  "{\"braketSchemaHeader\": /
     {\"name\": \"braket.device_schema.rigetti.rigetti_device_parameters\", /
      \"version\": \"1\"}, \"paradigmParameters\": /
       {\"braketSchemaHeader\": /
         {\"name\": \"braket.device_schema.gate_model_parameters\", /
          \"version\": \"1\"}, /
          \"qubitCount\": 2}}" /
          --tags {\"state\":\"Washington\"}
```

Marcar com o Amazon Braket API

- Se você estiver usando o Amazon API Braket para configurar tags em um recurso, ligue para o. TagResourceAPI

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tags {\"city\":
\"Seattle\"}
```

- Para remover tags de um recurso, chame UntagResourceAPIo.

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- Para listar todas as tags anexadas a um recurso específico, chame ListTagsForResourceAPIo.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city
\",\"state\"]"
```

Eventos e ações automatizadas para o Amazon Braket com a Amazon EventBridge

A Amazon EventBridge monitora eventos de mudança de status nas tarefas quânticas do Amazon Braket. Os eventos da Amazon Braket são entregues EventBridge, quase em tempo real. É possível

escrever regras simples para indicar quais eventos são interessantes para você e quais ações automatizadas devem ser realizadas quando um evento corresponder a uma regra. As ações automáticas que podem ser acionadas incluem:

- Invocar uma função do AWS Lambda
- Ativar a máquina de estado do AWS Step Functions
- Notificar um tópico do Amazon SNS

EventBridge monitora esses eventos de mudança de status do Amazon Braket:

- O estado das mudanças na tarefa quântica

AmazonO 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 Eventos e padrões de eventos em 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 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 em https://console.aws.amazon.com/events/.
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 `detail-type` seção conforme mostrado no código a seguir:

```
{
  "source": [
    "aws.braket"
  ]
}
```

- Para Tipos de destino, escolha AWS service (Serviço da AWS), e para 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 fazer 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 da tarefa do Amazon Braket Quantum, consulte Eventos e padrões de eventos em. 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 shots solicitações 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 um evento Amazon Braket Quantum Task Status Change.

```
{
    "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 o Amazon Braket com a Amazon 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 sobre o desempenho do Amazon Braket. Para saber mais, consulte Como usar CloudWatch métricas.

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	Descrição
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 `deviceArn` parâmetro, que tem o formato arn:aws:braket: ::device/xxx.

Dispositivos compatíveis

Para obter uma lista de dispositivos compatíveis e ARNs de dispositivos, consulte Dispositivos Braket.

> **Note**
>
> Você pode visualizar os fluxos de CloudWatch log dos notebooks Amazon Braket navegando até a página de detalhes do Notebook no console da Amazon. SageMaker Configurações adicionais do notebook Amazon Braket estão disponíveis no console. SageMaker

Registro da API Amazon Braket com CloudTrail

AmazonO Braket é integrado comAWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou um AWS service (Serviço da AWS) no Amazon Braket. CloudTrail captura todas as API chamadas para Amazon Braket como eventos. As chamadas capturadas incluem chamadas do console do Amazon Braket e chamadas de código para as operações do Amazon Braket Braket. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo eventos para 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 à Amazon Braket, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais sobre isso CloudTrail, consulte o Guia AWS CloudTrail do usuário.

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 um CloudTrail evento junto com outros AWS service (Serviço da AWS) eventos no histórico de eventos. Você pode exibir, pesquisar e baixar eventos recentes em sua Conta da AWS. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

Para um registro contínuo dos eventos em seuConta 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 loga eventos de todas as Regiões na partição da AWS e entrega os arquivos de log para o bucket do Amazon S3 especificado por você. Além disso, você pode configurar outros Serviços da AWS para analisar e agir com base nos dados do evento coletados nos CloudTrail registros. Para obter mais informações, 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:

- Se a solicitação foi feita com credenciais de segurança temporárias de uma função 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 .

Entendendo as entradas do arquivo 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 registro da `GetQuantumTask` ação, 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 mostrada uma entrada de registro para a `GetDevice` ação, 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"
}
```

Crie uma instância de notebook Amazon Braket usando AWS CloudFormation

Você pode usar AWS CloudFormation para gerenciar suas instâncias de notebook Amazon Braket. As instâncias do notebook Braket são criadas na Amazon. SageMaker 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. Você pode criar, atualizar e excluir instâncias de forma ordenada e repetível. Você pode achar isso útil ao gerenciar várias instâncias do notebook Braket em você. Conta da AWS

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

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

1. Crie um script de configuração do SageMaker ciclo de vida da Amazon.
2. Crie uma função AWS Identity and Access Management (IAM) a ser assumida pela SageMaker.
3. Crie uma instância de SageMaker notebook 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 a função do IAM para os notebooks Braket aos quais você atribui as mesmas permissões de execução.

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

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

```
BraketNotebookInstanceLifecycleConfig:
    Type: "AWS::SageMaker::NotebookInstanceLifecycleConfig"
    Properties:
      NotebookInstanceLifecycleConfigName: BraketLifecycleConfig-${AWS::StackName}
      OnStart:
```

```
      - Content:
          Fn::Base64: |
            #!/usr/bin/env bash

            sudo -u ec2-user -i #EOS
            aws s3 cp s3://braketnotebookcdk-prod-i-
notebooklccs3bucketb3089-1cysh30vzj2ju/notebook/braket-notebook-lcc.zip braket-
notebook-lcc.zip
            unzip braket-notebook-lcc.zip
            ./install.sh
            EOS

            exit 0
```

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

Quando você usa uma instância do notebook Braket, SageMaker executa operações em seu nome. Por exemplo, suponha que você execute um notebook Braket usando um circuito em um dispositivo compatível. Dentro da instância do notebook, SageMaker executa a operação no Braket para você. A função de execução do notebook define as operações exatas SageMaker que podem ser executadas em seu nome. Para obter mais informações, consulte as SageMaker funções no guia do SageMaker desenvolvedor da Amazon.

Use o exemplo a seguir para criar uma função de execução do notebook Braket com as permissões necessárias. Você pode modificar as políticas de acordo com suas necessidades.

> **Note**
>
> Certifique-se de que a função tenha permissão para as `s3:GetObject` operações `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 notebook 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 SageMaker notebook da Amazon com o prefixo **`amazon-braket-`**

Use o script de SageMaker ciclo de vida e a função do IAM criada nas etapas 1 e 2 para criar uma instância de SageMaker notebook. A instância do notebook 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 AWS 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
```

Registro avançado

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 `poll_interval_seconds` parâmetros `poll_timeout_seconds` e, 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 notebook 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 digitando o seguinte comando.

```
# print logs
! cat {log_file}
```

```
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: start polling for completion
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status CREATED
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status CREATED
```

```
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status QUEUED
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status RUNNING
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status RUNNING
Task arn:aws:braket:us-west-2:123412341234:quantum-
task/5088ec6c-89cf-4338-9750-9f5bb12a0dc4: task status COMPLETED
Counter({'00001': 493, '00011': 493, '01001': 5, '10111': 4, '01011': 3, '10101': 2})
```

Obtenha o ARN do arquivo de log

Da saída do arquivo de log que é 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()
```

Segurança no Amazon Braket

Este capítulo ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Amazon Braket. Ele mostra como configurar o Amazon Braket para atender aos seus objetivos de segurança e conformidade. Você também aprende a usar outros Serviços da AWS que ajudam você a monitorar e proteger seus recursos do Amazon Braket.

A segurança na nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você contará com um datacenter e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança. Você é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da sua empresa e as leis e regulamentações aplicáveis.

Responsabilidade compartilhada pela segurança

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isto como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa os Serviços da AWS na Nuvem AWS. A 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 saber mais sobre os programas de conformidade que se aplicam ao Amazon Braket, AWSconsulte Serviços no escopo por 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 dos 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 nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para mais

informações sobre a proteção de dados na Europa, consulte o artigo AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja asConta da AWS credenciais da e configure as contas de usuário individuais com o AWS IAM Identity Center ou o 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 [multi-factor authentication]) com cada conta.
- Use SSL/TLS para se comunicar com os atributos da AWS. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure o registro em log das atividades da API e do usuário com o .AWS CloudTrail
- Use AWS as soluções de criptografia da , juntamente com todos os controles de segurança padrão dos Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar a AWS por meio de uma interface de linha de comandos ou uma API, use um endpoint do FIPS. Para ter mais informações sobre endpoints do FIPS, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Name (Nome). Isso inclui quando você trabalha com o Amazon Braket ou Serviços da AWS outro usando o console, a API AWS CLI ou os SDKs. AWS Quaisquer dados inseridos em tags ou 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, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Retenção de dados

Depois de 90 dias, o Amazon Braket remove automaticamente todas as IDs de 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.

Gerenciando o acesso ao Amazon Braket

Este capítulo descreve as permissões necessárias para executar o Amazon Braket ou restringir o acesso de usuários e funções específicos. Você pode conceder (ou negar) as permissões necessárias para qualquer usuário ou função em sua conta. Para fazer isso, anexe a política apropriada do Amazon Braket a esse usuário ou função 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 funções
- Sobre a AmazonBraketFullAccess política
- Sobre a AmazonBraketJobsExecutionPolicy política
- Restringir o acesso do usuário a determinados dispositivos
- Atualizações do Amazon Braket para políticas gerenciadas AWS
- Restrinja o acesso do usuário a determinadas instâncias do notebook
- 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 Amazon Resource Name (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 funções

Você pode usar o tipo de recurso notebook no Braket. Um notebook é um SageMaker recurso da Amazon que o Braket pode compartilhar. Para usar um notebook com o Braket, você deve especificar uma função do IAM com um nome que comece com. `AmazonBraketServiceSageMakerNotebook`

Para criar um notebook, você deve usar uma função com permissões de administrador ou que tenha a seguinte política embutida anexada a ela.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": "iam:CreateRole",
           "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
       },
       {
           "Effect": "Allow",
           "Action": "iam:CreatePolicy",
           "Resource": [
               "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
               "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": "iam:AttachRolePolicy",
           "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
           "Condition": {
               "StringLike": {
                   "iam:PolicyARN": [
                       "arn:aws:iam::aws:policy/AmazonBraketFullAccess",
                       "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
                       "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
                   ]
```

```
                }
            }
        }
    ]
}
```

Para criar a função, siga as etapas fornecidas na página Criar um caderno de anotações ou peça ao administrador que a crie para você. Certifique-se de que a AmazonBraketFullAccesspolítica esteja anexada.

Depois de criar a função, você pode reutilizá-la em todos os cadernos que você lançar no futuro.

Sobre a AmazonBraketFullAccess política

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 funções para controlar recursos — Para criar uma função vinculada 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 de 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 registros de trabalhos híbridos. Abrange o caminho correto do Braket e permita colocar dados de registro. Insira dados métricos CloudWatch.
- Crie e armazene dados nos buckets do Amazon S3 e liste todos os buckets — Para criar buckets do S3, liste os buckets do S3 em sua conta e coloque objetos 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.

- Passe as funções do IAM — Para passar as funções do IAM para `CreateJob` API o.
- Amazon SageMaker Notebook — Para criar e gerenciar instâncias de SageMaker notebook com o escopo do recurso “arn:aws:sagemaker: ::notebook-instance/amazon-braket-”.
- Valide as cotas de serviço — Para criar SageMaker notebooks e trabalhos do Amazon Braket Hybrid, sua contagem de recursos não pode exceder as cotas da sua conta.

Conteúdo da política

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:CreateBucket",
                "s3:PutBucketPublicAccessBlock",
                "s3:PutBucketPolicy"
            ],
            "Resource": "arn:aws:s3:::amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "servicequotas:GetServiceQuota",
                "cloudwatch:GetMetricData"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ],
```

```
            "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:ListRolePolicies",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:ListNotebookInstances"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreatePresignedNotebookInstanceUrl",
```

```
                "sagemaker:CreateNotebookInstance",
                "sagemaker:DeleteNotebookInstance",
                "sagemaker:DescribeNotebookInstance",
                "sagemaker:StartNotebookInstance",
                "sagemaker:StopNotebookInstance",
                "sagemaker:UpdateNotebookInstance",
                "sagemaker:ListTags",
                "sagemaker:AddTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "arn:aws:sagemaker:*:*:notebook-instance/amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeNotebookInstanceLifecycleConfig",
                "sagemaker:CreateNotebookInstanceLifecycleConfig",
                "sagemaker:DeleteNotebookInstanceLifecycleConfig",
                "sagemaker:ListNotebookInstanceLifecycleConfigs",
                "sagemaker:UpdateNotebookInstanceLifecycleConfig"
            ],
            "Resource": "arn:aws:sagemaker:*:*:notebook-instance-lifecycle-config/
amazon-braket-*"
        },
        {
            "Effect": "Allow",
            "Action": "braket:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/braket.amazonaws.com/
AWSServiceRoleForAmazonBraket*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "braket.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
```

```
            ],
            "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "sagemaker.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketJobsExecutionRole*",
            "Condition": {
                "StringLike": {
                    "iam:PassedToService": [
                        "braket.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:GetQueryResults"
            ],
            "Resource": [
                "arn:aws:logs:*:*:log-group:*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "/aws/braket"
                }
            }
        }
    ]
}
```

Sobre a AmazonBraketJobsExecutionPolicy política

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 os 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 registros de trabalhos híbridos. Abrange o caminho correto do Braket e permita colocar dados de registro. 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. As funções devem estar em conformidade com o formato arn:aws:iam: :* *. :role/service-role/ AmazonBraketJobsExecutionRole

```
"Version": "2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
```

```
    "Action": [
     "s3:GetObject",
     "s3:PutObject",
     "s3:ListBucket",
     "s3:CreateBucket",
     "s3:PutBucketPublicAccessBlock",
     "s3:PutBucketPolicy"
    ],
    "Resource": "arn:aws:s3:::amazon-braket-*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "ecr:GetDownloadUrlForLayer",
     "ecr:BatchGetImage",
     "ecr:BatchCheckLayerAvailability"
    ],
    "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "ecr:GetAuthorizationToken"
    ],
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "braket:CancelJob",
     "braket:CancelQuantumTask",
     "braket:CreateJob",
     "braket:CreateQuantumTask",
     "braket:GetDevice",
     "braket:GetJob",
     "braket:GetQuantumTask",
     "braket:SearchDevices",
     "braket:SearchJobs",
     "braket:SearchQuantumTasks",
     "braket:ListTagsForResource",
     "braket:TagResource",
     "braket:UntagResource"
    ],
    "Resource": "*"
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": "arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*",
   "Condition": {
    "StringLike": {
     "iam:PassedToService": [
      "braket.amazonaws.com"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:ListRoles"
   ],
   "Resource": "arn:aws:iam::*:role/*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "logs:GetQueryResults"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "logs:PutLogEvents",
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:GetLogEvents",
    "logs:DescribeLogStreams",
    "logs:StartQuery",
    "logs:StopQuery"
   ],
   "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
  },
```

```
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": "/aws/braket"
    }
   }
  }
 ]
}
```

Restringir o acesso do usuário a determinados dispositivos

Para restringir o acesso de determinados usuários a determinados dispositivos Braket, você pode adicionar uma política de negação de permissões a uma função específicaIAM.

As ações a seguir podem ser restringidas com essas permissões:

- `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 todas as QPUs do. Conta da AWS `123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
  ]
```

```
}
```

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>:<account id>:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:<account id>:device/quantum-simulator/<provider>/<device_id>`

Aqui estão alguns exemplos de vários tipos de acesso a dispositivos

- Para selecionar todas as QPUs em todas as regiões: `arn:aws:braket:*:*:device/qpu/*`
- Para selecionar todas as QPUs SOMENTE na região us-west-2: `arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- Da mesma forma, para selecionar SOMENTE todas as 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:* :123456789012:device/quantum-simulator/*`
- Para restringir o acesso ao IonQ Harmony dispositivo na região us-east-1: `arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- Para restringir o acesso a dispositivos de um determinado provedor (por exemplo, a Rigetti QPU dispositivos): `arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- Para restringir o acesso ao TN1 dispositivo: `arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

Atualizações do Amazon Braket para políticas gerenciadas AWS

A tabela a seguir fornece detalhes sobre as atualizações das políticas AWS gerenciadas do Braket desde que esse serviço começou a rastrear essas alterações.

Alteração	Descrição	Data
AmazonBraketFullAccess- Política de acesso total para Braket	Foram adicionadas as ações servicequotas: GetServiceQuota e cloudwatch: GetMetricData a serem incluídas na política. AmazonBraketFullAccess .	24 de março de 2023
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 híbridos para Amazon Braket Hybrid Jobs	Braket atualizou o ARN da função de execução de trabalhos híbridos para incluir o caminho. `service-role/`	29 de novembro de 2021
Braket começou a rastrear as mudanças	A Braket começou a monitorar as mudanças em suas políticas AWS gerenciadas.	29 de novembro de 2021

Restrinja o acesso do usuário a determinadas instâncias do notebook

Para restringir o acesso de determinados usuários a instâncias específicas do notebook Braket, você pode adicionar uma política de negação de permissões a uma função, 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`).

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

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 uma função, 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.

```
{
  "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 notebook, você pode adicionar a política anterior à função de execução do notebook.

Função vinculada ao serviço Amazon Braket

Quando você ativa o Amazon Braket, uma função vinculada ao serviço é criada em sua conta.

Uma função vinculada ao serviço é um tipo exclusivo de função do IAM que, nesse caso, está vinculada diretamente ao Amazon Braket. A função vinculada ao serviço Amazon Braket é predefinida para incluir todas as permissões que a Braket exige ao ligar para outra pessoa em seu nome. Serviços da AWS

Uma função vinculada ao serviço facilita a configuração do Amazon Braket porque você não precisa adicionar as permissões necessárias manualmente. O Amazon Braket define as permissões de suas funções vinculadas ao serviço. A menos que você altere essas definições, somente o Amazon Braket pode assumir suas funções. 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.

A função vinculada ao serviço que o Amazon Braket configura faz parte da capacidade de funções vinculadas ao serviço AWS Identity and Access Management (IAM). Para obter informações sobre outros Serviços da AWS que oferecem suporte a funções vinculadas a serviços, consulte AWSServiços que funcionam com o IAM e procure os serviços que têm Sim na coluna Função vinculada ao serviço. Escolha um Sim com um link para visualizar a documentação da função vinculada a esse serviço.

Permissões de função vinculada ao serviço para o Amazon Braket

O Amazon Braket usa a função `AWSServiceRoleForAmazonBraket` vinculada ao serviço que confia na entidade braket.amazonaws.com para assumir a função.

Você deve configurar permissões para permitir que uma entidade do IAM (como um grupo ou função) crie, edite ou exclua uma função vinculada ao serviço. Para obter mais informações, consulte Permissões de funções vinculadas ao serviço.

Por padrão, a função vinculada ao serviço no Amazon Braket recebe as seguintes permissões:

- Amazon S3 — permissões para listar os buckets em sua conta e colocar objetos em 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 registros, criar os fluxos de log associados e colocar eventos no grupo de registros criado para o Amazon Braket.

A política a seguir está anexada à função **`AWSServiceRoleForAmazonBraket`**vinculada ao serviço:

```
{"Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket"
```

```
            ],
            "Resource": "arn:aws:s3:::amazon-braket*"
        },
        {"Effect": "Allow",
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket/*"
        },
        {"Effect": "Allow",
            "Action": "braket:*",
            "Resource": "*"
        },
        {"Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/braket.amazonaws.com/
AWSServiceRoleForAmazonBraket*",
            "Condition": {"StringEquals": {"iam:AWSServiceName": "braket.amazonaws.com"
                }
            }
        }
    ]
}
```

Resiliência no Amazon Braket

A infraestrutura AWS global é construída em torno Regiões da AWS de zonas de disponibilidade.

Cada região fornece várias zonas de disponibilidade que são fisicamente separadas e isoladas. Essas zonas de disponibilidade (AZs) são conectadas por meio de redes de baixa latência, alto rendimento e altamente redundantes. Como resultado, as zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis do que as infraestruturas tradicionais de um ou vários datacenters.

Você pode projetar e operar aplicativos e bancos de dados que realizam o failover entre AZs automaticamente, sem interrupção.

Para obter mais informações Regiões da AWS e zonas de disponibilidade, consulte Infraestrutura AWS global.

Validação de conformidade para Amazon Braket

Auditores terceirizados avaliam regularmente a segurança e a conformidade do Amazon Braket e nossa integração com fornecedores de hardware terceirizados. Para obter uma up-to-date lista de informações de conformidade do Braket, consulte Serviços da AWSescopo por programa de conformidade. Para obter informações gerais, consulte AWSconformidade.

Você pode baixar relatórios de auditoria de terceiros usando o AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact.

> **Note**
>
> AWSos relatórios de conformidade não abrangem QPUs de fornecedores de hardware terceirizados que podem optar por passar por suas próprias auditorias independentes.

Sua responsabilidade de conformidade ao usar o Amazon Braket é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade da sua empresa e pelas leis e regulamentações aplicáveis. AWSfornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido de segurança e conformidade – Esses guias de implantação discutem considerações sobre arquitetura e fornecem medidas para implantar ambientes de linha de base focados em segurança e conformidade na AWS.
- Recursos de compatibilidade da AWS – Esta coleção de guias e pastas de trabalho pode ser aplicada ao seu setor e local.

Segurança de infraestrutura no Amazon Braket

Como um serviço gerenciado, o Amazon Braket é protegido pelos procedimentos AWS globais de segurança de rede descritos no whitepaper AWS: Visão geral dos processos de segurança.

Para acessar o Amazon Braket pela rede, você faz chamadas para AWS APIs publicadas. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.2 ou posterior. Os clientes também devem oferecer suporte a conjuntos de cifras com sigilo direto perfeito (PFS), como

Ephemeral Diffie-Hellman (DHE) ou Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Segurança dos fornecedores de hardware Amazon Braket

As QPUs no Amazon Braket são hospedadas por fornecedores de hardware terceirizados. 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 pelaAWS. Informações sobre a localização física eAWSA região em que cada QPU está disponível pode ser encontrada noDetalhes do dispositivoseção do console Amazon Braket.

Seu conteúdo é anonimizado. Somente o conteúdo necessário para processar o circuito é enviado a terceiros.Conta da AWSas informações não são transmitidas a terceiros.

Seus dados são criptografados em repouso e em trânsito. 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 Amazon VPC para Amazon Braket

Você pode estabelecer uma conexão privada entre sua VPC e o Amazon Braket criando uma interface VPC endpoint. Os endpoints de interface são alimentados por AWS PrivateLinkuma tecnologia que permite o acesso às APIs do Braket sem um gateway de internet, dispositivo NAT,

conexão VPN ou conexão. AWS Direct Connect As instâncias em sua VPC não precisam de endereços IP públicos para se comunicar com as APIs do Braket.

Cada endpoint de interface é representado por uma ou mais Interfaces de Rede Elástica nas sub-redes.

Com isso 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 Interface VPC endpoints (AWS PrivateLink) no Guia do usuário da Amazon VPC.

Considerações sobre os endpoints Amazon Braket VPC

Antes de configurar uma interface VPC endpoint para Braket, certifique-se de revisar as propriedades e limitações do endpoint de interface no Guia do usuário da Amazon VPC.

O Braket oferece suporte para fazer chamadas para todas as suas ações de API a partir da sua VPC.

Por padrão, o acesso total ao Braket é permitido por meio do VPC endpoint. Você pode controlar o acesso se especificar políticas de VPC endpoint. Para obter mais informações, consulte Controlar o acesso a serviços com endpoints da VPC 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 Braket. Amazon 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.
- Crie um endpoint Amazon VPC para Braket
- Conecte e execute tarefas quânticas do Braket em seu endpoint

Etapa 1: inicie uma Amazon VPC, se necessário

Lembre-se de que você pode pular essa etapa se sua conta já tiver uma VPC em operação.

Uma VPC controla suas configurações de 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 informações sobre como criar suas próprias 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 VPC endpoint para 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 Criar um endpoint da interface no Guia do usuário da Amazon VPC.

Para criar um VPC endpoint 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 `–endpoint-url` bandeira quando você está fazendo determinadas chamadas para o BraketAPI.

Crie o VPC endpoint para Braket usando o seguinte nome de serviço:

- `com.amazonaws.substitute_your_region.braket`

Observação: se você habilitar o DNS privado para o endpoint, poderá fazer API solicitações ao Braket usando seu nome DNS padrão para a região, por exemplo,. `braket.us-east-1.amazonaws.com`

Para obter mais informações, consulte Acessar um serviço por um endpoint de interface 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 VPC endpoint, você pode executar comandos da CLI que incluem o `endpoint-url` parâmetro para especificar endpoints de interface para o tempo de execução API ou, como no exemplo a seguir:

```
aws braket search-quantum-tasks --endpoint-url
 VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

Se você habilitar nomes de host DNS privados para seu VPC endpoint, não precisará especificar o endpoint como uma URL nos comandos da CLI. 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 VPC endpoint. Ele tem a forma mostrada no exemplo a seguir:

```
https://braket.substituteYourRegionHere.amazonaws.com
```

A postagem do blog chamada Acesso direto aos SageMaker notebooks 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 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.

Saiba mais 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 aAWS CLI , consulte Creating an Interface Endpoint (“Criar um endpoint da interface”) no Manual do usuário da Amazon VPC.
- Para obter informações sobre como criar e configurar um endpoint usandoAWS CloudFormation, consulte o recurso AWS: :EC2: :VPCEndpoint no Guia do usuário. AWS CloudFormation

Controle o acesso com as políticas de endpoint da Amazon VPC

Para controlar o acesso à conectividade ao Amazon Braket, você pode anexar uma política de endpoint AWS Identity and Access Management (IAM) ao seu endpoint da Amazon VPC. Essa política especifica as seguintes informações:

- O principal (usuário ou função) que pode realizar ações.
- As ações que podem ser executadas.
- Os recursos sobre os quais as ações podem ser realizadas.

Para obter mais informações, consulte Controlar o acesso a serviços com endpoints da VPC no Guia do usuário da Amazon VPC.

Exemplo: política de VPC endpoint para ações do Braket

O exemplo a seguir mostra uma política de endpoint para Braket. Quando anexada a um endpoint, essa política concede acesso às ações listadas do Braket para todos os diretores 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 obter 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 do acesso a serviços com VPC endpoints

Solução de problemas do Amazon Braket

Use as informações e soluções de solução de problemas nesta seção para ajudar a resolver problemas com o Amazon Braket.

Nesta seção:

- AccessDeniedException
- Ocorreu um erro (ValidationException) ao chamar a CreateQuantumTask operação
- Um recurso do SDK não funciona
- O trabalho híbrido falha devido a ServiceQuotaExceededException
- Os componentes pararam de funcionar na instância do notebook
- Cotas do Amazon Braket
- Solucionar 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, você deve entrar em contato com seu AWS administrador interno para entender quais das seguintes condições se aplicam:

- Se houver restrições de função impedindo o acesso a uma região.
- Se a função que você está tentando usar tiver permissão para usar o Braket.

Se sua função 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 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 do SDK não funciona

Sua versão do Python deve ser 3.9 ou superior. Para Amazon Braket Hybrid Jobs, recomendamos o Python 3.10.

Verifique se o SDK e os esquemas estão. up-to-date Para atualizar o SDK a partir do notebook ou do seu editor 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 AWS sua região está definida como uma região suportada 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 obter mais informações sobre os limites do serviço, 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 o `search-quantum-tasks`API, 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 da cota de serviço para o número de tarefas quânticas simultâneas para o dispositivo simulador. Isso só se aplica ao SV1 dispositivo.
2. Gerencie `ServiceQuotaExceeded` exceções em seu código e tente novamente.

Os componentes pararam de funcionar na instância do notebook

Se alguns componentes do seu notebook pararem de funcionar, tente o seguinte:

1. Baixe todos os notebooks que você criou ou modificou em uma unidade local.
2. Pare sua instância do notebook.
3. Exclua sua instância do notebook.
4. Crie uma nova instância de notebook com um nome diferente.
5. Faça o upload dos notebooks para a nova instância.

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.

Algumas cotas podem ser aumentadas. Para obter mais informações, consulte as AWS service (Serviço da AWS) cotas.

- As cotas de taxa de estouro não podem ser aumentadas.
- O aumento máximo da taxa para cotas ajustáveis (exceto a taxa de intermitência, 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 permitido de instâncias de computação para um trabalho híbrido é 5, e as cotas são ajustáveis.

Recurso	Descrição	Limites	Ajustável
Taxa de solicitações API	O número máximo de solicitações por segundo que você pode enviar nesta conta na atual região.	140	Sim
Taxa de intermitência de solicitações API	O número máximo de solicitações adicionais por segundo (RPS) que você pode enviar em uma intermitência nesta conta na região atual.	600	Não
Taxa de solicitações `CreateQuantumTask`	O número máximo de `CreateQuantumTask` solicitações que você pode enviar por segundo nessa conta por região.	20	Sim
Taxa de intermitência de solicitaç	O número máximo de `CreateQuantumTask` solicitaç	40	Não

Recurso	Descrição	Limites	Ajustável
ões `CreateQuantumTask`	ões adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.		
Taxa de solicitações `SearchQuantumTasks`	O número máximo de `SearchQuantumTasks` solicitações que você pode enviar por segundo nessa conta por região.	5	Sim
Taxa de intermitência de solicitações `SearchQuantumTasks`	O número máximo de `SearchQuantumTasks` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	50	Não
Taxa de solicitações `GetQuantumTask`	O número máximo de `GetQuantumTask` solicitações que você pode enviar por segundo nessa conta por região.	100	Sim

Recurso	Descrição	Limites	Ajustável
Taxa de intermitência de solicitações `GetQuantumTask`	O número máximo de `GetQuantumTask` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	500	Não
Taxa de solicitações `CancelQuantumTask`	O número máximo de `CancelQuantumTask` solicitações que você pode enviar por segundo nessa conta por região.	2	Sim
Taxa de intermitência de solicitações `CancelQuantumTask`	O número máximo de `CancelQuantumTask` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	20	Não
Taxa de solicitações `GetDevice`	O número máximo de `GetDevice` solicitações que você pode enviar por segundo nessa conta por região.	5	Sim

Recurso	Descrição	Limites	Ajustável
Taxa de intermitência de solicitações `GetDevice`	O número máximo de `GetDevice` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	50	Não
Taxa de solicitações `SearchDevices`	O número máximo de `SearchDevices` solicitações que você pode enviar por segundo nessa conta por região.	5	Sim
Taxa de intermitência de solicitações `SearchDevices`	O número máximo de `SearchDevices` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	50	Não
Taxa de solicitações `CreateJob`	O número máximo de `CreateJob` solicitações que você pode enviar por segundo nessa conta por região.	1	Sim

Recurso	Descrição	Limites	Ajustável
Taxa de intermitência de solicitações `CreateJob`	O número máximo de `CreateJob` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	5	Não
Taxa de solicitações `SearchJob`	O número máximo de `SearchJob` solicitações que você pode enviar por segundo nessa conta por região.	5	Sim
Taxa de intermitência de solicitações `SearchJob`	O número máximo de `SearchJob` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	50	Não
Taxa de solicitações `GetJob`	O número máximo de `GetJob` solicitações que você pode enviar por segundo nessa conta por região.	5	Sim

Recurso	Descrição	Limites	Ajustável
Taxa de intermitência de solicitações `GetJob`	O número máximo de `GetJob` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	25	Não
Taxa de solicitações `CancelJob`	O número máximo de `CancelJob` solicitações que você pode enviar por segundo nessa conta por região.	2	Sim
Taxa de intermitência de solicitações `CancelJob`	O número máximo de `CancelJob` solicitações adicionais por segundo (RPS) que você pode enviar de uma só vez nessa conta na região atual.	5	Não
Número de tarefas SV1quânticas simultâneas	O número máximo de tarefas quânticas simultâneas em execução no simulador vetorial de estado (SV1) na região atual.	100 us-leste-1, 50 us-oeste-1, 100 us-oeste-2, 50 eu-west-2	Não

Recurso	Descrição	Limites	Ajustável
Número de tarefas DM1quânticas simultâneas	O número máximo de tarefas quânticas simultâneas em execução no simulador de matriz de densidade (DM1) na região atual.	100 us-leste-1, 50 us-oeste-1, 100 us-oeste-2, 50 eu-west-2	Não
Número de tarefas TN1quânticas simultâneas	O número máximo de tarefas quânticas simultâneas em execução no simulador de rede tensorial (TN1) na região atual.	10 us-leste-1, 10 us-oeste-2, 5 eu-west-2,	Sim
Número de trabalhos híbridos simultâneos	O número máximo de trabalhos híbridos simultâneos na região atual.	5	Sim
Limite de execução de trabalhos híbridos	O tempo máximo em dias em que uma tarefa híbrida pode ser executada.	5	Não

A seguir estão as cotas padrão de instância de computação clássica para trabalhos híbridos. Para aumentar essas cotas, entre em contato AWS Support. Além disso, as regiões disponíveis são especificadas para cada instância.

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo	O número	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
de instâncias de ml.c4.xlarge para trabalhos híbridos	máximo de instâncias do tipo ml.c4.xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.							

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c4.2xlarge para trabalhos híbridos	O número máximo de instâncias do tipo ml.c4.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Não

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.c4.4xlarge para trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	O número máximo de instâncias do tipo ml.m5.4xlarge permitido para Amazon Braket Hybrid Jobs nessa conta e região.	5	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	O número máximo de instâncias do tipo ml.m5.24› large permitido para Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Sim	Sim	Sim	Sim

Recurso	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	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 trabalhos híbridos	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	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p3.2xlarge para trabalhos híbridos	O número máximo de instâncias do tipo ml.p3.2xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Recurso	Descrição	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 trabalhos híbridos	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

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p3dn.24xlarge para trabalhos híbridos	O número máximo de instâncias do tipo ml.p3dn.24xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Não	Não

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p3.8xlarge para trabalhos híbridos	O número máximo de instâncias do tipo ml.p3.8xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Sim	Não

Recurso	Descrição	Limites	Ajustável	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Número máximo de instâncias de ml.p3.16xlarge para trabalhos híbridos	O número máximo de instâncias do tipo ml.p3.16xlarge permitido para todos os Amazon Braket Hybrid Jobs nessa conta e região.	0	Sim	Sim	Não	Sim	Sim	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 sua tarefa híbrida 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.
- O número máximo de disparos por tarefa permitido paraSV1,DM1, e Rigetti dispositivos é 100.000.
- O número máximo de disparos por tarefa permitido TN1 é 1000.
- Para IonQ os dispositivos Aria-1 e Aria-2, o máximo é 5.000 fotos por tarefa. Para IonQ dispositivos e dispositivos Harmony e OQC Forte, o máximo é 10.000.
- PoisQuEra, o máximo permitido de disparos por tarefa é 1000.
- Para TN1 e para os QPU dispositivos, as fotos por tarefa devem ser > 0.

Solucionar 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 qubits
- Misturando erro físico qubits com qubits erro virtual
- Solicitando tipos de resultados e medindo qubits no mesmo erro de programa
- Os limites clássicos e de qubit registro excederam o erro
- Caixa não precedida por um erro de pragma literal
- Erro de caixas textuais sem portas nativas
- Caixas textuais sem erro físico qubits
- O pragma literal não contém o erro “braket”
- Erro único qubits não pode ser indexado
- O físico qubits em um erro de duas qubit portas não está conectado
- GetDevice não retorna o erro de resultados do OpenQASM
- 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 exemplo a seguir gera um erro do analisador.

```
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 qubits

O uso não contíguo qubits em dispositivos `requiresContiguousQubitIndices` configurados como na capacidade do dispositivo resulta `true` em um erro.

Ao executar tarefas quânticas em simuladores 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].`

Misturando erro físico qubits com qubits erro virtual

Não é permitido misturar o físico qubits com o virtual qubits no mesmo programa e 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.`

Solicitando tipos de resultados e medindo qubits no mesmo erro de programa

Solicitar tipos de resultados que qubits sejam medidos 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 qubit registro excederam o erro

Somente um registro clássico e um qubit registro 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 nativas e físicas. qubits 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 físico qubits

As caixas textuais devem ser físicas. qubits O código a seguir gera o qubits erro 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 single 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, qubit matrizes únicas podem ser indexadas da seguinte forma:

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

O físico qubits em um erro de duas qubit portas não está conectado

Para usar o físicoqubits, primeiro confirme se o dispositivo usa o físico qubits 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`

GetDevice não retorna o erro de resultados do OpenQASM

Se você não vê resultados do OpenQASM na GetDevice resposta ao usar um SDK do Braket, talvez seja necessário definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário. Veja os exemplos de código fornecidos abaixo para saber como fazer isso com os SDKs Go e Java.

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o: AWS CLI

```
% export AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0"
# Or for single execution
% AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0" aws braket <cmd> [options]
```

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o Boto3:

```
import boto3
import botocore

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

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o/(SDK v2): JavaScript TypeScript

```
import Braket from 'aws-sdk/clients/braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0'  });
```

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o/(SDK v3): JavaScript TypeScript

```
import { Braket } from '@aws-sdk/client-braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0' });
```

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o Go SDK:

```
os.Setenv("AWS_EXECUTION_ENV", "BraketGo BraketSchemas/1.8.0")
mySession := session.Must(session.NewSession())
svc := braket.New(mySession)
```

Para definir a variável de ambiente AWS_EXECUTION_ENV para configurar o agente de usuário ao usar o Java SDK:

```
ClientConfiguration config = new ClientConfiguration();
config.setUserAgentSuffix("BraketSchemas/1.8.0");
BraketClient braketClient =
 BraketClientBuilder.standard().withClientConfiguration(config).build();
```

Aviso de suporte do simulador local

`LocalSimulator`Ele suporta recursos avançados no OpenQASM que podem não estar disponíveis em QPUs ou 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 QPUs ou simuladores sob demanda.

Para obter mais informações sobre os recursos suportados do OpenQASM, clique aqui.

Guia de referência de API e SDK para o Amazon Braket

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

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

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

- Tutoriais de braket GitHub

Histórico do documento

A tabela a seguir descreve a documentação dessa versão do Amazon Braket.

- APIversão: 28 de abril de 2022
- Última atualização API de referência: 25 de setembro de 2023
- Última atualização da documentação: 22 de maio de 2024

Alteração	Descrição	Data
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 notebook	Ao criar uma instância do notebook, ative o gerenciad or de inatividade e defina um tempo de inatividade para redefinir automaticamente a instância do notebook Braket.	27 de março de 2024
Reformulação do índice	Reorganizou o índice do Amazon Braket para atender aos requisitos do guia de AWS estilo e melhorar o fluxo de	12 de dezembro de 2023

	conteúdo para a experiência do cliente.	
Braket Direct lançado	Foi adicionado suporte para recursos diretos do Braket, incluindo: • Reservas • Conselhos de especialistas • Capacidades experimentais	27 de novembro de 2023
Atualização do Crie uma instância do notebook Amazon Braket	A documentação foi atualizada para adicionar informações para criar uma instância de notebook 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	Execute seu código local como um trabalho híbridoPágina adicionada. Contém exemplos: • Crie um trabalho híbrido a partir do código Python local • Instale pacotes e código-fonte adicionais do Python • Salve e carregue dados em uma instância de trabalho híbrida • Melhores práticas para decoradores de trabalho híbridos	16 de outubro de 2023
Visibilidade de fila adicionada	A documentação do Guia do Desenvolvedor foi atualizada para incluir queue depth queue position e. A documentação da API foi atualizada para refletir as novas alterações da API para visibilidade da fila.	25 de setembro de 2023
Padronize 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 IonQ Aria 2 dispositivo	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
Xanadupartida	Atualizou a documentação para remover todos os Xanadu dispositivos	2 de junho de 2023
Novo dispositivo IonQ Aria	Suporte adicionado para o IonQ Aria dispositivo	16 de maio de 2023
RigettiDispositivo retirado	Suporte descontinuado 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 Rigetti Aspen-M-3 dispositivo	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
D-Wavepartida	A documentação foi atualizada para acomodar a remoção de todos os D-Wave dispositivos	17 de novembro de 2022
Novo dispositivo QuEra Aquila	Suporte adicionado para o QuEra Aquila dispositivo	31 de outubro de 2022
Support para Braket Pulse	Foi adicionado suporte para Braket Pulse, que permite que o controle de pulso seja usado em dispositivos Rigetti OQC	20 de outubro de 2022
Support 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 instância	Atualizou as cotas padrão de instância de computaçã o clássica associadas a trabalhos híbridos	22 de agosto de 2022
Novo painel de serviços	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 Rigetti Aspen-M-2 dispositivo	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 controle de custos	Foi adicionado como obter estimativas de custo máximo quase em tempo real para simuladores e cargas de trabalho de hardware	18 de julho de 2022
Novo Xanadu Borealis dispositivo	Suporte adicionado para o Xanadu Borealis dispositivo	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 D-Wave Advantage_system6.1 dispositivo	12 de maio de 2022
Support 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 Amazon Braket	Adicionado s3: 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
Support para OpenQASM	Foi adicionado suporte ao OpenQASM 3.0 para dispositivos e simuladores quânticos baseados em portas	7 de março de 2022

Novo fornecedor de hardware quântico Oxford Quantum Circuits e nova região, eu-west-2	Adicionado suporte para OQC e eu-west-2	28 de fevereiro de 2022
Novo Rigetti dispositivo	Adicionado o suporte para Rigetti Aspen M-1	15 de fevereiro de 2022
Novos limites de recursos	Aumentamos o número máximo de SV1 tarefas DM1 e tarefas simultâneas de 55 para 100	5 de janeiro de 2022
Novo Rigetti dispositivo	Adicionado o suporte para Rigetti Aspen-11	20 de dezembro de 2021
RigettiDispositivo retirado	Suporte descontinuado para o 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 atualizada da política	O Amazon Braket atualizou o ARN da função para incluir o caminho. `servicerole/` Para obter informações sobre atualizações de políticas, consulte a tabela de atualizaç ões do Amazon Braket AWS para políticas gerenciadas.	29 de novembro de 2021
Empregos na Amazon Braket	Guia do usuário para Amazon Braket Hybrid Jobs e adicionado API	29 de novembro de 2021

Novo Rigetti dispositivo	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 D-Wave dispositivo	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
qubitAlocação manual	Braket suporta qubit alocação manual 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 D-Wave dispositivo	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

AWS Glossário

Para obter a AWS terminologia mais recente, consulte o AWSglossário na Referência do AWSGlossário.

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.