aws

Entwicklerhandbuch

Amazon Braket

Amazon Braket: Entwicklerhandbuch

Die Handelsmarken und die Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Die Handelsmarken und die Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist Amazon Braket? 1
Funktionsweise 4
Amazon Braket-Quanten-Taskflow 5
Third-party Datenverarbeitung 6
Begriffe und Konzepte von Amazon Braket 6
AWS Terminologie und Tipps für Amazon Braket 11
Kostenverfolgung und Kosteneinsparung 12
Ausgabenlimits für Amazon Braket QPUs festlegen 12
Kostenverfolgung nahezu in Echtzeit 16
Bewährte Methoden zur Kosteneinsparung 19
API-Referenzen und Repos 20
Kern-Repositorien 21
Plugins 21
Unterstützte Regionen und Geräte 22
Regionen und Endpunkte 26
Erste Schritte 28
Amazon Braket aktivieren 28
Voraussetzungen 29
Schritte zur Aktivierung von Amazon Braket 29
Erstellen Sie eine Amazon Braket-Notebook-Instance 30
(Erweitert) Erstellen Sie ein Braket-Notizbuch mit CloudFormation 32
Schritt 1: Erstellen Sie ein SageMaker AI-Lifecycle-Konfigurationsskript 33
Schritt 2: Erstellen Sie die von Amazon SageMaker AI übernommene IAM-Rolle 34
Schritt 3: Erstellen Sie eine SageMaker AI-Notebook-Instanz mit dem Präfix `amazon-braket-` 35
Entwicklung 37
Erstellen Sie Ihre erste Schaltung 37
Entwickeln Sie Ihre ersten Quantenalgorithmen 42
Schaltkreise im SDK konstruieren 43
Der Stromkreis wird überprüft 59
Liste der Ergebnistypen 61
Expertenrat einholen 67
(Fortgeschritten) Betreiben Sie Ihre Schaltungen mit OpenQASM 3.0 68
Was ist OpenQASM 3.0? 69

Wann sollte OpenQASM 3.0 verwendet werden 70
Wie funktioniert OpenQASM 3.0 70
Voraussetzungen 70
Welche OpenQASM-Funktionen unterstützt Braket? 71
Erstellen Sie eine OpenQASM 3.0-Beispiel-Quantenaufgabe und reichen Sie sie ein 77
Support für OpenQASM auf verschiedenen Braket-Geräten 80
Simulieren Sie Geräusche 90
QubitNeuverkabelung 92
Wörtliche Kompilierung 92
Die Braket-Konsole 93
Weitere Ressourcen 93
Berechnung von Gradienten 93
Messung bestimmter Qubits 94
(Fortgeschritten) Erkunden Sie experimentelle Möglichkeiten 95
Zugriff auf die lokale Abstimmung auf Aquila QuEra 96
Zugang zu hohen Geometrien auf Aquila QuEra 98
Zugang zu engen Geometrien auf Aquila QuEra 99
Dynamische Schaltungen auf IQM-Geräten 101
(Fortgeschrittene) Pulssteuerung auf Amazon Braket 103
Frames (Frames) 104
Ports 104
Wellenformen 105
Ich arbeite mit Hello Pulse 106
Zugreifen auf native Gates mithilfe von Impulsen 110
(Fortgeschrittene) Analoge Hamiltonsche Simulation 112
Hallo AHS: Führe deine erste analoge Hamilton-Simulation aus 112
Reichen Sie ein analoges Programm mit Aquila ein QuEra 125
(Fortgeschritten) Arbeiten mit AWS Boto3 146
Schalten Sie den Amazon Braket Boto3-Client ein 146
Konfiguration AWS CLI Profile für Boto3 und das Braket SDK 150
Test 153
Einreichung von Quantenaufgaben an Simulatoren 153
Vektorsimulator für lokalen Zustand () `braket_sv` 155
Simulator für lokale Dichtematrix () `braket_dm` 155
Lokaler AHS-Simulator () `braket_ahs` 156
Zustandsvektorsimulator () SV1 156

Dichtematrix-Simulator (DM1) 157
Tensor-Netzwerksimulator () TN1 158
Über eingebettete Simulatoren 159
Simulatoren vergleichen 160
Beispiele für Quantenaufgaben auf Amazon Braket 164
Testen einer Quantenaufgabe mit dem lokalen Simulator 170
Lokaler Emulator für Quantengeräte 172
Vorteile der lokalen Emulation 172
Erstellen Sie einen lokalen Emulator 173
Ausführen 174
Quantenaufgaben an QPUs einreichen 175
AQT 177
IonQ 178
IQM 178
Rigetti 179
QuEra 180
Beispiel: Eine Quantenaufgabe an eine QPU einreichen 180
Inspektion kompilierter Schaltungen 183
Mehrere Programme ausführen 184
Informationen zum Programmumfang und zu den Kosten 186
Informationen zur Batchverarbeitung und zu den Kosten von Quantenaufgaben 186
Batching von Quantenaufgaben und PennyLane 187
Batching von Aufgaben und parametrisierte Schaltungen 187
Wann wird meine Quantenaufgabe ausgeführt? 188
QPU-Verfügbarkeitsfenster und Status 189
Sichtbarkeit der Warteschlange 189
Richten Sie E-Mail- oder SMS-Benachrichtigungen ein 191
(Fortgeschritten) Arbeiten mit Reservierungen 192
Wie erstelle ich eine Reservierung 193
Ausführung von Quantenaufgaben während einer Reservierung 194
Ausführung von Hybridaufträgen während einer Reservierung 198
Was passiert am Ende Ihrer Reservierung 199
Stornieren oder verschieben Sie eine bestehende Reservierung 200
(Fortgeschrittene) Techniken zur Fehlerminimierung 200
Techniken zur Fehlerminimierung auf ionQ Geräte 200
Jobs bei Amazon Braket Hybrid 203

Wann sollten Sie Amazon Braket Hybrid Jobs verwenden 204
Einen Hybrid-Job mit Amazon Braket Hybrid Jobs ausführen 205
Die wichtigsten Konzepte 207
Eingaben 207
Outputs 208
Umgebungsvariablen 209
Hilfsfunktionen 210
Voraussetzungen 210
Erstellen Sie einen Hybrid-Job 214
Erstellen und ausführen 215
Überwachen Sie Ihre Ergebnisse 218
Speichern Sie Ihre Ergebnisse 220
Checkpoints verwenden 222
Führen Sie Ihren lokalen Code als Hybrid-Job aus 223
Verwendung der API mit Hybrid-Jobs 232
Erstellen und debuggen Sie einen Hybrid-Job im lokalen Modus 236
Einen Hybrid-Job stornieren 237
Personalisieren Sie Ihren Hybrid-Job 238
Definieren Sie die Umgebung für Ihr Algorithmus-Skript 239
Hyperparameter verwenden 251
Konfigurieren Sie Ihre Hybrid-Job-Instance 252
Verwendung der parametrischen Kompilierung zur Beschleunigung von Hybrid-Jobs 257
(Fortgeschritten) PennyLane mit Amazon Braket 258
Amazon Braket mit PennyLane 259
Hybride Algorithmen in Amazon Braket-Beispielnotizbüchern 261
Hybride Algorithmen mit eingebetteten Simulatoren PennyLane 261
Adjoint Gradient aktiviert PennyLane mit Amazon Braket-Simulatoren 262
Hybrid-Jobs verwenden und einen QAOA-Algorithmus ausführen PennyLane 263
Führen Sie hybride Workloads mit eingebetteten Simulatoren aus PennyLane 266
(Fortgeschritten) CUDA-Q mit Amazon Braket 272
CUDA-Q in NBIs 273
CUDA-Q in hybriden Aufträgen 273
Fehlerbehebung 277
AccessDeniedException 277
Beim Aufrufen des Vorgangs ist ein Fehler aufgetreten (ValidationException) CreateQuantumTask 277

Eine SDK-Funktion funktioniert nicht ... 278
Der Hybrid-Job schlägt fehl aufgrund von ServiceQuotaExceededException ... 278
Komponenten funktionieren in der Notebook-Instanz nicht mehr ... 279
Problembehandlung beim Python 3.12-Upgrade ... 279
-Übersicht ... 280
Allgemeine Fehlermeldungen ... 280
Von Braket verwaltete Notizbücher ... 281
Hybrider Jobdekorateur ... 281
Bring-Your-Own-Container (BYOC) ... 282
Aktualisierung der Braket-Notebook-Instanz ... 283
Fehlerbehebung bei OpenQASM ... 284
Fehler in der Anweisung einschließen ... 284
Nicht zusammenhängender Fehler qubits ... 284
Mischen von qubits physischem und virtuellem qubits Fehler ... 285
Fehler beim Abrufen von Ergebnistypen und Messen qubits im selben Programm ... 285
Die klassischen Grenzwerte und die qubit Registergrenzen haben den Fehler überschritten ... 286
Feld, dem kein wörtlicher Pragma-Fehler vorausgeht ... 286
Fehler bei verbatim-Boxen, bei denen native Gates fehlen ... 286
Bei den verbatim-Boxen fehlt ein physischer Fehler qubits ... 287
Im wörtlichen Pragma fehlt der Fehler „Braket“ ... 287
Fehler „Single qubits kann nicht indexiert werden“ ... 287
Fehler: Die physikalischen qubits Verbindungen in zwei qubit Gates sind nicht verbunden ... 288
Warnung zur Unterstützung des lokalen Simulators ... 288
Sicherheit ... 290
Gemeinsame Verantwortung für die Sicherheit ... 291
Datenschutz ... 291
Datenaufbewahrung ... 292
Zugriff auf Amazon Braket verwalten ... 292
Ressourcen für Amazon Braket ... 293
Notizbücher und Rollen ... 293
AWS verwaltete Richtlinien ... 295
Beschränken Sie den Benutzerzugriff auf bestimmte Geräte ... 299
Beschränken Sie den Benutzerzugriff auf bestimmte Notebook-Instanzen ... 301
Beschränken Sie den Benutzerzugriff auf bestimmte S3-Buckets ... 302
Servicegebundene Rolle ... 303

Compliance-Validierung 304
Infrastruktursicherheit 305
Sicherheit durch Dritte 305
VPC-Endpunkte (PrivateLink) 306
Überlegungen zu Amazon Braket VPC-Endpunkten 306
Richten Sie Braket ein und PrivateLink 307
Zusätzliche Informationen zum Erstellen eines Endpunkts 309
Steuern Sie den Zugriff mit Amazon VPC-Endpunktrichtlinien 309
Protokollierung und Überwachung 311
Verfolgung von Quantenaufgaben mit dem Amazon Braket SDK 312
Überwachung von Quantenaufgaben über die Amazon Braket-Konsole 314
Taggen von Ressourcen 317
Verwenden von Markierungen 318
Unterstützte Ressourcen für das Tagging in Amazon Braket 318
Markierung mit der Amazon Braket API 319
Tagging-Einschränkungen 319
Verwaltung von Tags in Amazon Braket 320
Beispiel für AWS CLI Tagging in Amazon Braket 321
Überwachen Sie Ihre Quantenaufgaben mit EventBridge 322
Überwachen Sie den Status von Quantenaufgaben mit EventBridge 323
Beispiel für eine Amazon EventBridge Braket-Veranstaltung 324
Überwachen Sie Ihre Metriken mit CloudWatch 325
Amazon Braket-Metriken und -Dimensionen 326
Protokollieren Sie Ihre Quantenaufgaben mit CloudTrail 326
Informationen zu Amazon Braket in CloudTrail 327
Grundlegendes zu Amazon Braket-Protokolldateieinträgen 328
(Erweiterte) Protokollierung 330
Kontingente 333
Zusätzliche Kontingente und Limits 394
Dokumentverlauf 395
..... cdix

Was ist Amazon Braket?

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Amazon Braket ist ein vollständig verwaltetes Programm, AWS-Service das Forschern, Wissenschaftlern und Entwicklern den Einstieg in das Quantencomputing erleichtert. Quantencomputer haben das Potenzial, Rechenprobleme zu lösen, die für klassische Computer unerreichbar sind, da sie die Gesetze der Quantenmechanik nutzen, um Informationen auf neue Weise zu verarbeiten.

Der Zugang zu Quantencomputer-Hardware kann teuer und umständlich sein. Eingeschränkter Zugriff macht es schwierig, Algorithmen auszuführen, Designs zu optimieren, den aktuellen Stand der Technologie zu bewerten und zu planen, wann Sie Ihre Ressourcen optimal einsetzen sollten. Braket hilft Ihnen, diese Herausforderungen zu meistern.

Braket bietet einen zentralen Zugangspunkt zu einer Vielzahl von Quantencomputertechnologien. Mit Braket können Sie:

- Erforschen und entwerfen Sie Quanten- und Hybridalgorithmen.
- Testen Sie Algorithmen auf verschiedenen Quantenschaltkreissimulatoren.
- Führen Sie Algorithmen auf verschiedenen Arten von Quantencomputern aus.
- Erstellen Sie Machbarkeitsnachweisanwendungen.

Die Definition von Quantenproblemen und die Programmierung von Quantencomputern zu ihrer Lösung erfordern neue Fähigkeiten. Um Ihnen beim Erwerb dieser Fähigkeiten zu helfen, bietet Braket verschiedene Umgebungen zur Simulation und Ausführung Ihrer Quantenalgorithmen. Mit einer Reihe von Beispielumgebungen, den sogenannten Notebooks, können Sie den Ansatz finden, der Ihren Anforderungen am besten entspricht, und schnell loslegen.

Die Entwicklung von Braket besteht aus drei Phasen:

- Build — Braket bietet vollständig verwaltete Jupyter-Notebook-Umgebungen, die den Einstieg erleichtern. Braket-Notebooks sind mit Beispielalgorithmen, Ressourcen und Entwicklertools, einschließlich des Amazon Braket-SDK, vorinstalliert. Mit dem Amazon Braket SDK können Sie Quantenalgorithmen erstellen und diese dann auf verschiedenen Quantencomputern und Simulatoren testen und ausführen, indem Sie eine einzige Codezeile ändern.
- Test — Braket bietet Zugriff auf vollständig verwaltete, leistungsstarke Quantenschaltkreissimulatoren. Sie können Ihre Schaltungen testen und validieren. Braket verwaltet alle zugrunde liegenden Softwarekomponenten und Amazon Elastic Compute Cloud (Amazon EC2) -Cluster, um die Simulation von Quantenschaltkreisen auf der klassischen HPC-Infrastruktur (High Performance Computing) zu vereinfachen.
- Run — Braket bietet sicheren On-Demand-Zugriff auf verschiedene Arten von Quantencomputern. Sie haben Zugriff auf Gate-basierte Quantencomputer vonAQT, IonQ IQMRigetti, und sowie auf einen analogen Hamilton-Simulator von. QuEra Sie haben auch keine Vorabverpflichtung und müssen sich den Zugang nicht über einzelne Anbieter sichern.

Über Quantencomputer und Braket

Quantencomputer befinden sich in einem frühen Entwicklungsstadium. Es ist wichtig zu verstehen, dass es derzeit keinen universellen, fehlertoleranten Quantencomputer gibt. Daher sind bestimmte Arten von Quantenhardware für jeden Anwendungsfall besser geeignet, und es ist entscheidend, Zugang zu einer Vielzahl von Computerhardware zu haben. Braket bietet eine Vielzahl von Hardware über Drittanbieter an.

Bestehende Quantenhardware ist aufgrund von Rauschen, das zu Fehlern führt, eingeschränkt. Die Branche befindet sich im Zeitalter des Noisy Intermediate Scale Quantum (NISQ). In der NISQ-Ära sind Quantencomputer zu laut, um reine Quantenalgorithmen wie den Algorithmus von Shor oder den Algorithmus von Grover aufrechtzuerhalten. Bis eine bessere Quantenfehlerkorrektur verfügbar ist, erfordert das praktischste Quantencomputing die Kombination von klassischen (traditionellen) Rechenressourcen mit Quantencomputern, um hybride Algorithmen zu entwickeln. Braket hilft Ihnen bei der Arbeit mit hybriden Quantenalgorithmen.

In hybriden Quantenalgorithmen werden Quantenverarbeitungseinheiten (QPUs) als Coprozessoren für CPUs verwendet, wodurch spezifische Berechnungen in einem klassischen Algorithmus beschleunigt werden. Diese Algorithmen nutzen eine iterative Verarbeitung, bei der die Berechnung zwischen klassischen Computern und Quantencomputern erfolgt. Aktuelle Anwendungen des Quantencomputers in den Bereichen Chemie, Optimierung und maschinelles Lernen basieren beispielsweise auf variationellen Quantenalgorithmen, bei denen es sich um eine Art

hybrider Quantenalgorithmen handelt. Bei variationalen Quantenalgorithmen passen klassische Optimierungsroutinen die Parameter eines parametrisierten Quantenschaltkreises iterativ an, ähnlich wie die Gewichte eines neuronalen Netzwerks iterativ auf der Grundlage des Fehlers in einem Trainingssatz für maschinelles Lernen angepasst werden. Braket bietet Zugriff auf die PennyLane Open-Source-Softwarebibliothek, die Sie bei variationellen Quantenalgorithmen unterstützt.

Quantencomputer gewinnen bei Berechnungen in vier Hauptbereichen an Bedeutung:

- Zahlentheorie — einschließlich Faktoring und Kryptografie (der Algorithmus von Shor ist beispielsweise eine primäre Quantenmethode für zahlentheoretische Berechnungen)
- Optimierung — einschließlich Erfüllung von Beschränkungen, Lösung linearer Systeme und maschinelles Lernen
- Oracular Computing — einschließlich Suche, verborgener Untergruppen und Ordnungsfindung (der Grover-Algorithmus ist beispielsweise eine primäre Quantenmethode für orakulare Berechnungen)
- Simulation — einschließlich direkter Simulation, Knoteninvarianten und Anwendungen für quantennahe Optimierungsalgorithmen (QAOA)

Anwendungen für diese Kategorien von Berechnungen finden sich in den Bereichen Finanzdienstleistungen, Biotechnologie, Fertigung und Pharmazie, um nur einige zu nennen. Braket bietet Funktionen und Beispiel-Notebooks, die neben bestimmten praktischen Problemen bereits auf viele Machbarkeitsnachweise angewendet werden können.

In diesem Abschnitt:

- So funktioniert Amazon Braket
- Begriffe und Konzepte von Amazon Braket
- Kostenverfolgung und Kosteneinsparung
- API-Referenzen und Repos für Amazon Braket
- Von Amazon Braket unterstützte Regionen und Geräte

So funktioniert Amazon Braket

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Amazon Braket bietet On-Demand-Zugriff auf Quantencomputergeräte, darunter On-Demand-Schaltungssimulatoren und verschiedene Arten von Quantenverarbeitungseinheiten (QPUs). In Amazon Braket ist die atomare Anfrage an ein Gerät eine Quantenaufgabe. Bei Gate-basierten Geräten umfasst diese Anfrage den Quantenschaltkreis (einschließlich der Messanweisungen und der Anzahl der Schüsse) und andere Metadaten der Anfrage. Bei analogen Hamilton-Simulatoren beinhaltet die Quantenaufgabe den physikalischen Aufbau des Quantenregisters und die Zeit- und Raumabhängigkeit der manipulierenden Felder.

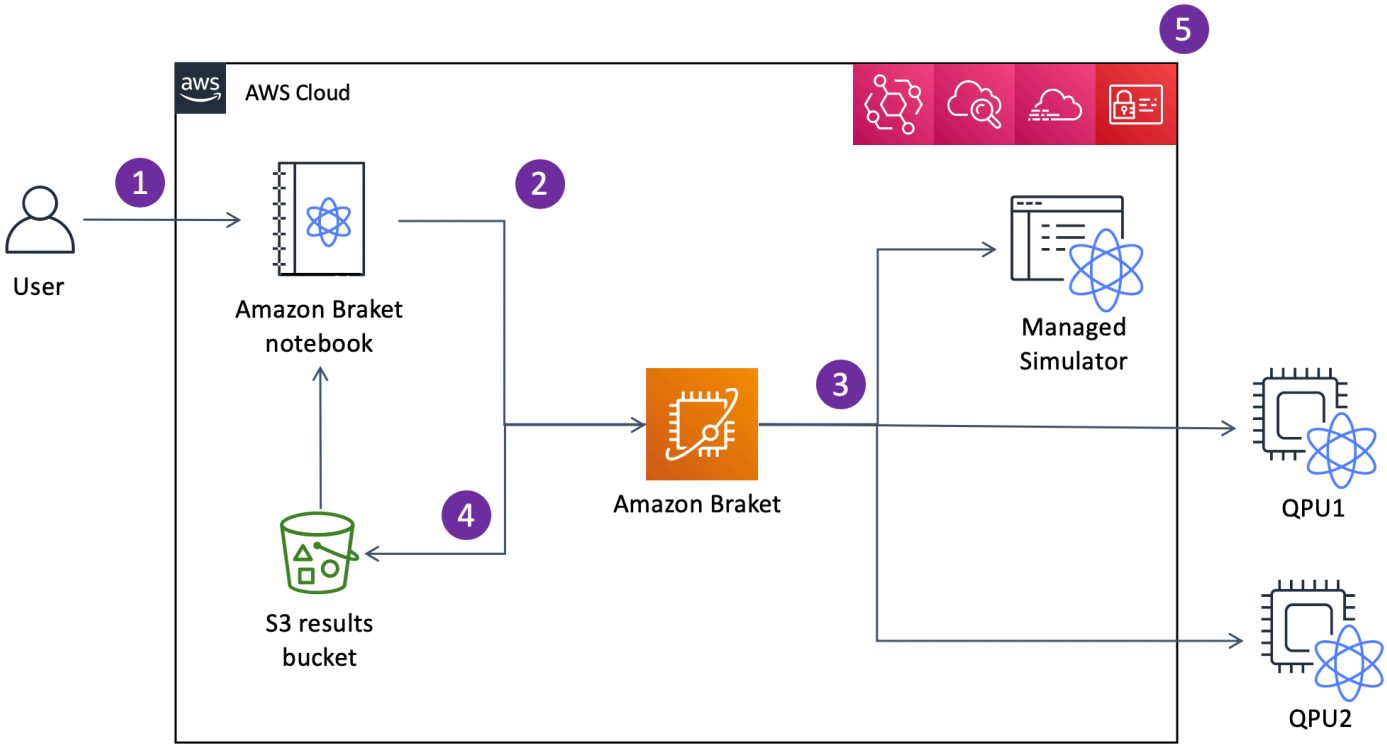
Braket Direct ist ein Programm, das die Möglichkeiten zur Erforschung von Quantencomputern erweitert und damit Forschung und Innovation beschleunigt. AWS Sie können dedizierte Kapazitäten für verschiedene Quantengeräte reservieren, direkt mit Spezialisten für Quantencomputer Kontakt aufnehmen und frühzeitig auf Funktionen der nächsten Generation zugreifen, einschließlich des neuesten Trapped-Ionen-Geräts von Forte. IonQ

In diesem Abschnitt erfahren wir mehr über den Ablauf der Ausführung von Quantenaufgaben auf Amazon Braket auf hoher Ebene.

In diesem Abschnitt:

- Amazon Braket-Quanten-Taskflow
- Third-party Datenverarbeitung

Amazon Braket-Quanten-Taskflow



Mit Jupyter Notizbüchern können Sie Ihre Quantenaufgaben über die Amazon Braket-Konsole oder mithilfe des Amazon Braket-SDK definieren, einreichen und überwachen. Sie können Ihre Quantenschaltkreise direkt im SDK erstellen. Für analoge Hamilton-Simulatoren definieren Sie jedoch das Registerlayout und die Steuerfelder (1). Nachdem Ihre Quantenaufgabe definiert wurde, können Sie ein Gerät auswählen, auf dem sie ausgeführt werden soll, und sie an die Amazon Braket-API senden (2). Je nachdem, welches Gerät Sie ausgewählt haben, wird die Quantenaufgabe in die Warteschlange gestellt, bis das Gerät verfügbar ist, und die Aufgabe wird zur Implementierung an die QPU oder den Simulator gesendet (3). Amazon Braket bietet Ihnen Zugriff auf eine Vielzahl unterstützter Quantengeräte, darunter QPUs, On-Demand-Simulatoren, lokale Simulatoren und einen eingebetteten Simulator.

Nach der Bearbeitung Ihrer Quantenaufgabe gibt Amazon Braket die Ergebnisse an einen Amazon S3 S3-Bucket zurück, wo die Daten in Ihrem AWS-Konto (4) gespeichert werden. Gleichzeitig fragt das SDK im Hintergrund nach den Ergebnissen ab und lädt sie nach Abschluss der Quantenaufgabe in das Jupyter-Notebook. Sie können Ihre Quantenaufgaben auch auf der Seite Quantum Tasks in der Amazon Braket-Konsole oder mithilfe der `GetQuantumTask` Bedienung von Amazon API Braket anzeigen und verwalten.

Amazon Braket ist in AWS Identity and Access Management (IAM), Amazon AWS CloudTrail und Amazon EventBridge für die Verwaltung CloudWatch, Überwachung und Protokollierung des Benutzerzugriffs sowie für die ereignisbasierte Verarbeitung integriert (5).

Third-party Datenverarbeitung

Quantenaufgaben, die an ein QPU-Gerät gesendet werden, werden auf Quantencomputern verarbeitet, die sich in Einrichtungen befinden, die von Drittanbietern betrieben werden. Weitere Informationen zur Sicherheit und Verarbeitung durch Dritte in Amazon Braket finden Sie unter Sicherheit von Amazon Braket-Hardwareanbietern.

Begriffe und Konzepte von Amazon Braket

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Die folgenden Begriffe und Konzepte werden in Braket verwendet:

Analoge Hamiltonsche Simulation

Die analoge Hamiltonsche Simulation (AHS) ist ein eigenständiges Quantencomputer-Paradigma für die direkte Simulation der zeitabhängigen Quantendynamik von Vielteilchensystemen. In AHS spezifizieren Benutzer direkt einen zeitabhängigen Hamilton-Operator, und der Quantencomputer ist so eingestellt, dass er die kontinuierliche Zeitentwicklung unter diesem Hamilton-Operator direkt emuliert. AHS-Geräte sind in der Regel Spezialgeräte und keine universellen Quantencomputer wie Gate-basierte Geräte. Sie sind auf eine Klasse von Hamiltonianern beschränkt, die sie simulieren können. Da diese Hamiltonianer jedoch von Natur aus auf dem Gerät implementiert sind, leidet AHS nicht unter dem Aufwand, der erforderlich ist, um Algorithmen als Schaltungen zu formulieren und Gate-Operationen zu implementieren.

Klammer

Wir haben den Braket-Service nach der Bra-Ket-Notation benannt, einer Standardnotation in der Quantenmechanik. Sie wurde 1939 von Paul Dirac eingeführt, um den Zustand von Quantensystemen zu beschreiben. Sie ist auch als Dirac-Notation bekannt.

Braket Direct

Mit Braket Direct können Sie einen dedizierten Zugang zu verschiedenen Quantengeräten Ihrer Wahl reservieren, sich mit Quantencomputerspezialisten in Verbindung setzen, um Beratung für Ihre Arbeitslast zu erhalten, und frühzeitig auf Funktionen der nächsten Generation zugreifen, z. B. auf neue Quantengeräte mit begrenzter Verfügbarkeit.

Hybrid-Job bei Braket

Amazon Braket verfügt über eine Funktion namens Amazon Braket Hybrid Jobs, die vollständig verwaltete Ausführungen von Hybrid-Algorithmen ermöglicht. Ein Braket-Hybrid-Job besteht aus drei Komponenten:

1. Die Definition Ihres Algorithmus, die als Skript, Python-Modul oder Docker-Container bereitgestellt werden kann.
2. Die auf Amazon EC2 basierende Hybrid-Job-Instance, auf der Ihr Algorithmus ausgeführt werden soll. Die Standardinstanz ist eine ml.m5.xlarge-Instance.
3. Das Quantengerät, auf dem die Quantenaufgaben ausgeführt werden sollen, die Teil Ihres Algorithmus sind. Ein einzelner Hybrid-Job enthält in der Regel eine Sammlung vieler Quantenaufgaben.

Gerät

In Amazon Braket ist ein Gerät ein Backend, das Quantenaufgaben ausführen kann. Ein Gerät kann eine QPU oder ein Quantenschaltkreissimulator sein. Weitere Informationen finden Sie unter Von Amazon Braket unterstützte Geräte.

Minimierung von Fehlern

Zur Fehlerminimierung werden mehrere physische Schaltkreise betrieben und ihre Messungen kombiniert, um ein besseres Ergebnis zu erzielen. Weitere Informationen finden Sie unter Techniken zur Fehlerminimierung.

Gate-based Quantencomputer

Beim Gate-Based Quantum Computing (QC), auch schaltkreisgestütztes QC genannt, werden Berechnungen in elementare Operationen (Gates) unterteilt. Bestimmte Gruppen von Gattern sind universell, was bedeutet, dass jede Berechnung als endliche Folge dieser Gatter ausgedrückt werden kann. Gatter sind die Bausteine von Quantenschaltungen und entsprechen den Logikgattern klassischer digitaler Schaltungen.

Gateshot-Limit

Ein Gateshot-Limit bezieht sich auf die Gesamtzahl der Tore pro Schuss (die Summe aller Tortypen) und die Anzahl der Schüsse pro Aufgabe. Mathematisch kann das Gateshot-Limit wie folgt ausgedrückt werden:

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

Hamiltonisch

Die Quantendynamik eines physikalischen Systems wird durch seinen Hamilton-Operator bestimmt, der alle Informationen über die Wechselwirkungen zwischen den Bestandteilen des Systems und die Auswirkungen exogener Antriebskräfte kodiert. Der Hamilton-Operator eines N-qubit Systems wird auf klassischen Maschinen üblicherweise als eine $2^{\text{N mal 2 N große Matrix komplexer}}$ Zahlen dargestellt. Durch die Ausführung einer analogen Hamilton-Simulation auf einem Quantengerät können Sie diese exponentiellen Ressourcenanforderungen vermeiden.

Puls

Ein Impuls ist ein vorübergehendes physikalisches Signal, das an die Qubits übertragen wird. Es wird durch eine in einem Frame abgespielte Wellenform beschrieben, die als Unterstützung für das Trägersignal dient und an den Hardwarekanal oder Port gebunden ist. Kunden können ihre eigenen Impulse entwerfen, indem sie die analoge Hüllkurve bereitstellen, die das hochfrequente sinusförmige Trägersignal moduliert. Der Frame wird eindeutig durch eine Frequenz und eine Phase beschrieben, die häufig so gewählt werden, dass sie in Resonanz mit der Energietrennung zwischen den Energieniveaus für |0¤ und |1# des Qubits stehen. Gates werden also als Impulse mit einer vorbestimmten Form und kalibrierten Parametern wie Amplitude, Frequenz und Dauer erzeugt. Anwendungsfälle, die nicht durch Template-Wellenformen abgedeckt werden, werden durch benutzerdefinierte Wellenformen ermöglicht, die für die Auflösung eines einzelnen Samples spezifiziert werden, indem eine Liste von Werten bereitgestellt wird, die durch eine feste physikalische Zykluszeit getrennt sind.

Quantenschaltung

Ein Quantenschaltkreis ist der Befehlssatz, der eine Berechnung auf einem Gate-basierten Quantencomputer definiert. Ein Quantenschaltkreis ist eine Abfolge von Quantengattern, bei denen es sich um umkehrbare Transformationen in einem qubit Register handelt, zusammen mit Messanweisungen.

Quantenschaltkreis-Simulator

Ein Quantenschaltkreissimulator ist ein Computerprogramm, das auf klassischen Computern läuft und die Messergebnisse eines Quantenschaltkreises berechnet. Bei allgemeinen Schaltkreisen wächst der Ressourcenbedarf einer Quantensimulation exponentiell mit der Anzahl der qubits zu simulierenden Schaltkreise. Braket bietet Zugriff sowohl auf verwaltete (Zugriff über BraketAPI) als auch auf lokale (Teil des Amazon Braket-SDK) Quantenschaltkreissimulatoren.

Quantencomputer

Ein Quantencomputer ist ein physikalisches Gerät, das quantenmechanische Phänomene wie Superposition und Verschränkung verwendet, um Berechnungen durchzuführen. Es gibt verschiedene Paradigmen für Quantencomputer (QC), wie z. B. die Gate-basierte QC.

Quantenverarbeitungseinheit (QPU)

Eine QPU ist ein physisches Quantencomputergerät, das auf einer Quantenaufgabe ausgeführt werden kann. QPUs können auf verschiedenen QC-Paradigmen basieren, beispielsweise auf Gate-basierter QC. Weitere Informationen finden Sie unter Von Amazon Braket unterstützte Geräte.

Native QPU-Gates

Native QPU-Gates können vom QPU-Steuersystem direkt Steuerimpulsen zugeordnet werden. Native Gates können ohne weitere Kompilierung auf dem QPU-Gerät ausgeführt werden. Teilmenge der von QPU unterstützten Gates. Sie finden die systemeigenen Gates eines Geräts auf der Geräteseite in der Amazon Braket-Konsole und über das Braket-SDK.

Von QPU unterstützte Gates

QPU-unterstützte Gates sind die Gates, die vom QPU-Gerät akzeptiert werden. Diese Gates laufen möglicherweise nicht direkt auf der QPU, was bedeutet, dass sie möglicherweise in native Gates zerlegt werden müssen. Sie finden die unterstützten Gates eines Geräts auf der Geräteseite in der Amazon Braket-Konsole und über das Amazon Braket-SDK.

Quantenaufgabe

In Braket ist eine Quantenaufgabe die atomare Anfrage an ein Gerät. Bei Gate-basierten QC-Geräten umfasst dies den Quantenschaltkreis (einschließlich der Messanweisungen und der Anzahl der Messbefehleshots) und andere Anforderungsmetadaten. Sie können Quantenaufgaben über das Amazon Braket SDK oder direkt mithilfe des CreateQuantumTask API Vorgangs erstellen. Nachdem Sie eine Quantenaufgabe erstellt haben, wird sie in die Warteschlange gestellt, bis das angeforderte Gerät verfügbar ist. Sie können Ihre

Quantenaufgaben auf der Seite Quantum Tasks der Amazon Braket-Konsole oder mithilfe der SearchQuantumTasks API Operationen GetQuantumTask oder anzeigen.

Qubit

Die grundlegende Informationseinheit in einem Quantencomputer wird als qubit (Quantenbit) bezeichnet, ähnlich wie ein Bit in der klassischen Datenverarbeitung. A qubit ist ein zweistufiges Quantensystem, das durch verschiedene physikalische Implementierungen realisiert werden kann, beispielsweise durch supraleitende Schaltkreise oder einzelne Ionen und Atome. Andere qubit Typen basieren auf Photonen, elektronischen oder nuklearen Spins oder exotischeren Quantensystemen.

Queue depth

Queue depthbezieht sich auf die Anzahl der Quantenaufgaben und Hybridjobs, die sich für ein bestimmtes Gerät in der Warteschlange befinden. Auf die Anzahl der Warteschlangen für Quantenaufgaben und Hybrid-Jobs eines Geräts kann über das Symbol Braket Software Development Kit (SDK) oder Amazon Braket Management Console zugegriffen werden.

1. Die Tiefe der Aufgabenwarteschlange bezieht sich auf die Gesamtzahl der Quantenaufgaben, die darauf warten, mit normaler Priorität ausgeführt zu werden.
2. Die Tiefe der Warteschlange für Prioritätsaufgaben bezieht sich auf die Gesamtzahl der eingereichten Quantenaufgaben, die darauf warten, bearbeitet zu Amazon Braket Hybrid Jobs werden. Sobald ein Hybrid-Job gestartet wird, haben diese Aufgaben Vorrang vor eigenständigen Aufgaben.
3. Die Warteschlangentiefe für Hybridaufträge bezieht sich auf die Gesamtzahl der Hybridaufträge, die sich derzeit auf einem Gerät in der Warteschlange befinden. Quantum tasksDie im Rahmen eines Hybridauftrags eingereichten Aufträge haben Priorität und werden in der zusammengefasst. Priority Task Queue

Queue position

Queue positionbezieht sich auf die aktuelle Position Ihrer Quantenaufgabe oder Ihres Hybrid-Jobs innerhalb einer entsprechenden Gerätewarteschlange. Sie kann für Quantenaufgaben oder Hybridjobs über das Braket Software Development Kit (SDK) oder abgerufen Amazon Braket Management Console werden.

Shots

Da Quantencomputer von Natur aus probabilistisch sind, muss jeder Schaltkreis mehrfach evaluiert werden, um ein genaues Ergebnis zu erhalten. Die Ausführung und Messung eines einzelnen Schaltkreises wird als Schuss bezeichnet. Die Anzahl der Schüsse (wiederholte

Ausführungen) für eine Schaltung wird auf der Grundlage der gewünschten Genauigkeit für das Ergebnis ausgewählt.

AWS Terminologie und Tipps für Amazon Braket

IAM-Richtlinien

Eine IAM-Richtlinie ist ein Dokument, das Berechtigungen für und Ressourcen gewährt oder verweigert. AWS-Services Mit IAM-Richtlinien können Sie die Zugriffsebenen der Benutzer auf Ressourcen anpassen. Sie können Benutzern beispielsweise Zugriff auf alle Amazon S3 S3-Buckets in Ihrem AWS-Konto oder nur auf einen bestimmten Bucket gewähren.

- Bewährtes Verfahren: Halten Sie sich bei der Erteilung von Berechtigungen an das Sicherheitsprinzip der geringsten Rechte. Indem Sie diesem Prinzip folgen, verhindern Sie, dass Benutzer oder Rollen über mehr Berechtigungen verfügen, als für die Ausführung ihrer Quantenaufgaben erforderlich sind. Wenn ein Mitarbeiter beispielsweise nur Zugriff auf einen bestimmten Bucket benötigt, geben Sie den Bucket in der IAM-Richtlinie an, anstatt dem Mitarbeiter Zugriff auf alle Buckets in Ihrem zu gewähren. AWS-Konto

IAM-Rollen

Eine IAM-Rolle ist eine Identität, von der Sie annehmen können, dass sie temporären Zugriff auf Berechtigungen gewährt. Bevor ein Benutzer, eine Anwendung oder ein Dienst eine IAM-Rolle übernehmen kann, müssen ihm die Berechtigungen erteilt werden, um zu der Rolle zu wechseln. Wenn jemand eine IAM-Rolle annimmt, gibt er alle vorherigen Berechtigungen auf, die er unter einer früheren Rolle hatte, und übernimmt die Berechtigungen der neuen Rolle.

- Bewährtes Verfahren: IAM-Rollen eignen sich ideal für Situationen, in denen der Zugriff auf Dienste oder Ressourcen vorübergehend statt langfristig gewährt werden muss.

Amazon S3 S3-Bucket

Mit Amazon Simple Storage Service (Amazon S3) können Sie Daten als Objekte in Buckets speichern. AWS-Service Amazon S3 S3-Buckets bieten unbegrenzten Speicherplatz. Die maximale Größe für ein Objekt in einem Amazon S3 S3-Bucket beträgt 5 TB. Sie können jede Art von Dateidaten in einen Amazon S3 S3-Bucket hochladen, z. B. Bilder, Videos, Textdateien, Sicherungsdateien, Mediendateien für eine Website, archivierte Dokumente und Ihre Braket-Quantenaufgabenergebnisse.

- Bewährtes Verfahren: Sie können Berechtigungen festlegen, um den Zugriff auf Ihren S3-Bucket zu kontrollieren. Weitere Informationen finden Sie unter Bucket-Richtlinien in der Amazon S3 S3-Dokumentation.

Kostenverfolgung und Kosteneinsparung

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Mit Amazon Braket haben Sie bei Bedarf Zugriff auf Quantencomputer-Ressourcen, ohne dass Sie sich vorab verpflichten müssen. Sie zahlen nur das, was Sie nutzen. Weitere Informationen zur Preisgestaltung finden Sie auf unserer Preisseite.

In diesem Abschnitt:

- Ausgabenlimits für Amazon Braket QPUs festlegen
- Kostenverfolgung nahezu in Echtzeit
- Bewährte Methoden zur Kosteneinsparung

Ausgabenlimits für Amazon Braket QPUs festlegen

Die Ausgabenlimits von Amazon Braket bieten optionale Kostenkontrollen pro Gerät für Quantenverarbeitungseinheiten (QPUs).

So funktionieren Ausgabenlimits: Amazon Braket verfolgt Ihre kumulierten Ausgaben und validiert jede Anfrage zur Aufgabenerstellung anhand Ihres konfigurierten Limits. Wenn die geschätzten Kosten einer Aufgabe Ihr verbleibendes Ausgabenlimit überschreiten, lehnt Amazon Braket die Aufgabe sofort mit einem Validierungsfehler ab. Sie können optional einen Zeitraum für Ihr Ausgabenlimit konfigurieren. Durch die Konfiguration eines Zeitraums können Sie sicherstellen, dass Aufgaben nur in diesem angegebenen Zeitraum eingereicht werden können. Aufgaben, die außerhalb des Zeitraums eingereicht wurden, werden abgelehnt.

Opt-in Design: Bestehende Workflows bleiben davon unberührt, es sei denn, Sie aktivieren die Steuerelemente ausdrücklich. Sie können alle Einschränkungen aufheben, indem Sie das Ausgabenlimit löschen.

> **Note**
>
> Ausgabenlimits gelten nur für QPU-Aufgaben auf Abruf und hybride Jobs. Sie schließen Simulatoren, verwaltete Notebooks, Hybrid Job EC2-Instance-Kosten und Braket Direct-Reservierungen aus. Für ein umfassendes Kostenmanagement für alle AWS-Services nutzen Sie weiterhin AWS Budgets.

Liste der Maßnahmen zur Ausgabenbegrenzung

Suchen

Mit dem folgenden AWS-CLI-Befehl können Sie Ausgabenlimits in einer bestimmten AWS-Region und für ein bestimmtes Braket-Gerät suchen und auflisten.

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

Erstellen

Mit dem folgenden AWS-CLI-Befehl können Sie ein neues Ausgabenlimit für ein bestimmtes Quantengerät in einer bestimmten Region erstellen. Die Anfrage wird abgelehnt, wenn für das Gerät bereits ein Ausgabenlimit besteht.

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

Aktualisieren

Mit dem folgenden AWS-CLI-Befehl können Sie ein vorhandenes Ausgabenlimit auf einen neuen maximalen Ausgabenwert aktualisieren. Die Anfrage wird abgelehnt, wenn die Summe der aktuellen Ausgaben und der Ausgaben in der Warteschlange bereits über dem angeforderten neuen Höchstbetrag liegt.

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

Sie können anstelle oder zusätzlich zu den neuen Höchstausgaben einen Zeitraum angeben, wie im obigen Beispiel.

Löschen

Mit dem folgenden AWS-CLI-Befehl können Sie ein vorhandenes Ausgabenlimit löschen.

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

Sie können anstelle oder zusätzlich zu den neuen Höchstausgaben einen Zeitraum angeben, wie im obigen Beispiel.

Als bewährte Methode sollten Sie immer den Regionsparameter angeben, obwohl dies optional ist. Befehle, die in einer anderen Region als der des Geräts ausgeführt werden, schlagen fehl oder geben im `SearchSpendingLimits` Fall von falsche Ergebnisse zurück.

Weitere Beispiele zur Verwendung von Ausgabenlimits finden Sie im Beispiel-Notizbuch.

So funktioniert die Aufgabenvalidierung

Wenn das AWS-Konto eine ansonsten gültige `CreateQuantumTask` Anfrage sendet, unterliegt es dem folgenden Gating-Verhalten. Hinweis: Das verbleibende Budget ist die Differenz zwischen dem Ausgabenlimit und der Summe der Ausgaben in der Warteschlange und der aktuellen Ausgaben. (Siehe nächsten Abschnitt)

- Fall 1: Es gibt kein Ausgabenlimit für das Task-Gerät: Die Aufgabe wurde erstellt.
- Fall 2: Es gibt ein Ausgabenlimit für das Zielgerät, und die aktuelle Uhrzeit liegt innerhalb des Zeitraums, für den das Ausgabenlimit gilt:
  - Wenn die geschätzten Kosten der Aufgabe niedriger oder gleich dem verbleibenden Budget sind: CreateQuantumTask erfolgreich, wird die Aufgabe erstellt.
  - Wenn die geschätzten Kosten höher sind als das verbleibende Budget: `CreateQuantumTask` schlägt fehl und es wird keine Aufgabe erstellt.
- Fall 3: Es gibt ein Ausgabenlimit für das Zielgerät und die aktuelle Uhrzeit liegt außerhalb des Zeitraums des Ausgabenlimits: `CreateQuantumTask` schlägt fehl und es wird keine Aufgabe erstellt.

Wie wird das verbleibende Budget berechnet

Das verbleibende Budget ist die Differenz zwischen dem Ausgabenlimit und der Summe der aktuellen Ausgaben und der Ausgaben in der Warteschlange.

Wenn eine Aufgabe für ein Gerät mit einem Ausgabenlimit erstellt wird, werden die Ausgaben in der Warteschlange um die geschätzten Kosten der Aufgabe erhöht. Dieses Ereignis ist in der ersten Zeile der folgenden Tabelle aufgeführt. Die folgende Tabelle zeigt, was mit den Ausgaben in der Warteschlange und den aktuellen Ausgaben je nach Fortschritt der Aufgabe passiert.

Alter Zustand der Quantenaufgabe	Neuer Quanten-Aufgabenzustand	Wechseln Sie zu Ausgaben in der Warteschlange	Zu aktuellen Ausgaben wechseln
-	CREATED	Erhöht um die geschätzten Kosten	Keine Änderung
CREATED	IN WARTESCHLANGE	Keine Änderung	Keine Änderung
Beliebig	AUSFÜHREN	Keine Änderung	Keine Änderung
Beliebig	CANCELLING	Keine Änderung	Keine Änderung
CANCELLING	CANCELLED	Reduziert um die geschätzten Kosten	Keine Änderung
Beliebig	FEHLGESCHLAGEN	Reduziert um die geschätzten Kosten	Keine Änderung
AUSFÜHREN	COMPLETED	Reduziert um die geschätzten Kosten	Erhöht um die geschätzten Kosten (entsprechend angepasst für teilweise erledigte Aufgaben)

Sonderfälle

F: Werden Aufgaben, die sich bereits in der Warteschlange befinden, bei der Erstellung eines Ausgabenlimits auf die Ausgaben in der Warteschlange angerechnet?

A: Nein. Aufgaben, die bereits erstellt wurden, sich in der Warteschlange befinden oder anderweitig in Bearbeitung sind, werden nicht auf die Ausgaben in der Warteschlange eines neu erstellten Ausgabenlimits angerechnet.

F: Führt die Senkung des Ausgabenlimits durch dessen Aktualisierung dazu, dass erstellte, in der Warteschlange befindliche oder anderweitig in Bearbeitung befindliche Quantenaufgaben vorzeitig beendet werden?

A: Nein.

F: Führt das Erreichen der Endzeit des Ausgabenlimits dazu, dass eine erstellte, in der Warteschlange befindliche oder anderweitig in Bearbeitung befindliche Quantenaufgabe vorzeitig beendet wird?

A: Nein. Erstellte, in der Warteschlange befindliche und anderweitig in Bearbeitung befindliche Aufgaben können unabhängig vom Status des Ausgabenlimits abgeschlossen werden.

F: Wie unterscheidet sich ein fehlendes Ausgabenlimit von einem Ausgabenlimit von null Dollar?

A: Da es kein Ausgabenlimit gibt, können Quantenaufgaben ohne Einschränkungen erstellt werden. Ein Ausgabenlimit von null Dollar blockiert alle Quantenaufgaben.

F: Blockiert ein Ausgabenlimit von Null in der Vergangenheit oder future die gesamte Erstellung von Quantenaufgaben?

A: Ja.

F: Werden bei der Festlegung eines Ausgabenlimits die geschätzten Kosten für Aufgaben, die sich bereits in der Warteschlange befinden, auf die aktuellen Ausgaben angerechnet, sobald diese Aufgaben abgeschlossen sind?

A: Nein. Nur Aufgaben, die eingereicht werden, solange ein Ausgabenlimit aktiv ist, werden auf die kumulierten Ausgaben angerechnet.

Kostenverfolgung nahezu in Echtzeit

Das Braket SDK bietet Ihnen die Möglichkeit, Ihre Quanten-Workloads um eine Kostenverfolgung nahezu in Echtzeit zu erweitern. Jedes unserer Beispiel-Notebooks enthält einen Code zur Kostenverfolgung, mit dem Sie einen maximalen Kostenvoranschlag für die Quantenverarbeitungseinheiten (QPUs) und On-Demand-Simulatoren von Braket erhalten. Die geschätzten Höchstkosten werden in USD angezeigt und beinhalten keine Gutschriften oder Rabatte.

> **Note**
>
> Die angegebenen Gebühren sind Schätzungen, die auf der Nutzung Ihres Amazon Braket-Simulators und Ihrer QPU-Aufgaben (Quantum Processing Unit) basieren. Die angezeigten geschätzten Gebühren können von Ihren tatsächlichen Gebühren abweichen. In den geschätzten Gebühren sind keine Rabatte oder Gutschriften enthalten, und es können zusätzliche Gebühren anfallen, wenn Sie andere Dienste wie Amazon Elastic Compute Cloud (Amazon EC2) nutzen.

Kostenverfolgung für SV1

Um zu demonstrieren, wie die Kostenverfolgungsfunktion verwendet werden kann, werden wir eine Bell-State-Schaltung konstruieren und sie auf unserem SV1-Simulator ausführen. Importieren Sie zunächst die Braket SDK-Module, definieren Sie einen Bell State und fügen Sie die `Tracker()` Funktion zu unserer Schaltung hinzu:

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

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

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

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

Wenn Sie Ihr Notebook ausführen, können Sie die folgende Ausgabe für Ihre Bell State-Simulation erwarten. Die Tracker-Funktion zeigt Ihnen die Anzahl der gesendeten Schüsse, die abgeschlossenen Quantenaufgaben, die Ausführungsdauer, die in Rechnung gestellte Ausführungsdauer und Ihre maximalen Kosten in USD an. Ihre Ausführungszeit kann für jede Simulation variieren.

```
import datetime

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

tracker.simulator_tasks_cost()
```

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

Verwenden Sie den Cost Tracker, um die maximalen Kosten festzulegen

Sie können den Cost Tracker verwenden, um die Höchstkosten für ein Programm festzulegen. Möglicherweise haben Sie einen Höchstbetrag dafür, wie viel Sie für ein bestimmtes Programm ausgeben möchten. Auf diese Weise können Sie den Cost Tracker verwenden, um die Kostenkontrolllogik in Ihrem Ausführungscode zu integrieren. Das folgende Beispiel verwendet dieselbe Schaltung auf einer Rigetti QPU und begrenzt die Kosten auf 1 USD. Die Kosten für die Ausführung einer Iteration der Schaltung in unserem Code betragen 0,30 USD. Wir haben die Logik so eingestellt, dass die Iterationen wiederholt werden, bis die Gesamtkosten 1 USD überschreiten. Daher wird der Codeausschnitt dreimal ausgeführt, bis die nächste Iteration 1 USD übersteigt. Im Allgemeinen würde ein Programm so lange iterieren, bis die von Ihnen gewünschten maximalen Kosten erreicht sind, in diesem Fall drei Iterationen.

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

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

> **Note**
>
> Der Cost Tracker erfasst nicht die Dauer fehlgeschlagener TN1 Quantenaufgaben. Wenn Ihre Probe während einer TN1 Simulation abgeschlossen ist, der Kontraktionsschritt jedoch fehlschlägt, wird Ihre Probengebühr nicht im Cost Tracker angezeigt.

Bewährte Methoden zur Kosteneinsparung

Beachten Sie die folgenden bewährten Methoden für die Verwendung von Amazon Braket. Sparen Sie Zeit, minimieren Sie Kosten und vermeiden Sie häufige Fehler.

Überprüfen Sie mit Simulatoren

- Überprüfen Sie Ihre Schaltungen mithilfe eines Simulators, bevor Sie sie auf einer QPU ausführen, sodass Sie Ihre Schaltung fein abstimmen können, ohne dass Gebühren für die Nutzung der QPU anfallen.
- Auch wenn die Ergebnisse der Ausführung der Schaltung auf einem Simulator möglicherweise nicht mit den Ergebnissen der Ausführung der Schaltung auf einer QPU identisch sind, können Sie Codierungsfehler oder Konfigurationsprobleme mithilfe eines Simulators identifizieren.

Beschränken Sie den Benutzerzugriff auf bestimmte Geräte

- Sie können Einschränkungen einrichten, die verhindern, dass unbefugte Benutzer Quantenaufgaben auf bestimmten Geräten einreichen. Die empfohlene Methode zur Zugriffsbeschränkung ist AWS IAM. Weitere Informationen dazu finden Sie unter Zugriff einschränken.
- Wir empfehlen Ihnen, Ihr Administratorkonto nicht zu verwenden, um Benutzern Zugriff auf Amazon Braket-Geräte zu gewähren oder einzuschränken.

Stellen Sie Abrechnungsalarme ein

- Sie können einen Abrechnungsalarm einrichten, der Sie benachrichtigt, wenn Ihre Rechnung ein voreingestelltes Limit erreicht. Die empfohlene Methode zum Einrichten eines Alarms ist die folgende AWS Budgets. Sie können benutzerdefinierte Budgets festlegen und Benachrichtigungen erhalten, wenn Ihre Kosten oder Nutzung Ihren budgetierten Betrag überschreiten könnten. Informationen finden Sie unter. AWS Budgets

Testen Sie TN1 Quantenaufgaben mit niedrigen Schusszahlen

- Simulatoren kosten weniger als QPUs, aber bestimmte Simulatoren können teuer sein, wenn Quantenaufgaben mit hohen Schusszahlen ausgeführt werden. Wir empfehlen Ihnen, Ihre TN1 Aufgaben mit einer niedrigen Anzahl zu testen. shot ShotDie Anzahl hat keinen Einfluss auf die Kosten für SV1 und lokale Simulatoraufgaben.

Suchen Sie in allen Regionen nach Quantenaufgaben

- In der Konsole werden Quantenaufgaben nur für Ihre aktuellen Aufgaben angezeigt AWS-Region. Wenn Sie nach abrechnungsfähigen Quantenaufgaben suchen, die eingereicht wurden, achten Sie darauf, alle Regionen zu überprüfen.
- Eine Liste der Geräte und der zugehörigen Regionen finden Sie auf der Dokumentationsseite Unterstützte Geräte.

API-Referenzen und Repos für Amazon Braket

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Amazon Braket bietet APIs, SDKs und eine Befehlszeilenschnittstelle, mit der Sie Notebook-Instances erstellen und verwalten sowie Modelle trainieren und bereitstellen können.

- Amazon Braket Python SDK (empfohlen)
- Amazon Braket-API-Referenz
- AWS Command Line Interface
- AWS SDK für .NET
- AWS SDK für C++
- AWS SDK für GoAPI Reference
- AWS SDK für Java
- AWS SDK für JavaScript

- AWS SDK für PHP
- AWS SDK für Python (Boto)
- AWS SDK für Ruby

Sie können Codebeispiele auch aus dem Amazon Braket GitHub Tutorials-Repository abrufen.

- Braket-Tutorials GitHub

Kern-Repositorien

Im Folgenden wird eine Liste der Core-Repositorys angezeigt, die wichtige Pakete enthalten, die für Braket verwendet werden:

- Braket Python SDK - Verwenden Sie das Braket Python SDK, um Ihren Code auf Jupyter Notebooks in der Programmiersprache Python einzurichten. Nachdem Ihre Jupyter Notebooks eingerichtet sind, können Sie Ihren Code auf Braket-Geräten und -Simulatoren ausführen
- Braket-Schemas — Der Vertrag zwischen dem Braket-SDK und dem Braket-Service.
- Braket Default Simulator — All unsere lokalen Quantensimulatoren für Braket (Zustandsvektor und Dichtematrix).

Plugins

Dann gibt es die verschiedenen Plugins, die zusammen mit verschiedenen Geräten und Programmierwerkzeugen verwendet werden. Dazu gehören von Braket unterstützte Plugins sowie Community-Plugins, wie unten gezeigt.

Amazon Braket unterstützt:

- Amazon Braket-Algorithmusbibliothek — Ein Katalog vorgefertigter Quantenalgorithmen, die in Python geschrieben wurden. Führen Sie sie so aus, wie sie sind, oder verwenden Sie sie als Ausgangspunkt, um komplexere Algorithmen zu erstellen.
- Braket-PennyLane Plugin — Wird PennyLane als QML-Framework auf Braket verwendet.
- Qiskit-Braket provider — Verwenden Sie das Qiskit SDK, um auf Braket-Ressourcen zuzugreifen.

Gemeinschaft:

- Braket-Julia SDK — (EXPERIMENTELL) Eine native Julia-Version des Braket-SDK

Von Amazon Braket unterstützte Regionen und Geräte

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit AWS! Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

In Amazon Braket steht ein Gerät für eine Quantenverarbeitungseinheit (QPU) oder einen Simulator, den Sie aufrufen können, um Quantenaufgaben auszuführen. Amazon Braket bietet Zugriff auf QPU-Geräte vonAQT,, IonQ IQMQuEra, und. Rigetti Darüber hinaus AWS bietet es Zugriff auf On-Demand-Simulatoren, lokale und eingebettete Simulatoren. Weitere Informationen zu eingebetteten Simulatoren finden Sie unter Über eingebettete Simulatoren.

Informationen zu unterstützten Quantenhardwareanbietern finden Sie unter Quantenaufgaben an QPUs senden. Informationen zu verfügbaren Simulatoren finden Sie unter Quantenaufgaben an Simulatoren senden. Die folgende Tabelle zeigt die Liste der verfügbaren Geräte und Simulatoren.

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
AQT	IBEX-Q1	Gate-basiert	QPU	arn:aws:braket:eu-north-1::/-Q1 device/qpu aqt/Ibex	eu-north-1
IonQ	Forte-1	Gate-basiert	QPU	arn:aws:braket:us-east-1::/-1 device/qpu ionq/Forte	us-east-1
IonQ	Forte-Enterprise-1	Gate-basiert	QPU	arn:aws:braket:us-east-1::/- device/qpu ionq/Forte Enterprise-1	us-east-1

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
IQM	Garnet	Gate-basiert	QPU	arn:aws:braket:eu-north-1::/device/qpuiqm/Garnet	eu-north-1
IQM	Emerald	Gate-basiert	QPU	arn:aws:braket:eu-north-1::/device/qpuiqm/Emerald	eu-north-1
QuEra	Aquila	Analoge Hamiltonsche Simulation	QPU	arn:aws:braket:us-east-1::/device/qpuquera/Aquila	us-east-1
Rigetti	Ankaa-3	Gate-basiert	QPU	arn:aws:braket:us-west-1::/-3 device/qpu rigetti/Ankaa	us-west-1
Rigetti	Cepheus-1-108Q	Gate-basiert	QPU	arn:aws:braket:us-west-1::/-1-108Q device/qpu rigetti/Cepheus	us-west-1
AWS	braket_sv	Gate-basiert	Lokaler Simulator	N/A (lokaler Simulator im Braket SDK)	N/A
AWS	braket_dm	Gate-basiert	Lokaler Simulator	N/A (lokaler Simulator im Braket SDK)	N/A
AWS	braket_ahs	Analoge Hamiltonsche Simulation	Lokaler Simulator	N/A (lokaler Simulator im Braket SDK)	N/A

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
AWS	SV1	Gate-basiert	On-demand Simulator	arn:aws:braket::: - / sv1 device/quantum simulator/amazon	us-east-1, us-west-1, us-west-2, eu-west-2
AWS	DM1	Gate-basiert	On-demand Simulator	arn:aws:braket::: - / dm1 device/quantum simulator/amazon	us-east-1, us-west-1, us-west-2, eu-west-2
AWS	TN1	Gate-basiert	On-demand Simulator	arn:aws:braket::: - / tn1 device/quantum simulator/amazon	us-east-1, us-west-2 und eu-west-2

> **Note**
>
> Geräte-ARNs unterscheiden zwischen Groß- und Kleinschreibung. Stellen Sie beispielsweise bei der Verwendung des AQT IBEX-Q1 Geräts sicher, dass der Geräte-ARN ARN enthält'Ibex-Q1'.

Weitere Informationen zu den QPUs, die Sie mit Amazon Braket verwenden können, finden Sie unter Amazon Braket Quantum Computers.

Eigenschaften des Geräts

Für alle Geräte finden Sie weitere Geräteeigenschaften wie Gerätetopologie, Kalibrierungsdaten und native Gate-Sets auf der Registerkarte Geräte der Amazon Braket-Konsole oder über die `GetDevice` API. Bei der Konstruktion einer Schaltung mit den Simulatoren verlangt Amazon Braket,

dass Sie zusammenhängende Qubits oder Indizes verwenden. Bei der Arbeit mit dem SDK zeigt das folgende Codebeispiel, wie Sie auf die Geräteeigenschaften für jedes verfügbare Gerät und jeden verfügbaren Simulator zugreifen können.

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

device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/sv1')
 # SV1
# device = LocalSimulator()
 # Local State Vector Simulator
# device = LocalSimulator("default")
 # Local State Vector Simulator
# device = LocalSimulator(backend="default")
 # Local State Vector Simulator
# device = LocalSimulator(backend="braket_sv")
 # Local State Vector Simulator
# device = LocalSimulator(backend="braket_dm")
 # Local Density Matrix Simulator
# device = LocalSimulator(backend="braket_ahs")
 # Local Analog Hamiltonian Simulation
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/tn1')
 # TN1
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/dm1')
 # DM1
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1')
 # AQT IBEX-Q1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
 # IonQ Forte-1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1')
 # IonQ Forte-Enterprise-1
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet')
 # IQM Garnet
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald')
 # IQM Emerald
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/quera/Aquila')
 # QuEra Aquila
# device = AwsDevice('arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3')
 # Rigetti Ankaa-3
# device = AwsDevice('arn:aws:braket:us-west-1::device/qpu/rigetti/Cepheus-1-108Q')
 # Rigetti Cepheus-1-108Q

# Get device properties
```

```
device.properties
```

Regionen und Endpunkte für Amazon Braket

Eine vollständige Liste der Regionen und Endpunkte finden Sie in der AWS Allgemeinen Referenz.

Quantum-Aufgaben, die auf einem QPU-Gerät ausgeführt werden, können in der Amazon Braket-Konsole in der Region dieses Geräts angezeigt werden. Wenn Sie das Amazon Braket SDK verwenden, können Sie Quantenaufgaben an jedes QPU-Gerät senden, unabhängig von der Region, in der Sie arbeiten. Das SDK erstellt automatisch eine Sitzung in der Region für die angegebene QPU.

Amazon Braket ist in den folgenden AWS-Regionen Ländern verfügbar:

Name der Region	Region	Braket-Endpunkte
USA Ost (Nord-Virginia)	us-east-1	braket.us-east-1.amazonaws.com (nur IPv4) braket.us-east-1.api.aws (Doppelstapel)
USA West (Nordkalifornien)	us-west-1	braket.us-west-1.amazonaws.com (nur IPv4) braket.us-west-1.api.aws (Doppelstapel)
USA West 2 (Oregon)	us-west-2	braket.us-west-2.amazonaws.com (nur IPv4) braket.us-west-2.api.aws (Doppelstapel)
EU Nord 1 (Stockholm)	eu-north-1	braket.eu-north-1.amazonaws.com (nur IPv4) braket.eu-north-1.api.aws (Doppelstapel)

Name der Region	Region	Braket-Endpunkte
EU West 2 (London)	eu-west-2	braket.eu-west-2.amazonaws.com (nur IPv4) braket.eu-west-2.api.aws (Doppelstapel)

> **Note**
>
> Das Amazon Braket SDK unterstützt keine IPv6-only Netzwerke.

Erste Schritte mit Amazon Braket

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit! AWS Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Nachdem Sie die Anweisungen unter Amazon Braket aktivieren befolgt haben, können Sie mit Amazon Braket beginnen.

Zu den ersten Schritten gehören:

- Amazon Braket aktivieren
- Erstellen Sie eine Amazon Braket-Notebook-Instance
- Erstellen Sie eine Braket-Notebook-Instanz mit CloudFormation

Amazon Braket aktivieren

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit! AWS Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Sie können Amazon Braket in Ihrem Konto über die AWS Konsole aktivieren.

In diesem Abschnitt:

- Voraussetzungen
- Schritte zur Aktivierung von Amazon Braket

Voraussetzungen

Um Amazon Braket zu aktivieren und auszuführen, benötigen Sie einen Benutzer oder eine Rolle mit der Berechtigung, Amazon Braket-Aktionen zu initiieren. Diese Berechtigungen sind in der AmazonBraketFullAccess IAM-Richtlinie (arn:aws:iam: :aws:policy/) enthalten. AmazonBraketFullAccess

> **Note**
>
> Wenn Sie Administrator sind:
> Um anderen Benutzern Zugriff auf Amazon Braket zu gewähren, gewähren Sie Benutzern Berechtigungen, indem Sie die AmazonBraketFullAccessRichtlinie oder eine von Ihnen erstellte benutzerdefinierte Richtlinie anhängen. Weitere Informationen zu den für die Nutzung von Amazon Braket erforderlichen Berechtigungen finden Sie unter Zugriff auf Amazon Braket verwalten.

Schritte zur Aktivierung von Amazon Braket

1. Melden Sie sich mit Ihrem AWS-Konto bei der Amazon Braket-Konsole an.
2. Öffnen Sie die Amazon Braket-Konsole.
3. Klicken Sie auf der Braket-Landingpage auf Get Started, um zur Service Dashboard-Seite zu gelangen. Die Warnung oben in Ihrem Service-Dashboard führt Sie durch die folgenden drei Schritte:
   a. Serviceverknüpfte Rollen (SLR) erstellen
   b. Aktivierung des Zugriffs auf Quantencomputer von Drittanbietern
   c. Eine neue Jupyter-Notebook-Instanz erstellen

Um Quantengeräte von Drittanbietern verwenden zu können, müssen Sie bestimmten Bedingungen für die Datenübertragung zwischen Ihnen und diesen Geräten zustimmen. AWS Die Bedingungen dieser Vereinbarung finden Sie auf der Seite „Berechtigungen und Einstellungen“ in der Amazon Braket-Konsole auf der Registerkarte „Allgemein“.

> **Note**
>
> Quanten-Geräte, an denen keine Drittanbieter beteiligt sind, wie z. B. die lokalen Braket-Simulatoren oder On-Demand-Simulatoren, können verwendet werden, ohne der Vereinbarung zur Aktivierung von Drittanbietergeräten zuzustimmen.
> Wenn Sie auf Hardware von Drittanbietern zugreifen, müssen Sie diese Bedingungen nur einmal pro Konto akzeptieren, um die Nutzung von Geräten von Drittanbietern zu ermöglichen.

Erstellen Sie eine Amazon Braket-Notebook-Instance

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit! AWS Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Amazon Braket bietet vollständig verwaltete Jupyter-Notebooks für den Einstieg. Die Amazon Braket-Notebook-Instances basieren auf Amazon SageMaker AI-Notebook-Instances. In den folgenden Schritten wird beschrieben, wie Sie eine neue Notebook-Instance für neue und bestehende Kunden erstellen.

Neue Amazon Braket-Kunden:

1. Öffnen Sie die Amazon Braket-Konsole und navigieren Sie zur Dashboard-Seite im linken Bereich.
2. Klicken Sie im Modal Willkommen bei Amazon Braket in der Mitte Ihrer Dashboard-Seite auf Erste Schritte. Geben Sie einen Notizbuchnamen ein, um ein Standard-Jupyter-Notizbuch zu erstellen.
   a. Die Erstellung Ihres Notizbuchs kann mehrere Minuten dauern.
   b. Ihr Notizbuch wird auf der Seite Notizbücher mit dem Status Ausstehend aufgeführt.
   c. Wenn Ihre Notebook-Instanz einsatzbereit ist, ändert sich der Status auf InService.
   d. Aktualisieren Sie die Seite, um den aktualisierten Status des Notebooks anzuzeigen.

Bestehende Amazon Braket-Kunden:

1. Öffnen Sie die Amazon Braket-Konsole und wählen Sie im linken Bereich Notebooks aus.

2. Wählen Sie Notebook-Instanz erstellen aus.
   a. Wenn Sie keine Notebooks haben, wählen Sie das Standard-Setup aus, um ein Standard-Jupyter-Notebook zu erstellen.
3. Geben Sie einen Notebook-Instanznamen ein, der nur alphanumerische Zeichen und Bindestriche enthält, und wählen Sie Ihren bevorzugten visuellen Modus aus.
4. Aktivieren oder deaktivieren Sie den Notebook-Inaktivitätsmanager für Ihr Notebook.
   a. Wenn diese Option aktiviert ist, wählen Sie die gewünschte Dauer im Leerlauf aus, bevor das Notebook zurückgesetzt wird. Wenn ein Notebook zurückgesetzt wird, fallen keine Rechenkosten mehr an, die Speichergebühren bleiben jedoch bestehen.
   b. Um zu überprüfen, wie viel Leerlaufzeit für Ihre Notebook-Instanz noch übrig ist, navigieren Sie zur Befehlsleiste, wählen Sie die Registerkarte Braket und dann die Registerkarte Inactivity Manager aus.

> **Note**
>
> Um Ihre Arbeit zu speichern, integrieren Sie Ihre SageMaker KI-Notebook-Instanz in ein Git-Repository. Verschieben Sie Ihre Arbeit alternativ aus den `/Braket Examples` Ordnern `/Braket Algorithms` und, damit sie nicht durch den Neustart der Notebook-Instanz überschrieben werden.

5. (Optional) Mit den erweiterten Einstellungen können Sie ein Notizbuch mit Zugriffsberechtigungen, zusätzlichen Konfigurationen und Netzwerkzugriffseinstellungen erstellen:
   a. Wählen Sie in der Notebook-Konfiguration Ihren Instanztyp aus.
      i. Standardmäßig wird der kostengünstige Standard-Instance-Typ ml.t3.medium ausgewählt. Weitere Informationen zu den Instance-Preisen finden Sie unter Amazon SageMaker AI-Preise.
   b. Um ein öffentliches Github-Repository mit deiner Notebook-Instanz zu verknüpfen, klicke auf das Drop-down-Menü Git-Repository und wähle im Dropdownmenü Repository die Option Öffentliches Git-Repository von URL klonen aus. Geben Sie die URL des Repos in die URL-Textleiste des Git-Repositorys ein.
   c. Konfigurieren Sie unter Zugriffsberechtigungen alle optionalen IAM-Rollen, Root-Zugriff und Verschlüsselungsschlüssel.
   d. Konfigurieren Sie unter Netzwerkzugriff benutzerdefinierte Netzwerk- und Zugriffseinstellungen für Ihre Jupyter Notebook Instance.

6. Überprüfen Sie Ihre Einstellungen und legen Sie alle Tags fest, um Ihre Notebook-Instance zu identifizieren. Klicken Sie auf Launch.

> **Note**
>
> Zeigen Sie Ihre Amazon Braket-Notebook-Instances in den Amazon Braket- und Amazon SageMaker AI-Konsolen an und verwalten Sie sie. Zusätzliche Amazon Braket-Notebook-Einstellungen sind über die SageMaker Konsole verfügbar.

Wenn Sie in der Amazon Braket-Konsole innerhalb AWS des Amazon Braket-SDK arbeiten und Plugins in den von Ihnen erstellten Notizbüchern vorinstalliert sind. Um es auf Ihrem eigenen Computer auszuführen, installieren Sie das SDK und die Plug-ins, wenn Sie den Befehl ausführen `pip install amazon-braket-sdk` oder wenn Sie den Befehl `pip install amazon-braket-pennylane-plugin` für Plugins ausführen. PennyLane

Erstellen Sie eine Braket-Notebook-Instanz mit CloudFormation

> **Tip**
>
> Lernen Sie die Grundlagen des Quantencomputers kennen mit! AWS Melden Sie sich für den Amazon Braket Digital Learning Plan an und verdienen Sie sich Ihr eigenes digitales Badge, nachdem Sie eine Reihe von Lernkursen und eine digitale Prüfung abgeschlossen haben.

Sie können CloudFormation damit Ihre Amazon Braket-Notebook-Instances verwalten. Braket-Notebook-Instances basieren auf Amazon SageMaker AI. Mit CloudFormation können Sie eine Notebook-Instance mit einer Vorlagendatei bereitstellen, die die beabsichtigte Konfiguration beschreibt. Die Vorlagendatei ist im JSON- oder YAML-Format geschrieben. Sie können Instanzen geordnet und wiederholbar erstellen, aktualisieren und löschen. Dies kann nützlich sein, wenn Sie mehrere Braket-Notebook-Instanzen in Ihrem verwalten. AWS-Konto

Nachdem Sie eine CloudFormation Vorlage für ein Braket-Notizbuch erstellt haben, verwenden Sie diese für CloudFormation die Bereitstellung der Ressource. Weitere Informationen finden Sie im CloudFormation Benutzerhandbuch unter Erstellen eines Stacks auf der CloudFormation Konsole.

Um eine Braket-Notebook-Instanz mit zu erstellen CloudFormation, führen Sie die folgenden drei Schritte aus:

1. Erstellen Sie ein SageMaker AI-Lifecycle-Konfigurationsskript.
2. Erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle, die von SageMaker KI übernommen werden soll.
3. Erstellen Sie eine SageMaker KI-Notebook-Instanz mit dem Präfix **`amazon-braket-`**

Sie können die Lebenszykluskonfiguration für alle von Ihnen erstellten Braket-Notebooks wiederverwenden. Sie können die IAM-Rolle auch für die Braket-Notebooks wiederverwenden, denen Sie dieselben Ausführungsberechtigungen zuweisen.

In diesem Abschnitt:

- Schritt 1: Erstellen Sie ein SageMaker AI-Lifecycle-Konfigurationsskript
- Schritt 2: Erstellen Sie die von Amazon SageMaker AI übernommene IAM-Rolle
- Schritt 3: Erstellen Sie eine SageMaker AI-Notebook-Instanz mit dem Präfix amazon-braket-

Schritt 1: Erstellen Sie ein SageMaker AI-Lifecycle-Konfigurationsskript

Verwenden Sie die folgende Vorlage, um ein SageMaker AI-Lifecycle-Konfigurationsskript zu erstellen. Das Skript passt eine SageMaker KI-Notebook-Instanz für Braket an. Die Konfigurationsoptionen für die CloudFormation Lifecycle-Ressource finden Sie AWS::SageMaker::NotebookInstanceLifecycleConfigim CloudFormation Benutzerhandbuch.

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

```
            exit 0
```

Schritt 2: Erstellen Sie die von Amazon SageMaker AI übernommene IAM-Rolle

Wenn Sie eine Braket-Notebook-Instance verwenden, führt SageMaker KI Operationen in Ihrem Namen aus. Nehmen wir zum Beispiel an, Sie führen ein Braket-Notebook mithilfe eines Circuits auf einem unterstützten Gerät aus. Innerhalb der Notebook-Instanz führt SageMaker KI den Vorgang auf Braket für Sie aus. Die Notebook-Ausführungsrolle definiert die genauen Operationen, die SageMaker KI in Ihrem Namen ausführen darf. Weitere Informationen finden Sie unter SageMaker KI-Rollen im Amazon SageMaker AI-Entwicklerhandbuch.

Verwenden Sie das folgende Beispiel, um eine Braket-Notebook-Ausführungsrolle mit den erforderlichen Berechtigungen zu erstellen. Sie können die Richtlinien Ihren Bedürfnissen entsprechend ändern.

> **Note**
>
> Stellen Sie sicher, dass die Rolle über die Berechtigung für die `s3:ListBucket` und für `s3:GetObject` Operationen auf Amazon S3 S3-Buckets verfügt, mit dem Präfix. `braketnotebookcdk-"` Das Lifecycle-Konfigurationsskript benötigt diese Berechtigungen, um das Braket-Notebook-Installationsskript zu kopieren.

```
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: "*"
```

Schritt 3: Erstellen Sie eine SageMaker AI-Notebook-Instanz mit dem Präfix **`amazon-braket-`**

Verwenden Sie das SageMaker AI-Lifecycle-Skript und die in Schritt 1 und Schritt 2 erstellte IAM-Rolle, um eine SageMaker AI-Notebook-Instance zu erstellen. Die Notebook-Instance ist für Braket angepasst und kann über die Amazon Braket-Konsole aufgerufen werden. Weitere Informationen zu den Konfigurationsoptionen für diese CloudFormation Ressource finden Sie AWS::SageMaker::NotebookInstanceim CloudFormation Benutzerhandbuch.

```
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
```

Erstellen Sie Ihre Quantenaufgaben mit Amazon Braket

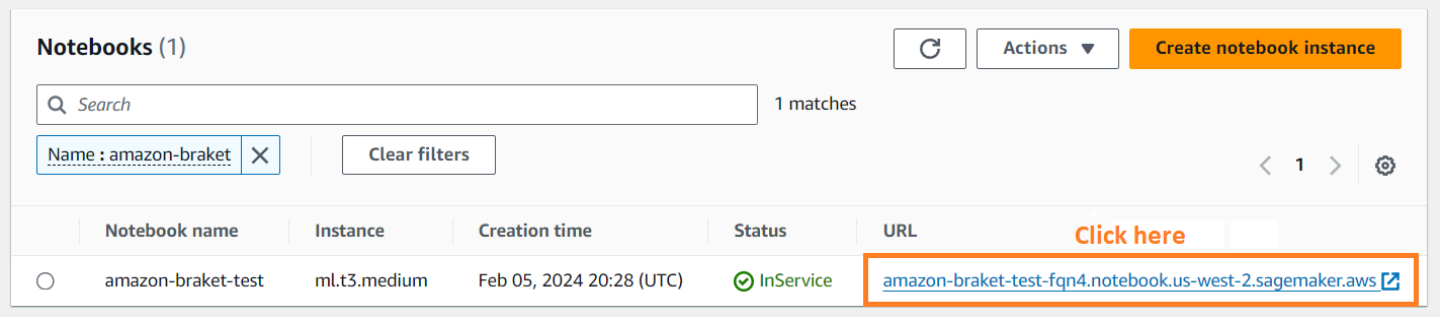
Braket bietet vollständig verwaltete Jupyter-Notebook-Umgebungen, die den Einstieg erleichtern. Braket-Notebooks sind mit Beispielalgorithmen, Ressourcen und Entwicklertools, einschließlich des Amazon Braket-SDK, vorinstalliert. Mit dem Amazon Braket SDK können Sie Quantenalgorithmen erstellen und diese dann auf verschiedenen Quantencomputern und Simulatoren testen und ausführen, indem Sie eine einzige Codezeile ändern.

In diesem Abschnitt:

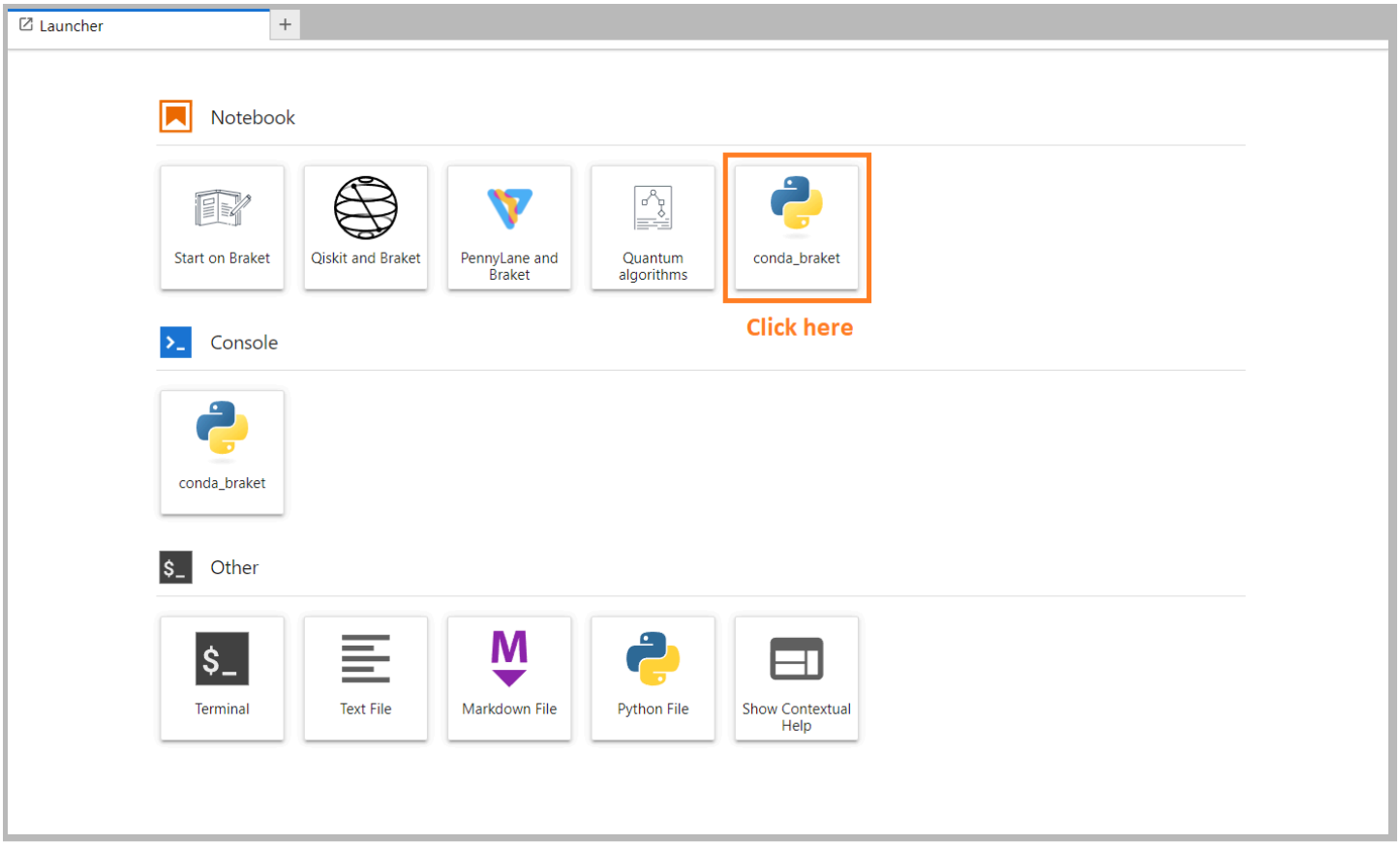
- Erstellen Sie Ihre erste Schaltung
- Expertenrat einholen
- Betreiben Sie Ihre Schaltungen mit OpenQASM 3.0
- Erkunden Sie experimentelle Möglichkeiten
- Pulssteuerung auf Amazon Braket
- Analoge Hamiltonsche Simulation
- Arbeitet mit AWS Boto3

Erstellen Sie Ihre erste Schaltung

Nachdem Ihre Notebook-Instanz gestartet wurde, öffnen Sie die Instanz mit einer Standard-Jupyter-Oberfläche, indem Sie das gerade erstellte Notebook auswählen.



Amazon Braket-Notebook-Instances sind mit dem Amazon Braket-SDK und all seinen Abhängigkeiten vorinstalliert. Erstellen Sie zunächst ein neues Notizbuch mit Kernel. `conda_braket`



Sie können mit einem einfachen „Hallo, Welt!“ beginnen Beispiel. Konstruieren Sie zunächst eine Schaltung, die einen Bell-Zustand vorbereitet, und führen Sie diese Schaltung dann auf verschiedenen Geräten aus, um die Ergebnisse zu erhalten.

Importieren Sie zunächst die Module Begin, indem Sie die Amazon Braket SDK-Module importieren und ein SimpleBraketLong; SDK-Modul definieren und einen grundlegenden Bell State-Schaltkreis definieren.

```
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)
```

Sie können den Schaltkreis mit diesem Befehl visualisieren:

```
print(bell)
```

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

Führen Sie Ihre Schaltung auf dem lokalen Simulator aus

Wählen Sie als Nächstes das Quantengerät aus, auf dem die Schaltung ausgeführt werden soll. Das Amazon Braket SDK enthält einen lokalen Simulator für schnelles Prototyping und Testen. Wir empfehlen die Verwendung des lokalen Simulators für kleinere Schaltungen, die bis zu 25 sein können qubits (abhängig von Ihrer lokalen Hardware).

Um den lokalen Simulator zu instanziieren:

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

und führe die Schaltung aus:

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

Sie sollten ein Ergebnis sehen, das in etwa so aussieht:

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

Der spezifische Bell-Zustand, den Sie vorbereitet haben, besteht erwartungsgemäß aus einer gleichmäßigen Überlagerung von |00# und |11# und einer etwa gleichen Verteilung (bis zum shot Rauschen) von 00 und 11 als Messergebnisse.

Lassen Sie Ihre Schaltung auf einem On-Demand-Simulator laufen

Amazon Braket bietet auch Zugriff auf einen leistungsstarken On-Demand-Simulator für den Betrieb größerer Schaltungen. SV1 SV1ist ein On-Demand-Zustandsvektorsimulator, der die Simulation von Quantenschaltkreisen von bis zu 34 ermöglicht. qubits Weitere Informationen finden Sie SV1

im Abschnitt Unterstützte Geräte und in der AWS Konsole. Wenn Sie Quantenaufgaben auf SV1 (und auf TN1 oder einer beliebigen QPU) ausführen, werden die Ergebnisse Ihrer Quantenaufgabe in einem S3-Bucket in Ihrem Konto gespeichert. Wenn Sie keinen Bucket angeben, erstellt das Braket SDK einen Standard-Bucket `amazon-braket-{region}-{accountID}` für Sie. Weitere Informationen finden Sie unter Zugriff auf Amazon Braket verwalten.

> **Note**
>
> Geben Sie Ihren tatsächlichen, vorhandenen Bucket-Namen ein, wobei das folgende Beispiel `amazon-braket-s3-demo-bucket` als Ihr Bucket-Name angezeigt wird. Bucket-Namen für Amazon Braket beginnen immer mit, `amazon-braket-` gefolgt von anderen identifizierenden Zeichen, die Sie hinzufügen. Informationen zur Einrichtung eines S3-Buckets finden Sie unter Erste Schritte mit Amazon S3.

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

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

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

Um einen Circuit auszuführenSV1, müssen Sie den Speicherort des S3-Buckets angeben, den Sie zuvor als Positionsargument im `.run()` Aufruf ausgewählt haben.

```
# 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)
```

Die Amazon Braket-Konsole bietet weitere Informationen zu Ihrer Quantenaufgabe. Navigieren Sie in der Konsole zum Tab Quantum Tasks und Ihre Quantenaufgabe sollte ganz oben auf der Liste

stehen. Alternativ können Sie anhand der eindeutigen Quantenaufgaben-ID oder anderer Kriterien nach Ihrer Quantenaufgabe suchen.

> **Note**
>
> Nach 90 Tagen entfernt Amazon Braket automatisch alle Quantenaufgaben-IDs und andere Metadaten, die mit Ihren Quantenaufgaben verknüpft sind. Weitere Informationen finden Sie unter Datenspeicherung.

Läuft auf einer QPU

Mit Amazon Braket können Sie das vorherige Quantenschaltungsbeispiel auf einem physikalischen Quantencomputer ausführen, indem Sie einfach eine einzige Codezeile ändern. Amazon Braket bietet Zugriff auf eine Vielzahl von QPU-Geräten (Quantum Processing Unit). Informationen zu den verschiedenen Geräten und Verfügbarkeitsfenstern finden Sie im Abschnitt Unterstützte Geräte und in der AWS Konsole unter dem Tab Geräte. Das folgende Beispiel zeigt, wie ein IQM Gerät instanziiert wird.

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

Oder wählen Sie ein IonQ Gerät mit diesem Code:

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

Nachdem Sie ein Gerät ausgewählt haben und bevor Sie Ihren Workload ausführen, können Sie mit dem folgenden Code die Tiefe der Gerätewarteschlange abfragen, um die Anzahl der Quantenaufgaben oder Hybrid-Jobs zu ermitteln. Darüber hinaus können sich Kunden die gerätespezifischen Warteschlangentiefen auf der Geräteseite von anzeigen lassenAmazon Braket Management Console.

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

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

Wenn Sie Ihre Aufgabe ausführen, fragt das Amazon Braket SDK nach einem Ergebnis ab (mit einem Standard-Timeout von 5 Tagen). Sie können diese Standardeinstellung ändern, indem Sie den `poll_timeout_seconds` Parameter im `.run()` Befehl ändern, wie im folgenden Beispiel gezeigt. Denken Sie daran, dass bei einem zu kurzen Abfrage-Timeout möglicherweise keine Ergebnisse innerhalb der Abfragezeit zurückgegeben werden, z. B. wenn eine QPU nicht verfügbar ist und ein lokaler Timeout-Fehler zurückgegeben wird. Sie können die Abfrage erneut starten, indem Sie die Funktion aufrufen. `task.result()`

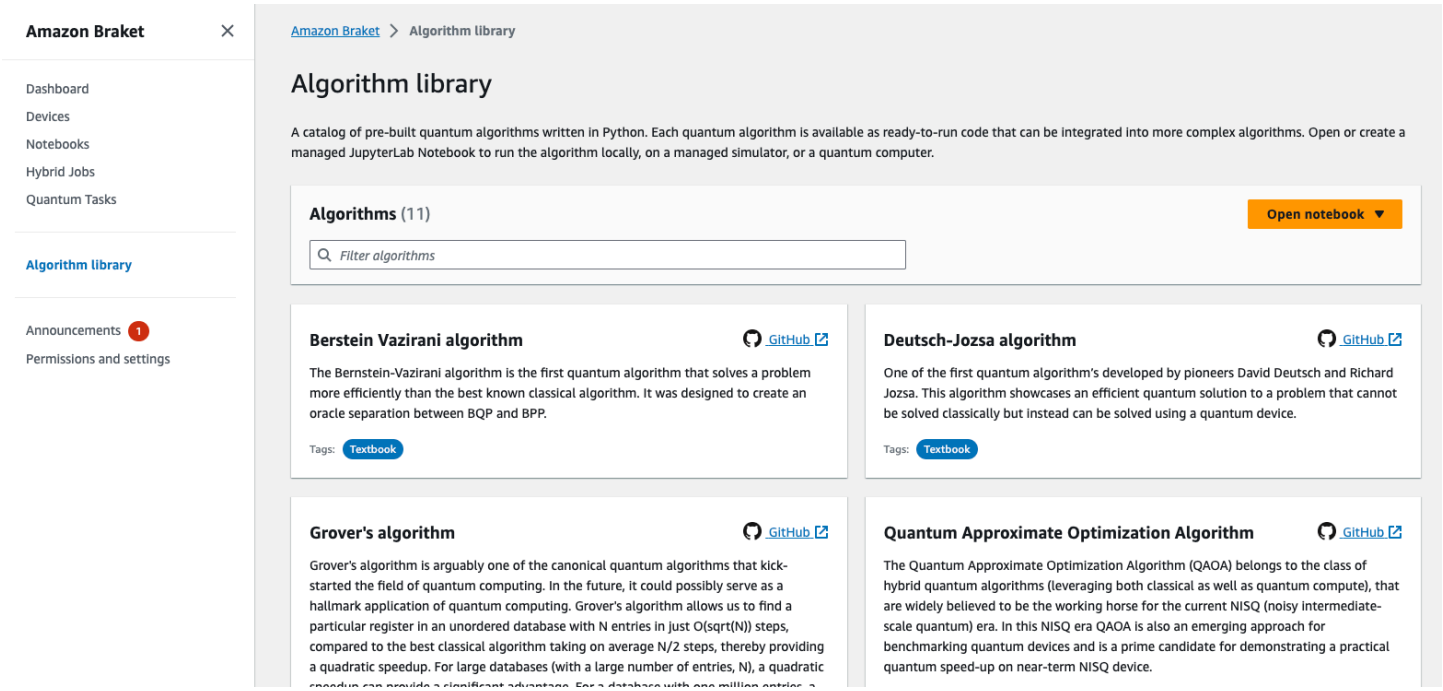
```
# Define quantum task with 1 day polling timeout
task = device.run(bell, s3_folder, poll_timeout_seconds=24*60*60)
print(task.result().measurement_counts)
```

Darüber hinaus können Sie nach dem Absenden Ihrer Quantenaufgabe oder Ihres Hybrid-Jobs die `queue_position()` Funktion aufrufen, um Ihre Warteschlangenposition zu überprüfen.

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

Entwickeln Sie Ihre ersten Quantenalgorithmen

Die Amazon Braket-Algorithmusbibliothek ist ein Katalog vorgefertigter Quantenalgorithmen, die in Python geschrieben wurden. Führen Sie diese Algorithmen unverändert aus oder verwenden Sie sie als Ausgangspunkt für die Erstellung komplexerer Algorithmen. Sie können von der Braket-Konsole aus auf die Algorithmusbibliothek zugreifen. Weitere Informationen finden Sie in der Braket Github-Algorithmusbibliothek.



Die Braket-Konsole bietet eine Beschreibung aller verfügbaren Algorithmen in der Algorithmusbibliothek. Wählen Sie einen GitHub Link, um die Details der einzelnen Algorithmen zu sehen, oder wählen Sie Notizbuch öffnen, um ein Notizbuch zu öffnen oder zu erstellen, das alle verfügbaren Algorithmen enthält. Wenn Sie die Option Notizbuch wählen, finden Sie die Braket-Algorithmusbibliothek anschließend im Stammordner Ihres Notizbuchs.

Schaltkreise im SDK konstruieren

Dieser Abschnitt enthält Beispiele für das Definieren eines Schaltkreises, das Anzeigen verfügbarer Gatter, das Erweitern eines Schaltkreises und das Anzeigen von Gates, die jedes Gerät unterstützt. Er enthält auch Anweisungen zur manuellen Zuweisung, zur Anweisung an den Compilerqubits, Ihre Schaltungen exakt wie definiert auszuführen, und zur Erstellung von verrauschten Schaltungen mit einem Geräuschsimulator.

Sie können in Braket auch auf Pulsebene für verschiedene Gatter mit bestimmten QPUs arbeiten. Weitere Informationen finden Sie unter Pulse Control auf Amazon Braket.

In diesem Abschnitt:

- Tore und Stromkreise
- Programmsätze
- Teilweise Messung
- Manuell Qubit Zuweisung

- Wörtliche Zusammenstellung
- Simulation von Geräuschen

Tore und Stromkreise

Quantengatter und -schaltungen sind in der `braket.circuits`Klasse des Amazon Braket Python SDK definiert. Über das SDK können Sie ein neues Circuit-Objekt instanziieren, indem Sie es aufrufen. `Circuit()`

Beispiel: Definieren Sie einen Schaltkreis

Das Beispiel beginnt mit der Definition eines Beispielschaltkreises mit vier qubits (mit,, und beschriftetq3) `q0` `q1q2`, der aus standardmäßigen Single-Qubit-Hadamard-Gattern und Zwei-Qubit-CNOT-Gattern besteht. Sie können diesen Schaltkreis visualisieren, indem Sie die Funktion aufrufen, wie das folgende Beispiel zeigt. `print`

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

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

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

Beispiel: Definieren Sie einen parametrisierten Schaltkreis

In diesem Beispiel definieren wir einen Schaltkreis mit Gattern, die von freien Parametern abhängen. Wir können die Werte dieser Parameter angeben, um eine neue Schaltung zu erstellen oder, wenn wir die Schaltung einreichen, sie als Quantenaufgabe auf bestimmten Geräten laufen zu lassen.

```
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)
```

Sie können aus einem parametrisierten Schaltkreis einen neuen, nicht parametrisierten Schaltkreis erstellen, indem Sie entweder einzelne Argumente `float` (das ist der Wert, den alle freien Parameter annehmen) oder Schlüsselwortargumente angeben, die den Wert jedes Parameters für den Schaltkreis wie folgt angeben.

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

Beachten Sie, dass er unverändert `my_circuit` ist, sodass Sie ihn verwenden können, um viele neue Schaltungen mit festen Parameterwerten zu instanziieren.

Beispiel: Ändern Sie Gates in einem Schaltkreis

Das folgende Beispiel definiert einen Schaltkreis mit Gattern, die Steuerungs- und Leistungsmodifikatoren verwenden. Sie können diese Änderungen verwenden, um neue Tore zu erstellen, z. B. das gesteuerte Ry Tor.

```
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)
```

Tormodifikatoren werden nur im lokalen Simulator unterstützt.

Beispiel: Alle verfügbaren Gates anzeigen

Das folgende Beispiel zeigt, wie Sie sich alle verfügbaren Gates in Amazon Braket ansehen können.

```
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)
```

Die Ausgabe dieses Codes listet alle Gates auf.

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

Jedes dieser Gatter kann an einen Schaltkreis angehängt werden, indem die Methode für diesen Schaltungstyp aufgerufen wird. Rufen Sie beispielsweise auf`circ.h(0)`, um dem ersten ein Hadamard-Gate hinzuzufügen. qubit

> **Note**
>
> Gatter werden an der richtigen Stelle angefügt, und im folgenden Beispiel werden alle im vorherigen Beispiel aufgelisteten Gatter demselben Schaltkreis hinzugefügt.

```
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)
```

Neben dem vordefinierten Gattersatz können Sie dem Schaltkreis auch selbstdefinierte einheitliche Gatter zuweisen. Dabei kann es sich um Single-Qubit-Gates (wie im folgenden Quellcode gezeigt) oder um Multi-Qubit-Gates handeln, die auf die durch den Parameter definierten Gates angewendet werden. qubits `targets`

```
import numpy as np

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

Beispiel: Erweitern Sie bestehende Schaltungen

Sie können bestehende Schaltungen erweitern, indem Sie Anweisungen hinzufügen. An `Instruction` ist eine Quantenrichtlinie, die die Quantenaufgabe beschreibt, die auf einem Quantengerät ausgeführt werden muss. `Instruction`Operatoren schließen `Gate` nur Objekte des Typs ein.

```
# Import the Gate and Instruction modules
```

```
from braket.circuits import Gate, Instruction

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

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

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

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

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

Beispiel: Sehen Sie sich die Gates an, die jedes Gerät unterstützt

Simulatoren unterstützen alle Gates im Braket-SDK, aber QPU-Geräte unterstützen eine kleinere Teilmenge. Sie finden die unterstützten Gates eines Geräts in den Geräteeigenschaften. Das Folgende zeigt ein Beispiel mit einem IonQ-Gerät:

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

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

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

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

Unterstützte Gates müssen möglicherweise zu nativen Gates kompiliert werden, bevor sie auf Quantenhardware laufen können. Wenn Sie eine Schaltung einreichen, führt Amazon Braket diese Kompilierung automatisch durch.

Beispiel: Rufen Sie programmgesteuert die Genauigkeit systemeigener Gates ab, die von einem Gerät unterstützt werden

Sie können die Genauigkeitsinformationen auf der Geräteseite der Braket-Konsole einsehen. Manchmal ist es hilfreich, programmgesteuert auf dieselben Informationen zuzugreifen. Der folgende Code zeigt, wie die qubit Zwei-Gate-Treue zwischen zwei Gates einer QPU extrahiert wird.

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

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

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

Programmsätze

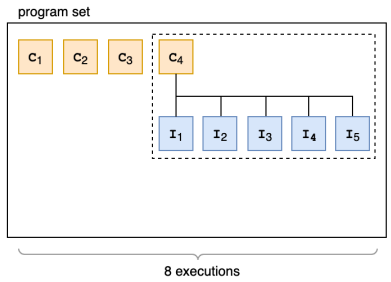
Programmsätze steuern effizient mehrere Quantenschaltkreise in einer einzigen Quantenaufgabe. In dieser einen Aufgabe können Sie bis zu 100 Quantenschaltungen oder einen einzelnen parametrischen Schaltkreis mit bis zu 100 verschiedenen Parametersätzen einreichen. Dieser Vorgang minimiert die Zeit zwischen aufeinanderfolgenden Schaltungsausführungen und reduziert den Verarbeitungsaufwand für Quantenaufgaben. Derzeit werden Programmpakete auf Amazon Braket Local Simulator und anderen AQT Rigetti Geräten unterstützt. IQM

Definition eines ProgramSet

Das folgende erste Codebeispiel zeigt, wie Sie eine erstellen, `ProgramSet` indem Sie sowohl parametrisierte Schaltungen als auch Schaltungen ohne Parameter verwenden.

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

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

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

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

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

Dieses Programmset enthält vier einzigartige Programme:`circ1`, `circ2circ3`, und. `circuit_binding` Das `circuit_binding` Programm wird mit fünf verschiedenen Parameterbindungen ausgeführt, wodurch fünf ausführbare Dateien erstellt werden. Die anderen drei parameterfreien Programme erstellen jeweils eine ausführbare Datei. Dies führt zu insgesamt acht ausführbaren Dateien, wie in der folgenden Abbildung dargestellt.



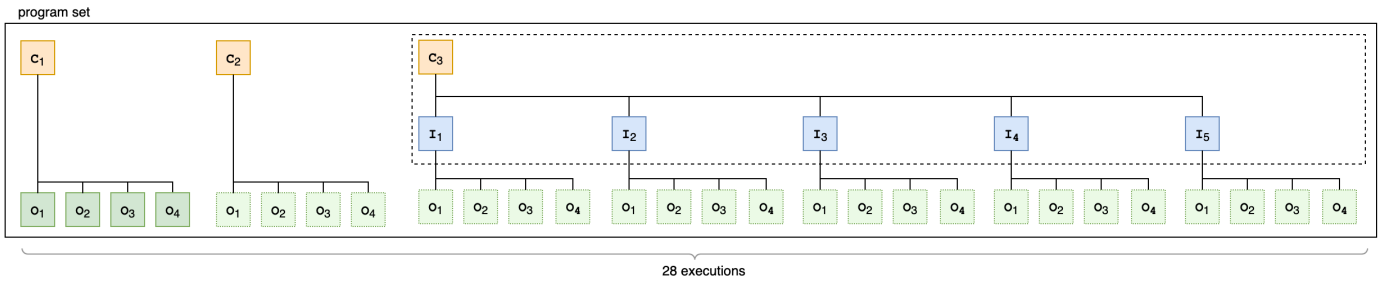

```
ProgramSet(
    [c1,
     c2,
     c3,
     CircuitBinding(c4, [i1, i2, i3, i4, i5])]
)
```

Das folgende zweite Codebeispiel zeigt, wie die `product()` Methode verwendet wird, um denselben Satz von Observablen an jede ausführbare Datei der Programmgruppe anzuhängen.

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

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

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

Bei parameterfreien Programmen wird jedes Observable für jeden Schaltkreis gemessen. Bei parametrischen Programmen wird jedes Observable für jeden Eingangssatz gemessen, wie in der folgenden Abbildung dargestellt.

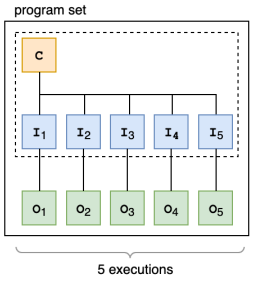
```
ProgramSet.product(
    circuits=[c1, c2, CircuitBinding(c3, [i1, i2, i3, i4, i5])],
    observables=[o1, o2, o3, o4]
)
```



Das folgende dritte Codebeispiel zeigt, wie die `zip()` Methode verwendet wird, um einzelne Observablen mit bestimmten Parametersätzen in der zu koppeln. `ProgramSet`

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

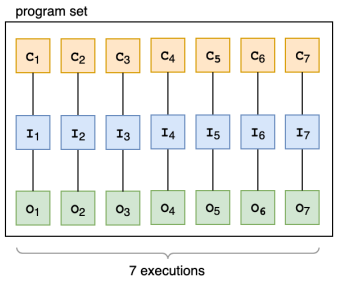
```
    observables=observables + [Y(0) @ Y(1)]
)
```




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

Stattdessen können Sie direkt eine Liste von `CircuitBinding()` Observablen mit einer Liste von Schaltungen und Eingabesätzen komprimieren.

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




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

Weitere Informationen und Beispiele zu Programmgruppen finden Sie im Ordner Program Set auf Github von amazon-braket-examples.

Untersuchen Sie ein Programmset auf einem Gerät und führen Sie es aus

Die Anzahl der ausführbaren Dateien eines Programmsatzes entspricht der Anzahl der eindeutigen parametergebundenen Schaltungen. Berechnen Sie die Gesamtzahl der ausführbaren Dateien und Shots von Schaltkreisen anhand des folgenden Codebeispiels.

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

```
shots = 10
num_executables = program_set_1.total_executables

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

> **Note**
>
> Bei Programmsätzen zahlen Sie eine einzige Gebühr pro Aufgabe und eine Gebühr pro Schuss, die auf der Gesamtzahl der Aufnahmen aller Schaltungen in einem Programmsatz basiert.

Verwenden Sie das folgende Codebeispiel, um den Programmsatz auszuführen.

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

Wenn Sie Rigetti Geräte verwenden, kann es sein, dass Ihr Programmsatz den `RUNNING` Status beibehält, während Aufgaben teilweise abgeschlossen und teilweise in die Warteschlange gestellt werden. Um schnellere Ergebnisse zu erzielen, sollten Sie erwägen, Ihr Programmset als Hybrid-Job einzureichen.

Ergebnisse analysieren

Führen Sie den folgenden Code aus, um die Ergebnisse der ausführbaren Dateien in a zu analysieren und zu messen. `ProgramSet`

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

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

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

Teilweise Messung

Anstatt alle Qubits in einem Quantenschaltkreis zu messen, verwenden Sie Teilmessungen, um einzelne Qubits oder eine Teilmenge von Qubits zu messen.

> **Note**
>
> Zusätzliche Funktionen wie Messungen in der Mitte des Schaltkreises und Feed-Forward-Operationen sind als experimentelle Funktionen verfügbar. Weitere Informationen finden Sie unter Zugriff auf dynamische Schaltungen auf IQM-Geräten.

Beispiel: Messen Sie eine Teilmenge von Qubits

Das folgende Codebeispiel zeigt eine Teilmessung, bei der nur Qubit 0 in einem Bell-State-Stromkreis gemessen wird.

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

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

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

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

# Get the results
result = task.result()

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

Manuell Qubit Zuweisung

Wenn Sie einen Quantenschaltkreis auf Quantencomputern von aus ausführenRigetti, können Sie optional die manuelle qubit Zuordnung verwenden, um zu steuern, welche für Ihren Algorithmus

verwendet qubits werden. Die Amazon Braket-Konsole und das Amazon Braket-SDK helfen Ihnen dabei, die neuesten Kalibrierungsdaten Ihres ausgewählten QPU-Geräts (Quantum Processing Unit) zu überprüfen, sodass Sie das Beste qubits für Ihr Experiment auswählen können.

Durch die manuelle qubit Zuordnung können Sie Schaltungen mit höherer Genauigkeit ausführen und einzelne Eigenschaften untersuchen. qubit Forscher und fortgeschrittene Anwender optimieren ihr Schaltungsdesign auf der Grundlage der neuesten Gerätekalibrierungsdaten und können genauere Ergebnisse erzielen.

Das folgende Beispiel zeigt, wie eine qubits explizite Zuordnung vorgenommen wird.

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

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

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

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

Weitere Informationen finden Sie in den Amazon Braket-Beispielen auf GitHub oder genauer gesagt in diesem Notizbuch: Zuweisen von Qubits auf QPU-Geräten.

Wörtliche Zusammenstellung

Wenn Sie eine Quantenschaltung auf Gate-basierten Quantencomputern ausführen, können Sie den Compiler anweisen, Ihre Schaltungen ohne Änderungen exakt so auszuführen, wie sie definiert sind. Mithilfe der wörtlichen Kompilierung können Sie entweder festlegen, dass ein ganzer Schaltkreis exakt wie spezifiziert erhalten bleibt oder dass nur bestimmte Teile davon erhalten bleiben (nur unterstützt von). Rigetti Bei der Entwicklung von Algorithmen für Hardware-Benchmarking- oder Fehlerminimierungsprotokolle müssen Sie die Möglichkeit haben, die Gates und Schaltkreislayouts, die auf der Hardware ausgeführt werden, genau zu spezifizieren. Die wörtliche Kompilierung gibt Ihnen direkte Kontrolle über den Kompilierungsprozess, indem Sie bestimmte Optimierungsschritte ausschalten und so sicherstellen, dass Ihre Schaltungen genau so laufen, wie sie entworfen wurden.

Die Verbatim-Kompilierung wird auf den Rigetti GerätenAQT, IonQIQM, und unterstützt und erfordert die Verwendung systemeigener Gatter. Bei der wörtlichen Kompilierung empfiehlt es sich, die

Topologie des Geräts zu überprüfen, um sicherzustellen, dass die Gates aufgerufen qubits und verbunden sind und dass die Schaltung die systemeigenen Gatter verwendet, die von der Hardware unterstützt werden. Das folgende Beispiel zeigt, wie Sie programmgesteuert auf die Liste der systemeigenen Gates zugreifen können, die von einem Gerät unterstützt werden.

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

Denn die Rigetti qubit Neuverkabelung muss durch die Einstellung `disableQubitRewiring=True` für die Verwendung mit wörtlicher Kompilierung ausgeschaltet werden. Wenn diese `disableQubitRewiring=False` Option gesetzt ist, wenn in einer Kompilierung wörtliche Boxen verwendet werden, schlägt der Quantenschaltkreis bei der Validierung fehl und kann nicht ausgeführt werden.

Wenn die wörtliche Kompilierung für eine Schaltung aktiviert ist und auf einer QPU ausgeführt wird, die sie nicht unterstützt, wird ein Fehler generiert, der darauf hinweist, dass ein nicht unterstützter Vorgang zum Fehlschlagen der Aufgabe geführt hat. Da immer mehr Quantenhardware Compilerfunktionen nativ unterstützt, wird diese Funktion um diese Geräte erweitert. Geräte, die die wörtliche Kompilierung unterstützen, schließen sie als unterstützten Vorgang ein, wenn sie mit dem folgenden Code abgefragt werden.

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

Mit der verbatim-Kompilierung sind keine zusätzlichen Kosten verbunden. Ihnen werden weiterhin Quantenaufgaben, die auf Braket QPU-Geräten, Notebook-Instances und On-Demand-Simulatoren ausgeführt werden, auf der Grundlage der aktuellen Tarife berechnet, die auf der Seite mit den Amazon Braket-Preisen angegeben sind. Weitere Informationen finden Sie im Beispiel-Notizbuch zur Verbatim-Kompilierung.

> **Note**
>
> Wenn Sie OpenQASM verwenden, um Ihre Schaltungen für die AQT IonQ AND-Geräte zu schreiben, und Sie Ihre Schaltung direkt den physikalischen Qubits zuordnen möchten, müssen Sie das verwenden, `#pragma braket verbatim` da das `disableQubitRewiring` Flag von OpenQASM ignoriert wird.

Simulation von Geräuschen

Um den lokalen Geräuschsimulator zu instanziieren, können Sie das Backend wie folgt ändern.

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

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

Sie können geräuschbehaftete Schaltungen auf zwei Arten aufbauen:

1. Baue den lauten Stromkreis von unten nach oben auf.
2. Nehmen Sie einen vorhandenen, rauschfreien Stromkreis und fügen Sie überall Rauschen ein.

Das folgende Beispiel zeigt die Ansätze, bei denen ein einfacher Schaltkreis mit depolarisierendem Rauschen und ein benutzerdefinierter Kraus-Kanal verwendet werden.

```
import scipy.stats
import numpy as np

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

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

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

```
from braket.circuits import Noise

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

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

Das Ausführen einer Schaltung bietet dieselbe Benutzererfahrung wie zuvor, wie in den folgenden beiden Beispielen gezeigt.

Beispiel 1

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

Oder

Beispiel 2

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

Weitere Beispiele finden Sie im einführenden Beispiel für einen Geräuschsimulator in Braket

Der Stromkreis wird überprüft

Quantenschaltungen in Amazon Braket haben ein Pseudozeitkonzept namens. `Moments` Jeder qubit kann pro ein einzelnes Gate erleben. `Moment` Der Zweck von `Moments` besteht darin, die Adressierung von Schaltkreisen und ihren Gates zu vereinfachen und eine zeitliche Struktur bereitzustellen.

> **Note**
>
> Die Zeitpunkte entsprechen im Allgemeinen nicht der Echtzeit, in der Gates auf einer QPU ausgeführt werden.

Die Tiefe einer Schaltung wird durch die Gesamtzahl der Momente in dieser Schaltung bestimmt. Sie können die Schaltkreistiefe anzeigen, indem Sie die Methode aufrufen, `circuit.depth` wie im folgenden Beispiel gezeigt.

```
from braket.circuits import Circuit

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

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

Die gesamte Schaltkreistiefe des obigen Schaltkreises beträgt 3 (dargestellt als Momente `0`1, und2). Sie können den Gate-Betrieb für jeden Moment überprüfen.

`Moments`funktioniert als Wörterbuch von Schlüssel-Wert-Paaren.

- Der Schlüssel ist`MomentsKey()`, der Pseudozeit und Informationen enthält. qubit
- Der Wert wird im Typ von zugewiesen. `Instructions()`

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

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

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

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

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

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

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

Sie können einem Schaltkreis auch Tore hinzufügen`Moments`.

```
from braket.circuits import Instruction, Gate

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

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

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

Liste der Ergebnistypen

Amazon Braket kann verschiedene Arten von Ergebnissen zurückgeben, wenn ein Stromkreis mit `ResultType` gemessen wird. Ein Stromkreis kann die folgenden Arten von Ergebnissen zurückgeben.

- `AdjointGradient`gibt den Gradienten (Vektorableitung) des Erwartungswerts einer angegebenen Observablen zurück. Diese Observable wirkt auf ein vorgegebenes Ziel in Bezug auf bestimmte Parameter, wobei die Methode der adjungierten Differenzierung verwendet wird. Sie können diese Methode nur verwenden, wenn shots=0 ist.
- `Amplitude`gibt die Amplitude der angegebenen Quantenzustände in der Ausgangswellenfunktion zurück. Sie ist nur auf den Simulatoren SV1 und vor Ort verfügbar.
- `Expectation`gibt den Erwartungswert einer bestimmten Observablen zurück, der mit der später in diesem Kapitel eingeführten `Observable` Klasse spezifiziert werden kann. Das zur Messung der beobachtbaren Größe qubits verwendete Ziel muss angegeben werden, und die Anzahl der angegebenen Ziele muss der Anzahl entsprechen, qubits auf die die beobachtbare Größe einwirkt. Wenn keine Ziele angegeben sind, darf die Observable nur mit 1 arbeiten qubit und sie wird auf alle qubits parallel.
- `Probability`gibt die Wahrscheinlichkeiten für die Messung von Zuständen auf rechnerischer Basis zurück. Wenn keine Ziele angegeben sind, wird die Wahrscheinlichkeit `Probability` zurückgegeben, mit der alle Basiszustände gemessen wurden. Wenn Ziele angegeben sind, werden nur die Randwahrscheinlichkeiten der Basisvektoren für die angegebenen qubits Werte zurückgegeben. Verwaltete Simulatoren und QPUs sind auf maximal 15 Qubits begrenzt, und lokale Simulatoren sind auf die Speichergröße des Systems beschränkt.
- `Reduced density matrix`gibt eine Dichtematrix für ein Subsystem mit einem angegebenen Ziel qubits aus einem System von zurück. qubits Um die Größe dieses Ergebnistyps zu begrenzen, begrenzt Braket die Anzahl der Ziele qubits auf maximal 8.
- `StateVector`gibt den vollständigen Zustandsvektor zurück. Er ist auf dem lokalen Simulator verfügbar.
- `Sample`gibt die Anzahl der Messungen eines bestimmten qubit Zielsatzes und eines beobachtbaren Zielwerts zurück. Wenn keine Ziele angegeben sind, darf die Observable nur mit 1 arbeiten qubit und sie wird auf alle qubits parallel. Wenn Ziele angegeben sind, muss die Anzahl der angegebenen Ziele der Anzahl entsprechen, qubits auf die die beobachtbare Größe einwirkt.
- `Variance`gibt die Varianz (`mean([x-mean(x)]`^{2}`)`) des angegebenen qubit Zielsatzes und Observable als angeforderten Ergebnistyp zurück. Wenn keine Ziele angegeben sind, darf die Observable nur mit 1 arbeiten qubit und sie wird auf alle qubits parallel. Andernfalls muss die Anzahl der angegebenen Ziele der Anzahl entsprechen, auf die qubits die beobachtbare Größe angewendet werden kann.

Die unterstützten Ergebnistypen für verschiedene Anbieter:

	Lokale SIM	SV1	DM1	TN1	AQT	IonQ	IQM	Rigetti
Adjunges Gefälle	N	Y	N	N	N	N	N	N
Amplitud	Y	Y	N	N	N	N	N	N
Erwartun	Y	Y	Y	Y	Y	Y	Y	Y
Probabili ty (Wahrscl inlichkei t)	Y	Y	Y	N	Y	Y	Y	Y
Matrix mit reduziert er Dichte	Y	N	Y	N	N	N	N	N
Zustands ektor	Y	N	N	N	N	N	N	N
Beispiel	Y	Y	Y	Y	Y	Y	Y	Y
Varianz	Y	Y	Y	Y	Y	Y	Y	Y

Sie können die unterstützten Ergebnistypen anhand der Geräteeigenschaften überprüfen, wie im folgenden Beispiel gezeigt.

```
from braket.aws import AwsDevice

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

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

```
    print(iter)
```

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

Um a aufzurufen`ResultType`, fügen Sie es an einen Schaltkreis an, wie im folgenden Beispiel gezeigt.

```
from braket.circuits import Circuit, Observable

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

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

> **Note**
>
> Verschiedene Quantengeräte liefern Ergebnisse in verschiedenen Formaten. RigettiGeräte geben beispielsweise Messwerte zurück, während IonQ Geräte Wahrscheinlichkeiten liefern. Das Amazon Braket SDK bietet eine Messeigenschaft für alle Ergebnisse. Bei Geräten, die Wahrscheinlichkeiten zurückgeben, werden diese Messungen jedoch nachberechnet und basieren auf den Wahrscheinlichkeiten, da keine Messungen pro Schuss verfügbar sind. Um festzustellen, ob ein Ergebnis nachberechnet wurde, überprüfen Sie den Wert auf dem Ergebnisobjekt. `measurements_copied_from_device` Dieser Vorgang ist in der Datei gate_model_quantum_task_result.py im Amazon Braket GitHub SDK-Repository detailliert beschrieben.

Observablen

Mit der `Observable` Klasse von Amazon Braket können Sie eine bestimmte beobachtbare Größe messen.

Sie können jeweils nur ein eindeutiges beobachtbares Objekt ohne Identität zuweisen. qubit Ein Fehler tritt auf, wenn Sie zwei oder mehr verschiedene Observablen ohne Identität für dasselbe angeben. qubit Zu diesem Zweck zählt jeder Faktor eines Tensorprodukts als einzelne beobachtbare Größe. Das bedeutet, dass Sie mehrere Tensorprodukte auf demselben System haben könnenqubit, solange die Faktoren, die darauf einwirken, gleich qubit bleiben.

Eine Observable kann skaliert werden und weitere Observablen (skaliert oder nicht) hinzugefügt werden. Dadurch entsteht eine`Sum`, die im Ergebnistyp verwendet werden kann. `AdjointGradient`

Die `Observable` Klasse umfasst die folgenden Observablen.

```
import numpy as np

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

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

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

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

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

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

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

Parameters

Schaltungen können freie Parameter enthalten. Diese freien Parameter müssen nur einmal erstellt werden, um sie mehrfach auszuführen, und können zur Berechnung von Gradienten verwendet werden.

Jeder freie Parameter verwendet einen in einer Zeichenfolge codierten Namen, der für folgende Zwecke verwendet wird:

- Parameterwerte festlegen
- Identifizieren Sie, welche Parameter verwendet werden sollen

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

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

Adjungierter Gradient

Das SV1 Gerät berechnet den adjungierten Gradienten eines beobachtbaren Erwartungswerts, einschließlich eines Hamilton-Werts mit mehreren Termen. Geben Sie zur Unterscheidung von Parametern ihren Namen (im Zeichenkettenformat) oder als direkte Referenz an.

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

device = AwsDevice(Devices.Amazon.SV1)

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

Wenn feste Parameterwerte als Argumente an einen parametrisierten Schaltkreis übergeben werden, werden die freien Parameter entfernt. Das Ausführen dieser Schaltung mit `AdjointGradient` erzeugt einen Fehler, da die freien Parameter nicht mehr existieren. Das folgende Codebeispiel zeigt die korrekte und falsche Verwendung:

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

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

Expertenrat einholen

Connect direkt in der Braket-Managementkonsole mit Experten für Quantencomputer in Verbindung, um weitere Informationen zu Ihren Workloads zu erhalten.

Um die Beratungsmöglichkeiten von Experten über Braket Direct zu erkunden, öffnen Sie die Braket-Konsole, wählen Sie im linken Bereich Braket Direct und navigieren Sie zum Abschnitt Expertentipps. Die folgenden Optionen für Expertenratschläge sind verfügbar:

- Sprechstunden in Braket: Die Sprechstunden in Braket sind Einzelsitzungen, also wer zuerst kommt, mahlt zuerst. Sie finden jeden Monat statt. Jede verfügbare Sprechstunde dauert 30 Minuten und ist kostenlos. Gespräche mit Braket-Experten können Ihnen helfen, schneller von der Idee zur Ausführung zu gelangen, indem Sie die Eignung von Anwendungsfall zu Gerät untersuchen, Optionen identifizieren, um Braket am besten für Ihren Algorithmus zu verwenden,

und Empfehlungen für die Verwendung bestimmter Braket-Funktionen wie Amazon Braket Hybrid Jobs, Braket Pulse oder Analog Hamiltonian Simulation erhalten.

- Um sich für die Sprechstunden von Braket anzumelden, wählen Sie Anmelden aus und geben Sie Kontaktinformationen, Workload-Details und Ihre gewünschten Diskussionsthemen ein.
- Sie erhalten per E-Mail eine Kalendereinladung zum nächsten verfügbaren Termin.

> **Note**
>
> Bei aufkommenden Problemen oder Fragen zur schnellen Fehlerbehebung empfehlen wir, sich an den AWS Support zu wenden. Für nicht dringende Fragen können Sie auch das AWS re:POST-Forum oder den Quantum Computing Stack Exchange nutzen, wo Sie zuvor beantwortete Fragen durchsuchen und neue stellen können.

- Angebote von Quanten-Hardwareanbietern:IonQ,QuEra, und Rigetti beide bieten professionelle Serviceangebote über. AWS Marketplace
  - Um ihre Angebote zu erkunden, wählen Sie Connect aus und stöbern Sie in ihren Angeboten.
  - Weitere Informationen zu den professionellen Dienstleistungsangeboten auf der AWS Marketplace finden Sie unter Produkte für professionelle Dienstleistungen.
- AmazonAdvanced Solutions Lab (ASL): Das ASL ist ein kollaboratives Forschungs- und Serviceteam mit Experten für Quantencomputer, die Ihnen helfen können, Quantencomputer effektiv zu erforschen und die aktuelle Leistungsfähigkeit dieser Technologie zu beurteilen.
  - Um die ASL zu kontaktieren, wählen Sie Connect aus und geben Sie Kontaktinformationen und Anwendungsfalldetails ein.
  - Das ASL-Team wird Sie per E-Mail mit den nächsten Schritten kontaktieren.

Betreiben Sie Ihre Schaltungen mit OpenQASM 3.0

AmazonBraket unterstützt jetzt OpenQASM 3.0 für Gate-basierte Quantengeräte und Simulatoren. Dieses Benutzerhandbuch enthält Informationen über die Teilmenge von OpenQASM 3.0, die von Braket unterstützt wird. Braket-Kunden haben jetzt die Wahl, Braket-Schaltungen mit dem SDK einzureichen oder OpenQASM 3.0-Strings direkt an alle Gate-basierten Geräte mit der Amazon Braket-API und dem Amazon Braket Python SDK bereitzustellen.

Die Themen in diesem Leitfaden führen Sie durch verschiedene Beispiele, wie Sie die folgenden Quantenaufgaben erledigen können.

- Erstellen und senden Sie OpenQASM-Quantenaufgaben auf verschiedenen Braket-Geräten
- Greifen Sie auf die unterstützten Operationen und Ergebnistypen zu
- Simulieren Sie Geräusche mit OpenQASM
- Verwenden Sie die wörtliche Kompilierung mit OpenQASM
- Beheben Sie OpenQASM-Probleme

Dieses Handbuch bietet auch eine Einführung in bestimmte hardwarespezifische Funktionen, die mit OpenQASM 3.0 auf Braket implementiert werden können, sowie Links zu weiteren Ressourcen.

In diesem Abschnitt:

- Was ist OpenQASM 3.0?
- Wann sollte OpenQASM 3.0 verwendet werden
- Wie funktioniert OpenQASM 3.0
- Voraussetzungen
- Welche OpenQASM-Funktionen unterstützt Braket?
- Erstellen Sie eine OpenQASM 3.0-Beispiel-Quantenaufgabe und reichen Sie sie ein
- Support für OpenQASM auf verschiedenen Braket-Geräten
- Simulieren Sie Geräusche mit OpenQASM 3.0
- Qubit Neuverkabelung mit OpenQASM 3.0
- Wörtliche Kompilierung mit OpenQASM 3.0
- Die Braket-Konsole
- Weitere Ressourcen
- Berechnung von Gradienten mit OpenQASM 3.0
- Messung bestimmter Qubits mit OpenQASM 3.0

Was ist OpenQASM 3.0?

Die Open Quantum Assembly Language (OpenQASM) ist eine Zwischendarstellung für Quantenbefehle. OpenQASM ist ein Open-Source-Framework und wird häufig für die Spezifikation von Quantenprogrammen für Gate-basierte Geräte verwendet. Mit OpenQASM können Benutzer die Quantengatter und Messoperationen programmieren, die die Bausteine der Quantenberechnung bilden. Die vorherige Version von OpenQASM (2.0) wurde von einer Reihe von Bibliotheken zur Quantenprogrammierung verwendet, um grundlegende Programme zu beschreiben.

Die neue Version von OpenQASM (3.0) erweitert die vorherige Version um weitere Funktionen wie Pulssteuerung, Gate-Timing und klassischen Kontrollfluss, um die Lücke zwischen der Endbenutzeroberfläche und der Hardwarebeschreibungssprache zu schließen. Einzelheiten und Spezifikationen zur aktuellen Version 3.0 sind in der OpenQASM 3.x Live Specification verfügbar. GitHub Die future Entwicklung von OpenQASM wird vom OpenQASM 3.0 Technical Steering Committee gesteuert, dem AWS neben IBM, Microsoft und der Universität Innsbruck auch angehört.

Wann sollte OpenQASM 3.0 verwendet werden

OpenQASM bietet ein ausdrucksstarkes Framework zur Spezifizierung von Quantenprogrammen durch einfache Steuerungen, die nicht architekturspezifisch sind, und eignet sich daher gut für die Darstellung mehrerer Gate-basierter Geräte. Die Braket-Unterstützung für OpenQASM fördert seine Akzeptanz als konsistenten Ansatz für die Entwicklung von Gate-basierten Quantenalgorithmen und reduziert so die Notwendigkeit für Benutzer, Bibliotheken in mehreren Frameworks zu erlernen und zu verwalten.

Wenn Sie bereits über Programmbibliotheken in OpenQASM 3.0 verfügen, können Sie diese für die Verwendung mit Braket anpassen, anstatt diese Schaltungen komplett neu zu schreiben. Forscher und Entwickler sollten auch von einer zunehmenden Anzahl verfügbarer Bibliotheken von Drittanbietern profitieren, die die Algorithmusentwicklung in OpenQASM unterstützen.

Wie funktioniert OpenQASM 3.0

Die Support von OpenQASM 3.0 von Braket bietet Funktionsparität mit der aktuellen Intermediate Representation. Das bedeutet, dass Sie alles, was Sie heute mit Braket auf Hardwaregeräten und On-Demand-Simulatoren tun können, auch mit OpenQASM und Braket tun können. API Sie können OpenQASM 3.0-Programme ausführen, indem Sie allen Gate-basierten Geräten OpenQASM-Strings direkt zur Verfügung stellen, ähnlich der Art und Weise, wie derzeit Schaltungen für Geräte auf Braket bereitgestellt werden. Braket-Benutzer können auch Bibliotheken von Drittanbietern integrieren, die OpenQASM 3.0 unterstützen. Der Rest dieses Handbuchs beschreibt, wie OpenQASM-Repräsentationen für die Verwendung mit Braket entwickelt werden.

Voraussetzungen

Um OpenQASM 3.0 auf Amazon Braket verwenden zu können, benötigen Sie Version v1.8.0 der Amazon Braket Python Schemas und Version v1.17.0 oder höher des Amazon Braket Python SDK.

Wenn Sie Amazon Braket zum ersten Mal verwenden, müssen Sie Amazon Braket aktivieren. Anweisungen finden Sie unter Amazon Braket aktivieren.

Welche OpenQASM-Funktionen unterstützt Braket?

Der folgende Abschnitt listet die OpenQASM 3.0-Datentypen, Anweisungen und Pragma-Anweisungen auf, die von Braket unterstützt werden.

In diesem Abschnitt:

- Unterstützte OpenQASM-Datentypen
- Unterstützte OpenQASM-Anweisungen
- Braket OpenQASM Pragmas
- Erweiterte Funktionsunterstützung für OpenQASM auf dem Local Simulator
- Unterstützte Operationen und Grammatik mit OpenPulse

Unterstützte OpenQASM-Datentypen

Die folgenden OpenQASM-Datentypen werden von Amazon Braket unterstützt.

- Non-negative Ganzzahlen werden für (virtuelle und physische) Qubit-Indizes verwendet:
  - `cnot q[0], q[1];`
  - `h $0;`
- Floating-point Zahlen oder Konstanten können für Tordrehwinkel verwendet werden:
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi ist eine eingebaute Konstante in OpenQASM und kann nicht als Parametername verwendet werden.

- Arrays komplexer Zahlen (mit der `im` OpenQASM-Notation für den Imaginärteil) sind in Ergebnistyp-Pragmas zur Definition allgemeiner hermitescher Observablen und in unitären Pragmas zulässig:
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

Unterstützte OpenQASM-Anweisungen

Die folgenden OpenQASM-Anweisungen werden von Amazon Braket unterstützt.

- `Header: OPENQASM 3;`
- Klassische Bit-Deklarationen:
  - `bit b1;`(äquivalent,) `creg b1;`
  - `bit[10] b2;`(gleichwertig,) `creg b2[10];`
- Qubit-Deklarationen:
  - `qubit b1;`(äquivalent,) `qreg b1;`
  - `qubit[10] b2;`(gleichwertig,) `qreg b2[10];`
- Indizierung innerhalb von Arrays: `q[0]`
- Eingabe: `input float alpha;`
- physikalische Spezifikationqubits: `$0`
- Unterstützte Tore und Operationen auf einem Gerät:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> Die unterstützten Gates eines Geräts finden Sie in den Geräteeigenschaften für OpenQASM-Aktionen. Für die Verwendung dieser Gates sind keine Gate-Definitionen erforderlich.

- Wörtliche Angaben in der Box. Derzeit unterstützen wir die Notation mit der Dauer von Boxen nicht. Bei wörtlichen Boxen qubits sind systemeigene Gates und physische Elemente erforderlich.

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

- Messung und Messzuweisung in einem qubits oder einem ganzen qubit Register.
  - `measure $0;`

  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`
- Barrierenanweisungen ermöglichen eine explizite Kontrolle über die Zusammenstellung und Ausführung von Schaltungen, indem sie eine Neuanordnung von Gates und Optimierungen über Barrierengrenzen hinweg verhindern. Sie setzen auch eine strikte zeitliche Reihenfolge bei der Ausführung durch und stellen sicher, dass alle Operationen vor einer Barriere abgeschlossen sind, bevor nachfolgende Operationen beginnen.
  - `barrier;`
  - `barrier q[0], q[1];`
  - `barrier $3, $6;`

Braket OpenQASM Pragmas

Die folgenden OpenQASM-Pragma-Anweisungen werden von Amazon Braket unterstützt.

- Noise-Pragmas
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- Wörtliche Pragmas
  - `#pragma braket verbatim`
- Pragmas vom Typ Ergebnis
  - Basisinvariante Ergebnistypen:
    - Zustandsvektor: `#pragma braket result state_vector`
    - Dichtematrix: `#pragma braket result density_matrix`
  - Pragmas zur Gradientenberechnung:
    - Adjungierter Gradient: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Z-Basis-Ergebnistypen:
    - Amplitude: `#pragma braket result amplitude "01"`

  - Wahrscheinlichkeit: `#pragma braket result probability q[0], q[1]`
- Auf Basis rotierte Ergebnistypen
  - Erwartung: `#pragma braket result expectation x(q[0]) @ y([q1])`
  - Varianz: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
  - Stichprobe: `#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0 ist abwärtskompatibel mit OpenQASM 2.0, sodass Programme, die mit 2.0 geschrieben wurden, auf Braket laufen können. Die von Braket unterstützten Funktionen von OpenQASM 3.0 weisen jedoch einige geringfügige Syntaxunterschiede auf, wie zum Beispiel vs und vs `qreg` `creg` `qubit` `bit` Es gibt auch Unterschiede in der Messsyntax, und diese müssen mit ihrer korrekten Syntax unterstützt werden.

Erweiterte Funktionsunterstützung für OpenQASM auf dem Local Simulator

Das `LocalSimulator` unterstützt erweiterte OpenQASM-Funktionen, die nicht als Teil der QPUs oder On-Demand-Simulatoren von Braket angeboten werden. Die folgende Liste von Funktionen wird nur in folgenden Versionen unterstützt: `LocalSimulator`

- Gate-Modifikatoren
- Integrierte OpenQASM-Gates
- Klassische Variablen
- Klassische Operationen
- Maßgeschneiderte Tore
- Klassische Steuerung
- QASM-Dateien
- Subroutinen

Beispiele für jede erweiterte Funktion finden Sie in diesem Beispielnotizbuch. Die vollständige OpenQASM-Spezifikation finden Sie auf der OpenQASM-Website.

Unterstützte Operationen und Grammatik mit OpenPulse

Unterstützte OpenPulse Datentypen

Blöcke aufrufen:

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

Aufkleberblöcke:

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

Rahmen:

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

Wellenformen:

```
// 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};
```

Beispiel für eine benutzerdefinierte Gate-Kalibrierung:

```
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;
```

Beispiel für einen beliebigen Impuls:

```
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);
```

```
}
```

Erstellen Sie eine OpenQASM 3.0-Beispiel-Quantenaufgabe und reichen Sie sie ein

Sie können das Amazon Braket Python SDK, Boto3 oder das verwenden, um OpenQASM 3.0-Quantenaufgaben AWS CLI an ein Amazon Braket-Gerät zu senden.

In diesem Abschnitt:

- Ein Beispiel für ein OpenQASM 3.0-Programm
- Verwenden Sie das Python-SDK, um OpenQASM 3.0-Quantenaufgaben zu erstellen
- Verwenden Sie Boto3, um OpenQASM 3.0-Quantenaufgaben zu erstellen
- Verwenden Sie den AWS CLI um OpenQASM 3.0-Aufgaben zu erstellen

Ein Beispiel für ein OpenQASM 3.0-Programm

Um eine OpenQASM 3.0-Aufgabe zu erstellen, können Sie mit einem OpenQASM 3.0-Grundprogramm (ghz.qasm) beginnen, das einen GHZ-Status vorbereitet, wie im folgenden Beispiel gezeigt.

```
// 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;
```

Verwenden Sie das Python-SDK, um OpenQASM 3.0-Quantenaufgaben zu erstellen

Sie können das Amazon Braket Python SDK verwenden, um dieses Programm mit dem folgenden Code an ein Amazon Braket-Gerät zu senden. Achten Sie darauf, den Amazon S3 S3-Bucket-

Beispielstandort „amzn-s3-demo-bucket“ durch Ihren eigenen Amazon S3 S3-Bucket-Namen zu ersetzen.

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

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

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

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

Verwenden Sie Boto3, um OpenQASM 3.0-Quantenaufgaben zu erstellen

Sie können auch das AWS Python SDK for Braket (Boto3) verwenden, um die Quantenaufgaben mithilfe von OpenQASM 3.0-Strings zu erstellen, wie im folgenden Beispiel gezeigt. Der folgende Codeausschnitt verweist auf ghz.qasm, das einen GHZ-Status wie oben gezeigt vorbereitet.

```
import boto3
import json

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

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

action = {
    "braketSchemaHeader": {
        "name": "braket.ir.openqasm.program",
        "version": "1"
```

```
    },
    "source": source
}
device_parameters = {}
device_arn = "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3"
shots = 100

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

Verwenden Sie den AWS CLI um OpenQASM 3.0-Aufgaben zu erstellen

Die AWS Command Line Interface (CLI) kann auch verwendet werden, um OpenQASM 3.0-Programme einzureichen, wie im folgenden Beispiel gezeigt.

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

Support für OpenQASM auf verschiedenen Braket-Geräten

Bei Geräten, die OpenQASM 3.0 unterstützen, unterstützt das `action` Feld eine neue Aktion über die `GetDevice` Antwort, wie im folgenden Beispiel für die Geräte und gezeigt. Rigetti IonQ

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

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

Bei Geräten, die die Impulssteuerung unterstützen, wird das `pulse` Feld in der `GetDevice` Antwort angezeigt. Das folgende Beispiel zeigt dieses `pulse` Feld für das Rigetti Gerät.

```
// 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
    }
  }
}
```

In den vorherigen Feldern wird Folgendes detailliert beschrieben:

Anschlüsse:

Beschreibt vorgefertigte externe (`extern`) Geräteanschlüsse, die auf der QPU deklariert sind, zusätzlich zu den zugehörigen Eigenschaften des angegebenen Anschlusses. Alle in dieser Struktur aufgelisteten Ports sind als gültige Bezeichner innerhalb des vom Benutzer übermittelten `OpenQASM 3.0` Programms vordeklariert. Zu den zusätzlichen Eigenschaften eines Ports gehören:

- Port-ID (PortID)
  - Der Portname, der in OpenQASM 3.0 als Identifier deklariert wurde.
- Richtung (Richtung)
  - Die Richtung des Hafens. Antriebsanschlüsse übertragen Impulse (Richtung „tx“), während Messanschlüsse Impulse empfangen (Richtung „rx“).
- Porttyp (PortType)
  - Der Aktionstyp, für den dieser Port verantwortlich ist (z. B. drive, capture oder ff — fast-flux).
- Dt (dt)
  - Die Zeit in Sekunden, die einen einzelnen Sample-Zeitschritt auf dem angegebenen Port darstellt.
- Qubit-Zuordnungen (QubitMappings)
  - Die Qubits, die dem angegebenen Port zugeordnet sind.
- Mittenfrequenzen (CenterFrequencies)
  - Eine Liste der zugehörigen Mittenfrequenzen für alle vordeklarierten oder benutzerdefinierten Frames am Port. Weitere Informationen finden Sie unter Frames.
- QHP-spezifische Eigenschaften (SpecificPropertiesqhp)
  - Eine optionale Karte, in der die vorhandenen Eigenschaften des für das QHP spezifischen Ports detailliert beschrieben werden.

Rahmen:

Beschreibt vorgefertigte externe Frames, die auf der QPU deklariert sind, sowie die zugehörigen Eigenschaften der Frames. Alle in dieser Struktur aufgelisteten Frames sind innerhalb des vom Benutzer eingereichten `OpenQASM 3.0` Programms als gültige Bezeichner vordeklariert. Zu den zusätzlichen Eigenschaften eines Frames gehören:

- Rahmen-ID (FrameID)
  - Der Frame-Name, der in OpenQASM 3.0 als Identifier deklariert wurde.
- Port-ID (Port-ID)

  - Der zugehörige Hardwareport für den Frame.
- Frequenz (Frequenz)
  - Die standardmäßige Anfangsfrequenz des Frames.
- Mittenfrequenz (CenterFrequency)
  - Die Mitte der Frequenzbandbreite für den Frame. Normalerweise können Frames nur auf eine bestimmte Bandbreite im Bereich der Mittenfrequenz eingestellt werden. Daher sollten Frequenzanpassungen innerhalb eines bestimmten Deltas der Mittenfrequenz bleiben. Sie finden den Bandbreitenwert in den Validierungsparametern.
- Phase (Phase)
  - Die standardmäßige Anfangsphase des Frames.
- Assoziiertes Tor (AssociatedGate)
  - Die Gates, die dem angegebenen Frame zugeordnet sind.
- Qubit-Mappings (QubitMappings)
  - Die Qubits, die dem angegebenen Frame zugeordnet sind.
- QHP-spezifische Eigenschaften (QHP) SpecificProperties
  - Eine optionale Karte, in der die vorhandenen Eigenschaften des Frames detailliert beschrieben werden, die für das QHP spezifisch sind.

SupportsDynamicFrames:

Beschreibt, ob ein Frame in der Funktion `cal` oder in `defcal` Blöcken deklariert werden kann. OpenPulse `newframe` Wenn dies falsch ist, dürfen nur die in der Frame-Struktur aufgelisteten Frames innerhalb des Programms verwendet werden.

SupportedFunctions:

Beschreibt die OpenPulse Funktionen, die für das Gerät unterstützt werden, zusätzlich zu den zugehörigen Argumenten, Argumenttypen und Rückgabetypen für die angegebenen Funktionen. Beispiele für die Verwendung der OpenPulse Funktionen finden Sie in der OpenPulseSpezifikation. Derzeit unterstützt Braket:

- shift_phase
  - Verschiebt die Phase eines Frames um einen bestimmten Wert
- set_phase

  - Setzt die Phase des Frames auf den angegebenen Wert
- swap_phases
  - Tauscht die Phasen zwischen zwei Frames aus.
- shift_frequency
  - Verschiebt die Frequenz eines Frames um einen bestimmten Wert
- set_frequency
  - Setzt die Frequenz des Frames auf den angegebenen Wert
- spielen
  - Plant eine Wellenform
- capture_v0
  - Gibt den Wert eines Capture-Frames in ein Bitregister zurück

SupportedQhpTemplateWaveforms:

Beschreibt die auf dem Gerät verfügbaren vorgefertigten Wellenformfunktionen sowie die zugehörigen Argumente und Typen. Standardmäßig bietet Braket Pulse auf allen Geräten vorgefertigte Wellenformroutinen. Diese sind:

Konstant

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

`τ`ist die Länge der Wellenform und `iq` ist eine komplexe Zahl.

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

Gaußsch

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

`τ`ist die Länge der Wellenform, ist die Breite der Gaußkurve und σ ist die Amplitude. `A` Bei Einstellung `ZaE` auf `True` wird der Gauß-Wert verschoben und neu skaliert, sodass er am Anfang und Ende der Wellenform gleich Null ist und das Maximum erreicht. `A`

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

ZIEHEN SIE Gaussian

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

τist die Länge der Wellenform, ist die Breite der Gaußkurve, σ ist ein freier Parameter und ist die Amplitude. β A Bei Einstellung ZaE auf True wird die Gaußsche Methode zur Entfernung von Ableitungen durch Adiabatic Gate (DRAG) verschoben und neu skaliert, sodass sie am Anfang und Ende der Wellenform gleich Null ist und der Realteil das Maximum erreicht. A Weitere Informationen zur DRAG-Wellenform finden Sie im paper Simple Pulses for Elimination of Leakage in Weakly Nonlinear Qubits.

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

Erf-Quadrat

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

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

Wo L ist die Länge, W ist die Breite der Wellenform, σ definiert, wie schnell die Kanten steigen und fallen, und, ist die Amplitude. t_1=(L-W)/2 $t_2$2=(L+W)/2 A Bei Einstellung ZaE auf True wird der Gauß-Wert verschoben und neu skaliert, sodass er am Anfang und Ende der Wellenform gleich Null ist und das Maximum erreicht. A Die folgende Gleichung ist die neu skalierte Version der Wellenform.

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

Wo und. a=erf(W/2σ) b=erf(-t_1/σ)/2+erf(t $_2$/σ)/2

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

SupportsLocalPulseElements:

Beschreibt, ob Impulselemente wie Ports, Frames und Wellenformen lokal in `defcal` Blöcken definiert werden können. Wenn der Wert ist`false`, müssen Elemente in `cal` Blöcken definiert werden.

SupportsNonNativeGatesWithPulses:

Beschreibt, ob wir nicht systemeigene Gatter in Kombination mit Impulsprogrammen verwenden können oder nicht. Zum Beispiel können Sie ein nicht systemeigenes Gate wie ein `H` Gate in einem Programm nicht verwenden, ohne zuerst das Gate Through `defcal` für das verwendete Qubit zu definieren. Die Liste der systemeigenen `nativeGateSet` Gates-Schlüssel finden Sie unter den Gerätefunktionen.

ValidationParameters:

Beschreibt die Grenzen der Validierung von Impulselementen, einschließlich:

- Werte für maximale Skala und maximale Amplitude für Wellenformen (willkürlich und vordefiniert)
- Maximale Frequenzbandbreite ausgehend von der eingegebenen Mittenfrequenz in Hz
- Minimaler Impuls length/duration in Sekunden
- Maximaler Puls length/duration in Sekunden

Unterstützte Operationen, Ergebnisse und Ergebnistypen mit OpenQASM

Um herauszufinden, welche OpenQASM 3.0-Funktionen jedes Gerät unterstützt, können Sie dem `braket.ir.openqasm.program` Schlüssel im `action` Feld auf der Ausgabe der Gerätefunktionen entnehmen. Im Folgenden sind beispielsweise die unterstützten Operationen und Ergebnistypen aufgeführt, die für den Braket State Vector-Simulator verfügbar sind. 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
        }
      ]
    }
  },
...
```

Simulieren Sie Geräusche mit OpenQASM 3.0

Um Rauschen mit OpenQASM3 zu simulieren, verwenden Sie Pragma-Befehle, um Rauschoperatoren hinzuzufügen. Um beispielsweise die zuvor bereitgestellte Version des GHZ-Programms zu simulieren, können Sie das folgende OpenQASM-Programm einreichen.

```
// 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;
```

Die Spezifikationen für alle unterstützten Pragma-Noise-Operatoren finden Sie in der folgenden Liste.

```
#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
```

Kraus-Operator

Um einen Kraus-Operator zu generieren, können Sie durch eine Liste von Matrizen iterieren und jedes Element der Matrix als komplexen Ausdruck drucken.

Beachten Sie bei der Verwendung von Kraus-Operatoren Folgendes:

- Die Anzahl von qubits darf 2 nicht überschreiten. Die aktuelle Definition in den Schemas legt diesen Grenzwert fest.
- Die Länge der Argumentliste muss ein Vielfaches von 8 sein. Das bedeutet, dass sie nur aus 2x2-Matrizen bestehen darf.
- Die Gesamtlänge überschreitet nicht 2 $^{2*\text{num\_qubits-Matrizen}}$. Das bedeutet 4 Matrizen für 1 und 16 für 2. qubit qubits
- Bei allen mitgelieferten Matrizen handelt es sich um eine vollständig positive Spurensicherung (CPTP).
- Das Produkt der Kraus-Operatoren mit ihren transponierten Konjugaten muss sich zu einer Identitätsmatrix addieren.

Qubit Neuverkabelung mit OpenQASM 3.0

Amazon Braket unterstützt die physische qubit Notation innerhalb von OpenQASM auf Rigetti Geräten (weitere Informationen finden Sie auf dieser Seite). Wenn Sie physische Geräte qubits zusammen mit der Naive-Rewiring-Strategie verwenden, stellen Sie sicher, dass sie am ausgewählten Gerät angeschlossen qubits sind. Wenn stattdessen qubit Register verwendet werden, ist alternativ die PARTIELLE Neuverkabelungsstrategie auf Geräten standardmäßig aktiviert. Rigetti

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

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

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

Wörtliche Kompilierung mit OpenQASM 3.0

Wenn Sie eine Quantenschaltung auf Quantencomputern von Anbietern wieRigetti, und betreiben, können Sie den Compiler anweisenIonQ, Ihre Schaltungen exakt wie definiert und ohne Änderungen auszuführen. Diese Funktion wird als wörtliche Kompilierung bezeichnet. Mit Rigetti-Geräten können Sie genau festlegen, was erhalten bleibt — entweder ein ganzer Schaltkreis oder nur bestimmte Teile davon. Um nur bestimmte Teile eines Schaltkreises beizubehalten, müssen Sie native Gates innerhalb der geschützten Bereiche verwenden. Derzeit wird IonQ nur die wörtliche Kompilierung für den gesamten Schaltkreis unterstützt, sodass jede Anweisung in der Schaltung in einer wörtlichen Box enthalten sein muss.

Mit OpenQASM können Sie explizit ein wörtliches Pragma für eine Codebox angeben, die dann unangetastet bleibt und nicht durch die Low-Level-Kompilierungsroutine der Hardware optimiert wird. Das folgende Codebeispiel zeigt, wie Sie die Direktive verwenden können, um dies zu erreichen. `#pragma braket verbatim`

```
OPENQASM 3;

bit[2] c;
```

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

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

Ausführlichere Informationen zum Prozess der wörtlichen Kompilierung, einschließlich Beispielen und bewährten Methoden, finden Sie im Beispielnotizbuch zur Verbatim-Kompilierung, das im Github-Repository Amazon-Braket-Examples verfügbar ist.

Die Braket-Konsole

OpenQASM 3.0-Aufgaben sind verfügbar und können in der Amazon Braket-Konsole verwaltet werden. Auf der Konsole haben Sie die gleiche Erfahrung mit dem Einreichen von Quantenaufgaben in OpenQASM 3.0 gemacht wie beim Einreichen vorhandener Quantenaufgaben.

Weitere Ressourcen

OpenQASM ist in allen Amazon Braket-Regionen verfügbar.

Ein Beispiel-Notizbuch für die ersten Schritte mit OpenQASM auf Amazon Braket finden Sie unter Braket-Tutorials. GitHub

Berechnung von Gradienten mit OpenQASM 3.0

Amazon Braket unterstützt die Berechnung von Gradienten sowohl auf On-Demand-Simulatoren als auch auf lokalen Simulatoren, wenn sie im `shots=0` (exakten) Modus ausgeführt werden. Dies wird durch die Verwendung der Methode der adjungierten Differenzierung erreicht. Um den Gradienten anzugeben, den Sie berechnen möchten, können Sie das entsprechende Pragma angeben, wie der Code im folgenden Beispiel zeigt.

```
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
```

Anstatt alle einzelnen Parameter explizit aufzulisten, können Sie das `all` Schlüsselwort auch innerhalb des Pragmas angeben. Dadurch wird der Gradient in Bezug auf alle aufgelisteten `input` Parameter berechnet. Dies kann eine praktische Option sein, wenn die Anzahl der Parameter sehr groß ist. In diesem Fall sieht das Pragma wie der Code im folgenden Beispiel aus.

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

Alle beobachtbaren Typen werden in der OpenQASM 3.0-Implementierung von Amazon Braket unterstützt, einschließlich einzelner Operatoren, Tensorprodukte, Hermitian-Observablen und Observablen. `Sum` Der spezifische Operator, den Sie bei der Berechnung von Gradienten verwenden möchten, muss innerhalb der `expectation()` Funktion enthalten sein, und die Qubits, auf die jeder Term der Observablen einwirkt, müssen explizit angegeben werden.

Messung bestimmter Qubits mit OpenQASM 3.0

Der von Amazon Braket bereitgestellte Local State Vector Simulator und der Local Density Matrix Simulator unterstützen die Einreichung von OpenQASM Programmen, mit denen eine Teilmenge der Qubits der Schaltung selektiv gemessen werden kann. Diese Fähigkeit, die oft als Teilmessung bezeichnet wird, ermöglicht gezieltere und effizientere Quantenberechnungen. Im folgenden Codeausschnitt können Sie beispielsweise einen Zwei-Qubit-Schaltkreis erstellen und festlegen, dass nur das erste Qubit gemessen wird, während das zweite Qubit nicht gemessen wird.

```
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];
"""
```

In diesem Beispiel haben wir einen Quantenschaltkreis mit zwei Qubits, `q[0]` und, aber wir sind nur daran interessiert`q[1]`, den Zustand des ersten Qubits zu messen. Dies wird durch die Linie erreicht`b[0] = measure q[0]`, die den Zustand von Qubit [0] misst und das Ergebnis im klassischen Bit b [0] speichert. Um dieses partielle Messszenario auszuführen, können wir den folgenden Code auf dem von Amazon Braket bereitgestellten Local State Vector Simulator ausführen.

```
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)
```

Sie können überprüfen, ob ein Gerät Teilmessungen unterstützt, indem Sie das `requiresAllQubitsMeasurement` Feld anhand seiner Aktionseigenschaften untersuchen. Ist dies der Fall`False`, wird eine Teilmessung unterstützt.

```
from braket.devices import Devices

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

Hier `requiresAllQubitsMeasurement` ist`False`, was darauf hindeutet, dass nicht alle Qubits gemessen werden müssen.

Erkunden Sie experimentelle Möglichkeiten

Experimentelle Funktionen ermöglichen den Zugriff auf Hardware mit begrenzter Verfügbarkeit und neue Softwarefunktionen. Diese Funktionen können die Geräteleistung über die Standardspezifikationen hinaus beeinträchtigen. Sie können experimentelle Softwarefunktionen automatisch für jede Aufgabe über das Amazon Braket SDK aktivieren.

Um experimentelle Funktionen zu verwenden, geben Sie den `experimental_capabilities` Parameter an, wenn Sie Quantenaufgaben erstellen. Stellen Sie diesen Parameter auf ein`"ALL"`, um alle verfügbaren experimentellen Funktionen für diese Aufgabe zu aktivieren. Das folgende Beispiel zeigt, wie experimentelle Funktionen aktiviert werden, wenn Sie eine Schaltung auf einem Gerät ausführen:

```
from braket.aws import AwsDevice

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

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

> **Note**
>
> Diese Funktionen sind experimentell und können sich ohne vorherige Ankündigung ändern. Die Geräteleistung kann von den veröffentlichten Spezifikationen abweichen, und die Ergebnisse können vom Standardbetrieb abweichen. Sie müssen experimentelle Funktionen für jede Aufgabe explizit aktivieren. Bei Aufgaben ohne diesen Parameter werden nur Standardgerätefunktionen verwendet.

In diesem Abschnitt:

- Zugriff auf die lokale Abstimmung auf Aquila QuEra
- Zugang zu hohen Geometrien auf Aquila QuEra
- Zugang zu engen Geometrien auf Aquila QuEra
- Dynamische Schaltungen auf IQM-Geräten

Zugriff auf die lokale Abstimmung auf Aquila QuEra

Local Detuning (LD) ist ein neues, zeitabhängiges Kontrollfeld mit einem anpassbaren räumlichen Muster. Das LD-Feld beeinflusst Qubits nach einem anpassbaren räumlichen Muster. Dabei werden für verschiedene Qubits unterschiedliche Hamiltonianer realisiert, die über das hinausgehen, was das einheitliche Antriebsfeld und die Interaktion erzeugen können. Rydberg-Rydberg

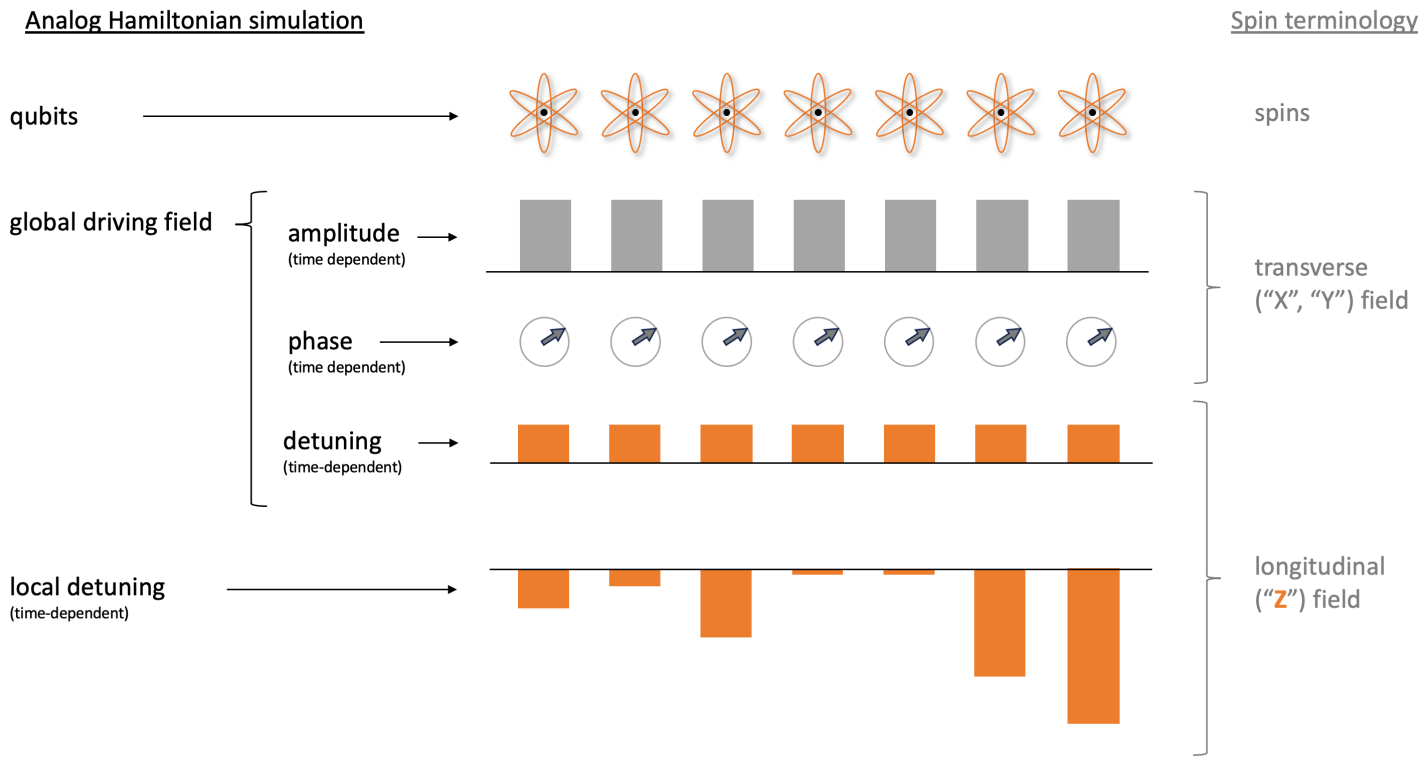
Einschränkungen:

Das räumliche Muster des lokalen Verstimmungsfeldes ist für jedes AHS-Programm anpassbar, bleibt aber im Verlauf eines Programms konstant. Die Zeitreihe des lokalen Verstimmungsfeldes muss bei Null beginnen und enden, wobei alle Werte kleiner oder gleich Null sein müssen. Darüber hinaus sind die Parameter des Felds für die lokale Abstimmung durch numerische Beschränkungen begrenzt, die

über das Braket-SDK im Abschnitt mit den spezifischen Geräteeigenschaften - eingesehen werden können. `aquila_device.properties.paradigm.rydberg.rydbergLocal`

Einschränkungen:

Bei der Ausführung von Quantenprogrammen, die das lokale Verstimmungsfeld verwenden (auch wenn dessen Größe im Hamiltonschen Wert konstant auf Null gesetzt ist), dekohärent das Gerät schneller als die T2-Zeit, die im Abschnitt „Leistung“ der Eigenschaften von Aquila angegeben ist. Wenn dies nicht erforderlich ist, empfiehlt es sich, das lokale Verstimmungsfeld aus dem Hamilton-Wert des AHS-Programms wegzulassen.



Beispiele:

1. Simulation der Wirkung eines ungleichmäßigen Longitudinalmagnetfeldes in Spinsystemen

   Während Amplitude und Phase des treibenden Feldes dieselbe Wirkung auf die Qubits haben wie das transversale Magnetfeld auf Spins, erzeugt die Summe aus der Verstimmung des treibenden Feldes und der lokalen Verstimmung auf die Qubits den gleichen Effekt wie das Längsfeld auf Spins. Mit der räumlichen Kontrolle über das lokale Verstimmungsfeld können komplexere Spinsysteme simuliert werden.
2. Vorbereitung von Anfangszuständen, die nicht im Gleichgewicht sind

Das Beispielheft Simulation der Gittereichtheorie mit Rydberg-Atomen zeigt, wie verhindert werden kann, dass das Zentralatom einer linearen Anordnung mit 9 Atomen angeregt wird, wenn das System in Richtung der Z2-geordneten Phase geglüht wird. Nach dem Vorbereitungsschritt wird das lokale Verstimmungsfeld heruntergefahren, und das AHS-Programm simuliert weiterhin die zeitliche Entwicklung des Systems ausgehend von diesem speziellen Nichtgleichgewichtszustand.

3. Lösung gewichteter Optimierungsprobleme

   Das Beispiel-Notizbuch Maximum Weight Independent Set (MWIS) zeigt, wie ein MWIS-Problem auf Aquila gelöst werden kann. Das lokale Verstimmungsfeld wird verwendet, um die Gewichte auf den Knoten des Einheitenscheibendiagramms zu definieren, deren Kanten durch den Effekt realisiert werden. Rybderg-blockage Ausgehend vom einheitlichen Grundzustand und allmählicher Erhöhung des lokalen Verstimmungsfeldes geht das System in den Grundzustand des MWIS-Hamiltonian über, um Lösungen für das Problem zu finden.

Zugang zu hohen Geometrien auf Aquila QuEra

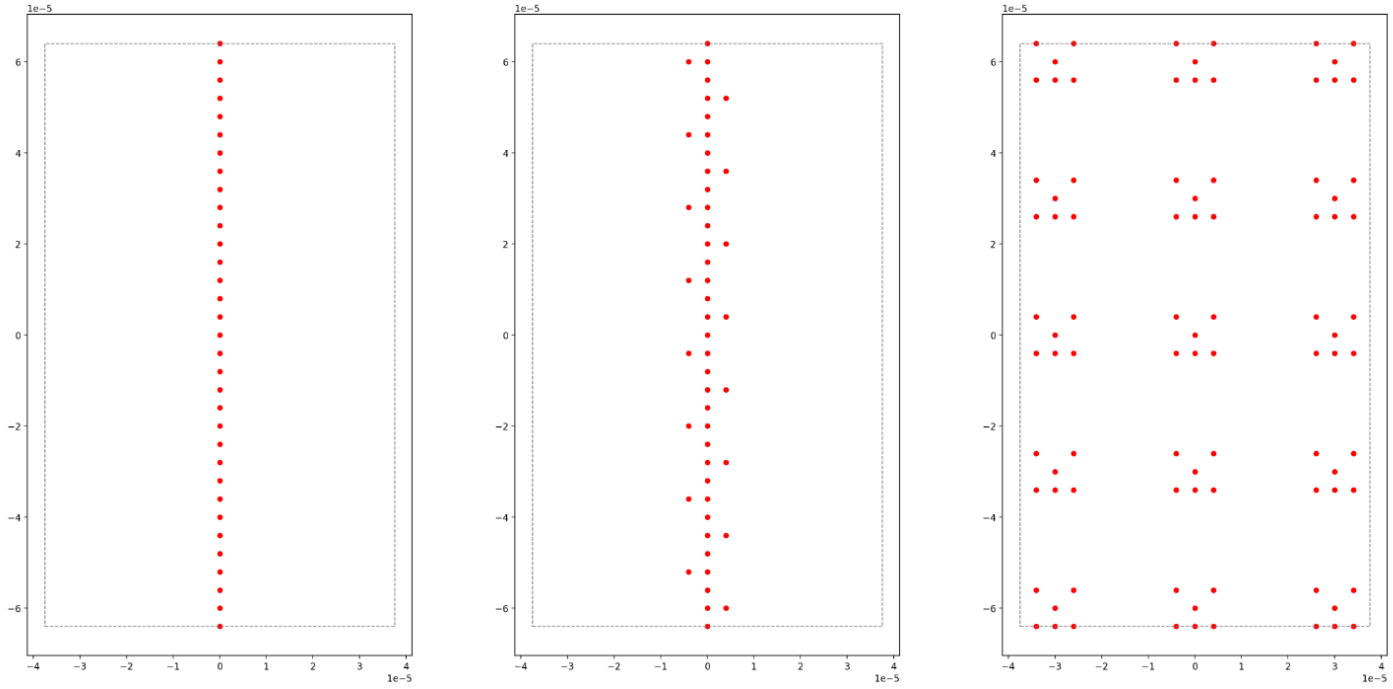
Mit der Funktion für hohe Geometrien können Sie Geometrien mit größerer Höhe angeben. Mit dieser Funktion können sich die Atomanordnungen Ihrer AHS-Programme über eine zusätzliche Länge in Y-Richtung erstrecken, die über die regulären Funktionen von Aquila hinausgeht.

Einschränkungen:

Die maximale Höhe für hohe Geometrien beträgt 0,000128 m (128 um).

Einschränkungen:

Die Funktionen, die auf der Seite mit den Geräteeigenschaften und dem `GetDevice` Aufruf angezeigt werden, entsprechen weiterhin dem regulären, unteren Grenzwert für die Höhe.
Wenn ein AHS-Programm Atom-Anordnungen verwendet, die über die regulären Funktionen hinausgehen, ist davon auszugehen, dass der Füllfehler zunimmt. Sie werden im `pre_sequence` Teil des Aufgabenergebnisses eine erhöhte Anzahl unerwarteter Nullen finden, was wiederum die Wahrscheinlichkeit verringert, eine perfekt initialisierte Anordnung zu erhalten. Dieser Effekt ist in Reihen mit vielen Atomen am stärksten.



Beispiele:

1. Größere 1D- und Quasi-1D-Anordnungen

   Atomketten und leiterartige Anordnungen können auf höhere Atomzahlen erweitert werden. Durch die Ausrichtung der langen Richtung parallel zu y können längere Instanzen dieser Modelle programmiert werden.
2. Mehr Spielraum für das Multiplexen bei der Ausführung von Aufgaben mit kleinen Geometrien

   Das Beispiel-Notizbuch Parallele Quantenaufgaben auf Aquila zeigt, wie man den verfügbaren Bereich optimal nutzt: indem man gemultiplexte Kopien der fraglichen Geometrie in einer Atomanordnung platziert. Je mehr Fläche zur Verfügung steht, desto mehr Kopien können platziert werden.

Zugang zu engen Geometrien auf Aquila QuEra

Mit der Funktion für enge Geometrien können Sie Geometrien mit kürzeren Abständen zwischen benachbarten Zeilen angeben. In einem AHS-Programm werden Atome in Reihen angeordnet, die durch einen minimalen vertikalen Abstand voneinander getrennt sind. Die Y-Koordinate zweier beliebiger Atompositionen muss entweder Null sein (gleiche Reihe) oder sich um mehr als den minimalen Zeilenabstand (unterschiedliche Reihe) unterscheiden. Durch die Möglichkeit enger Geometrien wird der minimale Reihenabstand reduziert, wodurch engere Atomanordnungen möglich

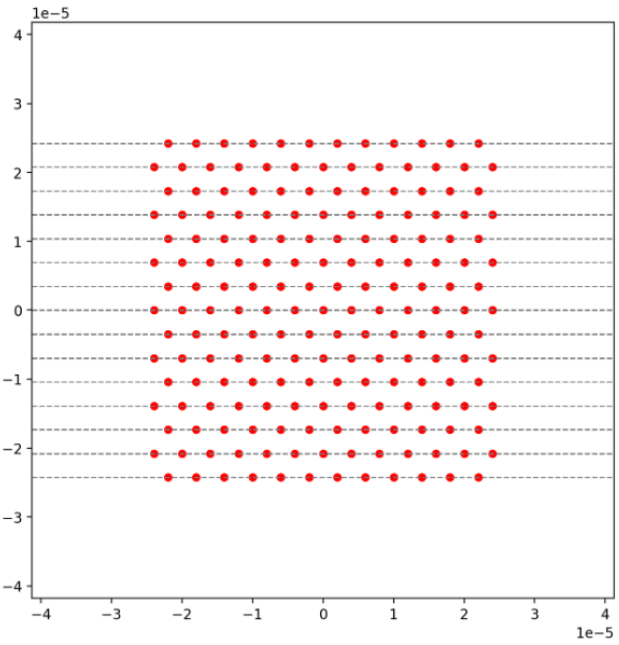
sind. Diese Erweiterung ändert zwar nichts an der euklidischen Mindestabstandsanforderung zwischen Atomen, ermöglicht aber die Erzeugung von Gittern, bei denen entfernte Atome benachbarte Reihen besetzen, die näher beieinander liegen. Ein bemerkenswertes Beispiel ist das Dreiecksgitter.

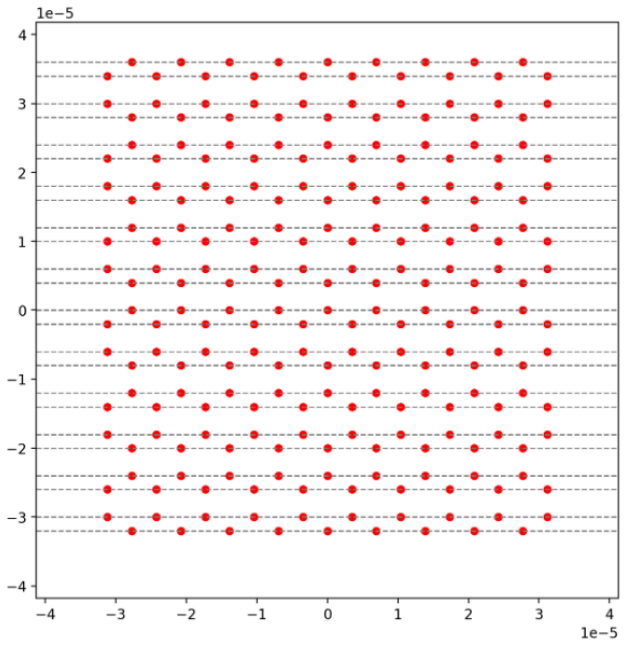
Einschränkungen:

Der minimale Reihenabstand für enge Geometrien beträgt 0,000002 m (2 um).

Einschränkungen:

Die Funktionen, die auf der Seite mit den Geräteeigenschaften und dem `GetDevice` Aufruf angezeigt werden, entsprechen weiterhin dem regulären, höheren Grenzwert für den Abstand. Wenn ein AHS-Programm Atom-Anordnungen verwendet, die über die regulären Funktionen hinausgehen, ist davon auszugehen, dass der Füllfehler zunimmt. Kunden werden im `pre_sequence` Teil des Aufgabenergebnisses eine erhöhte Anzahl unerwarteter Nullen feststellen, was wiederum die Wahrscheinlichkeit verringert, ein perfekt initialisiertes Arrangement zu erhalten. Dieser Effekt ist in Reihen mit vielen Atomen am stärksten.





Beispiele:

1. Non-rectangular Gitter mit kleinen Gitterkonstanten

Ein engerer Zeilenabstand ermöglicht die Bildung von Gittern, bei denen sich die nächsten Nachbarn einiger Atome in diagonaler Richtung befinden. Bemerkenswerte Beispiele sind dreieckige, sechseckige und Kagome-Gitter sowie einige Quasikristalle.

2. Einstellbare Familie von Gittern

   In AHS-Programmen werden Interaktionen durch Anpassung des Abstands zwischen Atompaaren abgestimmt. Ein engerer Reihenabstand ermöglicht es, die Wechselwirkungen verschiedener Atompaare relativ zueinander mit größerer Freiheit abzustimmen, da die Winkel und Abstände, die die Atomstruktur definieren, weniger durch die minimale Reihenabstandsbeschränkung begrenzt sind. Ein bemerkenswertes Beispiel ist die Familie von Shastry-Sutherland Gittern mit unterschiedlichen Bindungslängen.

Dynamische Schaltungen auf IQM-Geräten

Dynamische Schaltungen an IQM Geräten ermöglichen Messungen im mittleren Schaltkreis (MCM) und Feed-Forward-Operationen. Diese Funktionen ermöglichen es Quantenforschern und Entwicklern, fortschrittliche Quantenalgorithmen mit bedingter Logik und Funktionen zur Wiederverwendung von Qubits zu implementieren. Diese experimentelle Funktion hilft bei der Erforschung von Quantenalgorithmen mit verbesserter Ressourceneffizienz und bei der Untersuchung von Schemata zur Minderung und Fehlerkorrektur im Bereich Quantenfehler.

Die wichtigsten Anweisungen:

- `measure_ff`: Implementiert Messungen zur Feed-Forward-Steuerung, wobei ein Qubit gemessen und das Ergebnis mit einem Feedback-Schlüssel gespeichert wird.
- `cc_prx`: Implementiert eine klassisch gesteuerte Rotation, die nur gilt, wenn das mit dem angegebenen Feedback-Schlüssel verknüpfte Ergebnis einen Zustand von |1# misst.

Amazon Braket unterstützt dynamische Schaltungen durch OpenQASMAmazon Braket SDK, die und dieAmazon Braket Qiskit Provider.

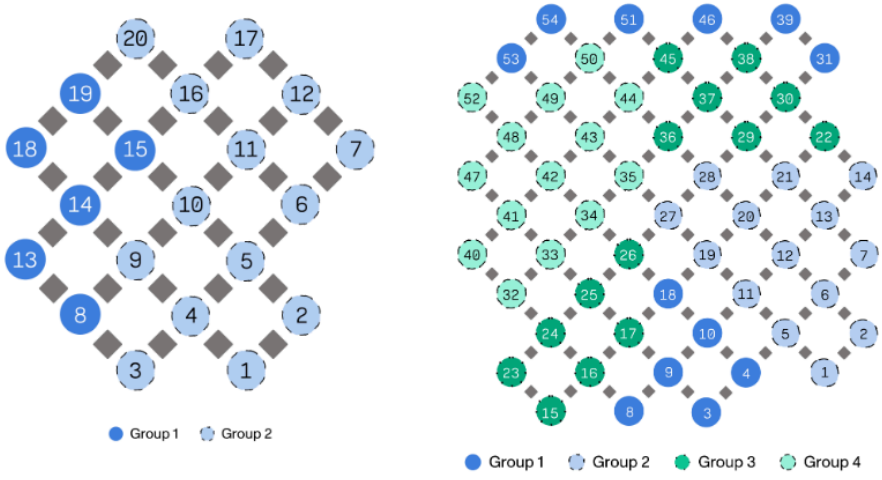
Einschränkungen:

1. Die Feedback-Schlüssel in den `measure_ff` Anweisungen müssen eindeutig sein.
2. Ein `cc_prx` muss danach `measure_ff` mit demselben Feedback-Schlüssel passieren.

3. In einem einzigen Stromkreis kann die Feed-Forward-Übertragung auf einem Qubit nur von einem Qubit gesteuert werden, entweder von ihm selbst oder von einem anderen Qubit. In verschiedenen Schaltungen können Sie unterschiedliche Steuerpaare haben.
   a. Wenn Qubit 1 beispielsweise von Qubit 2 gesteuert wird, kann es nicht von Qubit 3 in derselben Schaltung gesteuert werden. Es gibt keine Beschränkung, wie oft die Steuerung zwischen Qubit 1 und Qubit 2 angewendet wird. Qubit 2 kann durch Qubit 3 (oder Qubit 1) gesteuert werden, es sei denn, Qubit 2 wurde aktiv zurückgesetzt.
4. Die Steuerung kann nur auf Qubits innerhalb derselben Gruppe angewendet werden. Die Qubit-Gruppen für die Emerald Geräte IQM Garnet und sind in den folgenden Bildern dargestellt.
5. Programme mit diesen Funktionen müssen als unveränderte Programme eingereicht werden. Weitere Informationen zu verbatim-Programmen finden Sie unter Verbatim-Kompilierung mit OpenQASM 3.0.

Einschränkungen:

MCM kann nur für die Feed-Forward-Steuerung in einem Programm verwendet werden. Die MCM-Ergebnisse (0 oder 1) werden nicht als Teil eines Aufgabenergebnisses zurückgegeben.



Diese Bilder zeigen IQM die Qubit-Gruppierungen für beide Geräte. Das Garnet 20-Qubit-Gerät enthält 2 Gruppen von Qubits, während das Emerald 54-Qubit-Gerät 4 Gruppen von Qubits enthält.

Beispiele:

1. Wiederverwendung von Qubits durch aktiven Reset

MCM mit bedingten Reset-Operationen ermöglicht die Wiederverwendung von Qubits innerhalb einer einzigen Schaltkreisausführung. Dies reduziert die Anforderungen an die Schaltungstiefe und verbessert die Nutzung der Ressourcen von Quantengeräten.

2. Aktiver Bit-Flip-Schutz

   Dynamische Schaltungen erkennen Bit-Flip-Fehler und wenden auf der Grundlage der Messergebnisse Korrekturmaßnahmen an. Diese Implementierung dient als Experiment zur Erkennung von Quantenfehlern.

3. Teleportationsexperimente

   Bei der Zustandsteleportation werden Qubitzustände mithilfe lokaler Quantenoperationen und klassischer Informationen von MCMs übertragen. Die Gate-Teleportation implementiert Tore zwischen Qubits ohne direkte Quantenoperationen. Diese Experimente demonstrieren grundlegende Subroutinen in drei Schlüsselbereichen: Quantenfehlerkorrektur, messgestütztes Quantencomputing und Quantenkommunikation.

4. Offene Simulation von Quantensystemen

   Dynamische Schaltkreise modellieren das Rauschen in Quantensystemen durch Datenqubit- und Umgebungsverschränkung sowie durch Umweltmessungen. Dieser Ansatz verwendet spezifische Qubits zur Darstellung von Daten- und Umweltelementen. Ein Geräuschkanal kann durch die Tore und Messungen an der Umgebung entworfen werden.

Weitere Informationen zur Verwendung dynamischer Schaltungen finden Sie in zusätzlichen Beispielen im Amazon Braket-Notebook-Repository.

Pulssteuerung auf Amazon Braket

Impulse sind die analogen Signale, die die Qubits in einem Quantencomputer steuern. Bei bestimmten Geräten auf Amazon Braket können Sie auf die Impulssteuerungsfunktion zugreifen, um Stromkreise mithilfe von Impulsen zu übertragen. Sie können über das Braket-SDK, mit OpenQASM 3.0 oder direkt über die Braket-APIs auf die Impulssteuerung zugreifen. Stellen Sie zunächst einige wichtige Konzepte für die Impulssteuerung in Braket vor.

In diesem Abschnitt:

- Frames (Frames)
- Ports

- Wellenformen
- Ich arbeite mit Hello Pulse
- Zugreifen auf native Gates mithilfe von Impulsen

Frames (Frames)

Ein Frame ist eine Softwareabstraktion, die sowohl als Uhr innerhalb des Quantenprogramms als auch als Phase fungiert. Die Uhrzeit wird bei jeder Nutzung und einem zustandsbehafteten Trägersignal, das durch eine Frequenz definiert wird, inkrementiert. Bei der Übertragung von Signalen an das Qubit bestimmt ein Frame die Trägerfrequenz, den Phasenversatz und den Zeitpunkt, zu dem die Wellenformhüllkurve emittiert wird. In Braket Pulse hängt die Konstruktion von Frames vom Gerät, der Frequenz und der Phase ab. Je nach Gerät können Sie entweder einen vordefinierten Frame auswählen oder neue Frames instanziieren, indem Sie einen Port angeben.

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

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

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

Ports

Ein Port ist eine Softwareabstraktion, die jede input/output Hardwarekomponente darstellt, die Qubits steuert. Es hilft Hardwareanbietern, eine Schnittstelle bereitzustellen, mit der Benutzer interagieren können, um Qubits zu manipulieren und zu beobachten. Anschlüsse sind durch eine einzelne Zeichenfolge gekennzeichnet, die den Namen des Connectors darstellt. Diese Zeichenfolge zeigt auch ein Mindestzeitinkrement an, das angibt, wie genau wir die Wellenformen definieren können.

```
from braket.pulse import Port

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

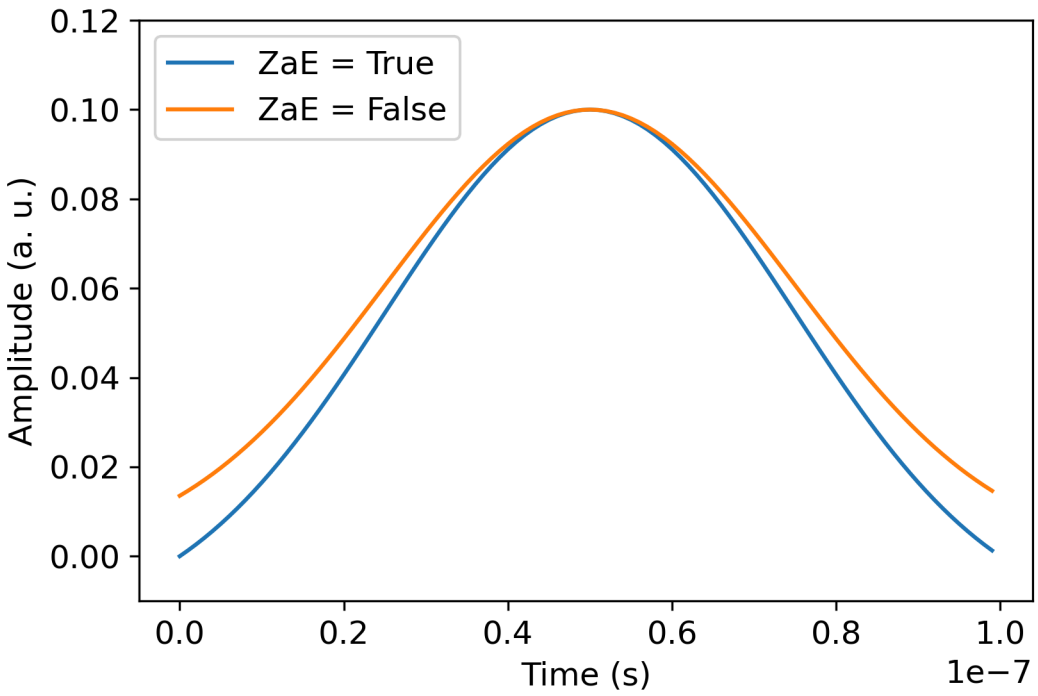
Wellenformen

Eine Wellenform ist eine zeitabhängige Hüllkurve, die wir verwenden können, um Signale an einem Ausgangsanschluss zu senden oder Signale über einen Eingangsanschluss zu erfassen. Sie können Ihre Wellenformen direkt angeben, entweder durch eine Liste komplexer Zahlen oder mithilfe einer Wellenformvorlage, um eine Liste vom Hardwareanbieter zu generieren.

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

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

Braket Pulse bietet eine Standardbibliothek mit Wellenformen, darunter eine konstante Wellenform, eine Gaußsche Wellenform und eine DRAG-Wellenform (Derivative Removal by Adiabatic Gate). Sie können die Wellenformdaten mithilfe der Funktion abrufen, um die Form der Wellenform zu zeichnen, wie im folgenden Beispiel gezeigt`sample`.

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

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

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

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



Das vorherige Bild zeigt die Gaußschen Wellenformen, die aus erzeugt wurden. `GaussianWaveform` Wir haben eine Pulslänge von 100 ns, eine Breite von 25 ns und eine Amplitude von 0,1 (willkürliche Einheiten) gewählt. Die Wellenformen sind im Pulsfenster zentriert. `GaussianWaveform`akzeptiert ein boolesches Argument `zero_at_edges` (zAe in der Legende). Wenn dieses Argument auf gesetzt ist`True`, verschiebt es die Gaußsche Wellenform so, dass die Punkte bei t=0 und t= bei Null `length` liegen, und skaliert seine Amplitude neu, sodass der Maximalwert dem Argument entspricht. `amplitude`

Ich arbeite mit Hello Pulse

In diesem Abschnitt erfahren Sie, wie Sie ein einzelnes Qubit-Gate direkt mithilfe eines Impulses an einem Rigetti Gerät charakterisieren und konstruieren. Das Anlegen eines elektromagnetischen Feldes an ein Qubit führt zu einer Rabi-Oszillation, bei der Qubits zwischen dem Zustand 0 und dem Zustand 1 umgeschaltet werden. Bei kalibrierter Länge und Phase des Impulses kann die Rabi-Oszillation einzelne Qubit-Gates berechnen. Hier werden wir die optimale Impulslänge für die Messung eines Impulses bestimmen. Dabei handelt es sich um einen elementaren Block, der zum Aufbau komplexerer pi/2 Impulssequenzen verwendet wird.

Um eine Impulssequenz zu erstellen, importieren Sie zunächst die `PulseSequence` Klasse.

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

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

import numpy as np
```

Als Nächstes instanziieren Sie ein neues Braket-Gerät mit dem Amazon Resource Name (ARN) der QPU. Der folgende Codeblock verwendet. Rigetti Ankaa-3

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

Die folgende Impulssequenz besteht aus zwei Komponenten: Abspielen einer Wellenform und Messen eines Qubits. Die Pulssequenz kann normalerweise auf Frames angewendet werden. Mit einigen Ausnahmen wie Barrier und Delay, die auf Qubits angewendet werden können. Bevor Sie die Impulssequenz erstellen, müssen Sie die verfügbaren Frames abrufen. Der Antriebsrahmen dient zum Anlegen des Impulses für die Rabi-Oszillation, und der Ausleserahmen dient zur Messung des Qubit-Zustands. In diesem Beispiel werden die Frames von Qubit 25 verwendet.

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

Erstellen Sie nun die Wellenform, die im Drive-Frame abgespielt werden soll. Ziel ist es, das Verhalten der Qubits für verschiedene Pulslängen zu charakterisieren. Sie spielen jedes Mal eine Wellenform mit unterschiedlichen Längen ab. Anstatt jedes Mal eine neue Wellenform zu instanziieren, verwenden Sie die In-Puls-Sequenz. Braket-supported `FreeParameter` Sie können die Wellenform und die Impulssequenz einmal mit freien Parametern erstellen und dann dieselbe Impulssequenz mit unterschiedlichen Eingabewerten ausführen.

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

Schließlich fügen Sie sie zu einer Impulssequenz zusammen. In der Impulssequenz wird die angegebene Wellenform auf dem Antriebsrahmen `play` abgespielt und der `capture_v0` Status wird vom Ausleserahmen aus gemessen.

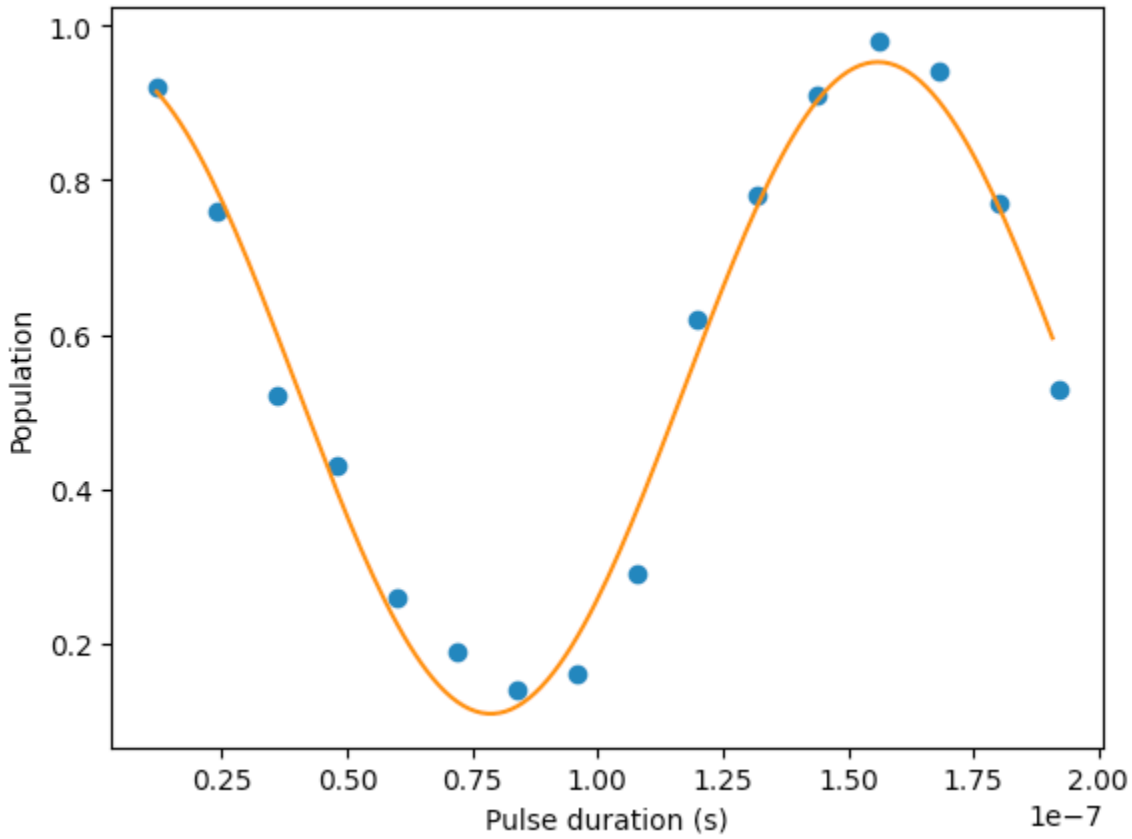
```
pulse_sequence = (
    PulseSequence()
    .play(drive_frame, waveform)
```

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

Scannt über einen bestimmten Pulslängenbereich und sendet sie an die QPU. Bevor Sie die Impulssequenzen auf einer QPU ausführen, binden Sie den Wert der freien Parameter.

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

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

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

Die Statistik der Qubit-Messung zeigt die Schwingungsdynamik des Qubits, das zwischen dem 0-Zustand und dem 1-Zustand oszilliert. Aus den Messdaten können Sie die Rabi-Frequenz extrahieren und die Länge des Impulses fein abstimmen, um ein bestimmtes 1-Qubit-Gate zu implementieren. Aus den Daten in der Abbildung unten geht beispielsweise hervor, dass die Periodizität etwa 154 ns beträgt. Ein pi/2 Rotationstor würde also der Impulssequenz mit einer Länge von 38,5 ns entsprechen.



Hallo Pulse benutzt OpenPulse

OpenPulseist eine Sprache zur Spezifikation der Impulssteuerung eines allgemeinen Quantengeräts und Teil der OpenQASM 3.0-Spezifikation. Amazon Braket unterstützt die OpenPulse direkte Programmierung von Impulsen mithilfe der OpenQASM 3.0-Darstellung.

Braket verwendet OpenPulse als zugrunde liegende Zwischendarstellung zum Ausdrücken von Impulsen in systemeigenen Befehlen. OpenPulseunterstützt das Hinzufügen von Befehlskalibrierungen in Form von Deklarationen `defcal` (kurz für „Kalibrierung definieren“). Mit diesen Deklarationen können Sie eine Implementierung einer Gate-Anweisung innerhalb einer Kontrollgrammatik auf niedrigerer Ebene spezifizieren.

Mit dem folgenden Befehl können Sie sich das OpenPulse Programm einer `PulseSequence` Braket-Datei ansehen.

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

Sie können ein OpenPulse Programm auch direkt erstellen.

```
from braket.ir.openqasm import Program

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

Erstellen Sie ein `Program` Objekt mit Ihrem Skript. Senden Sie das Programm dann an eine QPU.

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

program = Program(source=openpulse_script)

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

Zugreifen auf native Gates mithilfe von Impulsen

Forscher müssen oft genau wissen, wie die nativen Gates, die von einer bestimmten QPU unterstützt werden, als Impulse implementiert werden. Impulssequenzen werden von Hardwareanbietern sorgfältig kalibriert, aber der Zugriff auf sie bietet Forschern die Möglichkeit, bessere Gates zu entwerfen oder Protokolle zur Fehlerminimierung zu erforschen, wie z. B. eine rauschfreie Extrapolation durch Dehnen der Impulse bestimmter Gatter.

Amazon Braket unterstützt den programmatischen Zugriff auf native Gates von Rigetti.

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

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

calibrations = device.gate_calibrations
```

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

> **Note**
>
> Hardwareanbieter kalibrieren die QPU regelmäßig, oft mehr als einmal täglich. Das Braket SDK ermöglicht es Ihnen, die neuesten Gate-Kalibrierungen zu erhalten.

```
device.refresh_gate_calibrations()
```

Um ein bestimmtes natives Gate, wie z. B. das RX- oder XY-Gate, abzurufen, müssen Sie das `Gate` Objekt und die gewünschten Qubits übergeben. Sie können zum Beispiel die Impulsimplementierung des auf 0 angewendeten RX (π /2) überprüfen. qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

Mit der Funktion können Sie einen gefilterten Satz von Kalibrierungen erstellen. `filter` Sie bestehen an einer Liste von Toren oder einer Liste von`QubitSet`. Der folgende Code erstellt zwei Sätze, die alle Kalibrierungen für RX (π /2) und für 0 enthalten. qubit

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

Jetzt können Sie die Aktion nativer Gates angeben oder ändern, indem Sie einen benutzerdefinierten Kalibrierungssatz anhängen. Stellen Sie sich zum Beispiel die folgende Schaltung vor.

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

Sie können es mit einer benutzerdefinierten Gate-Kalibrierung für das eingeschaltete `rx` Gate ausführen, `qubit 0` indem Sie ein Wörterbuch mit `PulseSequence` Objekten an das `gate_definitions` Schlüsselwort-Argument übergeben. Sie können aus dem Attribut

`pulse_sequences` des `GateCalibrations` Objekts ein Wörterbuch erstellen. Alle nicht spezifizierten Gatter werden durch die Pulskalibrierung des Quantenhardwareanbieters ersetzt.

```
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)
```

Analoge Hamiltonsche Simulation

Die analoge Hamiltonsche Simulation (AHS) ist ein neues Paradigma im Quantencomputing, das sich erheblich vom traditionellen Quantenschaltkreismodell unterscheidet. Anstatt einer Abfolge von Gates, bei der jeder Schaltkreis nur auf ein paar Qubits gleichzeitig wirkt. Ein AHS-Programm wird durch die zeit- und raumabhängigen Parameter des betreffenden Hamiltonschen Parameters definiert. Der Hamilton-Wert eines Systems kodiert seine Energieniveaus und die Auswirkungen äußerer Kräfte, die zusammen die zeitliche Entwicklung seiner Zustände bestimmen. Für ein N-qubit System kann der Hamilton-Operator durch eine quadratische Matrix mit 2 N x 2 ^{N} aus komplexen Zahlen dargestellt werden.

Quantengeräte, die AHS ausführen können, sind so konzipiert, dass sie der zeitlichen Entwicklung eines Quantensystems unter einem benutzerdefinierten Hamiltonschen Wert sehr nahe kommen, indem ihre internen Steuerparameter sorgfältig abgestimmt werden. Zum Beispiel die Anpassung der Amplitude und die Verstimmung der Parameter eines kohärenten Antriebsfeldes. Das AHS-Paradigma eignet sich gut für die Simulation der statischen und dynamischen Eigenschaften von Quantensystemen mit vielen wechselwirkenden Teilchen, beispielsweise in der Physik der kondensierten Materie oder der Quantenchemie. Purpose-built Quantenverarbeitungseinheiten (QPUs), wie das Aquila-Gerät vonQuEra, wurden entwickelt, um die Leistungsfähigkeit von AHS zu nutzen und Probleme, die über die Reichweite herkömmlicher digitaler Quantencomputeransätze hinausgehen, auf innovative Weise anzugehen.

In diesem Abschnitt:

- Hallo AHS: Führe deine erste analoge Hamilton-Simulation aus
- Reichen Sie ein analoges Programm mit Aquila ein QuEra

Hallo AHS: Führe deine erste analoge Hamilton-Simulation aus

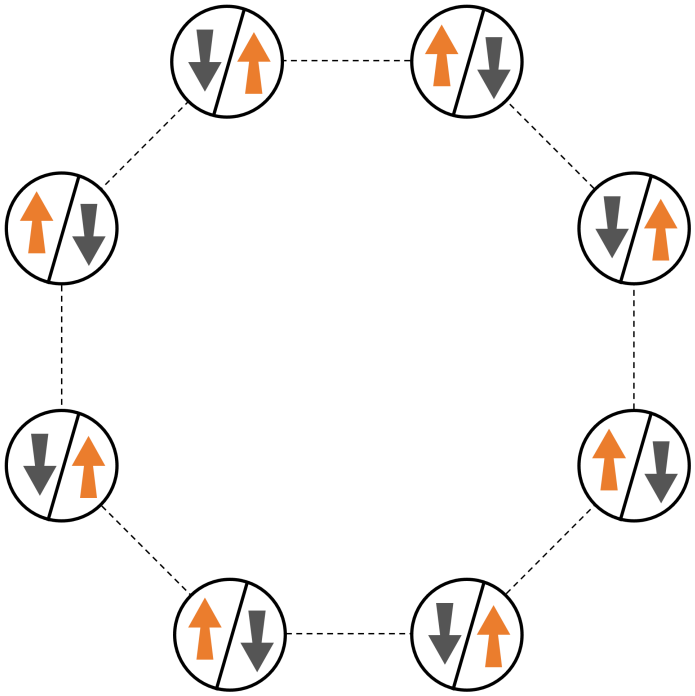
Dieser Abschnitt enthält Informationen zur Ausführung Ihrer ersten analogen Hamiltonschen Simulation.

In diesem Abschnitt:

- Interagierende Spin-Kette
- Anordnung
- Interaktionen
- Fahrfeld
- AHS-Programm
- Läuft auf einem lokalen Simulator
- Analysieren der Simulatorergebnisse
- Läuft auf der Aquila QuEra QPU
- QPU-Ergebnisse analysieren
- Nächste Schritte

Interagierende Spin-Kette

Als kanonisches Beispiel für ein System aus vielen wechselwirkenden Teilchen betrachten wir einen Ring aus acht Spins (von denen sich jeder in den Zuständen „oben“ ↑ und „unten“ ↓# befinden kann). Dieses Modellsystem ist zwar klein, weist aber bereits eine Handvoll interessanter Phänomene natürlich vorkommender magnetischer Materialien auf. In diesem Beispiel werden wir zeigen, wie man eine sogenannte antiferromagnetische Ordnung erzeugt, bei der aufeinanderfolgende Spins in entgegengesetzte Richtungen zeigen.



Anordnung

Wir verwenden ein neutrales Atom, das für jeden Spin steht, und die Spinzustände „hoch“ und „runter“ werden jeweils im angeregten Rydberg-Zustand und im Grundzustand der Atome kodiert. Zuerst erstellen wir die 2-D-Anordnung. Wir können den obigen Ring von Spins mit dem folgenden Code programmieren.

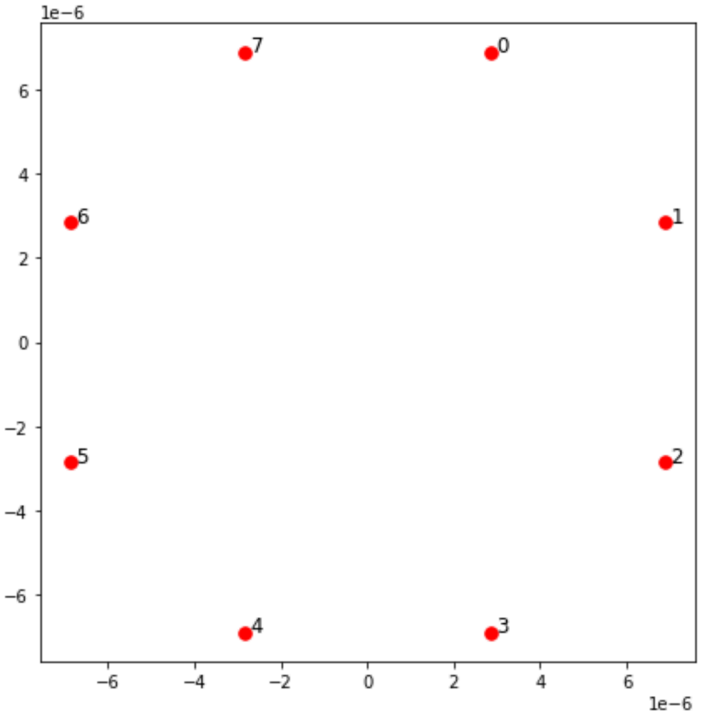
Voraussetzungen: Sie müssen das Braket SDK per Pip installieren. (Wenn Sie eine von Braket gehostete Notebook-Instanz verwenden, ist dieses SDK zusammen mit den Notebooks vorinstalliert.) Um die Plots zu reproduzieren, müssen Sie matplotlib auch separat mit dem Shell-Befehl installieren. `pip install matplotlib`

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

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

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

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

womit wir auch plotten können

```
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
```



Interaktionen

Um die antiferromagnetische Phase vorzubereiten, müssen wir Wechselwirkungen zwischen benachbarten Spins induzieren. Wir nutzen dafür die Van-der-Waals-Wechselwirkung, die nativ von neutralen Atomgeräten (wie dem Gerät von) implementiert wird. Aquila QuEra Mit Hilfe der Spin-Repräsentation kann der Hamiltonsche Term für diese Wechselwirkung als Summe über alle Spinpaare (j, k) ausgedrückt werden.

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

In diesem Fall ist nj=↑ bung↑ $_j$ ein $_j$ Operator, der den Wert 1 nur annimmt, wenn sich Spin j im Zustand „up“ befindet, andernfalls 0. Die Stärke ist V $_{j,k}$ =C_6/($d_{j,k}$) ^{6}, wobei C $_6$ der feste Koeffizient und d der euklidische Abstand zwischen den Spins j und k $_{j,k}$ ist. Der unmittelbare Effekt dieses Wechselwirkungsterms besteht darin, dass jeder Zustand, in dem sowohl Spin j als auch Spin k „oben“ sind, eine erhöhte Energie (um den Betrag V) aufweist. $_{j,k}$ Durch die sorgfältige Gestaltung des restlichen AHS-Programms wird durch diese Wechselwirkung verhindert, dass sich benachbarte Spins beide im „Up-Zustand“ befinden — ein Effekt, der allgemein als „Rydberg-Blockade“ bekannt ist.

Fahrfeld

Zu Beginn des AHS-Programms beginnen alle Spins (standardmäßig) in ihrem „down“ -Zustand, sie befinden sich in einer sogenannten ferromagnetischen Phase. Mit Blick auf unser Ziel, die antiferromagnetische Phase vorzubereiten, spezifizieren wir ein zeitabhängiges kohärentes Antriebsfeld, das die Spins sanft von diesem Zustand in einen Vielteilchenzustand überführt, in dem der Zustand „oben“ bevorzugt wird. Der entsprechende Hamilton-Operator kann geschrieben werden als

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

wobei Ω (t), φ (t), δ (t) die zeitabhängige globale Amplitude (auch bekannt als Rabi-Frequenz), Phase und Verstimmung des treibenden Feldes sind, die alle Spins gleichmäßig beeinflussen. Hier sind S $_{-,k}$ =↓ $_k$ bung↑ $_k$ und S $_{+,k}$ =(S$_{-,k}$) † =↑ $_k$ Chesinik↓ $_k$ die senkenden bzw. anhebenden Operatoren von Spin k, und n $_k$ =↑ $_k$ ↑ ←↑ $_k$ ist derselbe Operator wie zuvor. Der Ω Teil des Antriebsfeldes verbindet kohärent die Zustände „nach unten“ und „nach oben“ aller Spins gleichzeitig, während der Δ-Teil die Energiebelohnung für den Zustand „hoch“ steuert.

Um einen reibungslosen Übergang von der ferromagnetischen Phase zur antiferromagnetischen Phase zu programmieren, spezifizieren wir das treibende Feld mit dem folgenden Code.

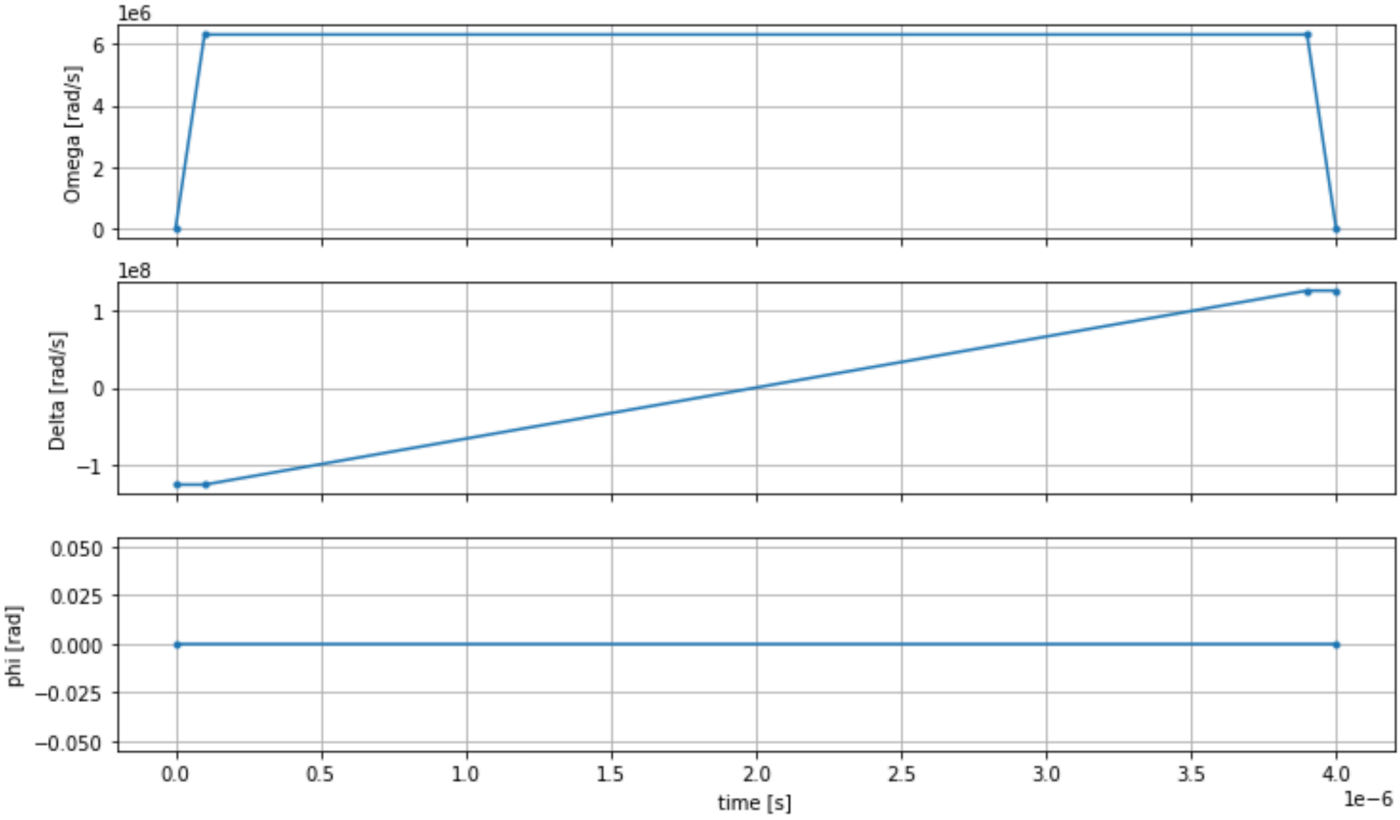
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField

# 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
)
```

Wir können die Zeitreihen des Fahrfeldes mit dem folgenden Skript visualisieren.

```
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
```



AHS-Programm

Das Register, das treibende Feld (und die impliziten Van-der-Waals-Interaktionen) bilden das Programm Analoge Hamiltonsche Simulation. `ahs_program`

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

Läuft auf einem lokalen Simulator

Da dieses Beispiel klein ist (weniger als 15 Spins), können wir es vor der Ausführung auf einer AHS-compatible QPU auf dem lokalen AHS-Simulator ausführen, der mit dem Braket-SDK geliefert wird. Da der lokale Simulator mit dem Braket-SDK kostenlos verfügbar ist, ist dies die bewährte Methode, um sicherzustellen, dass unser Code korrekt ausgeführt werden kann.

Hier können wir die Anzahl der Aufnahmen auf einen hohen Wert setzen (z. B. 1 Million), weil der lokale Simulator die zeitliche Entwicklung des Quantenzustands verfolgt und Proben aus dem Endzustand zieht. Dadurch wird die Anzahl der Aufnahmen erhöht, während die Gesamtlaufzeit nur geringfügig erhöht wird.

```
from braket.devices import LocalSimulator

device = LocalSimulator("braket_ahs")

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

Analysieren der Simulatorergebnisse

Wir können die Schussergebnisse mit der folgenden Funktion aggregieren, die den Status jedes Spins ableitet (der „d“ für „runter“, „u“ für „oben“ oder „e“ für leere Stelle sein kann) und zählt, wie oft jede Konfiguration in den Schüssen aufgetreten ist.

```
from collections import Counter


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

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

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

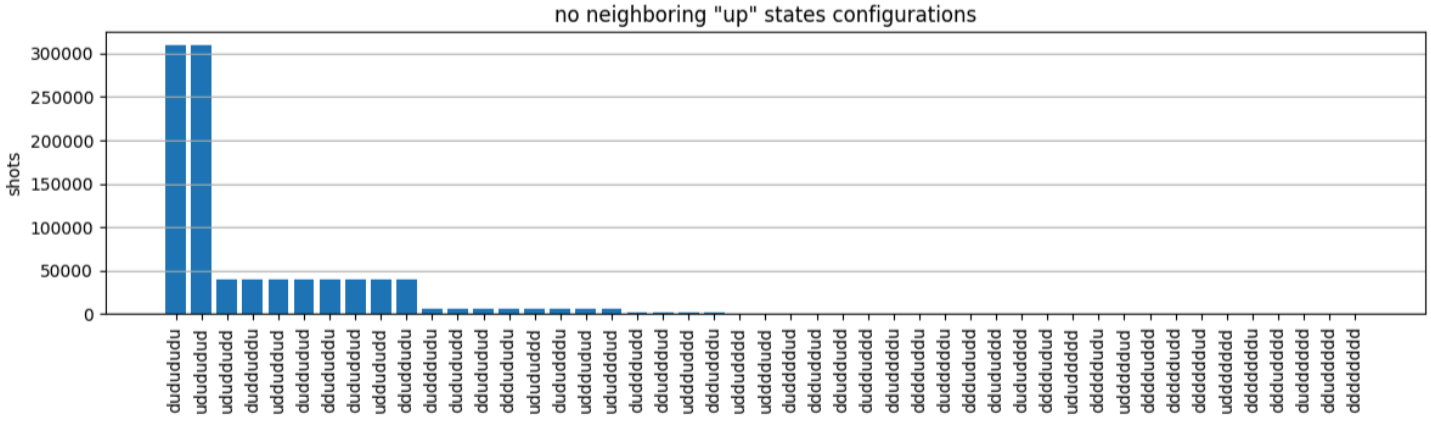
```
    Returns
       dict: number of times each state configuration is measured

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


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

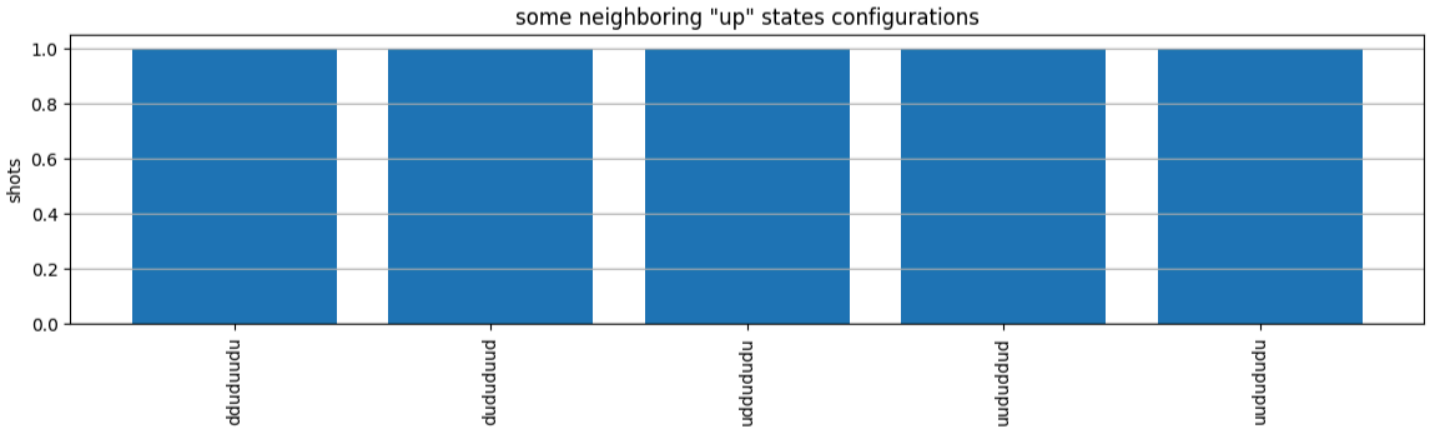
```
*[Output]*
{'dddddddd': 5, 'ddddddddu': 12, 'ddddddud': 15, ...}
```

Hier `counts` ist ein Wörterbuch, das zählt, wie oft jede Zustandskonfiguration in den einzelnen Schüssen beobachtet wurde. Wir können sie auch mit dem folgenden Code visualisieren.

```
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)
```





Aus den Diagrammen können wir die folgenden Beobachtungen ablesen, die belegen, dass wir die antiferromagnetische Phase erfolgreich vorbereitet haben.

1. Im Allgemeinen sind nicht blockierte Zustände (in denen sich keine zwei benachbarten Spins im „Up“ -Zustand befinden) häufiger als Zustände, in denen sich mindestens ein Paar benachbarter Spins beide im „Up“ -Zustand befinden.
2. Im Allgemeinen werden Zustände mit mehr Erregungen nach oben bevorzugt, es sei denn, die Konfiguration ist blockiert.
3. Die häufigsten Zustände sind in der Tat die perfekten antiferromagnetischen Zustände und. `"dudududu" "udududud"`
4. Die zweithäufigsten Zustände sind diejenigen, bei denen es nur 3 „nach oben“ gerichtete Erregungen mit aufeinanderfolgenden Abständen von 1, 2, 2 gibt. Dies zeigt, dass sich die Van-der-Waals-Wechselwirkung auch auf die nächsten Nachbarn auswirkt (wenn auch viel geringer).

Läuft auf der Aquila QuEra QPU

Voraussetzungen: Wenn Sie neu bei Amazon Braket sind, stellen Sie neben der Pip-Installation des Braket-SDK sicher, dass Sie die erforderlichen Schritte für den Einstieg abgeschlossen haben.

> **Note**
>
> Wenn Sie eine von Braket gehostete Notebook-Instance verwenden, ist das Braket-SDK zusammen mit der Instance vorinstalliert.

Wenn alle Abhängigkeiten installiert sind, können wir eine Verbindung zur QPU herstellen. Aquila

```
from braket.aws import AwsDevice

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

Damit unser AHS-Programm für die QuEra Maschine geeignet ist, müssen wir alle Werte runden, um die von der Aquila QPU zulässige Genauigkeit einzuhalten. (Diese Anforderungen werden durch die Geräteparameter bestimmt, deren Name „Auflösung“ enthält. Wir können sie sehen, indem wir sie `aquila_qpu.properties.dict()` in einem Notizbuch ausführen. Weitere Informationen zu den Funktionen und Anforderungen von Aquila finden Sie in der Einführung in das Aquila-Notizbuch.) Wir können das tun, indem wir die `discretize` Methode aufrufen.

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

Jetzt können wir das Programm (es laufen vorerst nur 100 Schüsse) auf der Aquila QPU ausführen.

> **Note**
>
> Das Ausführen dieses Programms auf dem Aquila Prozessor ist mit Kosten verbunden. Das Amazon Braket SDK enthält einen Cost Tracker, mit dem Kunden Kostenlimits festlegen und ihre Kosten nahezu in Echtzeit verfolgen können.

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

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

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

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

Aufgrund der großen Varianz, wie lange die Ausführung einer Quantenaufgabe dauern kann (abhängig von Verfügbarkeitsfenstern und QPU-Auslastung), ist es eine gute Idee, den ARN der

Quantenaufgabe zu notieren, damit wir ihren Status zu einem späteren Zeitpunkt mit dem folgenden Codeausschnitt überprüfen können.

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

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

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

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

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

Sobald der Status ABGESCHLOSSEN ist (was auch auf der Quantenaufgaben-Seite der Amazon Braket-Konsole überprüft werden kann), können wir die Ergebnisse abfragen mit:

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

QPU-Ergebnisse analysieren

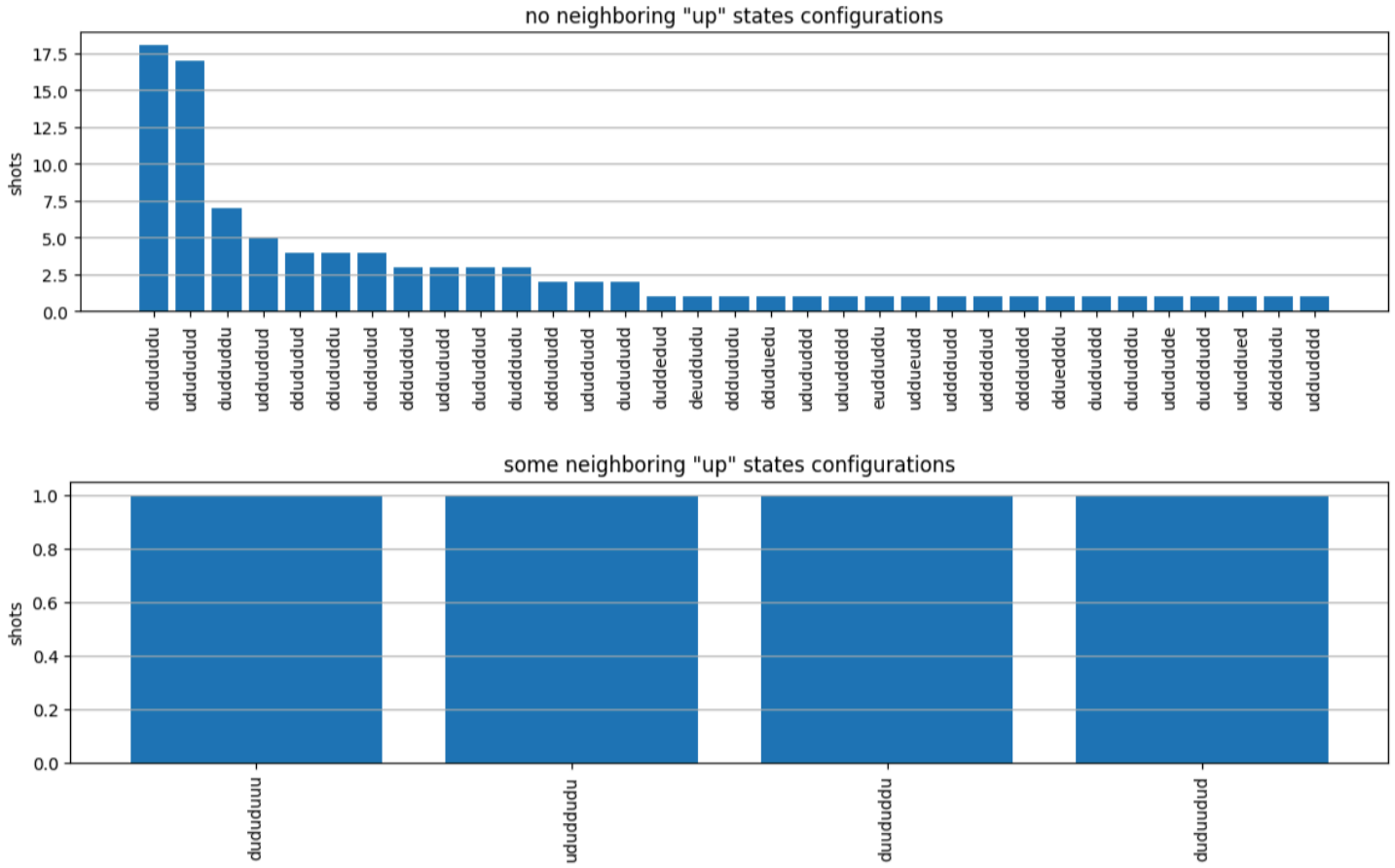
Mit den gleichen `get_counts` Funktionen wie zuvor können wir die Zählungen berechnen:

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

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

und plotten Sie sie mit`plot_counts`:

```
plot_counts(counts_aquila)
```



Beachten Sie, dass ein kleiner Teil der Aufnahmen leere Bereiche hat (mit „e“ gekennzeichnet). Dies ist auf eine Unvollkommenheit der QPU bei der Präparation pro Atom von 1— 2% zurückzuführen. Aquila Abgesehen davon stimmen die Ergebnisse innerhalb der erwarteten statistischen Fluktuation aufgrund der geringen Anzahl von Schüssen mit der Simulation überein.

Nächste Schritte

Herzlichen Glückwunsch, Sie haben jetzt Ihren ersten AHS-Workload auf Amazon Braket mit dem lokalen AHS-Simulator und der Aquila QPU ausgeführt.

Weitere Informationen zur Rydberg-Physik, zur analogen Hamiltonschen Simulation und zum Aquila Gerät finden Sie in unseren Beispiel-Notebooks.

Reichen Sie ein analoges Programm mit Aquila ein QuEra

Diese Seite enthält eine umfassende Dokumentation über die Funktionen der Aquila Maschine vonQuEra. Hier werden die folgenden Details behandelt:

1. Der parametrisierte Hamilton-Operator, simuliert von Aquila

2. AHS-Programmparameter
3. Inhalt der AHS-Ergebnisse
4. AquilaParameter „Fähigkeiten“

In diesem Abschnitt:

- Hamiltonisch
- AHS-Programmschema in Braket
- Ergebnisschema für die AHS-Aufgabe in Braket
- QuEra Schema der Geräteeigenschaften

Hamiltonisch

Die Aquila Maschine von QuEra simuliert nativ den folgenden (zeitabhängigen) Hamilton-Operator:

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

> **Note**
>
> Der Zugriff auf lokale Verstimmung ist eine experimentelle Funktion, die auf Anfrage über Braket Direct verfügbar ist.

where

- H drive,k (t) =($^{1}/_{2}$Ω (t) e $^{i(\ t)}$ S −,k + $^{1}/_{2}$Ω (t) e $^{-iφ\ (t)\ S) + (-δ\ (t_{+,k})}$ n), global k
  - Ω (t) ist die zeitabhängige globale Antriebsamplitude (auch bekannt als Rabi-Frequenz) in Einheiten von (rad/s)
  - φ (t) ist die zeitabhängige, globale Phase, gemessen im Bogenmaß
  - S −,k und S +,k sind die Operatoren zum Herabsetzen und Erhöhen des Spins des Atoms k (in der Basis |↓# =|g#, |↓ = |r#, sie lauten S =|g¤ ¤ r|, S =(S) † =|r-und rcontent/in|g|) − + −
  - δ (t_{global}) ist die zeitabhängige, globale Verstimmung
  - n k ist der Projektionsoperator für den Rydberg-Zustand des Atoms k (das heißt, n=|r¤ ¤ r|)
- H (t) =-Δ (t) h n local detuning,k local k k
  - Δ local (t) ist der zeitabhängige Faktor der lokalen Frequenzverschiebung in Einheiten von (rad/s)

- h_k ist der ortsabhängige Faktor, eine dimensionslose Zahl zwischen 0,0 und 1,0
- $V_{vdw,k,l} = C_6/(d_{k,l})\ 6\ n\ n_k\ n_l$
  - C_6 ist der Van-der-Waals-Koeffizient in Einheiten von (rad/s) * (m) ^6
  - $d_{k,l}$ ist der euklidische Abstand zwischen Atom k und l, gemessen in Metern.

Benutzer haben über das Braket AHS-Programmschema die Kontrolle über die folgenden Parameter.

- 2D-Atomanordnung (X_k - und $_k$ Y-Koordinaten jedes Atoms k, in Einheiten von um), die die paarweisen Atomabstände $d_{k,l}$ mit k steuert, l=1,2,... N
- Ω (t), die zeitabhängige, globale Rabi-Frequenz, in Einheiten von (rad/s)
- φ (t), die zeitabhängige, globale Phase, in Einheiten von (rad)
- Δ_{global} (t), die zeitabhängige, globale Verstimmung, in Einheiten von (rad/s)
- Δ_{local} (t), der zeitabhängige (globale) Faktor für die Größe der lokalen Verstimmung, in Einheiten von (rad/s)
- h_k, der (statische) ortsabhängige Faktor für das Ausmaß der lokalen Verstimmung, eine dimensionslose Zahl zwischen 0,0 und 1,0

> **Note**
>
> Der Benutzer kann weder kontrollieren, um welche Stufen es sich handelt (d. h. die Operatoren S_-, S_+, n sind fest) noch die Stärke des Rydberg-Rydberg Interaktionskoeffizienten (C_6).

AHS-Programmschema in Braket

Braket.IR.AHS.Program_V1.Program-Objekt (Beispiel)

> **Note**
>
> Wenn die Funktion zur lokalen Feinabstimmung für Ihr Konto nicht aktiviert ist, verwenden Sie das folgende Beispiel. `localDetuning=[]`

```
Program(
```

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

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

JSON (Beispiel)

> **Note**
>
> Wenn die Funktion zur lokalen Feinabstimmung für Ihr Konto nicht aktiviert ist, verwenden Sie sie `"localDetuning": []` im folgenden Beispiel.

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

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

Die wichtigsten Felder

Feld Programm	type	description
setup.ahs_register.sites	Liste [Liste [Dezimal]]	Liste der 2D-Koordinaten, an denen

Feld Programm	type	description
		die Pinzette Atome einfängt
setup.ahs_register.filling	Liste [int]	Markiert Atome, die die Fallenste llen besetzen, mit 1 und leere Stellen mit 0
Hamiltonian.DrivingFields [] .amplitude.time_se ries.times	Liste [Dezimal]	Zeitpunkte der Antriebsamplitude, Omega (t)
Hamiltonian.DrivingFields [] .amplitude.time_se ries.values	Liste [Dezimal]	Werte der Antriebsamplitude, Omega (t)
Hamiltonian.DrivingFields [] .amplitude.pattern	str	das räumliche Muster der Antriebsamplitude, Omega (t); muss „einheitlich“ sein
Hamiltonian.DrivingFields [] .phase.time_series.times	Liste [Dezimal]	Zeitpunkte der Fahrphase, phi (t)
Hamiltonian.DrivingFields [] .phase.time_series .values	Liste [Dezimal]	Werte der Antriebsphase, Phi (t)
Hamiltonian.DrivingFields [] .phase.pattern	str	räumliches Muster der Fahrphase, phi (t); muss „einheitl ich“ sein

Feld Programm	type	description
Hamiltonian.DrivingFields [] .detuning.time_series.times	Liste [Dezimal]	Zeitpunkte der Fahrverstimmung, Delta_global (t)
Hamiltonian.DrivingFields [] .detuning.time_series.values	Liste [Dezimal]	Werte der Fahrverstimmung, Delta_global (t)
Hamiltonian.DrivingFields [] .detuning.pattern	str	das räumliche Muster der Fahrverstimmung, (t); muss „einheitlich“ sein Delta_global
Hamiltonian.LocalDeTuning [] .magnitude.time_series.times	Liste [Dezimal]	Zeitpunkte des zeitabhängigen Faktors der lokalen Verstimmungsgröße, Delta_local (t)
Hamiltonian.LocalDeTuning [] .magnitude.time_series.values	Liste [Dezimal]	Werte des zeitabhängigen Faktors der lokalen Verstimmungsgröße, Delta_local (t)

Feld Programm	type	description
Hamiltonian.LocalDeTuning [] .magnitude.pattern	Liste [Dezimal]	ortsabhängiger Faktor der lokalen Verstimmungsgröße, h_k (Werte entsprechen Standorten in setup.ahs_register.sites)

Metadaten-Felder

Feld „Programm“	type	description
Klammer SchemaHeader.name	str	Name des Schemas; muss 'braket.ir.ahs.program' sein
Klammer SchemaHeader.version	str	Version des Schemas

Ergebnisschema für die AHS-Aufgabe in Braket

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQuantumTaskResult(Beispiel)

```
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 (Beispiel)

```
{
    "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
        }
    }
}
```

Hauptfelder

Ergebnisfeld der Aufgabe	type	description
Messungen [] .shot Result.preSequence	Liste [int]	Pre-sequence Messbits (eins für jede atomare Stelle) für jeden Schuss: 0, wenn die Stelle leer ist, 1, wenn die Stelle gefüllt ist, gemessen vor den Impulsseq uenzen, die die Quantenen twicklung bestimmen
Messungen [] .shot Result.postSequence	Liste [int]	Post-sequence Messbits für jeden Schuss: 0, wenn sich das Atom im Rydberg-Zustand befindet

Ergebnisfeld der Aufgabe	type	description
		oder die Stelle leer ist, 1, wenn sich das Atom im Grundzustand befindet, gemessen am Ende der Impulssequenzen, die die Quantenentwicklung bestimmen

Metadaten-Felder

Ergebnisfeld der Aufgabe	type	description
Klammer SchemaHeader.name	str	Name des Schemas; muss 'braket.task_result.analog_hamiltonian_simulation_task_result' sein
Klammer SchemaHeader.version	str	Version des Schemas
Aufgabe Metadata.braketSchemaHeader.name	str	Name des Schemas; muss 'braket.task_result.task_metadata' sein
Aufgabe Metadata.braketSchemaHeader.version	str	Version des Schemas
Aufgabe Metadata.id	str	Die ID der Quantenaufgabe.

Ergebnisfeld der Aufgabe	type	description
		Für AWS Quantenaufgaben ist dies die Quantenaufgabe ARN.
Aufgabe Metadata.shots	int	Die Anzahl der Schüsse für die Quantenaufgabe
Aufgabe Metadata.shots.deviceId	str	Die ID des Geräts, auf dem die Quantenaufgabe ausgeführt wurde. Für AWS Geräte ist dies der Geräte-ARN.

Ergebnisfeld der Aufgabe	type	description
Aufgabe Metadata.shots.createdAt	str	Der Zeitstempel der Erstellung; das Format muss im ISO-8601/RFC3339 Zeichenkettenformat YYYY-MM-DDTHH:mm:ss.sssZ sein. Die Standardeinstellung ist Keine.
Aufgabe Metadata.shots.endedAt	str	Der Zeitstempel, zu dem die Quantenaufgabe beendet wurde. Das Format muss im ISO-8601/RFC3339 Zeichenkettenformat vorliegen YYYY-MM-DDTHH:mm:ss.sssZ. Die Standardeinstellung ist Keine.

Ergebnisfeld der Aufgabe	type	description
Aufgabe Metadata.shots.status	str	Der Status der Quantenaufgabe (ERSTELLT, IN DER WARTESCHLANGE, LÄUFT, ABGESCHLOSSEN, FEHLGESCHLAGEN). Die Standardeinstellung ist Keine.
Aufgabe Metadata.shots.failureReason	str	Der Grund für das Scheitern der Quantenaufgabe. Die Standardeinstellung ist Keine.
zusätzlich Metadata.action	Braket.IR.AHS.Program_V1.Program	(Siehe den Abschnitt zum Braket AHS-Programmschema)
zusätzlich Metadata.action.braketSchemaHeader.queraMetadata.name	str	Name des Schemas; muss 'braket.task_result.quera_metadata' sein

Ergebnisfeld der Aufgabe	type	description
zusätzlich Metadata.action.braketSchemaHeader.queraMetadata.version	str	Version des Schemas
zusätzlich Metadata.action.numSuccessfulShots	int	Anzahl der vollständig erfolgreichen Schüsse; muss der angeforderten Anzahl von Schüssen entsprechen
Messungen [] .shot Metadata.shotStatus	int	Der Status des Schusses (Erfolg, Teilerfolg, Fehlschlag) muss „Erfolg“ lauten

QuEra Schema der Geräteeigenschaften

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

```
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 (Beispiel)

```
{
    "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
        ...
    }
}
```

Felder für Diensteigenschaften

Feld „Serviceeigenschaften“	type	description
service.executionWindows [] .executionDay	ExecutionDay	Tage des Ausführungsfensters; muss 'Jeden Tag', 'Wochentage', 'Wochenende', 'Montag', 'Dienstag', 'Mittwoch', Donnerstag ',' Freitag ',' Samstag 'oder' Sonntag 'sein
service.executionWindows [] .window StartHour	Datum/Uhrzeit.Uhrzeit	UTC-24-Stunden-Format der Uhrzeit, zu der das Ausführungsfenster gestartet wird
service.executionWindows [] .window EndHour	Datum/Uhrzeit.Uhrzeit	UTC-24-Stunden-Format der Uhrzeit, zu der das Ausführungsfenster endet
service.qpu_capabilities.service.shotsRange	Tupel [int, int]	Minimale und maximale Anzahl von Aufnahmen für das Gerät
service.qpu_capabilities.service.deviceCost.price	float	Preis des Geräts in US-Dollar
service.qpu_capabilities.service.deviceCost.unit	str	Einheit zur Berechnung des Preises, z. B.:

Feld „Serviceeigenschaften“	type	description
		'Minute', 'Stunde', 'Schuss', 'Aufgabe'

Metadaten-Felder

Metadaten-Feld	type	description
Aktion [] .version	str	Version des AHS-Programmschemas
Aktion [] .actionType	ActionType	Name des AHS-Programmschemas; muss 'braket.ir.ahs.program' sein
service.braket SchemaHeader.name	str	Name des Schemas; muss 'braket.device_schema.device_service_properties' sein
service.braket SchemaHeader.version	str	Version des Schemas
service.device Documentation.imageUrl	str	URL für das Bild des Geräts
service.device Documentation.summary	str	kurze Beschreibung auf dem Gerät
service.device Documentation.externalDocumentationUrl	str	URL der externen Dokumentation
service.deviceLocation	str	geografischer Standort des Geräts

Metadaten-Feld	type	description
Service.AktualisiertAt	datetime	Zeitpunkt der letzten Aktualisierung der Geräteeigenschaften

Arbeitet mit AWS Boto3

Boto3 ist das AWS SDK für Python. Mit Boto3 können Python-Entwickler beispielsweise Braket erstellen, konfigurieren und verwalten AWS-Services. Amazon Boto3 bietet sowohl objektorientierten als auch API einfachen Zugriff auf Amazon Braket.

Folgen Sie den Anweisungen in der Boto3-Schnellstartanleitung, um zu erfahren, wie Sie Boto3 installieren und konfigurieren.

Boto3 bietet die Kernfunktionen, die zusammen mit dem Amazon Braket Python SDK verwendet werden, um Sie bei der Konfiguration und Ausführung Ihrer Quantenaufgaben zu unterstützen. Python-Kunden müssen immer Boto3 installieren, da dies die Kernimplementierung ist. Wenn Sie zusätzliche Hilfsmethoden verwenden möchten, müssen Sie auch das Amazon Braket SDK installieren.

Wenn Sie beispielsweise anrufen`CreateQuantumTask`, sendet das Amazon Braket SDK die Anfrage an Boto3, das dann den aufruft. AWS API

In diesem Abschnitt:

- Schalten Sie den Amazon Braket Boto3-Client ein
- Konfiguration AWS CLI Profile für Boto3 und das Braket SDK

Schalten Sie den Amazon Braket Boto3-Client ein

Um Boto3 mit Amazon Braket zu verwenden, müssen Sie Boto3 importieren und dann einen Client definieren, den Sie für die Verbindung mit Amazon Braket verwenden. API Im folgenden Beispiel wird der Boto3-Client benannt. `braket`

```
import boto3
import botocore
```

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

> **Note**
>
> Braket unterstützt IPv6. Wenn Sie ein IPv6-only Netzwerk verwenden oder sicherstellen möchten, dass Ihr Workload IPv6-Verkehr verwendet, verwenden Sie die Dual-Stack-Endpunkte, wie im Leitfaden und im FIPS-Endpunkte-Leitfaden beschrieben. Dual-stack

Jetzt, da Sie einen `braket` Kunden eingerichtet haben, können Sie Anfragen stellen und Antworten über den Amazon Braket-Service bearbeiten. Weitere Informationen zu Anfrage- und Antwortdaten finden Sie in der API-Referenz.

Die folgenden Beispiele zeigen, wie man mit Geräten und Quantenaufgaben arbeitet.

- Suchen Sie nach Geräten
- Rufen Sie ein Gerät ab
- Erstellen Sie eine Quantenaufgabe
- Rufen Sie eine Quantenaufgabe ab
- Suchen Sie nach Quantenaufgaben
- Quantenaufgabe abbrechen

Suchen Sie nach Geräten

- `search_devices(**kwargs)`

Sucht mit den angegebenen Filtern nach Geräten.

```
# 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'])
```

Rufen Sie ein Gerät ab

- `get_device(deviceArn)`

Rufen Sie die in Amazon Braket verfügbaren Geräte ab.

```
# 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']}")
```

Erstellen Sie eine Quantenaufgabe

- `create_quantum_task(**kwargs)`

Erstellen Sie eine Quantenaufgabe.

```
# 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")
```

Rufen Sie eine Quantenaufgabe ab

- `get_quantum_task(quantumTaskArn)`

Ruft die angegebene Quantenaufgabe ab.

```
# 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'])
```

Suchen Sie nach Quantenaufgaben

- `search_quantum_tasks(**kwargs)`

Sucht nach Quantenaufgaben, die den angegebenen Filterwerten entsprechen.

```
# 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']}")
```

Quantenaufgabe abbrechen

- `cancel_quantum_task(quantumTaskArn)`

Storniert die angegebene Quantenaufgabe.

```
# 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']}")
```

Konfiguration AWS CLI Profile für Boto3 und das Braket SDK

Das Amazon Braket SDK verwendet die AWS CLI Standardanmeldedaten, sofern Sie nicht ausdrücklich etwas anderes angeben. Wir empfehlen, dass Sie bei der Ausführung auf einem verwalteten Amazon Braket-Notebook die Standardeinstellung beibehalten, da Sie eine IAM-Rolle angeben müssen, die über Berechtigungen zum Starten der Notebook-Instance verfügt.

Wenn Sie Ihren Code lokal ausführen (z. B. auf einer Amazon EC2 EC2-Instance), können Sie optional benannte AWS CLI Profile einrichten. Sie können jedem Profil einen anderen Berechtigungssatz zuweisen, anstatt das Standardprofil regelmäßig zu überschreiben.

In diesem Abschnitt wird kurz erklärt, wie eine solche CLI konfiguriert wird `profile` und wie dieses Profil in Amazon Braket integriert wird, sodass API Aufrufe mit den Berechtigungen dieses Profils getätigt werden.

In diesem Abschnitt:

- Schritt 1: Konfigurieren Sie ein lokales AWS CLI-Profil
- Schritt 2: Richten Sie ein Boto3-Sitzungsobjekt ein
- Schritt 3: Integrieren Sie die Boto3-Sitzung in das Braket AwsSession

Schritt 1: Konfigurieren Sie ein lokales AWS **`CLI-Profil`**

Es würde den Rahmen dieses Dokuments sprengen, zu erklären, wie ein Benutzer erstellt und wie ein nicht standardmäßiges Profil konfiguriert wird. Informationen zu diesen Themen finden Sie unter:

- Erste Schritte

- Konfiguration der AWS CLI zu verwendenden AWS IAM Identity Center

Um Amazon Braket verwenden zu können, müssen Sie diesem Benutzer — und der zugehörigen CLI `profile` — die erforderlichen Braket-Berechtigungen gewähren. Sie können die Richtlinie beispielsweise anhängen. AmazonBraketFullAccess

Schritt 2: Richten Sie ein Boto3-Sitzungsobjekt ein

Verwenden Sie das folgende Codebeispiel, um ein Boto3-Sitzungsobjekt einzurichten.

```
from boto3 import Session

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

> **Note**
>
> Wenn für die erwarteten API Aufrufe Region-based Einschränkungen gelten, die nicht mit Ihrer `profile` Standardregion übereinstimmen, können Sie eine Region für die Boto3-Sitzung angeben, wie im folgenden Beispiel gezeigt.

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

Ersetzen Sie das als `region` angegebene Argument durch einen Wert, der einem der Werte entspricht, AWS-Regionen in denen Amazon Braket verfügbar ist`us-east-1`, z. B.`us-west-1`, usw.

Schritt 3: Integrieren Sie die Boto3-Sitzung in das Braket AwsSession

Das folgende Beispiel zeigt, wie eine Boto3-Braket-Sitzung initialisiert und ein Gerät in dieser Sitzung instanziiert wird.

```
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)
```

Nachdem diese Einrichtung abgeschlossen ist, können Sie Quantenaufgaben an dieses instanziierte `AwsDevice` Objekt senden (indem Sie beispielsweise den Befehl aufrufen). `device.run(…)` Alle von diesem Gerät getätigten API Aufrufe können die IAM-Anmeldeinformationen verwenden, die dem CLI-Profil zugeordnet sind, das Sie zuvor als `profile` angegeben haben.

Testen Sie Ihre Quantenaufgaben mit Amazon Braket

Amazon Braket bietet eine Vielzahl von Hochleistungssimulatoren für Quantenschaltkreise, mit denen Sie Ihre Quantenalgorithmen testen und validieren können, bevor Sie sie auf echter Quantenhardware ausführen. Diese Simulatoren verarbeiten die komplexe zugrundeliegende Software und Infrastruktur sowie Amazon Elastic Compute Cloud (Amazon EC2) -Cluster, um die Simulation von Quantenschaltkreisen in der klassischen HPC-Infrastruktur (High Performance Computing) zu vereinfachen. Diese Ressourcen ermöglichen es Ihnen, sich auf die Entwicklung und Optimierung Ihrer Quantenanwendungen zu konzentrieren.

Mit den Simulatoren von Braket können Sie Ihre Quantenschaltkreise und Algorithmen gründlich testen, ohne die Einschränkungen und Einschränkungen physikalischer Quantengeräte. Auf diese Weise können Sie eine breite Palette von Quantencomputer-Konzepten erforschen, von grundlegenden Quantengattern und -schaltungen bis hin zu fortschrittlicheren Quantenalgorithmen und Techniken zur Fehlerminimierung.

Das Braket SDK vereinfacht die Übertragung Ihrer Quantenaufgaben an die Simulatoren und ermöglicht es Ihnen, die Simulationsparameter wie die Anzahl der Schüsse und das Rauschmodell zu steuern, um das Verhalten Ihrer Quantenalgorithmen besser zu verstehen. Sie können die Funktionen von Amazon Braket Hybrid Job auch verwenden, um klassische und Quantencomputer-Elemente zu kombinieren und so den Umfang Ihrer Tests und Validierungen weiter zu erweitern.

Indem Sie Ihre Quantenaufgaben auf den Simulatoren von Braket gründlich testen, können Sie wertvolle Erkenntnisse gewinnen, Ihre Algorithmen verfeinern und deren Richtigkeit sicherstellen, bevor Sie sie auf echter Quantenhardware einsetzen. Dies hilft, die Entwicklungszeit zu verkürzen, Fehler zu minimieren und letztendlich Ihre Fortschritte auf dem Gebiet des Quantencomputers zu beschleunigen.

In diesem Abschnitt:

- Einreichung von Quantenaufgaben an Simulatoren
- Lokaler Emulator für Quantengeräte

Einreichung von Quantenaufgaben an Simulatoren

Amazon Braket bietet Zugriff auf mehrere Simulatoren, mit denen Sie Ihre Quantenaufgaben testen können. Sie können Quantenaufgaben einzeln einreichen oder mehrere Programme ausführen.

Simulatoren

- Dichtematrixsimulator,: DM1 `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- Zustandsvektorsimulator, SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Tensor-Netzwerksimulator,: TN1 `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- Der lokale Simulator: `LocalSimulator()`

> **Note**
>
> Sie können Quantenaufgaben im `CREATED` Status „Für“ QPUs und „On-Demand-Simulatoren“ stornieren. Für On-Demand-Simulatoren und können Quantenaufgaben im `QUEUED` Status nach bestem Wissen und Gewissen storniert werden. QPUs Beachten Sie, dass es unwahrscheinlich ist, dass `QUEUED` QPU-Quantenaufgaben während der QPU-Verfügbarkeitsfenster erfolgreich storniert werden.

In diesem Abschnitt:

- Vektorsimulator für lokalen Zustand () braket_sv
- Simulator für lokale Dichtematrix () braket_dm
- Lokaler AHS-Simulator () braket_ahs
- Zustandsvektorsimulator () SV1
- Dichtematrix-Simulator (DM1)
- Tensor-Netzwerksimulator () TN1
- Über eingebettete Simulatoren
- Vergleichen Sie Amazon Braket-Simulatoren
- Beispiele für Quantenaufgaben auf Amazon Braket
- Testen einer Quantenaufgabe mit dem lokalen Simulator

Vektorsimulator für lokalen Zustand () `braket_sv`

Der Local State Vector Simulator (`braket_sv`) ist Teil des Amazon Braket-SDK, das lokal in Ihrer Umgebung ausgeführt wird. Er eignet sich gut für schnelles Prototyping auf kleinen Schaltungen (bis zu 25qubits), abhängig von den Hardwarespezifikationen Ihrer Braket-Notebook-Instance oder Ihrer lokalen Umgebung.

Der lokale Simulator unterstützt alle Gates im Amazon Braket SDK, aber QPU-Geräte unterstützen eine kleinere Teilmenge. Sie finden die unterstützten Gates eines Geräts in den Geräteeigenschaften.

> **Note**
>
> Der lokale Simulator unterstützt erweiterte OpenQASM-Funktionen, die auf QPU-Geräten oder anderen Simulatoren möglicherweise nicht unterstützt werden. Weitere Informationen zu den unterstützten Funktionen finden Sie in den Beispielen im OpenQASM Local Simulator-Notizbuch.

Weitere Informationen zur Arbeit mit Simulatoren finden Sie in den Amazon Braket-Beispielen.

Simulator für lokale Dichtematrix () `braket_dm`

Der lokale Dichtematrixsimulator (`braket_dm`) ist Teil des Amazon Braket-SDK, das lokal in Ihrer Umgebung ausgeführt wird. Er eignet sich gut für schnelles Prototyping auf kleinen Schaltungen mit Rauschen (bis zu 12qubits), abhängig von den Hardwarespezifikationen Ihrer Braket-Notebook-Instanz oder Ihrer lokalen Umgebung.

Mithilfe von Gate-Noise-Operationen wie Bit-Flip und Depolarizing-Error können Sie gängige, rauschbehaftete Schaltungen von Grund auf neu aufbauen. Sie können Rauschoperationen auch auf bestimmte qubits Gates vorhandener Schaltungen anwenden, die sowohl mit als auch ohne Rauschen betrieben werden sollen.

Der `braket_dm` lokale Simulator kann die folgenden Ergebnisse liefern, vorausgesetzt, die angegebene Anzahl vonshots:

- Matrix mit reduzierter Dichte: Shots = 0

> **Note**
>
> Der lokale Simulator unterstützt erweiterte OpenQASM-Funktionen, die auf QPU-Geräten oder anderen Simulatoren möglicherweise nicht unterstützt werden. Weitere Informationen zu den unterstützten Funktionen finden Sie in den Beispielen im OpenQASM Local Simulator-Notizbuch.

Weitere Informationen zum lokalen Dichtematrix-Simulator finden Sie im einführenden Beispiel für einen Geräuschsimulator in Braket.

Lokaler AHS-Simulator () `braket_ahs`

Der lokale AHS-Simulator (Analog Hamiltonian Simulation) (`braket_ahs`) ist Teil des Amazon Braket-SDK, das lokal in Ihrer Umgebung ausgeführt wird. Er kann verwendet werden, um Ergebnisse eines AHS-Programms zu simulieren. Es eignet sich gut für das Prototyping auf kleinen Registern (bis zu 10 bis 12 Atome), abhängig von den Hardwarespezifikationen Ihrer Braket-Notebook-Instanz oder Ihrer lokalen Umgebung.

Der lokale Simulator unterstützt AHS-Programme mit einem einheitlichen Antriebsfeld, einem (ungleichmäßigen) Verschiebungsfeld und beliebigen Atomanordnungen. Einzelheiten finden Sie in der Braket AHS-Klasse und im Braket AHS-Programmschema.

Weitere Informationen zum lokalen AHS-Simulator finden Sie auf der Seite Hello AHS: Run your first Analog Hamiltonian Simulation und in den Beispiel-Notebooks Analog Hamiltonian Simulation.

Zustandsvektorsimulator () SV1

SV1ist ein leistungsstarker, universeller Zustandsvektorsimulator auf Abruf. Er kann Schaltungen von bis zu 34 simulierenqubits. Sie können davon ausgehen34-qubit, dass die Fertigstellung eines dichten und quadratischen Stromkreises (Schaltkreistiefe = 34) etwa 1—2 Stunden in Anspruch nimmt, abhängig von der Art der verwendeten Gatter und anderen Faktoren. Schaltungen mit all-to-all Gates eignen sich gut fürSV1. Es gibt Ergebnisse in Formen wie einem vollständigen Zustandsvektor oder einer Reihe von Amplituden zurück.

SV1hat eine maximale Laufzeit von 6 Stunden. Es hat standardmäßig 35 gleichzeitige Quantenaufgaben und maximal 100 (50 in us-west-1 und eu-west-2) gleichzeitige Quantenaufgaben.

SV1Ergebnisse

SV1kann bei gegebener Anzahl von folgenden Ergebnissen die folgenden Ergebnisse liefernshots:

- Beispiel: Shots > 0
- Erwartung: Shots >= 0
- Varianz: >= 0 Shots
- Wahrscheinlichkeit: > 0 Shots
- Amplitude: Shots = 0
- Adjungierter Gradient: Shots = 0

Weitere Informationen zu Ergebnissen finden Sie unter Ergebnistypen.

SV1ist immer verfügbar, führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel ausführen. Die Laufzeit skaliert linear mit der Anzahl der Operationen und exponentiell mit der Anzahl von. qubits Die Anzahl von shots hat einen geringen Einfluss auf die Laufzeit. Weitere Informationen finden Sie unter Simulatoren vergleichen.

Simulatoren unterstützen alle Gates im Braket-SDK, aber QPU-Geräte unterstützen eine kleinere Teilmenge. Sie finden die unterstützten Gates eines Geräts in den Geräteeigenschaften.

Dichtematrix-Simulator (DM1)

DM1ist ein leistungsstarker Dichtematrixsimulator auf Abruf. Er kann Schaltungen von bis zu 17 simulierenqubits.

DM1hat eine maximale Laufzeit von 6 Stunden, eine Standardeinstellung von 35 gleichzeitigen Quantenaufgaben und ein Maximum von 50 gleichzeitigen Quantenaufgaben.

DM1Ergebnisse

DM1kann bei gegebener Anzahl von folgenden Ergebnissen die folgenden Ergebnisse liefernshots:

- Beispiel: Shots > 0
- Erwartung: Shots >= 0
- Varianz: >= 0 Shots
- Wahrscheinlichkeit: > 0 Shots
- Matrix mit reduzierter Dichte: Shots = 0, bis max. 8 qubits

Weitere Informationen zu Ergebnissen finden Sie unter Ergebnistypen.

DM1ist immer verfügbar, führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel ausführen. Die Laufzeit skaliert linear mit der Anzahl der Operationen und exponentiell mit der Anzahl von. qubits Die Anzahl von shots hat einen geringen Einfluss auf die Laufzeit. Weitere Informationen finden Sie unter Simulatoren vergleichen.

Lärmschutzbarrieren und Einschränkungen

```
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]
```

Tensor-Netzwerksimulator () TN1

TN1ist ein On-Demand-Hochleistungs-Tensor-Netzwerksimulator. TN1kann bestimmte Schaltungstypen mit bis zu 50 qubits und einer Schaltungstiefe von 100 oder weniger simulieren. TN1ist besonders leistungsfähig für Schaltungen mit geringer Dichte, Schaltungen mit lokalen Gattern und andere Schaltungen mit spezieller Struktur, wie z. B. Quanten-Fourier-Transformationsschaltungen (QFT). TN1arbeitet in zwei Phasen. In der Probenphase wird zunächst versucht, einen effizienten Rechenpfad für Ihre Schaltung zu ermitteln, sodass die Laufzeit der nächsten Phase, der sogenannten Kontraktionsphase, abgeschätzt werden TN1 kann. Wenn die geschätzte Kontraktionszeit das Laufzeitlimit der TN1 Simulation TN1 überschreitet, wird kein Kontraktionsversuch unternommen.

TN1hat ein Laufzeitlimit von 6 Stunden. Es ist auf maximal 10 (5 in eu-west-2) gleichzeitige Quantenaufgaben begrenzt.

TN1Ergebnisse

Die Kontraktionsphase besteht aus einer Reihe von Matrixmultiplikationen. Die Reihe von Multiplikationen wird fortgesetzt, bis ein Ergebnis erreicht ist oder bis festgestellt wird, dass ein Ergebnis nicht erreicht werden kann.

Hinweis: Shots muss > 0 sein.

Zu den Ergebnistypen gehören:

- Beispiel
- Erwartung
- Varianz

Weitere Informationen zu Ergebnissen finden Sie unter Ergebnistypen.

TN1ist immer verfügbar, führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel ausführen. Weitere Informationen finden Sie unter Simulatoren vergleichen.

Simulatoren unterstützen alle Gates im Braket-SDK, aber QPU-Geräte unterstützen eine kleinere Teilmenge. Sie finden die unterstützten Gates eines Geräts in den Geräteeigenschaften.

Im Amazon GitHub Braket-Repository finden Sie ein TN1 Beispiel-Notizbuch, das Ihnen den Einstieg erleichtern soll. TN1

Bewährte Methoden für die Arbeit mit TN1

- Vermeiden Sie all-to-all Stromkreise.
- Testen Sie eine neue Schaltung oder Klasse von Schaltungen mit einer kleinen Anzahl vonshots, um die „Härte“ des Schaltkreises zu TN1 ermitteln.
- Teilen Sie große shot Simulationen auf mehrere Quantenaufgaben auf.

Über eingebettete Simulatoren

Bei eingebetteten Simulatoren wird die Simulation direkt in den Algorithmuscode eingebettet. Außerdem ist es im selben Container enthalten und führt die Simulation direkt auf der Hybrid-

Job-Instanz aus. Dieser Ansatz ist nützlich, um Engpässe zu beseitigen, die typischerweise bei der Kommunikation zwischen der Simulation und einem Remote-Gerät auftreten. Indem alle Berechnungen in einer einzigen, zusammenhängenden Umgebung gespeichert werden, können eingebettete Simulatoren den Speicherbedarf und die Anzahl der Schaltungsausführungen, die zum Erreichen eines Zielergebnisses erforderlich sind, erheblich reduzieren. Dies kann im Vergleich zu herkömmlichen Konfigurationen, die auf Fernsimulationen basieren, zu erheblichen Leistungsverbesserungen führen, oft um den Faktor zehn oder mehr. Weitere Informationen darüber, wie eingebettete Simulatoren die Leistung verbessern und optimierte Hybrid-Jobs ermöglichen, finden Sie auf der Dokumentationsseite Run a hybrid job with Amazon Braket Hybrid Jobs.

PennyLaneDie Blitzsimulatoren

Sie können die PennyLane Blitzsimulatoren als eingebettete Simulatoren auf Braket verwenden. Mit PennyLane den Blitzsimulatoren können Sie fortschrittliche Methoden zur Berechnung von Gradienten verwenden, wie z. B. die Differenzierung von Adjungten, um Gradienten schneller auszuwerten. Der lightning.qubit-Simulator ist als Gerät über Braket NBIs und als eingebetteter Simulator verfügbar, wohingegen der lightning.gpu-Simulator als eingebetteter Simulator mit einer GPU-Instanz ausgeführt werden muss. Ein Beispiel für die Verwendung von lightning.gpu finden Sie im Notizbuch Integrierte Simulatoren in Braket Hybrid Jobs.

Vergleichen Sie Amazon Braket-Simulatoren

Dieser Abschnitt hilft Ihnen bei der Auswahl des Amazon Braket-Simulators, der für Ihre Quantenaufgabe am besten geeignet ist, indem einige Konzepte, Einschränkungen und Anwendungsfälle beschrieben werden.

Wählen Sie zwischen lokalen Simulatoren und On-Demand-Simulatoren (SV1,,) TN1 DM1

Die Leistung lokaler Simulatoren hängt von der Hardware ab, die die lokale Umgebung hostet, z. B. eine Braket-Notebook-Instanz, die zum Ausführen Ihres Simulators verwendet wird. On-Demand-Simulatoren werden in der AWS Cloud ausgeführt und sind so konzipiert, dass sie über typische lokale Umgebungen hinaus skaliert werden können. On-Demand-Simulatoren sind für größere Schaltungen optimiert, verursachen jedoch einen gewissen Latenzaufwand pro Quantenaufgabe oder Batch von Quantenaufgaben. Dies kann einen Kompromiss bedeuten, wenn es um viele Quantenaufgaben geht. Angesichts dieser allgemeinen Leistungsmerkmale können Ihnen die folgenden Anleitungen bei der Auswahl der Art der Durchführung von Simulationen helfen, auch bei Simulationen mit Rauschen.

Für Simulationen:

- Wenn Sie weniger als 18 Mitarbeiter beschäftigenqubits, verwenden Sie einen lokalen Simulator.
- Wenn Sie 18 bis 24 Mitarbeiter beschäftigenqubits, wählen Sie je nach Arbeitslast einen Simulator aus.
- Wenn Sie mehr als 24 Mitarbeiter beschäftigenqubits, verwenden Sie einen On-Demand-Simulator.

Für Geräuschsimulationen:

- Wenn Sie weniger als 9 einsetzenqubits, verwenden Sie einen lokalen Simulator.
- Wenn Sie 9—12 Mitarbeiter einsetzenqubits, wählen Sie einen Simulator, der auf der Arbeitslast basiert.
- Wenn Sie mehr als 12 Mitarbeiter beschäftigen, verwenden Sie. qubits DM1

Was ist ein Zustandsvektorsimulator?

SV1ist ein universeller Zustandsvektorsimulator. Er speichert die gesamte Wellenfunktion des Quantenzustands und wendet sequentiell Gate-Operationen auf den Zustand an. Es speichert alle Möglichkeiten, auch die extrem unwahrscheinlichen. Die Laufzeit des SV1 Simulators für eine Quantenaufgabe nimmt linear mit der Anzahl der Gates im Schaltkreis zu.

Was ist ein Dichtematrixsimulator?

DM1simuliert Quantenschaltkreise mit Rauschen. Es speichert die vollständige Dichtematrix des Systems und wendet sequentiell die Gatter- und Rauschoperationen der Schaltung an. Die endgültige Dichtematrix enthält vollständige Informationen über den Quantenzustand nach dem Betrieb der Schaltung. Die Laufzeit skaliert im Allgemeinen linear mit der Anzahl der Operationen und exponentiell mit der Anzahl von. qubits

Was ist ein Tensor-Netzwerksimulator?

TN1kodiert Quantenschaltkreise in einen strukturierten Graphen.

- Die Knoten des Graphen bestehen aus Quantengattern, oderqubits.
- Die Kanten des Graphen stellen Verbindungen zwischen Gates dar.

Als Ergebnis dieser Struktur TN1 können simulierte Lösungen für relativ große und komplexe Quantenschaltkreise gefunden werden.

TN1benötigt zwei Phasen

TN1Arbeitet in der Regel in einem zweiphasigen Ansatz zur Simulation von Quantenberechnungen.

- Die Probenphase: In dieser Phase wird eine Möglichkeit TN1 entwickelt, den Graphen auf effiziente Weise zu durchqueren. Dazu müssen Sie jeden Knoten besuchen, um die gewünschte Messung zu erhalten. Als Kunde sehen Sie diese Phase nicht, da beide Phasen gemeinsam für Sie TN1 durchgeführt werden. Sie schließt die erste Phase ab und entscheidet anhand praktischer Einschränkungen, ob die zweite Phase eigenständig durchgeführt werden soll. Sie haben keinen Einfluss auf diese Entscheidung, nachdem die Simulation begonnen hat.
- Die Kontraktionsphase: Diese Phase entspricht der Ausführungsphase einer Berechnung in einem klassischen Computer. Die Phase besteht aus einer Reihe von Matrixmultiplikationen. Die Reihenfolge dieser Multiplikationen hat einen großen Einfluss auf die Schwierigkeit der Berechnung. Daher wird zunächst die Probenphase durchgeführt, um die effektivsten Berechnungspfade im Graphen zu finden. Nachdem der Kontraktionspfad während der Probenphase ermittelt wurde, TN1 werden die Gates Ihres Schaltkreises zusammengezogen, um die Ergebnisse der Simulation zu erhalten.

TN1Graphen entsprechen einer Karte

Metaphorisch gesehen können Sie das zugrundeliegende TN1 Diagramm mit den Straßen einer Stadt vergleichen. In einer Stadt mit einem geplanten Raster ist es einfach, mithilfe einer Karte eine Route zu Ihrem Ziel zu finden. In einer Stadt mit ungeplanten Straßen, doppelten Straßennamen usw. kann es schwierig sein, auf einer Karte eine Route zu Ihrem Ziel zu finden.

Wenn Sie die Probenphase TN1 nicht durchführen würden, wäre es, als würden Sie durch die Straßen der Stadt laufen, um Ihr Ziel zu finden, anstatt zuerst auf eine Karte zu schauen. In Bezug auf die Gehzeit kann es sich wirklich auszahlen, mehr Zeit damit zu verbringen, auf die Karte zu schauen. Ebenso liefert die Probenphase wertvolle Informationen.

Man könnte sagen, dass der ein gewisses „Bewusstsein“ für die Struktur des zugrundeliegenden Stromkreises TN1 hat, den er durchquert. Dieses Bewusstsein erlangt es während der Probenphase.

Problemtypen, die für jeden dieser Simulatortypen am besten geeignet sind

SV1eignet sich gut für alle Arten von Problemen, die hauptsächlich auf einer bestimmten Anzahl von qubits UND-Gattern beruhen. Im Allgemeinen wächst die benötigte Zeit linear mit der Anzahl der Gates, obwohl sie nicht von der Anzahl der Gates abhängt. shots SV1ist im Allgemeinen schneller als TN1 bei Schaltungen unter 28. qubits

SV1kann bei höheren qubit Zahlen langsamer sein, weil es tatsächlich alle Möglichkeiten simuliert, auch die extrem unwahrscheinlichen. Es gibt keine Möglichkeit zu bestimmen, welche Ergebnisse wahrscheinlich sind. Für eine 30-qubit Auswertung SV1 müssen also 2^30 Konfigurationen berechnet werden. Das Limit von 34 qubits für den Amazon SV1 Braket-Simulator ist eine praktische Einschränkung aufgrund von Speicher- und Speicherbeschränkungen. Sie können sich das so vorstellen: Jedes Mal, wenn Sie ein qubit zu hinzufügenSV1, wird das Problem doppelt so schwierig.

TN1Kann bei vielen Problemklassen viel größere Schaltungen in realistischer Zeit auswerten, als SV1 weil es die Struktur des Graphen TN1 ausnutzt. Es verfolgt im Wesentlichen die Entwicklung von Lösungen von Anfang an und behält nur die Konfigurationen bei, die zu einer effizienten Bearbeitung beitragen. Anders ausgedrückt: Es speichert die Konfigurationen, um eine Reihenfolge der Matrixmultiplikation zu erstellen, die zu einem einfacheren Bewertungsprozess führt.

Denn TN1 die Anzahl der qubits UND-Gatter ist wichtig, aber die Struktur des Graphen ist viel wichtiger. eignet TN1 sich zum Beispiel sehr gut zur Auswertung von Schaltungen (Graphen), in denen die Gatter eine geringe Reichweite haben (d. h. jedes Tor qubit ist nur mit seinem nächsten Nachbarn verbundenqubits), und Schaltungen (Graphen), in denen die Verbindungen (oder Gatter) eine ähnliche Reichweite haben. Ein typischer Bereich für TN1 ist, dass jeder nur qubit mit anderen sprichtqubits, die 5 qubits entfernt sind. Wenn der Großteil der Struktur in einfachere Beziehungen wie diese zerlegt werden kann, die in mehr, kleineren oder einheitlicheren Matrizen dargestellt werden können, ist die TN1 Auswertung effizient.

Einschränkungen von TN1

TN1kann langsamer sein als SV1 je nach struktureller Komplexität des Graphen. TN1Beendet bei bestimmten Grafiken die Simulation nach der Probenphase und zeigt den Status an, und zwar aus `FAILED` einem der beiden folgenden Gründe:

- Es kann kein Pfad gefunden werden — Wenn der Graph zu komplex ist, ist es zu schwierig, einen guten Durchlaufpfad zu finden, und der Simulator gibt die Berechnung auf. TN1kann die Kontraktion nicht durchführen. Möglicherweise wird eine Fehlermeldung ähnlich der folgenden angezeigt: `No viable contraction path found.`
- Die Kontraktionsphase ist zu schwierig — In einigen Grafiken TN1 kann ein Durchlaufpfad gefunden werden, aber er ist sehr lang und extrem zeitaufwändig zu bewerten. In diesem Fall ist die Kontraktion so teuer, dass sie unerschwinglich wäre und stattdessen nach der Probenphase beendet wird. TN1 Möglicherweise wird eine Fehlermeldung ähnlich der folgenden angezeigt: `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> Ihnen wird die Probenphase in Rechnung gestellt, TN1 auch wenn die Kontraktion nicht durchgeführt wird und Sie einen Status sehen. `FAILED`

Die prognostizierte Laufzeit hängt auch von der Anzahl ab. shot Im schlimmsten Fall hängt die TN1 Kontraktionszeit linear von der Anzahl ab. shot Der Stromkreis kann mit weniger zusammengezogen werden. shots Sie könnten zum Beispiel eine Quantenaufgabe mit 100 einreichenshots, die dann TN1 entscheidet, dass sie nicht zusammengezogen werden kann. Wenn Sie jedoch erneut eine Aufgabe mit nur 10 einreichen, wird die Kontraktion fortgesetzt. In diesem Fall könnten Sie, um 100 Proben zu erhalten, 10 Quantenaufgaben mit jeweils 10 Proben shots für denselben Schaltkreis einreichen und die Ergebnisse am Ende kombinieren.

Als bewährte Methode empfehlen wir, dass Sie Ihre Schaltung oder Schaltungsklasse immer mit einigen shots (z. B. 10) testen, um herauszufinden, wie schwer Ihre Schaltung istTN1, bevor Sie mit einer höheren Anzahl von shots fortfahren.

> **Note**
>
> Die Reihe von Multiplikationen, die die Kontraktionsphase bilden, beginnt mit kleinen NxN-Matrizen. Ein 2-qubit Gate benötigt beispielsweise eine 4x4-Matrix. Die Zwischenmatrizen, die bei einer als zu schwierig eingestuften Kontraktion benötigt werden, sind gigantisch. Eine solche Berechnung würde Tage in Anspruch nehmen. Aus diesem Grund versucht Amazon Braket keine extrem komplexen Kontraktionen.

Concurrency (Nebenläufigkeit)

Alle Braket-Simulatoren bieten Ihnen die Möglichkeit, mehrere Schaltungen gleichzeitig auszuführen. Die Grenzwerte für Parallelität variieren je nach Simulator und Region. Weitere Informationen zu Parallelitätslimits finden Sie auf der Seite Kontingente.

Beispiele für Quantenaufgaben auf Amazon Braket

In diesem Abschnitt werden die einzelnen Phasen der Ausführung einer Beispiel-Quantenaufgabe beschrieben, von der Auswahl des Geräts bis hin zur Anzeige des Ergebnisses. Als bewährte Methode für Amazon Braket empfehlen wir, dass Sie die Schaltung zunächst auf einem Simulator ausführen, z. B. SV1

In diesem Abschnitt:

- Geben Sie das Gerät an
- Reichen Sie ein Beispiel für eine Quantenaufgabe ein
- Reichen Sie eine parametrisierte Aufgabe ein
- Angeben der shots
- Umfrage nach Ergebnissen
- Sehen Sie sich die Beispielergebnisse an

Geben Sie das Gerät an

Wählen und spezifizieren Sie zunächst das Gerät für Ihre Quantenaufgabe. Dieses Beispiel zeigt, wie Sie den Simulator auswählen,SV1.

```
from braket.aws import AwsDevice

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

Sie können einige Eigenschaften dieses Geräts wie folgt anzeigen:

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

```
SV1
('version', ['1.0', '1.1'])
('actionType', 'braket.ir.jaqcd.program')
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
 'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', 'ecr', 'h', 'i', 'iswap',
 'pswap', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 'swap', 't', 'ti', 'unitary', 'v',
 'vi', 'x', 'xx', 'xy', 'y', 'yy', 'z', 'zz'])
('supportedResultTypes', [ResultType(name='Sample', observables=['x', 'y', 'z', 'h',
 'i', 'hermitian'], minShots=1, maxShots=100000), ResultType(name='Expectation',
 observables=['x', 'y', 'z', 'h', 'i', 'hermitian'], minShots=0, maxShots=100000),
 ResultType(name='Variance', observables=['x', 'y', 'z', 'h', 'i', 'hermitian'],
 minShots=0, maxShots=100000), ResultType(name='Probability', observables=None,
 minShots=1, maxShots=100000), ResultType(name='Amplitude', observables=None,
 minShots=0, maxShots=0)])
```

```
('disabledQubitRewiringSupported', None)
```

Reichen Sie ein Beispiel für eine Quantenaufgabe ein

Reichen Sie eine Beispiel-Quantenaufgabe ein, die auf dem On-Demand-Simulator ausgeführt werden soll.

```
from braket.circuits import Circuit, Observable

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

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

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

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

Der `device.run()` Befehl erstellt über die `CreateQuantumTask` API eine Quantenaufgabe. Nach einer kurzen Initialisierungszeit wird die Quantenaufgabe in die Warteschlange gestellt, bis die Kapazität zur Ausführung der Quantenaufgabe auf einem Gerät vorhanden ist. In diesem Fall ist das Gerät. SV1 Nachdem das Gerät die Berechnung abgeschlossen hat, schreibt Amazon Braket die Ergebnisse an den im Anruf angegebenen Amazon S3 S3-Standort. Das Positionsargument `s3_location` ist für alle Geräte außer dem lokalen Simulator erforderlich.

> **Note**
>
> Die Braket-Quanten-Task-Aktion ist auf eine Größe von 5 MB begrenzt.

Reichen Sie eine parametrisierte Aufgabe ein

Amazon Braket On-Demand-Simulatoren und lokale Simulatoren und unterstützt QPUs auch die Angabe von Werten freier Parameter bei der Aufgabenübergabe. Sie können dies tun, indem Sie das `inputs` Argument to verwenden`device.run()`, wie im folgenden Beispiel gezeigt. `inputs`Es muss sich um ein Wörterbuch mit String-Float-Paaren handeln, wobei die Schlüssel die Parameternamen sind.

Die parametrische Kompilierung kann die Leistung bei der Ausführung parametrischer Schaltungen in bestimmten Fällen verbessern. QPUs Wenn eine parametrische Schaltung als Quantenaufgabe an eine unterstützte QPU gesendet wird, kompiliert Braket die Schaltung einmal und speichert das Ergebnis im Cache. Für nachfolgende Parameteraktualisierungen derselben Schaltung ist keine Neukompilierung erforderlich, was zu schnelleren Laufzeiten für Aufgaben führt, die dieselbe Schaltung verwenden. Braket verwendet bei der Kompilierung Ihrer Schaltung automatisch die aktualisierten Kalibrierungsdaten des Hardwareanbieters, um Ergebnisse von höchster Qualität zu gewährleisten.

> **Note**
>
> Die parametrische Kompilierung wird auf allen supraleitenden Gate-basierten Formularen unterstützt, Rigetti Computing mit Ausnahme QPUs von Pulspegelprogrammen.

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

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

# Create a circuit with a result type
circ = Circuit().rx(0, alpha).ry(1, alpha).cnot(0, 2).xx(0, 2, beta)
circ.variance(observable=Observable.Z(), target=0)

# Add another result type
circ.probability(target=[0, 2])

# Submit the quantum task to run
my_task = device.run(circ, inputs={'alpha': 0.1, 'beta': 0.2}, shots=100)
```

Angeben der shots

Das shots Argument bezieht sich auf die Anzahl der gewünschten Messungenshots. Simulatoren wie SV1 unterstützen zwei Simulationsmodi.

- Für shots = 0 führt der Simulator eine exakte Simulation durch und gibt die wahren Werte für alle Ergebnistypen zurück. (Nicht verfügbar amTN1.)
- Bei Werten von ungleich Null shots verwendet der Simulator Stichproben aus der Ausgangsverteilung, um das reale shot Rauschen zu emulieren. QPUs QPU-Geräte erlauben shots nur > 0.

Informationen zur maximalen Anzahl von Schüssen pro Quantenaufgabe finden Sie unter Braket Quotas.

Umfrage nach Ergebnissen

Bei der Ausführung `my_task.result()` beginnt das SDK mit der Abfrage nach einem Ergebnis mit den Parametern, die Sie bei der Erstellung der Quantenaufgabe definiert haben:

- `poll_timeout_seconds`ist die Anzahl der Sekunden, nach der die Quantenaufgabe abgefragt werden muss, bevor sie bei der Ausführung der Quantenaufgabe auf dem On-Demand-Simulator und/oder den QPU-Geräten abläuft. Der Standardwert ist 432.000 Sekunden, was 5 Tagen entspricht.
- Hinweis: Für QPU-Geräte wie Rigetti und empfehlen wirIonQ, einige Tage einzuplanen. Wenn Ihr Abfrage-Timeout zu kurz ist, werden die Ergebnisse möglicherweise nicht innerhalb der Abfragezeit zurückgegeben. Wenn beispielsweise eine QPU nicht verfügbar ist, wird ein lokaler Timeout-Fehler zurückgegeben.
- `poll_interval_seconds`ist die Frequenz, mit der die Quantenaufgabe abgefragt wird. Sie gibt an, wie oft Sie Braket aufrufen, API um den Status abzurufen, wenn die Quantenaufgabe auf dem On-Demand-Simulator und auf QPU-Geräten ausgeführt wird. Der Standardwert ist 1 Sekunde.

Diese asynchrone Ausführung erleichtert die Interaktion mit QPU-Geräten, die nicht immer verfügbar sind. Beispielsweise könnte ein Gerät während eines regulären Wartungsfensters nicht verfügbar sein.

Das zurückgegebene Ergebnis enthält eine Reihe von Metadaten, die mit der Quantenaufgabe verknüpft sind. Sie können das Messergebnis mit den folgenden Befehlen überprüfen:

```
print('Measurement results:\n', result.measurements)
print('Counts for collapsed states:\n', result.measurement_counts)
print('Probabilities for collapsed states:\n', result.measurement_probabilities)
```

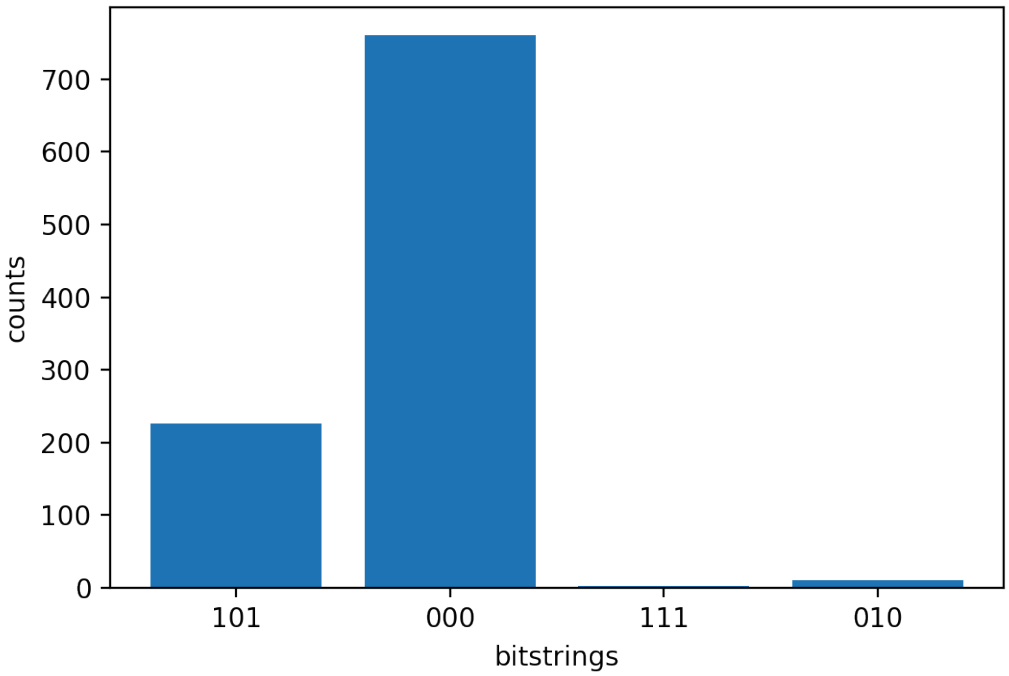
```
Measurement results:
 [[1 0 1]
 [0 0 0]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [1 0 1]]
Counts for collapsed states:
 Counter({'000': 766, '101': 220, '010': 11, '111': 3})
Probabilities for collapsed states:
 {'101': 0.22, '000': 0.766, '010': 0.011, '111': 0.003}
```

Sehen Sie sich die Beispielergebnisse an

Da Sie auch die angegeben haben`ResultType`, können Sie die zurückgegebenen Ergebnisse anzeigen. Die Ergebnistypen werden in der Reihenfolge angezeigt, in der sie dem Schaltkreis hinzugefügt wurden.

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

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

```
Result types include:
 [ResultTypeValue(type=Variance(observable=['z'], targets=[0], type=<Type.variance:
 'variance'>), value=0.693084), ResultTypeValue(type=Probability(targets=[0, 2],
 type=<Type.probability: 'probability'>), value=array([0.777, 0.   , 0.   , 0.223]))]
Variance= 0.693084
Probability= [0.777 0.    0.    0.223]
Text(0, 0.5, 'counts')
```



Testen einer Quantenaufgabe mit dem lokalen Simulator

Sie können Quantenaufgaben direkt an einen lokalen Simulator senden, um Prototypen schnell zu entwickeln und zu testen. Dieser Simulator wird in Ihrer lokalen Umgebung ausgeführt, sodass Sie keinen Amazon S3 S3-Standort angeben müssen. Die Ergebnisse werden direkt in Ihrer Sitzung berechnet. Um eine Quantenaufgabe auf dem lokalen Simulator auszuführen, müssen Sie nur den shots Parameter angeben.

> **Note**
>
> Die Ausführungsgeschwindigkeit und die maximale Anzahl, qubits die der lokale Simulator verarbeiten kann, hängen vom Amazon Braket-Notebook-Instance-Typ oder von Ihren lokalen Hardwarespezifikationen ab.

Die folgenden Befehle sind alle identisch und instanziieren den lokalen State-Vector-Simulator (geräuschfrei).

```
# 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")
```

Führen Sie dann eine Quantenaufgabe mit dem Folgenden aus.

```
my_task = device.run(circ, shots=1000)
```

Um den Simulator mit lokaler Dichtematrix (Rauschen) zu instanziieren, ändern Kunden das Backend wie folgt.

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

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

Messung bestimmter Qubits auf dem lokalen Simulator

Der lokale Zustandsvektorsimulator und der lokale Dichtematrixsimulator unterstützen laufende Schaltungen, bei denen eine Teilmenge der Qubits des Schaltkreises gemessen werden kann, was oft als Teilmessung bezeichnet wird.

Im folgenden Code können Sie beispielsweise eine Schaltung mit zwei Qubits erstellen und nur das erste Qubit messen, indem Sie am Ende der Schaltung eine `measure` Anweisung mit den Ziel-Qubits hinzufügen.

```
# 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)
```

Lokaler Emulator für Quantengeräte

Mit dem lokalen Emulator-Tool von Amazon Braket können Sie Ihre wörtlichen Quantenprogramme lokal emulieren, bevor Sie sie auf echter Quantenhardware ausführen. Der Emulator verwendet Gerätekalibrierungsdaten, um die verbatim-Schaltungen zu validieren, sodass Sie Kompatibilitätsprobleme früher erkennen können.

Darüber hinaus simuliert der lokale Emulator Quanten-Hardware-Rauschen mithilfe des folgenden Prozesses:

- Verwendung von Gerätekalibrierungsdaten zur Konstruktion des Geräuschmodells
- Wenden Sie depolarisierendes Rauschen auf jedes Gate in Ihrem Stromkreis an
- Wird ein Auslesefehler am Ende Ihres Stromkreises angewendet
- Simulation des verrauschten Stromkreises mit einem lokalen Dichtematrixsimulator

Weitere Informationen zur Verwendung eines lokalen Emulators finden Sie im Repository unter Lokale Emulation für verbatim Circuits auf Amazon Braket. amazon-braket-examples GitHub

In diesem Abschnitt:

- Vorteile der lokalen Emulation
- Erstellen Sie einen lokalen Emulator

Vorteile der lokalen Emulation

- Validieren Sie anhand von Kalibrierungsdaten in Echtzeit oder in der Vergangenheit erstellte Schaltkreise anhand von Geräteeinschränkungen im Wortlaut.
- Debuggen Sie Probleme, bevor Sie Aufgaben an Quantenhardware senden.
- Vergleichen Sie geräuschlose und verrauschte Emulationen mit Hardwareergebnissen, um die Geräuscheffekte zu verstehen.
- Optimieren Sie den Arbeitsablauf bei der Entwicklung rauschsensitiver Quantenalgorithmen.

Erstellen Sie einen lokalen Emulator

Ein lokaler Emulator für Quantengeräte kann direkt aus einem Quantengerät oder einer Reihe von Geräteeigenschaften erstellt werden. Bei der direkten Emulation eines Geräts verwendet der Emulator die neuesten Kalibrierungsdaten des instanziierten Geräts. Das folgende Codebeispiel zeigt, wie ein Gerät direkt emuliert wird. Rigetti's Ankaa-3

```
from braket.aws.aws_device import AwsDevice

ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_emulator = ankaa3.emulator()
```

Das folgende Beispiel zeigt die Erstellung eines lokalen Geräteemulators aus einer Reihe von Ankaa-3 Geräteeigenschaften im JSON-Format.

```
from braket.aws import AwsDevice
from braket.emulation.local_emulator import LocalEmulator
import json

# Instantiate the device
ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_properties = ankaa3.properties

# Put the Ankaa-3 properties in a file named ankaa3_device_properties.json
with open("ankaa3_device_properties.json", "w") as f:
    json.dump(ankaa3_properties.json(), f)

# Load the json into the ankaa3_data_json variable
with open("ankaa3_device_properties.json", "r") as json_file:
    ankaa3_data_json = json.load(json_file)

# Create the Ankaa-3 local emulator from the json file you created
ankaa3_emulator = LocalEmulator.from_json(ankaa3_data_json)
```

Sie können das Beispiel wie folgt anpassen:

- Verwenden von Eigenschaften eines anderen QPU-Geräts
- Angabe einer anderen JSON-Datei für den Emulator
- Ändern der Werte der Geräteeigenschaften vor der Instanziierung des Emulators

Ausführen Ihrer Quantenaufgaben mit Amazon Braket

Braket bietet sicheren On-Demand-Zugriff auf verschiedene Arten von Quantencomputern. Sie haben Zugriff auf Gate-basierte Quantencomputer vonAQT, IonQ IQMRigetti, und sowie auf einen analogen Hamilton-Simulator von. QuEra Sie haben auch keine Vorabverpflichtung und müssen sich den Zugang nicht über einzelne Anbieter sichern.

- Die Amazon Braket-Konsole bietet Geräteinformationen und Status, um Sie bei der Erstellung, Verwaltung und Überwachung Ihrer Ressourcen und Quantenaufgaben zu unterstützen.
- Senden Sie Quantenaufgaben über das Amazon Braket Python SDK sowie über die Konsole und führen Sie sie aus. Auf das SDK kann über vorkonfigurierte Amazon Braket-Notebooks zugegriffen werden.
- Auf die Amazon Braket-API kann über das Amazon Braket Python SDK und Notebooks zugegriffen werden. Sie können direkt das aufrufen, API wenn Sie Anwendungen entwickeln, die mit Quantencomputern programmgesteuert arbeiten.

Die Beispiele in diesem Abschnitt zeigen, wie Sie API direkt mit Amazon Braket arbeiten können, indem Sie das Amazon Braket Python SDK zusammen mit dem AWS Python SDK für Braket (Boto3) verwenden.

Mehr über das Amazon Braket Python SDK

Um mit dem Amazon Braket Python SDK zu arbeiten, installieren Sie zunächst das AWS Python SDK für Braket (Boto3), damit Sie mit dem kommunizieren können. AWS API Sie können sich das Amazon Braket Python SDK als praktischen Wrapper rund um Boto3 für Quantenkunden vorstellen.

- Boto3 enthält Schnittstellen, auf die Sie zugreifen müssen. AWS API (Beachten Sie, dass Boto3 ein großes Python-SDK ist, das mit dem kommuniziert. AWS API Die meisten AWS-Services unterstützen eine Boto3-Schnittstelle.)
- Das Amazon Braket Python SDK enthält Softwaremodule für Schaltungen, Gatter, Geräte, Ergebnistypen und andere Teile einer Quantenaufgabe. Jedes Mal, wenn Sie ein Programm erstellen, importieren Sie die Module, die Sie für diese Quantenaufgabe benötigen.
- Auf das Amazon Braket Python SDK kann über Notebooks zugegriffen werden, auf denen alle Module und Abhängigkeiten vorinstalliert sind, die Sie für die Ausführung von Quantenaufgaben benötigen.

- Sie können Module aus dem Amazon Braket Python SDK in jedes Python-Skript importieren, wenn Sie nicht mit Notebooks arbeiten möchten.

Nachdem Sie Boto3 installiert haben, sieht eine Übersicht der Schritte zum Erstellen einer Quantenaufgabe über das Amazon Braket Python SDK wie folgt aus:

1. (Optional) Öffnen Sie Ihr Notizbuch.
2. Importieren Sie die SDK-Module, die Sie für Ihre Schaltungen benötigen.
3. Geben Sie eine QPU oder einen Simulator an.
4. Instanziieren Sie die Schaltung.
5. Führen Sie die Schaltung aus.
6. Sammle die Ergebnisse.

Die Beispiele in diesem Abschnitt zeigen Details zu den einzelnen Schritten.

Weitere Beispiele finden Sie im Amazon Braket Examples Repository unter GitHub.

In diesem Abschnitt:

- Quantenaufgaben an QPUs einreichen
- Mehrere Programme ausführen
- Wann wird meine Quantenaufgabe ausgeführt?
- Mit Reservierungen arbeiten
- Techniken zur Fehlerminimierung

Quantenaufgaben an QPUs einreichen

Amazon Braket bietet Zugriff auf mehrere Geräte, die Quantenaufgaben ausführen können. Sie können Quantenaufgaben einzeln einreichen oder die Batchverarbeitung von Quantenaufgaben einrichten.

Quantenverarbeitungseinheiten (QPUs)

Sie können Quantenaufgaben jederzeit an QPUs senden, aber die Aufgabe wird innerhalb bestimmter Verfügbarkeitsfenster ausgeführt, die auf der Geräteseite der Amazon Braket-Konsole

angezeigt werden. Sie können die Ergebnisse der Quantenaufgabe mit der Quantenaufgaben-ID abrufen, die im nächsten Abschnitt vorgestellt wird.

- AQT IBEX-Q1 : `arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1`
- IonQ Forte-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1`
- IonQ Forte-Enterprise-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1`
- IQM Garnet : `arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet`
- IQM Emerald : `arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Ankaa-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3`
- Rigetti Cepheus-1-108Q : `arn:aws:braket:us-west-1::device/qpu/rigetti/Cepheus-1-108Q`

> **Note**
>
> Sie können Quantenaufgaben im `CREATED` Status für QPUs und On-Demand-Simulatoren stornieren. Für On-Demand-Simulatoren und QPUs können Sie Quantenaufgaben im `QUEUED` Status nach bestem Wissen stornieren. Beachten Sie, dass es unwahrscheinlich ist, dass `QUEUED` QPU-Quantenaufgaben während der QPU-Verfügbarkeitsfenster erfolgreich storniert werden.

In diesem Abschnitt:

- AQT
- IonQ
- IQM
- Rigetti
- QuEra
- Beispiel: Eine Quantenaufgabe an eine QPU einreichen
- Inspektion kompilierter Schaltungen

AQT

AQTDie IBEX-Q1 QPU basiert auf einem Kristall aus $^{40}Ca^{+}$ -Ionen in einer makroskopischen Hochfrequenzfalle, die sich in einer Ultrahochvakuumkammer befindet. Das Gerät läuft bei Raumtemperatur und passt in zwei 19-Zoll-Racks, die für Rechenzentren geeignet sind.

High-fidelity Gates werden durch die niedrigen Heizraten der Falle und die Verwendung eines direkten optischen Übergangs für die Qubit-Rotation ermöglicht. Der Qubit-Übergang wird durch einen Laser mit schmaler Linienbreite und sehr hoher relativer Frequenzstabilität angetrieben. Die Qubits zeichnen sich außerdem durch eine effiziente Zustandserfassung und -auslesung durch optische Ablage aus. All-to-all Die Konnektivität wird durch die weitreichende Coulomb-Wechselwirkung im Ionenkristall erreicht. Single-ion Adressierung und Auslesung werden durch die Verwendung einer Linse mit hoher numerischer Apertur erreicht.

Das AQT Gerät unterstützt die folgenden Quantengatter.

```
'ccnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01', 'cphaseshift10',
 'cswap', 'swap', 'iswap', 'pswap', 'ecr', 'cy', 'cz', 'xy', 'xx', 'yy', 'zz', 'h',
 'i', 'phaseshift', 'rx', 'ry', 'rz', 's', 'si', 't', 'ti', 'v', 'vi', 'x', 'y', 'z',
 'prx'
```

Bei wörtlicher Kompilierung unterstützt das AQT Gerät die folgenden systemeigenen Gatter.

```
'prx', 'xx', 'rz'
```

> **Note**
>
> Im Folgenden werden äquivalente Gates zwischen AQT nativen Gates und Amazon Braket beschrieben:
>
> - Das Tor von AQT Mølmer-Sørensen (MS oder RXX) entspricht dem Tor von Braket `'xx'`
> - Das AQT R-Gate entspricht dem Braket-Gate. `'prx'`
> - Die `'rz'` Gate-Benennung ist dieselbe

IonQ

IonQbietet Gate-basierte QPUs, die auf der Ionenfallentechnologie basieren. IonQ'sQPUs mit gefangenen Ionen bestehen aus einer Kette gefangener 171Yb+-Ionen, die durch eine mikrogefertigte Oberflächenelektrodenfalle in einer Vakuumkammer räumlich begrenzt werden.

IonQGeräte unterstützen die folgenden Quantengatter.

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

Bei wörtlicher Kompilierung unterstützen die IonQ QPUs die folgenden nativen Gatter.

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

Wenn Sie bei der Verwendung des nativen MS-Gates nur zwei Phasenparameter angeben, läuft ein vollständig verschränktes MS-Gate. Ein vollständig verschränktes MS-Gate führt immer eine π /2-Drehung durch. Um einen anderen Winkel anzugeben und ein teilweise verschränktes MS-Gatter auszuführen, geben Sie den gewünschten Winkel an, indem Sie einen dritten Parameter hinzufügen. Weitere Informationen finden Sie im Modul braket.circuits.gate.

Diese systemeigenen Gates können nur bei wörtlicher Kompilierung verwendet werden. Weitere Informationen zur wörtlichen Kompilierung finden Sie unter Verbatim-Kompilierung.

IQM

IQMQuantenprozessoren sind Geräte mit universellen Gate-Modellen, die auf supraleitenden Transmon-Qubits basieren. Dabei IQM Garnet handelt es sich um ein 20-Qubit-Gerät, während es sich um ein 54-Qubit-Gerät handelt. IQM Emerald Beide Geräte verwenden eine quadratische Gittertopologie, die auch als Kristallgittertopologie bezeichnet wird.

Die IQM Geräte unterstützen die folgenden Quantengatter.

```
"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"
```

Bei wörtlicher Kompilierung unterstützen die IQM Geräte die folgenden systemeigenen Gatter.

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

Rigetti

RigettiQuantenprozessoren sind universell einsetzbare Maschinen im Gate-Modell, die auf vollständig einstellbarer Supraleitung basieren. qubits

- Das Ankaa-3 System ist ein 84-Qubit-Gerät, das skalierbare Multi-Chip-Technologie nutzt.
- Das Cepheus-1-108Q System ist ein 108-Qubit-Gerät, das skalierbare Multi-Chip-Technologie verwendet.

Das Rigetti Gerät unterstützt die folgenden Quantengatter.

```
'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'
```

Ankaa-3Unterstützt bei wörtlicher Kompilierung die folgenden systemeigenen Gatter.

```
'rx', 'rz', 'iswap'
```

RigettiSupraleitende Quantenprozessoren können das „Rx“ -Gatter nur mit den Winkeln ±π /2 oder ±π betreiben.

Pulse-level Die Steuerung ist auf den Rigetti-Geräten verfügbar, die eine Reihe von vordefinierten Frames der folgenden Typen für das System unterstützen. Ankaa-3

```
`flux_tx`, `charge_tx`, `readout_rx`, `readout_tx`
```

Das Ankaa-3 Gerät hat eine maximale Grenze von 20.000 Gates pro Stromkreis. Schaltungen, die diesen Grenzwert überschreiten, werden mit einem Validierungsfehler zurückgewiesen. Dies ist ein fester Grenzwert, der nicht erhöht werden kann. Die Gatteranzahl bezieht sich auf die kompilierte Schaltung, die sich von der Gatteranzahl der ursprünglichen unkompilierten Schaltung unterscheiden kann. Um die Anzahl der kompilierten Gates vor der Übermittlung an die QPU abzuschätzen, können Sie die wörtliche Kompilierung lokal verwenden oder Ihre Schaltung auf das native Gate-Set (,,) transpilieren. `rx` `rz` `iswap`

QuEra

QuErabietet Geräte auf Basis neutraler Atome, mit denen Quantenaufgaben der analogen Hamiltonschen Simulation (AHS) ausgeführt werden können. Diese Spezialgeräte reproduzieren originalgetreu die zeitabhängige Quantendynamik von Hunderten von gleichzeitig wechselwirkenden Qubits.

Man kann diese Geräte nach dem Paradigma der analogen Hamiltonschen Simulation programmieren, indem man das Layout des Qubit-Registers und die zeitliche und räumliche Abhängigkeit der manipulierenden Felder vorschreibt. Amazon Braket bietet Hilfsprogramme zum Erstellen solcher Programme über das AHS-Modul des Python-SDK,`braket.ahs`.

Weitere Informationen finden Sie in den Beispielnotizbüchern zur analogen Hamiltonschen Simulation oder auf der Seite Submit an analog program using QuEra's Aquila.

Beispiel: Eine Quantenaufgabe an eine QPU einreichen

Mit Amazon Braket können Sie eine Quantenschaltung auf einem QPU-Gerät ausführen. Das folgende Beispiel zeigt, wie Sie eine Quantenaufgabe an Rigetti unsere Geräte senden. IonQ

Wählen Sie das Rigetti Ankaa-3 Gerät aus und schauen Sie sich dann das zugehörige Konnektivitätsdiagramm an

```
# import the QPU module
from braket.aws import AwsDevice
# choose the Rigetti device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")

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

```
{'fullyConnected': False,
 'connectivityGraph': {'0': ['1', '7'],
  '1': ['0', '2', '8'],
  '2': ['1', '3', '9'],
  '3': ['2', '4', '10'],
  '4': ['3', '5', '11'],
  '5': ['4', '6', '12'],
  '6': ['5', '13'],
  '7': ['0', '8', '14'],
  '8': ['1', '7', '9', '15'],
```

```
    '9': ['2', '8', '10', '16'],
    '10': ['3', '9', '11', '17'],
    '11': ['4', '10', '12', '18'],
    '12': ['5', '11', '13', '19'],
    '13': ['6', '12', '20'],
    '14': ['7', '15', '21'],
    '15': ['8', '14', '22'],
    '16': ['9', '17', '23'],
    '17': ['10', '16', '18', '24'],
    '18': ['11', '17', '19', '25'],
    '19': ['12', '18', '20', '26'],
    '20': ['13', '19', '27'],
    '21': ['14', '22', '28'],
    '22': ['15', '21', '23', '29'],
    '23': ['16', '22', '24', '30'],
    '24': ['17', '23', '25', '31'],
    '25': ['18', '24', '26', '32'],
    '26': ['19', '25', '33'],
    '27': ['20', '34'],
    '28': ['21', '29', '35'],
    '29': ['22', '28', '30', '36'],
    '30': ['23', '29', '31', '37'],
    '31': ['24', '30', '32', '38'],
    '32': ['25', '31', '33', '39'],
    '33': ['26', '32', '34', '40'],
    '34': ['27', '33', '41'],
    '35': ['28', '36', '42'],
    '36': ['29', '35', '37', '43'],
    '37': ['30', '36', '38', '44'],
    '38': ['31', '37', '39', '45'],
    '39': ['32', '38', '40', '46'],
    '40': ['33', '39', '41', '47'],
    '41': ['34', '40', '48'],
    '42': ['35', '43', '49'],
    '43': ['36', '42', '44', '50'],
    '44': ['37', '43', '45', '51'],
    '45': ['38', '44', '46', '52'],
    '46': ['39', '45', '47', '53'],
    '47': ['40', '46', '48', '54'],
    '48': ['41', '47', '55'],
    '49': ['42', '56'],
    '50': ['43', '51', '57'],
    '51': ['44', '50', '52', '58'],
    '52': ['45', '51', '53', '59'],
```

```
  '53': ['46', '52', '54'],
  '54': ['47', '53', '55', '61'],
  '55': ['48', '54', '62'],
  '56': ['49', '57', '63'],
  '57': ['50', '56', '58', '64'],
  '58': ['51', '57', '59', '65'],
  '59': ['52', '58', '60', '66'],
  '60': ['59'],
  '61': ['54', '62', '68'],
  '62': ['55', '61', '69'],
  '63': ['56', '64', '70'],
  '64': ['57', '63', '65', '71'],
  '65': ['58', '64', '66', '72'],
  '66': ['59', '65', '67'],
  '67': ['66', '68'],
  '68': ['61', '67', '69', '75'],
  '69': ['62', '68', '76'],
  '70': ['63', '71', '77'],
  '71': ['64', '70', '72', '78'],
  '72': ['65', '71', '73', '79'],
  '73': ['72', '80'],
  '75': ['68', '76', '82'],
  '76': ['69', '75', '83'],
  '77': ['70', '78'],
  '78': ['71', '77', '79'],
  '79': ['72', '78', '80'],
  '80': ['73', '79', '81'],
  '81': ['80', '82'],
  '82': ['75', '81', '83'],
  '83': ['76', '82']}}
```

Das vorherige Wörterbuch `connectivityGraph` listet die benachbarten Qubits für jedes Qubit im Rigetti Gerät auf.

Wählen Sie das Gerät IonQ Forte-Enterprise-1

Für das IonQ Forte-Enterprise-1 Gerät `connectivityGraph` ist das leer, wie im folgenden Beispiel gezeigt, da das Gerät All-in-All-Konnektivität bietet. Daher ist eine detaillierte Beschreibung nicht `connectivityGraph` erforderlich.

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

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

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

Wie im folgenden Beispiel gezeigt, haben Sie die Möglichkeit, den shots (Standard=1000), den `poll_timeout_seconds` (Standard = 432000 = 5 Tage), den `poll_interval_seconds` (Standard = 1) und den Speicherort des S3-Buckets (`s3_location`), in dem Ihre Ergebnisse gespeichert werden, anzupassen, wenn Sie einen anderen Speicherort als den Standard-Bucket angeben.

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

Die IonQ Rigetti Geräte kompilieren die bereitgestellte Schaltung automatisch in ihre jeweiligen nativen Gate-Sets und ordnen die abstrakten qubit Indizes den physikalischen qubits auf der jeweiligen QPU zu.

> **Note**
>
> QPU-Geräte haben eine begrenzte Kapazität. Sie können mit längeren Wartezeiten rechnen, wenn die Kapazität erreicht ist.

Amazon Braket kann QPU-Quantenaufgaben innerhalb bestimmter Verfügbarkeitsfenster ausführen, aber Sie können Quantenaufgaben trotzdem jederzeit einreichen (24/7), da alle entsprechenden Daten und Metadaten zuverlässig im entsprechenden S3-Bucket gespeichert werden. Wie im nächsten Abschnitt gezeigt, können Sie Ihre Quantenaufgabe mithilfe Ihrer eindeutigen `AwsQuantumTask` Quantenaufgaben-ID wiederherstellen.

Inspektion kompilierter Schaltungen

Wenn ein Quantenschaltkreis auf einem Hardwaregerät wie einer Quantenverarbeitungseinheit (QPU) ausgeführt werden muss, muss der Schaltkreis zunächst in ein akzeptables Format kompiliert werden, das das Gerät verstehen und verarbeiten kann. Zum Beispiel die Transpilierung des Quantenschaltkreises auf hoher Ebene bis zu den spezifischen nativen Gates, die von der QPU-Zielhardware unterstützt werden. Die Überprüfung der tatsächlich kompilierten Ausgabe

des Quantenschaltkreises kann für Debugging- und Optimierungszwecke äußerst nützlich sein. Dieses Wissen kann helfen, potenzielle Probleme, Engpässe oder Möglichkeiten zur Verbesserung der Leistung und Effizienz der Quantenanwendung zu identifizieren. Mithilfe des unten angegebenen Codes können Sie die kompilierte Ausgabe Ihrer Quantenschaltkreise sowohl für Quantencomputergeräte als auch für IQM Quantencomputergeräte anzeigen Rigetti und analysieren.

```
task = AwsQuantumTask(arn=task_id, aws_session=session)
# After the task has finished running
task_result = task.result()
compiled_circuit = task_result.get_compiled_circuit()
```

> **Note**
>
> Derzeit wird die Anzeige der kompilierten Schaltkreisausgabe für IonQ Geräte nicht unterstützt.

Mehrere Programme ausführen

Amazon Braket bietet zwei Ansätze für die effiziente Ausführung mehrerer Quantenprogramme: Programmsätze und Batching von Quantenaufgaben.

Programmgruppen sind die bevorzugte Methode, um Workloads mit mehreren Programmen auszuführen. Sie ermöglichen es Ihnen, mehrere Programme in einer einzigen Amazon Braket-Quantenaufgabe zu bündeln. Programmgruppen bieten Leistungsverbesserungen und Kosteneinsparungen im Vergleich zur individuellen Einreichung von Programmen, insbesondere wenn sich die Anzahl der Programmausführungen 100 nähert.

Derzeit unterstützen IQM alle Rigetti Geräte Programmpakete. Bevor Sie Programmsets an QPUs senden, wird empfohlen, zuerst auf dem Amazon Braket Local Simulator zu testen. Um zu überprüfen, ob ein Gerät Programmsets unterstützt, können Sie die Eigenschaften des Geräts mit dem Amazon Braket SDK oder die Geräteseite in der Amazon Braket-Konsole aufrufen.

Das folgende Beispiel zeigt, wie ein Programmsatz ausgeführt wird.

```
from math import pi
from braket.devices import LocalSimulator
from braket.program_sets import ProgramSet
from braket.circuits import Circuit
```

```
program_set = ProgramSet([
    Circuit().h(0).cnot(0,1),
    Circuit().rx(0, pi/4).ry(1, pi/8).cnot(1,0),
    Circuit().t(0).t(1).cz(0,1).s(0).cz(1,2).s(1).s(2),
])

device = LocalSimulator()
result = device.run(program_set, shots=300).result()
print(result[0][0].counts)  # The result of the first program in the program set
```

Weitere Informationen über verschiedene Methoden zum Erstellen einer Programmgruppe (z. B. zum Konstruieren einer Programmgruppe aus vielen Observablen oder Parametern mit einem einzigen Programm) und zum Abrufen von Programmgruppenergebnissen finden Sie im Abschnitt Programmgruppen im Amazon Braket Developer Guide und im Ordner Program Sets im Github-Repository Braket-Beispiele.

Quantum Task Batching ist auf jedem Amazon Braket-Gerät verfügbar. Batching ist besonders nützlich für Quantenaufgaben, die Sie auf den On-Demand-Simulatoren (SV1, DM1 oderTN1) ausführen, da sie mehrere Quantenaufgaben parallel verarbeiten können. Durch Batching können Sie Quantenaufgaben parallel starten. Wenn Sie beispielsweise eine Berechnung durchführen möchten, für die 10 Quantenaufgaben erforderlich sind und die Programme in diesen Quantenaufgaben unabhängig voneinander sind, empfiehlt es sich, Task-Batching zu verwenden. Verwenden Sie Quanten-Task-Batching, wenn Sie Workloads mit mehreren Programmen auf einem Gerät ausführen, das keine Programmgruppen unterstützt.

Das folgende Beispiel zeigt, wie ein Stapel von Quantenaufgaben ausgeführt wird.

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

bell = Circuit().h(0).cnot(0, 1)
circuits = [bell for _ in range(5)]

device = LocalSimulator()
batch = device.run_batch(circuits, shots=100)
print(batch.results()[0].measurement_counts)  # The result of the first quantum task in
 the batch
```

Genauere Informationen zur Batchverarbeitung finden Sie in den Amazon Braket-Beispielen unter. GitHub

In diesem Abschnitt:

- Informationen zum Programmumfang und zu den Kosten
- Informationen zur Batchverarbeitung und zu den Kosten von Quantenaufgaben
- Batching von Quantenaufgaben und PennyLane
- Batching von Aufgaben und parametrisierte Schaltungen

Informationen zum Programmumfang und zu den Kosten

Mit Programmsätzen können mehrere Quantenprogramme effizient ausgeführt werden, indem sie bis zu 100 Programme oder Parametersätze in einer einzigen Quantenaufgabe zusammenfassen. Bei Programmsets zahlen Sie nur eine Gebühr pro Aufgabe zuzüglich der Gebühren pro Schuss, die auf der Gesamtzahl der Aufnahmen aller Programme basieren, wodurch die Kosten im Vergleich zur Einreichung einzelner Programme erheblich gesenkt werden. Dieser Ansatz ist besonders vorteilhaft bei Workloads mit vielen Programmen und einer geringen Anzahl von Aufnahmen pro Programm. Programmsätze werden derzeit auf IQM allen Rigetti Geräten sowie im Amazon Braket Local Simulator unterstützt.

Weitere Informationen finden Sie im Abschnitt Programmgruppen mit detaillierten Implementierungsschritten, bewährten Methoden und Codebeispielen.

Informationen zur Batchverarbeitung und zu den Kosten von Quantenaufgaben

Einige Vorbehalte, die Sie in Bezug auf die Batch- und Abrechnungskosten für Quantenaufgaben beachten sollten:

- Standardmäßig wiederholt die Batchverarbeitung bei Quantenaufgaben die Zeitüberschreitung oder schlägt dreimal fehl.
- Ein Stapel lang andauernder Quantenaufgaben, wie z. B. 34 qubits fürSV1, kann hohe Kosten verursachen. Achten Sie darauf, die `run_batch` Zuweisungswerte sorgfältig zu überprüfen, bevor Sie mit einer Reihe von Quantenaufgaben beginnen. Wir empfehlen nicht, TN1 mit zu verwenden`run_batch`.
- TN1kann Kosten für fehlgeschlagene Aufgaben in der Probenphase verursachen (weitere Informationen finden Sie in der TN1-Beschreibung). Automatische Wiederholungsversuche können die Kosten in die Höhe treiben. Wir empfehlen daher, die Anzahl der 'max_retries' beim Batching auf 0 zu setzen, wenn sie verwendet werden TN1 (siehe Quantum Task Batching, Zeile 186).

Batching von Quantenaufgaben und PennyLane

Nutzen Sie die Batching-Funktion, wenn Sie PennyLane Amazon Braket verwenden, indem Sie festlegen, `parallel = True` wann Sie ein Amazon Braket-Gerät instanziieren, wie im folgenden Beispiel gezeigt.

```
import pennylane as qml

# Define the number of wires (qubits) you want to use
wires = 2  # For example, using 2 qubits

# Define your S3 bucket
my_bucket = "amazon-braket-s3-demo-bucket"
my_prefix = "pennylane-batch-output"
s3_folder = (my_bucket, my_prefix)

device = qml.device("braket.aws.qubit",
                    device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
                    wires=wires,
                    s3_destination_folder=s3_folder,
                    parallel=True)
```

Weitere Informationen zur Stapelverarbeitung mit finden Sie unter Parallelisierte Optimierung von PennyLane Quantenschaltkreisen.

Batching von Aufgaben und parametrisierte Schaltungen

Wenn Sie einen Quanten-Task-Batch einreichen, der parametrisierte Schaltungen enthält, können Sie entweder ein `inputs` Wörterbuch bereitstellen, das für alle Quantenaufgaben im Stapel verwendet wird, oder ein `list` Eingabewörterbuch, in welchem Fall `i` das -te Wörterbuch mit `i` der -ten Aufgabe verknüpft wird, wie im folgenden Beispiel gezeigt.

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

# Define your quantum device
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")

# Create the free parameters
alpha = FreeParameter('alpha')
beta = FreeParameter('beta')
```

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

circ_b = Circuit().rx(0, alpha).rz(1, alpha).cnot(0, 2).zz(0, 2, beta)
circ_b.expectation(observable=Observable.Z(), target=2)

# Use the same inputs for both circuits in one batch
tasks = device.run_batch([circ_a, circ_b], inputs={'alpha': 0.1, 'beta': 0.2})

# Or provide each task its own set of inputs
inputs_list = [{'alpha': 0.3, 'beta': 0.1}, {'alpha': 0.1, 'beta': 0.4}]

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

Sie können auch eine Liste von Eingabewörterbüchern für einen einzelnen parametrischen Schaltkreis erstellen und diese als Quanten-Task-Batch einreichen. Wenn die Liste N Eingabewörterbücher enthält, enthält der Stapel N Quantenaufgaben. Die `i` -te Quantenaufgabe entspricht der Schaltung, die mit dem `i` -th Eingabewörterbuch ausgeführt wird.

```
from braket.circuits import Circuit, FreeParameter

# Create a parametric circuit
circ = Circuit().rx(0, FreeParameter('alpha'))

# Provide a list of inputs to execute with the circuit
inputs_list = [{'alpha': 0.1}, {'alpha': 0.2}, {'alpha': 0.3}]

tasks = device.run_batch(circ, inputs=inputs_list, shots=100)
```

Wann wird meine Quantenaufgabe ausgeführt?

Wenn Sie einen Circuit einreichen, sendet Amazon Braket ihn an das von Ihnen angegebene Gerät. Die Quantum Processing Unit (QPU) und die Quantenaufgaben des On-Demand-Simulators werden in der Reihenfolge ihres Eingangs in die Warteschlange gestellt und verarbeitet. Die Zeit, die benötigt wird, um Ihre Quantenaufgabe nach dem Absenden zu bearbeiten, hängt von der Anzahl und Komplexität der von anderen Amazon Braket-Kunden eingereichten Aufgaben und der Verfügbarkeit der ausgewählten QPU ab.

In diesem Abschnitt:

- QPU-Verfügbarkeitsfenster und Status
- Sichtbarkeit der Warteschlange
- Richten Sie E-Mail- oder SMS-Benachrichtigungen ein

QPU-Verfügbarkeitsfenster und Status

Die Verfügbarkeit von QPU variiert von Gerät zu Gerät.

Auf der Geräteseite der Amazon Braket-Konsole können Sie die aktuellen und bevorstehenden Verfügbarkeitsfenster und den Gerätestatus sehen. Darüber hinaus zeigt jede Geräteseite individuelle Warteschlangentiefen für Quantenaufgaben und Hybrid-Jobs.

Ein Gerät gilt unabhängig vom Verfügbarkeitsfenster als offline, wenn es für Kunden nicht verfügbar ist. Beispielsweise könnte es aufgrund von geplanten Wartungsarbeiten, Upgrades oder Betriebsproblemen offline sein.

Sichtbarkeit der Warteschlange

Bevor Sie eine Quantenaufgabe oder einen Hybridauftrag einreichen, können Sie anhand der Warteschlangentiefe des Geräts überprüfen, wie viele Quantenaufgaben oder Hybridaufträge noch vor Ihnen liegen.

Tiefe der Warteschlange

Queue depthbezieht sich auf die Anzahl der Quantenaufgaben und Hybrid-Jobs, die sich für ein bestimmtes Gerät in der Warteschlange befinden. Auf die Anzahl der Warteschlangen für Quantenaufgaben und Hybrid-Jobs eines Geräts kann über das Symbol Braket Software Development Kit (SDK) oder Amazon Braket Management Console zugegriffen werden.

1. Die Tiefe der Aufgabenwarteschlange bezieht sich auf die Gesamtzahl der Quantenaufgaben, die derzeit darauf warten, mit normaler Priorität ausgeführt zu werden.
2. Die Tiefe der Warteschlange für Prioritätsaufgaben bezieht sich auf die Gesamtzahl der eingereichten Quantenaufgaben, die darauf warten, bearbeitet zu Amazon Braket Hybrid Jobs werden. Diese Aufgaben werden vor eigenständigen Aufgaben ausgeführt.
3. Die Warteschlangentiefe für Hybridaufträge bezieht sich auf die Gesamtzahl der Hybridaufträge, die sich derzeit auf einem Gerät in der Warteschlange befinden. Quantum tasksDie im Rahmen eines Hybridauftrags eingereichten Aufträge haben Priorität und werden in der zusammengefasst. Priority Task Queue

Kunden, die die Tiefe der Warteschlange über einsehen möchten, Braket SDK können den folgenden Codeausschnitt ändern, um die Warteschlangenposition ihrer Quantenaufgabe oder ihres Hybrid-Jobs zu ermitteln:

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

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


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

Wenn Sie eine Quantenaufgabe oder einen Hybrid-Job an eine QPU senden, kann dies dazu führen, dass sich Ihr Workload in einem Zustand befindet. `QUEUED` Amazon Braket bietet Kunden Einblick in ihre Warteschlangenposition für Quantenaufgaben und Hybrid-Jobs.

Position in der Warteschlange

Queue positionbezieht sich auf die aktuelle Position Ihrer Quantenaufgabe oder Ihres Hybrid-Jobs innerhalb einer entsprechenden Gerätewarteschlange. Sie kann für Quantenaufgaben oder Hybridjobs über das Braket Software Development Kit (SDK) oder abgerufen Amazon Braket Management Console werden.

Kunden, die die Warteschlangenposition über das einsehen möchten, Braket SDK können den folgenden Codeausschnitt ändern, um die Warteschlangenposition ihrer Quantenaufgabe oder ihres Hybrid-Jobs zu ermitteln:

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

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

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

# Returns the number of Quantum Tasks queued ahead of you
```

```
'2'


from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=False
)

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

Richten Sie E-Mail- oder SMS-Benachrichtigungen ein

Amazon Braket sendet Ereignisse an Amazon, EventBridge wenn sich die Verfügbarkeit einer QPU ändert oder wenn sich der Status Ihrer Quantenaufgabe ändert. Gehen Sie wie folgt vor, um Benachrichtigungen über die Änderung des Geräte- und Quanten-Task-Status per E-Mail oder SMS zu erhalten:

1. Erstellen Sie ein Amazon SNS SNS-Thema und ein Abonnement für E-Mail oder SMS. Die Verfügbarkeit von E-Mail oder SMS hängt von Ihrer Region ab. Weitere Informationen finden Sie unter Erste Schritte mit Amazon SNS und Senden von SMS-Nachrichten.
2. Erstellen Sie eine Regel EventBridge , die Benachrichtigungen zu Ihrem SNS-Thema auslöst. Weitere Informationen finden Sie unter Amazon Braket mit Amazon EventBridge überwachen.

(Optional) Richten Sie SNS-Benachrichtigungen ein

Sie können Benachrichtigungen über den Amazon Simple Notification Service (SNS) einrichten, sodass Sie eine Benachrichtigung erhalten, wenn Ihre Amazon Braket-Quantenaufgabe abgeschlossen ist. Aktive Benachrichtigungen sind nützlich, wenn Sie mit einer langen Wartezeit rechnen, z. B. wenn Sie eine umfangreiche Quantenaufgabe einreichen oder wenn Sie eine Quantenaufgabe außerhalb des Verfügbarkeitsfensters eines Geräts einreichen. Wenn Sie nicht warten möchten, bis die Quantenaufgabe abgeschlossen ist, können Sie eine SNS-Benachrichtigung einrichten.

Ein Amazon Braket-Notizbuch führt Sie durch die Einrichtungsschritte. Weitere Informationen finden Sie in den Amazon Braket-Beispielen GitHub und insbesondere im Beispiel-Notizbuch zum Einrichten von Benachrichtigungen.

Mit Reservierungen arbeiten

Reservierungen geben Ihnen exklusiven Zugriff auf das Quantengerät Ihrer Wahl. Sie können nach Belieben eine Reservierung vereinbaren, sodass Sie genau wissen, wann die Ausführung Ihres Workloads beginnt und endet. Reservierungen sind für alle Braket-Geräte in Abständen von 1 Stunde möglich und können bis zu 48 Stunden im Voraus ohne zusätzliche Kosten storniert werden. Wir empfehlen, Quantenaufgaben und Hybrid-Jobs für eine bevorstehende Reservierung im Voraus in die Warteschlange zu stellen, indem Sie Ihren Braket Direct-Reservierungs-ARN verwenden oder Workloads während Ihrer Reservierung einreichen.

Die Kosten für den Zugriff auf dedizierte Geräte richten sich nach der Dauer Ihrer Reservierung, unabhängig davon, wie viele Quantenaufgaben und Hybridjobs Sie auf der Quantenverarbeitungseinheit (QPU) ausführen. Eine aktualisierte Liste der Quantencomputer, die für Reservierungen verfügbar sind, finden Sie auf unserer Preisseite oder über die Amazon Braket-Managementkonsole.

> **Note**
>
> Bei einer Reservierung gibt es keine Gateshot-Limits. Darüber hinaus wurde bei IonQ Geräten die Mindestanzahl von Schüssen für Aufgaben zur Fehlerminimierung auf 500 reduziert (gegenüber 2500 für On-Demand-Aufgaben).

Wann sollte man eine Reservierung nutzen

Die Nutzung des Reservierungszugriffs bietet Ihnen den Komfort und die Vorhersehbarkeit, genau zu wissen, wann die Ausführung Ihres Quanten-Workloads beginnt und endet. Im Vergleich zur Übermittlung von Aufgaben und hybriden Aufträgen auf Abruf müssen Sie nicht in einer Warteschlange mit anderen Kundenaufgaben warten. Da Sie während Ihrer Reservierung exklusiven Zugriff auf das Gerät haben, werden während der gesamten Reservierung nur Ihre Workloads auf dem Gerät ausgeführt.

Wir empfehlen, den On-Demand-Zugriff für die Entwurfs- und Prototypingphase Ihrer Forschung zu verwenden, um eine schnelle und kostengünstige Iteration Ihrer Algorithmen zu ermöglichen. Sobald Sie bereit sind, die endgültigen Versuchsergebnisse zu erstellen, sollten Sie erwägen, nach

Belieben eine Gerätereservierung zu vereinbaren, um sicherzustellen, dass Sie die Projekt- oder Veröffentlichungsfristen einhalten können. Wir empfehlen außerdem, Reservierungen zu nutzen, wenn Sie die Ausführung von Aufgaben zu bestimmten Zeiten wünschen, z. B. wenn Sie eine Live-Demo oder einen Workshop auf einem Quantencomputer durchführen.

In diesem Abschnitt:

- Wie erstelle ich eine Reservierung
- Ausführung von Quantenaufgaben während einer Reservierung
- Ausführung von Hybridaufträgen während einer Reservierung
- Was passiert am Ende Ihrer Reservierung
- Stornieren oder verschieben Sie eine bestehende Reservierung

Wie erstelle ich eine Reservierung

Um eine Reservierung zu erstellen, wenden Sie sich wie folgt an das Braket-Team:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Bereich Braket Direct und dann im Bereich Reservierungen die Option Gerät reservieren aus.
3. Wählen Sie das Gerät aus, das Sie reservieren möchten.
4. Geben Sie Ihre Kontaktinformationen einschließlich Name und E-Mail an. Stellen Sie sicher, dass Sie eine gültige E-Mail-Adresse angeben, die Sie regelmäßig überprüfen.
5. Geben Sie unter Erzählen Sie uns von Ihrem Workload alle Informationen über den Workload an, der mit Ihrer Reservierung ausgeführt werden soll. Zum Beispiel die gewünschte Reservierungsdauer, relevante Einschränkungen oder der gewünschte Zeitplan.

Nachdem Sie das Formular abgeschickt haben, erhalten Sie vom Braket-Team eine E-Mail mit den nächsten Schritten. Sobald Ihre Reservierung bestätigt ist, erhalten Sie den Reservierungs-ARN per E-Mail. Sie benötigen den Reservierungs-ARN, um Reservierungsaufgaben zu erstellen. Aufgaben, die ohne den Reservierungs-ARN erstellt wurden, werden an die reguläre On-Demand-Warteschlange gestellt und NICHT während Ihrer Reservierung ausgeführt.

> **Note**
>
> Ihre Reservierung ist erst bestätigt, wenn Sie den Reservierungs-ARN erhalten haben.

Reservierungen sind in Abständen von mindestens einer Stunde möglich, und für bestimmte Geräte gelten möglicherweise zusätzliche Einschränkungen der Reservierungsdauer (einschließlich Mindest- und Höchstdauer der Reservierung). Das Braket-Team teilt Ihnen vor der Bestätigung der Reservierung alle relevanten Informationen mit.

Das Braket-Team wird Sie per E-Mail kontaktieren, um eine 30-minütige Sitzung mit einem Braket-Experten zu vereinbaren.

Ausführung von Quantenaufgaben während einer Reservierung

Nachdem Sie einen gültigen Reservierungs-ARN von Create a reservation erhalten haben, können Sie Quantenaufgaben erstellen, die während der Reservierung ausgeführt werden sollen. Quantenaufgaben und Hybrid-Jobs, die mit einem Reservierungs-ARN eingereicht wurden, werden nicht in einer Gerätewarteschlange angezeigt. Aufgaben, die vor Beginn der Reservierung eingereicht wurden, bleiben so lange `QUEUED` erhalten, bis Ihre Reservierung beginnt.

> **Note**
>
> Reservierungen sind AWS konto- und gerätespezifisch. Nur das AWS Konto, das die Reservierung erstellt hat, kann Ihren Reservierungs-ARN verwenden.
> Während einer Reservierung können sowohl Reservierungen als auch reguläre Aufgaben erstellt werden. Um zu überprüfen, ob eine erstellte Braket-Quantenaufgabe mit einer Reservierung verknüpft ist, überprüfen Sie das Feld „Reservierungs-ARN“ auf der Seite der Quantenaufgabe in der Braket-Konsole oder fragen Sie dasselbe Feld in den Aufgabenmetadaten mithilfe des SDK ab. Im Rest dieser Seite wird beschrieben, wie Sie angeben, welche Aufgaben mit der Reservierung verknüpft sind.

Sie können Quantenaufgaben Python SDKs beispielsweise mit Braket,, CUDA-QPennyLaneQiskit, oder direkt mit boto3 (Working with Boto3) erstellen. Um Reservierungen nutzen zu können, benötigen Sie Version v1.79.0 oder höher von Amazon Braket. Python SDK Mit dem folgenden Code können Sie auf das neueste Braket-SDK, den neuesten Qiskit Anbieter und PennyLane das neueste Plugin aktualisieren.

```
pip install --upgrade amazon-braket-sdk amazon-braket-pennylane-plugin qiskit-braket-provider
```

Führen Sie Aufgaben mit dem **`DirectReservation`** Kontext-Manager aus

Die empfohlene Methode, eine Aufgabe innerhalb Ihrer geplanten Reservierung auszuführen, ist die Verwendung des `DirectReservation` Kontext-Managers. Durch die Angabe Ihres Zielgeräts und des Reservierungs-ARN stellt der Kontext-Manager sicher, dass alle in der `with` Python-Anweisung erstellten Aufgaben mit exklusivem Zugriff auf das Gerät ausgeführt werden.

Definieren Sie zunächst einen Quantenschaltkreis und das Gerät. Verwenden Sie dann den Reservierungskontext und führen Sie die Aufgabe aus. Stellen Sie sicher, dass Ihre gesamte Arbeitslast innerhalb des **`with`** Blocks ausgeführt wird. Alles, was außerhalb des Bereichs des **`with`** Blocks ausgeführt wird, wird nicht mit Ihrer Reservierung verknüpft!

```
from braket.aws import AwsDevice, DirectReservation
from braket.circuits import Circuit
from braket.devices import Devices

bell = Circuit().h(0).cnot(0, 1)
device = AwsDevice(Devices.IonQ.ForteEnterprise1)

# run the circuit in a reservation
with DirectReservation(device, reservation_arn="<my_reservation_arn>"):
    task = device.run(bell, shots=100)
```

Sie können Quantenaufgaben in einer Reservierung mithilfe der Qiskit Plug-ins CUDA-QPennyLane, und erstellen, sofern der `DirectReservation` Kontext beim Erstellen von Quantenaufgaben aktiv ist. Mit dem Qiskit-Braket Anbieter können Sie Aufgaben beispielsweise wie folgt ausführen.

```
from braket.devices import Devices
from braket.aws import DirectReservation
from qiskit import QuantumCircuit
from qiskit_braket_provider import BraketProvider


qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)

qpu = BraketProvider().get_backend("Forte Enterprise 1")

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    qpu_task = qpu.run(qc, shots=10)
```

In ähnlicher Weise führt der folgende Code während einer Reservierung mithilfe des Braket-PennyLane Plug-ins eine Verbindung aus.

```
from braket.devices import Devices
from braket.aws import DirectReservation
import pennylane as qml


dev = qml.device("braket.aws.qubit", device_arn=Devices.IonQ.ForteEnterprise1.value,
 wires=2, shots=10)

@qml.qnode(dev)
def bell_state():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    return qml.probs(wires=[0, 1])

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>"):
    probs = bell_state()
```

Manuelles Einstellen des Reservierungskontextes

Alternativ können Sie den Reservierungskontext mit dem folgenden Code manuell festlegen.

```
# set reservation context
reservation_context = DirectReservation(device,
 reservation_arn="<my_reservation_arn>").start()

# run circuit during reservation
task = device.run(bell, shots=100)
```

Dies ist ideal für Jupyter-Notebooks, bei denen der Kontext in der ersten Zelle ausgeführt werden kann und alle nachfolgenden Aufgaben in der Reservierung ausgeführt werden.

> **Note**
>
> Die Zelle, die den `.start()` Aufruf enthält, sollte nur einmal ausgeführt werden.

Um zum On-Demand-Modus zurückzukehren: Starten Sie das Jupyter-Notebook neu, oder rufen Sie den folgenden Befehl auf, um den Kontext wieder in den On-Demand-Modus zu ändern.

```
reservation_context.stop()  # unset reservation context
```

> **Note**
>
> Reservierungen haben eine vorab festgelegte Start- und Endzeit (siehe Reservierung erstellen). Die `reservation_context.stop()` Methoden `reservation_context.start()` und bedeuten weder den Beginn noch das Ende einer Reservierung. Stattdessen werden alle Quantenaufgaben, die Sie erstellen, Ihrer Reservierung zugeordnet und nur während Ihrer geplanten Reservierung ausgeführt, solange der Kontext aktiv ist. Der Reservierungskontext hat keine Auswirkung auf die geplante Reservierungszeit.

Übergeben Sie den Reservierungs-ARN beim Erstellen der Aufgabe explizit

Eine andere Möglichkeit, Aufgaben während einer Reservierung zu erstellen, besteht darin, den Reservierungs-ARN beim Anrufen explizit zu übergeben`device.run()`.

```
task = device.run(bell, shots=100, reservation_arn="<my_reservation_arn>")
```

Diese Methode verknüpft die Quantenaufgabe direkt mit dem Reservierungs-ARN und stellt so sicher, dass sie während des reservierten Zeitraums ausgeführt wird. Fügen Sie für diese Option den Reservierungs-ARN zu jeder Aufgabe hinzu, die Sie während einer Reservierung ausführen möchten. Beachten Sie jedoch, dass es bei der Verwendung von Bibliotheken von Drittanbietern wie Qiskit oder schwierig sein kannPennyLane, sicherzustellen, dass die eingereichten Aufgaben den richtigen Reservierungs-ARN verwenden. Aus diesem Grund wird die Verwendung des DirectReservation Kontext-Managers empfohlen.

Wenn Sie boto3 direkt verwenden, übergeben Sie den Reservierungs-ARN als Assoziation, wenn Sie eine Aufgabe erstellen.

```
import boto3

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

```
kwargs["associations"] = [
    {
        "arn": "<my_reservation_arn>",
        "type": "RESERVATION_TIME_WINDOW_ARN",
    }
]

response = braket_client.create_quantum_task(**kwargs)
```

Ausführung von Hybridaufträgen während einer Reservierung

Sobald Sie eine Python-Funktion als Hybrid-Job ausführen können, können Sie den Hybrid-Job in einer Reservierung ausführen, indem Sie das `reservation_arn` Schlüsselwort-Argument übergeben. Alle Aufgaben innerhalb des Hybrid-Jobs verwenden den Reservierungs-ARN. Wichtig ist, dass der Hybrid-Job, bei dem die klassische Rechenleistung *`reservation_arn`* erst dann aktiviert wird, wenn Ihre Reservierung beginnt.

> **Note**
>
> Ein Hybrid-Job, der während einer Reservierung ausgeführt wird, führt nur erfolgreich Quantenaufgaben auf dem reservierten Gerät aus. Der Versuch, ein anderes On-Demand-Braket-Gerät zu verwenden, führt zu einem Fehler. Wenn Sie Aufgaben sowohl auf einem On-Demand-Simulator als auch auf dem reservierten Gerät innerhalb desselben Hybrid-Jobs ausführen müssen, verwenden Sie `DirectReservation` stattdessen.

Der folgende Code zeigt, wie ein Hybridjob während einer Reservierung ausgeführt wird.

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

@hybrid_job(device=Devices.IonQ.ForteEnterprise1,
 reservation_arn="<my_reservation_arn>")
def example_hybrid_job():
    # declare AwsDevice within the hybrid job
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)

    task = device.run(bell, shots=10)
```

Hybrid-Jobs, die ein Python-Skript verwenden (siehe Abschnitt zum Erstellen Ihres ersten Hybrid-Jobs im Entwicklerhandbuch), können Sie sie innerhalb der Reservierung ausführen, indem Sie bei der Erstellung des Jobs das `reservation_arn` Schlüsselwort-Argument übergeben.

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

job = AwsQuantumJob.create(
    Devices.IonQ.ForteEnterprise1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="<my_reservation_arn>"
)
```

Was passiert am Ende Ihrer Reservierung

Nach Ablauf Ihrer Reservierung haben Sie keinen dedizierten Zugriff mehr auf das Gerät. Alle verbleibenden Workloads, die sich mit dieser Reservierung in der Warteschlange befinden, werden automatisch storniert.

> **Note**
>
> Jeder Job, der sich am Ende der Reservierung im `RUNNING` Status befand, wird storniert. Wir empfehlen, Checkpoints zu verwenden, um Jobs nach Belieben zu speichern und neu zu starten.

Eine laufende Reservierung, z. B. nach dem Beginn der Reservierung und vor dem Ende der Reservierung, kann nicht verlängert werden, da jede Reservierung einen eigenständigen, dedizierten Gerätezugriff darstellt. Beispielsweise werden zwei aufeinanderfolgende Reservierungen als getrennt betrachtet, und alle ausstehenden Aufgaben aus der ersten Reservierung werden automatisch storniert. Sie werden in der zweiten Reservierung nicht wieder aufgenommen.

> **Note**
>
> Reservierungen stellen einen dedizierten Gerätezugriff für Ihr AWS Konto dar. Selbst wenn das Gerät inaktiv bleibt, können es keine anderen Kunden verwenden. Daher wird Ihnen unabhängig von der genutzten Zeit die Dauer der reservierten Zeit in Rechnung gestellt.

Stornieren oder verschieben Sie eine bestehende Reservierung

Sie können Ihre Reservierung mindestens 48 Stunden vor dem geplanten Beginn der Reservierung stornieren. Um zu stornieren, antworten Sie auf die Reservierungsbestätigungs-E-Mail, die Sie mit Ihrer Stornierungsanfrage erhalten haben.

Um einen neuen Termin zu vereinbaren, müssen Sie Ihre bestehende Reservierung stornieren und dann eine neue erstellen.

Techniken zur Fehlerminimierung

Bei der Minimierung von Quantenfehlern handelt es sich um eine Reihe von Techniken, die darauf abzielen, die Auswirkungen von Fehlern in Quantencomputern zu reduzieren.

Quantengeräte sind Umgebungsgeräuschen ausgesetzt, die die Qualität der durchgeführten Berechnungen beeinträchtigen. Fehlertolerante Quantencomputer versprechen zwar eine Lösung für dieses Problem, aber aktuelle Quantengeräte sind durch die Anzahl der Qubits und relativ hohe Fehlerraten begrenzt. Um dem kurzfristig entgegenzuwirken, untersuchen Forscher Methoden zur Verbesserung der Genauigkeit verrauschter Quantenberechnungen. Dieser Ansatz, bekannt als Quantenfehlerminimierung, beinhaltet den Einsatz verschiedener Techniken, um das beste Signal aus verrauschten Messdaten zu extrahieren.

In diesem Abschnitt:

- Techniken zur Fehlerminimierung auf ionQ Geräte

Techniken zur Fehlerminimierung auf ionQ Geräte

Zur Fehlerminimierung werden mehrere physische Schaltkreise betrieben und ihre Messungen kombiniert, um ein besseres Ergebnis zu erzielen.

> **Note**
>
> Für alle IonQ Geräte gilt: Bei Verwendung eines On-Demand-Modells gilt ein Limit von 1 Million Gateshots und ein Minimum von 2500 Schüssen für Aufgaben zur Fehlerminimierung. Bei einer direkten Reservierung gibt es kein Gateshot-Limit und ein Minimum von 500 Schüssen für Aufgaben zur Fehlerminimierung.

Entschärfung

IonQGeräte verfügen über eine Methode zur Fehlerminimierung, die als Debiasing bezeichnet wird.

Beim Debiasing wird ein Schaltkreis in mehrere Varianten eingeteilt, die auf unterschiedliche Qubit-Permutationen oder unterschiedliche Gate-Zerlegungen einwirken. Dadurch werden die Auswirkungen systematischer Fehler wie Gatterüberdrehungen oder eines einzelnen fehlerhaften Qubits reduziert, indem unterschiedliche Implementierungen einer Schaltung verwendet werden, die andernfalls die Messergebnisse verfälschen könnten. Dies geht auf Kosten des zusätzlichen Aufwands für die Kalibrierung mehrerer Qubits und Gates.

Weitere Informationen zum Debiasing finden Sie unter Verbesserung der Leistung von Quantencomputern durch Symmetrisierung.

> **Note**
>
> Für die Verwendung von Debiasing sind mindestens 2500 Aufnahmen erforderlich.

Mit dem folgenden Code können Sie eine Quantenaufgabe mit Debiasing auf einem IonQ Gerät ausführen:

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

# choose an IonQ device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1")
circuit = Circuit().h(0).cnot(0, 1)

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

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

Wenn die Quantenaufgabe abgeschlossen ist, können Sie die Messwahrscheinlichkeiten und alle Ergebnistypen der Quantenaufgabe sehen. Die Messwahrscheinlichkeiten und Zählungen aller Varianten werden zu einer einzigen Verteilung zusammengefasst. Alle im Schaltkreis angegebenen Ergebnistypen, wie z. B. Erwartungswerte, werden anhand der aggregierten Messzahlen berechnet.

Scharfzeichnen

Sie können auch auf Messwahrscheinlichkeiten zugreifen, die mit einer anderen Nachbearbeitungsstrategie, dem sogenannten Sharpening, berechnet wurden. Beim Schärfen werden die Ergebnisse der einzelnen Varianten verglichen und widersprüchliche Aufnahmen verworfen, sodass das wahrscheinlichste Messergebnis aller Varianten bevorzugt wird. Weitere Informationen finden Sie unter Verbesserung der Leistung von Quantencomputern durch Symmetrisierung.

Wichtig ist, dass beim Scharfzeichnen davon ausgegangen wird, dass die Form der Ausgangsverteilung dünn ist und nur wenige Zustände mit hoher Wahrscheinlichkeit und viele Zustände mit einer Wahrscheinlichkeit von Null aufweist. Wenn diese Annahme nicht zutrifft, kann es zu einer Verzerrung der Wahrscheinlichkeitsverteilung kommen.

Sie können auf die Wahrscheinlichkeiten aus einer geschärften Verteilung im `additional_metadata` Feld `GateModelTaskResult` im Braket Python SDK zugreifen. Beachten Sie, dass beim Scharfzeichnen nicht die Messwerte zurückgegeben werden, sondern stattdessen eine neu normalisierte Wahrscheinlichkeitsverteilung zurückgegeben wird. Der folgende Codeausschnitt zeigt, wie Sie nach dem Scharfzeichnen auf die Verteilung zugreifen können.

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

Arbeiten mit Amazon Braket Hybrid Jobs

Amazon Braket Hybrid Jobs bietet Ihnen die Möglichkeit, hybride quantenklassische Algorithmen auszuführen, die sowohl klassische AWS Ressourcen als auch Quantenverarbeitungseinheiten (QPUs) erfordern. Hybrid Jobs ist darauf ausgelegt, die angeforderten klassischen Ressourcen hochzufahren, Ihren Algorithmus auszuführen und die Instances nach Abschluss freizugeben, sodass Sie nur für das bezahlen, was Sie tatsächlich nutzen.

Hybrid Jobs ist ideal für lang andauernde, iterative Algorithmen, bei denen sowohl klassische Computerressourcen als auch Quantencomputer-Ressourcen verwendet werden. Bei Hybrid Jobs führt Braket Ihren Algorithmus, nachdem Sie ihn zur Ausführung eingereicht haben, in einer skalierbaren, containerisierten Umgebung aus. Sobald der Algorithmus abgeschlossen ist, können Sie die Ergebnisse abrufen.

Darüber hinaus profitieren Quantenaufgaben, die aus einem Hybrid-Job erstellt werden, von einer Warteschlange mit höherer Priorität für das QPU-Zielgerät. Diese Priorisierung stellt sicher, dass Ihre Quantenberechnungen verarbeitet und vor anderen Aufgaben ausgeführt werden, die in der Warteschlange warten. Dies ist besonders vorteilhaft für iterative Hybridalgorithmen, bei denen die Ergebnisse einer Quantenaufgabe von den Ergebnissen früherer Quantenaufgaben abhängen. Beispiele für solche Algorithmen sind der Quantum Approximate Optimization Algorithm (QAOA), der Variational Quantum Eigensolver oder das quantenmechanische Lernen. Sie können den Fortschritt Ihres Algorithmus auch nahezu in Echtzeit überwachen, sodass Sie Kosten, Budget oder benutzerdefinierte Kennzahlen wie Trainingsverlust oder Erwartungswerte verfolgen können.

Sie können auf hybride Jobs in Braket zugreifen, indem Sie:

- Das Amazon Braket Python SDK.
- Die Amazon Braket-Konsole.
- Das Amazon BraketAPI.

In diesem Abschnitt:

- Wann sollten Sie Amazon Braket Hybrid Jobs verwenden
- Einen Hybrid-Job mit Amazon Braket Hybrid Jobs ausführen
- Schlüsselkonzepte für hybride Jobs
- Voraussetzungen

- Erstellen Sie einen Hybrid-Job
- Einen Hybrid-Job stornieren
- Personalisieren Sie Ihren Hybrid-Job
- Verwendung PennyLane mit Amazon Braket
- Verwendung CUDA-Q mit Amazon Braket

Wann sollten Sie Amazon Braket Hybrid Jobs verwenden

Mit Amazon Braket Hybrid Jobs können Sie hybride quantenklassische Algorithmen wie den Variational Quantum Eigensolver (VQE) und den Quantum Approximate Optimization Algorithm (QAOA) ausführen, die klassische Rechenressourcen mit Quantencomputern kombinieren, um die Leistung der heutigen Quantensysteme zu optimieren. Amazon Braket Hybrid Jobs bietet drei Hauptvorteile:

1. Leistung: Amazon Braket Hybrid Jobs bietet eine bessere Leistung als das Ausführen von Hybrid-Algorithmen aus Ihrer eigenen Umgebung. Während Ihr Job ausgeführt wird, hat er vorrangigen Zugriff auf die ausgewählte Ziel-QPU. Aufgaben aus Ihrem Job werden vor anderen Aufgaben ausgeführt, die sich auf dem Gerät in der Warteschlange befinden. Dies führt zu kürzeren und besser vorhersehbaren Laufzeiten für hybride Algorithmen. Amazon Braket Hybrid Jobs unterstützt auch die parametrische Kompilierung. Sie können eine Schaltung mit freien Parametern einreichen und Braket kompiliert die Schaltung einmal, ohne dass für nachfolgende Parameter-Aktualisierungen derselben Schaltung erneut kompiliert werden muss, was zu noch schnelleren Laufzeiten führt.
2. Komfort: Amazon Braket Hybrid Jobs vereinfacht die Einrichtung und Verwaltung Ihrer Computerumgebung und deren Betrieb, während Ihr Hybrid-Algorithmus ausgeführt wird. Sie stellen einfach Ihr Algorithmus-Skript bereit und wählen ein Quantengerät (entweder eine Quantenverarbeitungseinheit oder einen Simulator) aus, auf dem es ausgeführt werden soll. Amazon Braket wartet, bis das Zielgerät verfügbar ist, aktiviert die klassischen Ressourcen, führt die Arbeitslast in vorgefertigten Container-Umgebungen aus, gibt die Ergebnisse an Amazon Simple Storage Service (Amazon S3) zurück und gibt die Rechenressourcen frei.
3. Metriken: Amazon Braket Hybrid Jobs bietet sofort Einblicke in laufende Algorithmen und stellt anpassbare Algorithmusmetriken nahezu in Echtzeit an Amazon CloudWatch und die Amazon Braket-Konsole bereit, sodass Sie den Fortschritt Ihrer Algorithmen verfolgen können.

Einen Hybrid-Job mit Amazon Braket Hybrid Jobs ausführen

Um einen Hybrid-Job mit Amazon Braket Hybrid Jobs auszuführen, müssen Sie zunächst Ihren Algorithmus definieren. Sie können es definieren, indem Sie das Algorithmus-Skript und optional andere Abhängigkeitsdateien mit dem Amazon Braket Python SDK oder PennyLaneschreiben. Wenn Sie andere (Open-Source-oder proprietäre) Bibliotheken verwenden möchten, können Sie mithilfe von Docker, das diese Bibliotheken enthält, Ihr eigenes benutzerdefiniertes Container-Image definieren. Weitere Informationen finden Sie unter Bring Your Own Container (BYOC).

In beiden Fällen erstellen Sie als Nächstes einen Hybrid-Job mithilfe von Amazon BraketAPI, in dem Sie Ihr Algorithmus-Skript oder Ihren Container angeben, das Ziel-Quantengerät auswählen, das der Hybrid-Job verwenden soll, und dann aus einer Vielzahl optionaler Einstellungen wählen. Die für diese optionalen Einstellungen bereitgestellten Standardwerte funktionieren für die meisten Anwendungsfälle. Damit das Zielgerät Ihren Hybrid-Job ausführen kann, haben Sie die Wahl zwischen einer QPU, einem On-Demand-Simulator (wieSV1, DM1 oderTN1) oder der klassischen Hybrid-Job-Instanz selbst. Bei einem On-Demand-Simulator oder einer QPU führt Ihr Hybrid-Job-Container API-Aufrufe an ein Remote-Gerät durch. Bei den eingebetteten Simulatoren ist der Simulator in denselben Container eingebettet wie Ihr Algorithmus-Skript. Die Blitzsimulatoren von PennyLane sind in den vorgefertigten Standard-Container für Hybrid-Jobs eingebettet, den Sie verwenden können. Wenn Sie Ihren Code mit einem eingebetteten PennyLane Simulator oder einem benutzerdefinierten Simulator ausführen, können Sie einen Instanztyp sowie die Anzahl der Instanzen angeben, die Sie verwenden möchten. Die mit den einzelnen Optionen verbundenen Kosten finden Sie auf der Seite mit den Amazon Braket-Preisen.

Wenn es sich bei Ihrem Zielgerät um einen On-Demand-Simulator oder einen eingebetteten Simulator handelt, beginnt Amazon Braket sofort mit der Ausführung des Hybrid-Jobs. Es startet die Hybrid-Job-Instance (Sie können den Instance-Typ im API Aufruf anpassen), führt Ihren Algorithmus aus, schreibt die Ergebnisse in Amazon S3 und gibt Ihre Ressourcen frei. Diese Ressourcenfreigabe stellt sicher, dass Sie nur für das bezahlen, was Sie tatsächlich nutzen.

Die Gesamtzahl der gleichzeitigen Hybrid-Jobs pro Quantenverarbeitungseinheit (QPU) ist begrenzt. Heute kann jeweils nur ein Hybrid-Job auf einer QPU ausgeführt werden. Warteschlangen werden verwendet, um die Anzahl der Hybrid-Jobs zu kontrollieren, die ausgeführt werden dürfen, sodass das zulässige Limit nicht überschritten wird. Wenn es sich bei Ihrem Zielgerät um eine QPU handelt, wird Ihr Hybrid-Job zuerst in die Job-Warteschlange der ausgewählten QPU aufgenommen. Amazon Braket startet die benötigte Hybrid-Job-Instance und führt Ihren Hybrid-Job auf dem Gerät aus. Für die Dauer Ihres Algorithmus hat Ihr Hybrid-Job bevorzugten Zugriff. Das bedeutet, dass Quantenaufgaben aus Ihrem Hybrid-Job vor anderen Braket-Quantenaufgaben in der Warteschlange auf dem Gerät ausgeführt werden, vorausgesetzt, die Job-Quantenaufgaben werden alle paar Minuten an die QPU übermittelt. Sobald Ihr Hybrid-Job abgeschlossen ist, werden Ressourcen freigegeben, was bedeutet, dass Sie nur für das bezahlen, was Sie tatsächlich nutzen.

> **Note**
>
> Die Geräte sind regional und Ihr Hybrid-Job wird auf demselben Gerät ausgeführt AWS-Region wie Ihr primäres Gerät.

Sowohl im Simulator- als auch im QPU-Zielszenario haben Sie die Möglichkeit, benutzerdefinierte Algorithmusmetriken, z. B. die Energie Ihres Hamiltonian, als Teil Ihres Algorithmus zu definieren. Diese Metriken werden automatisch an Amazon gemeldet CloudWatch und von dort aus nahezu in Echtzeit in der Amazon Braket-Konsole angezeigt.

> **Note**
>
> Wenn Sie eine GPU-basierte Instance verwenden möchten, stellen Sie sicher, dass Sie einen der GPU-based Simulatoren verwenden, die mit den eingebetteten Simulatoren auf Braket verfügbar sind (z. B.). `lightning.gpu` Wenn Sie sich für einen der CPU-based eingebetteten Simulatoren entscheiden (z. B., oder`braket:default-simulator`)`lightning.qubit`, wird die GPU nicht verwendet und es können Ihnen unnötige Kosten entstehen.

Schlüsselkonzepte für hybride Jobs

In diesem Abschnitt werden die wichtigsten Konzepte der vom Amazon Braket Python SDK bereitgestellten `AwsQuantumJob.create` Funktion und die Zuordnung zur Container-Dateistruktur erklärt.

Zusätzlich zu der Datei oder den Dateien, aus denen Ihr komplettes Algorithmus-Skript besteht, kann Ihr Hybrid-Job zusätzliche Eingaben und Ausgaben haben. Wenn Ihr Hybrid-Job gestartet wird, kopiert Amazon Braket die im Rahmen der Erstellung des Hybrid-Jobs bereitgestellten Eingaben in den Container, in dem das Algorithmus-Skript ausgeführt wird. Wenn der Hybrid-Job abgeschlossen ist, werden alle während des Algorithmus definierten Ausgaben an den angegebenen Amazon S3 S3-Speicherort kopiert.

> **Note**
>
> Algorithmus-Metriken werden in Echtzeit gemeldet und folgen nicht diesem Ausgabeverfahren.

Amazon Braket bietet auch mehrere Umgebungsvariablen und Hilfsfunktionen, um die Interaktionen mit Container-Eingaben und -Ausgaben zu vereinfachen. Weitere Informationen finden Sie im Paket braket.jobs im Amazon Braket SDK.

In diesem Abschnitt:

- Eingaben
- Outputs
- Umgebungsvariablen
- Hilfsfunktionen

Eingaben

Eingabedaten: Eingabedaten können dem Hybrid-Algorithmus zur Verfügung gestellt werden, indem die Eingabedatendatei, die als Wörterbuch eingerichtet ist, mit dem `input_data` Argument angegeben wird. Der Benutzer definiert das `input_data` Argument innerhalb der `AwsQuantumJob.create` Funktion im SDK. Dadurch werden die Eingabedaten in das Container-Dateisystem an dem in der Umgebungsvariablen angegebenen Speicherort kopiert`"AMZN_BRAKET_INPUT_DIR"`. Einige Beispiele dafür, wie Eingabedaten in einem hybriden

Algorithmus verwendet werden, finden Sie unter QAOA mit Amazon Braket Hybrid Jobs PennyLane und Quantum Machine Learning in Amazon Braket Hybrid Jobs Jupyter-Notebooks.

> **Note**
>
> Wenn die Eingabedaten groß sind (> 1 GB), dauert es lange, bis der Hybrid-Job eingereicht wird. Dies liegt an der Tatsache, dass die lokalen Eingabedaten zuerst in einen S3-Bucket hochgeladen werden, dann der S3-Pfad zur Hybrid-Job-Anfrage hinzugefügt wird und schließlich die Hybrid-Job-Anfrage an den Braket-Service übermittelt wird.

Hyperparameter: Wenn Sie sie weitergeben`hyperparameters`, sind sie unter der Umgebungsvariablen verfügbar. `"AMZN_BRAKET_HP_FILE"`

> **Note**
>
> Weitere Informationen darüber, wie Sie Hyperparameter und Eingabedaten erstellen und diese Informationen dann an das Hybrid-Job-Skript übergeben, finden Sie im Abschnitt Hyperparameter verwenden und auf dieser Github-Seite.

Checkpoints: Um einen Checkpoint anzugeben`job-arn`, dessen Checkpoint Sie in einem neuen Hybrid-Job verwenden möchten, verwenden Sie den Befehl. `copy_checkpoints_from_job` Mit diesem Befehl werden die Checkpoint-Daten in den `checkpoint_configs3Uri` des neuen Hybrid-Jobs kopiert, sodass sie `AMZN_BRAKET_CHECKPOINT_DIR` während der Ausführung des Jobs unter dem in der Umgebungsvariablen angegebenen Pfad verfügbar sind. Die Standardeinstellung ist`None`, was bedeutet, dass Checkpoint-Daten aus einem anderen Hybrid-Job im neuen Hybrid-Job nicht verwendet werden.

Outputs

Quantenaufgaben: Die Ergebnisse der Quantenaufgaben werden am S3-Standort `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks` gespeichert.

Arbeitsergebnisse: Alles, was Ihr Algorithmus-Skript in dem von der Umgebungsvariablen angegebenen Verzeichnis speichert, `"AMZN_BRAKET_JOB_RESULTS_DIR"` wird an den unter angegebenen S3-Speicherort kopiert`output_data_config`. Wenn der Wert nicht angegeben ist, wird standardmäßig der Wert verwendet. `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data` Wir bieten die SDK-Hilfsfunktion **`save_job_result`**, mit

der Sie Ergebnisse bequem in Form eines Wörterbuchs speichern können, wenn Sie sie von Ihrem Algorithmus-Skript aus aufrufen.

Checkpoints: Wenn Sie Checkpoints verwenden möchten, können Sie diese in dem durch die Umgebungsvariable angegebenen Verzeichnis speichern. `"AMZN_BRAKET_CHECKPOINT_DIR"` Sie können stattdessen auch die SDK-Hilfsfunktion `save_job_checkpoint` verwenden.

Algorithmus-Metriken: Sie können Algorithmus-Metriken als Teil Ihres Algorithmus-Skripts definieren, die an Amazon gesendet CloudWatch und in Echtzeit in der Amazon Braket-Konsole angezeigt werden, während Ihr Hybrid-Job ausgeführt wird. Ein Beispiel für die Verwendung von Algorithmus-Metriken finden Sie unter Verwenden von Amazon Braket-Hybrid-Jobs zur Ausführung eines QAOA-Algorithmus.

Weitere Informationen zum Speichern Ihrer Job-Ausgaben finden Sie unter Speichern Ihrer Ergebnisse in der Dokumentation zu Hybrid-Jobs.

Umgebungsvariablen

Amazon Braket bietet mehrere Umgebungsvariablen, um die Interaktionen mit Container-Eingaben und -Ausgaben zu vereinfachen. Der folgende Code listet die Umgebungsvariablen auf, die Braket verwendet.

- `AMZN_BRAKET_INPUT_DIR`— Das Eingabedatenverzeichnis opt/braket/. input/data
- `AMZN_BRAKET_JOB_RESULTS_DIR`— Das Ausgabeverzeichnis opt/braket /model, in das die Job-Ergebnisse geschrieben werden sollen.
- `AMZN_BRAKET_JOB_NAME`— Der Name des Jobs.
- `AMZN_BRAKET_CHECKPOINT_DIR`— Das Checkpoint-Verzeichnis.
- `AMZN_BRAKET_HP_FILE`— Die Datei, die die Hyperparameter enthält.
- `AMZN_BRAKET_DEVICE_ARN`— Der Geräte-ARN (AWS Ressourcenname).
- `AMZN_BRAKET_OUT_S3_BUCKET`— Der Amazon S3 S3-Ausgabe-Bucket, wie in der `CreateJob` Anfrage angegeben`OutputDataConfig`.
- `AMZN_BRAKET_SCRIPT_ENTRY_POINT`— Der Einstiegspunkt, wie in der `CreateJob` Anfrage angegeben`ScriptModeConfig`.
- `AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE`— Der in der `CreateJob` Anfrage angegebene Komprimierungstyp`ScriptModeConfig`.
- `AMZN_BRAKET_SCRIPT_S3_URI`— Der Amazon S3 S3-Speicherort des Benutzerskripts, wie in der `CreateJob` Anfrage angegeben`ScriptModeConfig`.

- `AMZN_BRAKET_TASK_RESULTS_S3_URI`— Der Amazon S3 S3-Speicherort, an dem das SDK standardmäßig die Ergebnisse der Quantenaufgabe für den Job speichern würde.
- `AMZN_BRAKET_JOB_RESULTS_S3_PATH`— Der Amazon S3 S3-Standort, an dem die Auftragsergebnisse gespeichert würden, wie in den `CreateJob` Anfragen angegeben`OutputDataConfig`.
- `AMZN_BRAKET_JOB_TOKEN`— Die Zeichenfolge, die an `CreateQuantumTask` den `jobToken` Parameter für Quantenaufgaben übergeben werden soll, die im Jobcontainer erstellt wurden.

Hilfsfunktionen

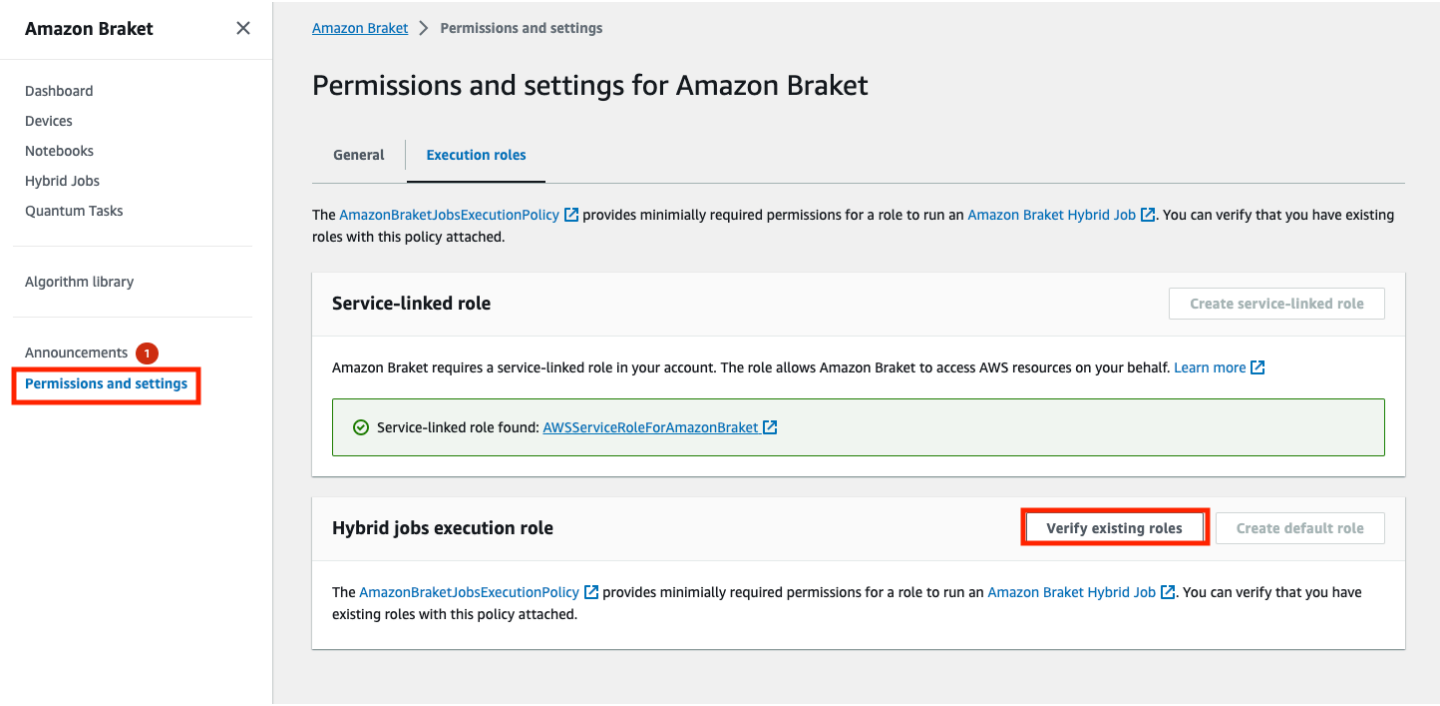
Amazon Braket bietet mehrere Hilfsfunktionen, um die Interaktionen mit Container-Eingaben und -Ausgaben zu vereinfachen. Diese Hilfsfunktionen würden innerhalb des Algorithmus-Skripts aufgerufen, das zur Ausführung Ihres Hybrid-Jobs verwendet wird. Das folgende Beispiel zeigt, wie sie verwendet werden.

```
from braket.jobs import get_checkpoint_dir, get_hyperparameters, get_input_data_dir,
 get_job_device_arn, get_job_name, get_results_dir, save_job_result,
 save_job_checkpoint, load_job_checkpoint

get_checkpoint_dir() # Get the checkpoint directory
get_hyperparameters() # Get the hyperparameters as strings
get_input_data_dir() # Get the input data directory
get_job_device_arn() # Get the device specified by the hybrid job
get_job_name() # Get the name of the hybrid job.
get_results_dir() # Get the path to a results directory
save_job_result(result_data='data') # Save hybrid job results
save_job_checkpoint(checkpoint_data={'key': 'value'}) # Save a checkpoint
load_job_checkpoint() # Load a previously saved checkpoint
```

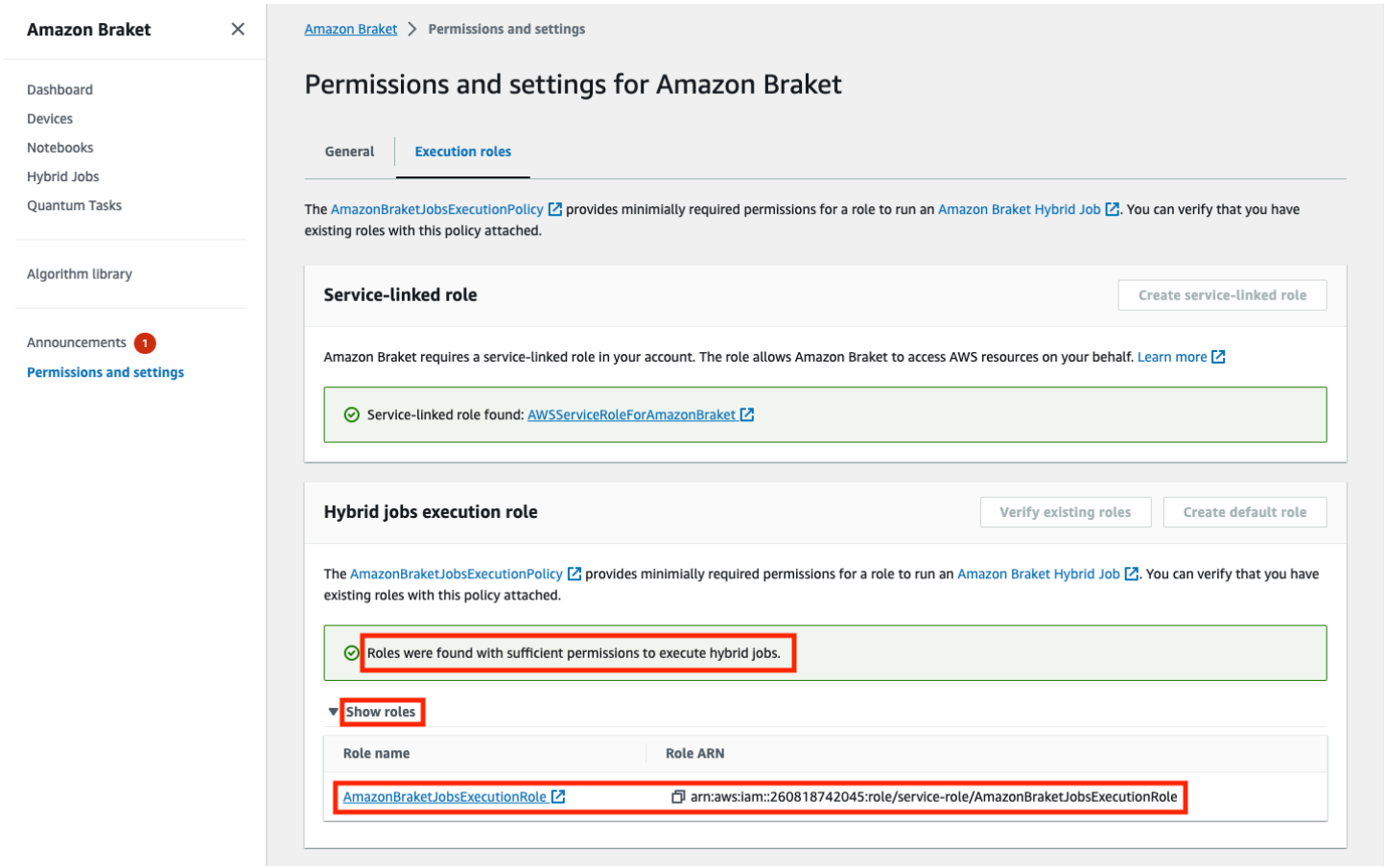
Voraussetzungen

Bevor Sie Ihren ersten Hybridauftrag ausführen, müssen Sie sicherstellen, dass Sie über ausreichende Berechtigungen verfügen, um mit dieser Aufgabe fortzufahren. Um festzustellen, ob Sie über die richtigen Berechtigungen verfügen, wählen Sie im Menü auf der linken Seite der Braket-Konsole die Option Berechtigungen aus. Auf der Seite „Berechtigungsverwaltung für Amazon Braket“ können Sie überprüfen, ob eine Ihrer vorhandenen Rollen über ausreichende Berechtigungen verfügt, um Ihren Hybrid-Job auszuführen, oder führt Sie durch die Erstellung einer Standardrolle, die zur

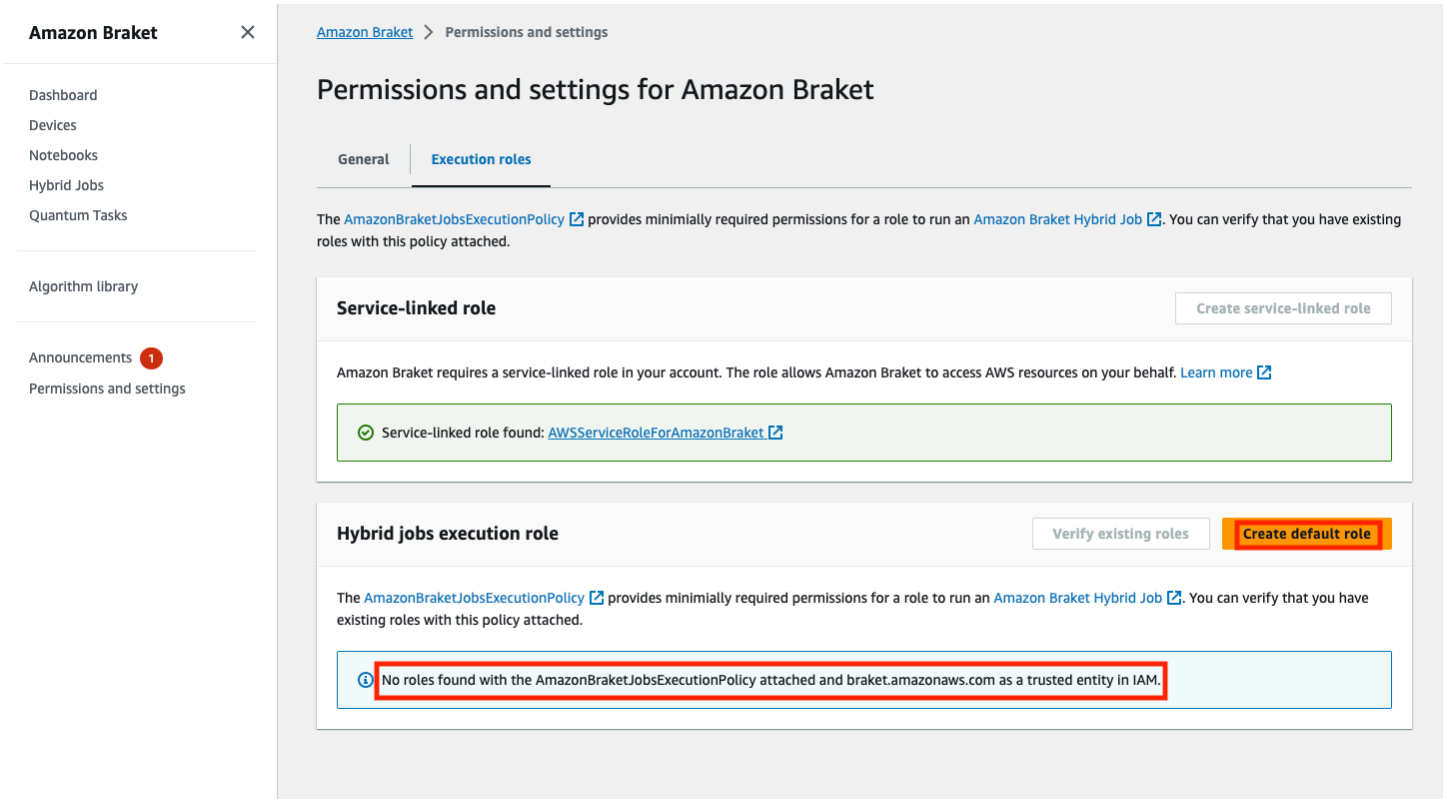
Ausführung Ihres Hybrid-Jobs verwendet werden kann, falls Sie noch nicht über eine solche Rolle verfügen.



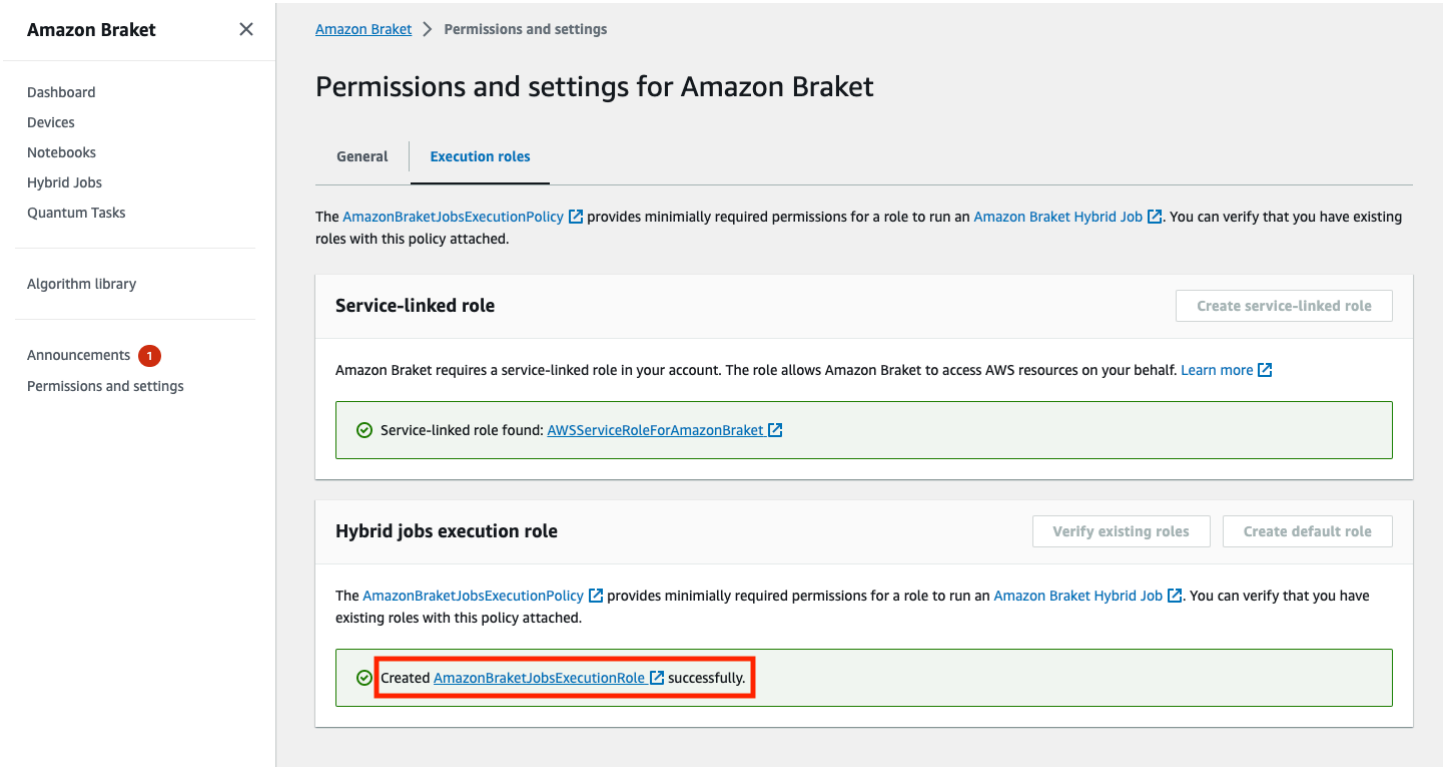
Um zu überprüfen, ob Sie über Rollen mit ausreichenden Berechtigungen für die Ausführung eines Hybrid-Jobs verfügen, klicken Sie auf die Schaltfläche Bestehende Rolle überprüfen. Wenn Sie dies tun, erhalten Sie eine Meldung, dass die Rollen gefunden wurden. Um die Namen der Rollen und ihre Rollen-ARNs zu sehen, klicken Sie auf die Schaltfläche Rollen anzeigen.



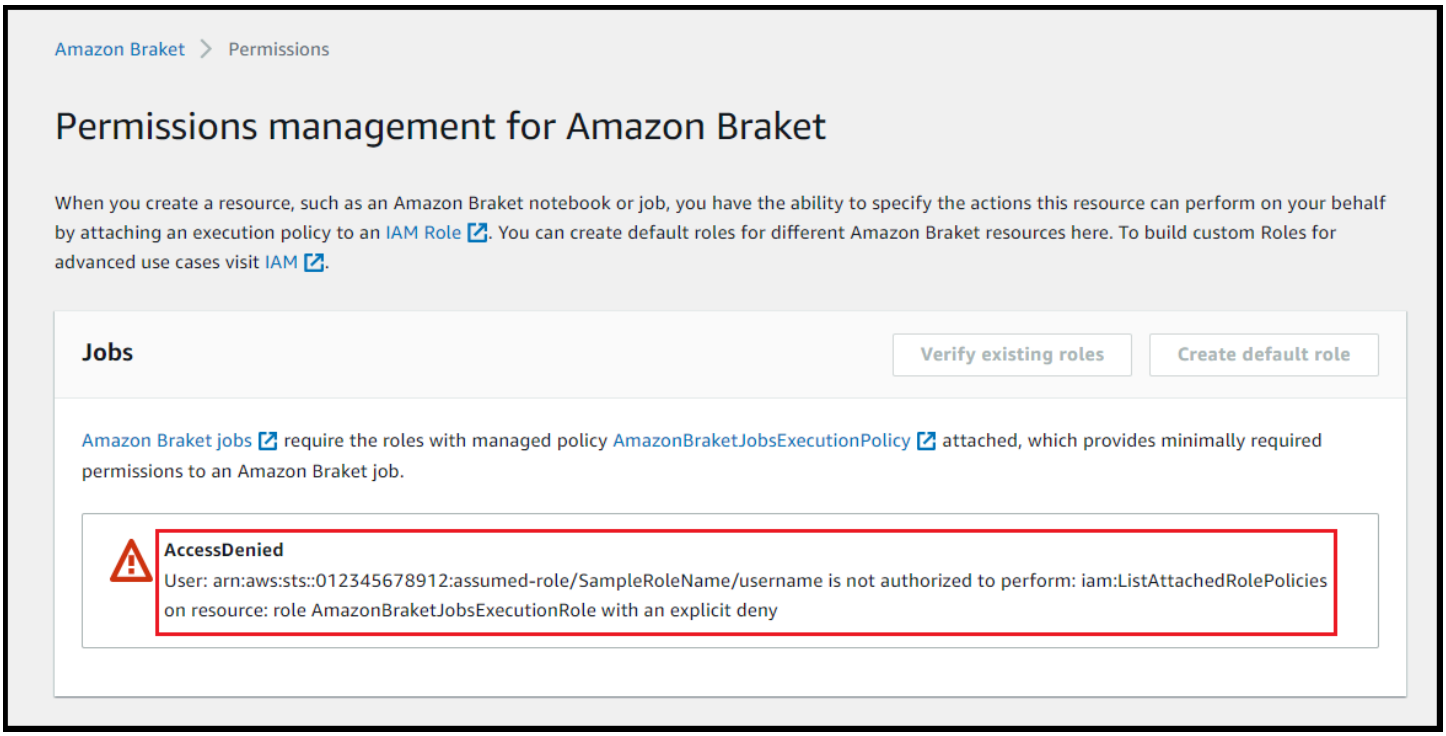
Wenn Sie nicht über eine Rolle mit ausreichenden Berechtigungen verfügen, um einen Hybrid-Job auszuführen, erhalten Sie eine Meldung, dass keine solche Rolle gefunden wurde. Wählen Sie die Schaltfläche Standardrolle erstellen, um eine Rolle mit ausreichenden Berechtigungen zu erhalten.



Wenn die Rolle erfolgreich erstellt wurde, erhalten Sie eine Bestätigungsnachricht.



Wenn Sie nicht berechtigt sind, diese Anfrage zu stellen, wird Ihnen der Zugriff verweigert. Wenden Sie sich in diesem Fall an Ihren internen AWS Administrator.



Erstellen Sie einen Hybrid-Job

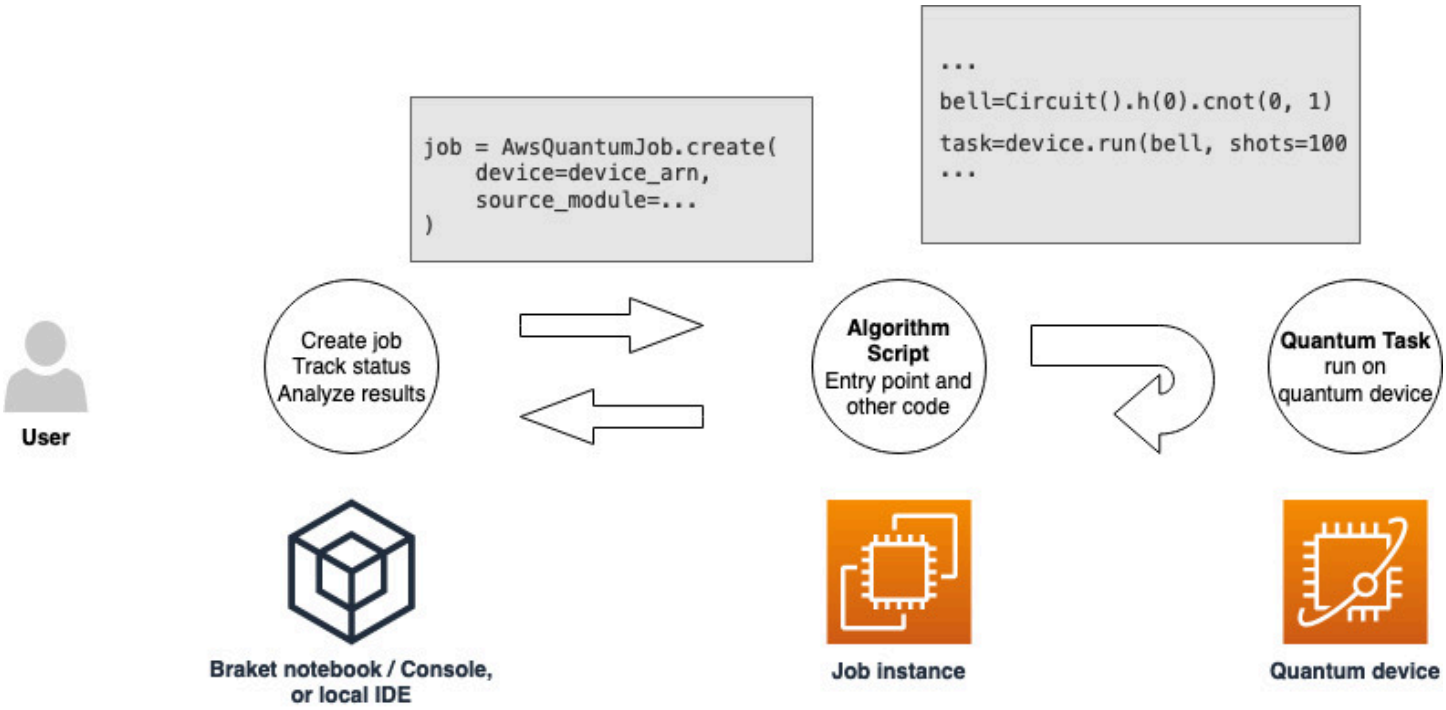
In diesem Abschnitt erfahren Sie, wie Sie mit einem Python-Skript einen Hybrid-Job erstellen. Informationen zum Erstellen eines Hybrid-Jobs aus lokalem Python-Code, z. B. Ihrer bevorzugten integrierten Entwicklungsumgebung (IDE) oder einem Braket-Notizbuch, finden Sie alternativ unterFühren Sie Ihren lokalen Code als Hybrid-Job aus.

In diesem Abschnitt:

- Erstellen und ausführen
- Überwachen Sie Ihre Ergebnisse
- Speichern Sie Ihre Ergebnisse
- Checkpoints verwenden
- Führen Sie Ihren lokalen Code als Hybrid-Job aus
- Verwendung der API mit Hybrid-Jobs
- Erstellen und debuggen Sie einen Hybrid-Job im lokalen Modus

Erstellen und ausführen

Sobald Sie über eine Rolle mit den Berechtigungen zur Ausführung eines Hybrid-Jobs verfügen, können Sie fortfahren. Das Herzstück Ihres ersten Braket-Hybrid-Jobs ist das Algorithmus-Skript. Es definiert den Algorithmus, den Sie ausführen möchten, und enthält die klassischen Logik- und Quantenaufgaben, die Teil Ihres Algorithmus sind. Zusätzlich zu Ihrem Algorithmus-Skript können Sie weitere Abhängigkeitsdateien bereitstellen. Das Algorithmusskript zusammen mit seinen Abhängigkeiten wird als Quellmodul bezeichnet. Der Einstiegspunkt definiert die erste Datei oder Funktion, die in Ihrem Quellmodul ausgeführt wird, wenn der Hybrid-Job gestartet wird.



Betrachten Sie zunächst das folgende grundlegende Beispiel für ein Algorithmus-Skript, das fünf Glockenzustände erzeugt und die entsprechenden Messergebnisse ausgibt.

```
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!")
```

Speichern Sie diese Datei mit dem Namen algorithm_script.py in Ihrem aktuellen Arbeitsverzeichnis auf Ihrem Braket-Notebook oder in der lokalen Umgebung. Die Datei algorithm_script.py hat `start_here()` den geplanten Einstiegspunkt.

Erstellen Sie als Nächstes eine Python-Datei oder ein Python-Notebook im selben Verzeichnis wie die Datei algorithm_script.py. Dieses Skript startet den Hybrid-Job und kümmert sich um jede asynchrone Verarbeitung, z. B. das Drucken des Status oder der wichtigsten Ergebnisse, an denen wir interessiert sind. Dieses Skript muss mindestens Ihr Hybrid-Job-Skript und Ihr primäres Gerät angeben.

> **Note**
>
> Weitere Informationen darüber, wie Sie ein Braket-Notizbuch erstellen oder eine Datei, z. B. die Datei algorithm_script.py, in dasselbe Verzeichnis wie die Notizbücher hochladen, finden Sie unter Run your first circuit using the Amazon Braket Python SDK

In diesem grundlegenden ersten Fall zielen Sie auf einen Simulator ab. Welchen Typ von Quantengerät Sie auch anvisieren, ob es sich um einen Simulator oder eine tatsächliche Quantenverarbeitungseinheit (QPU) handelt, das Gerät, das Sie `device` im folgenden Skript angeben, wird zur Planung des Hybrid-Jobs verwendet und steht den Algorithmus-Skripten als Umgebungsvariable zur Verfügung. `AMZN_BRAKET_DEVICE_ARN`

> **Note**
>
> Sie können nur Geräte verwenden, die in Ihrem Hybrid-Job AWS-Region verfügbar sind. Das Amazon Braket SDK wählt dies AWS-Region auto aus. Beispielsweise kann ein Hybrid-Job in us-east-1 GeräteIonQ,SV1, und verwendenDM1, aber keine TN1 Rigetti Geräte.

Wenn Sie sich für einen Quantencomputer anstelle eines Simulators entscheiden, plant Braket Ihre Hybrid-Jobs so, dass alle ihre Quantenaufgaben mit bevorzugtem Zugriff ausgeführt werden.

```
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
)
```

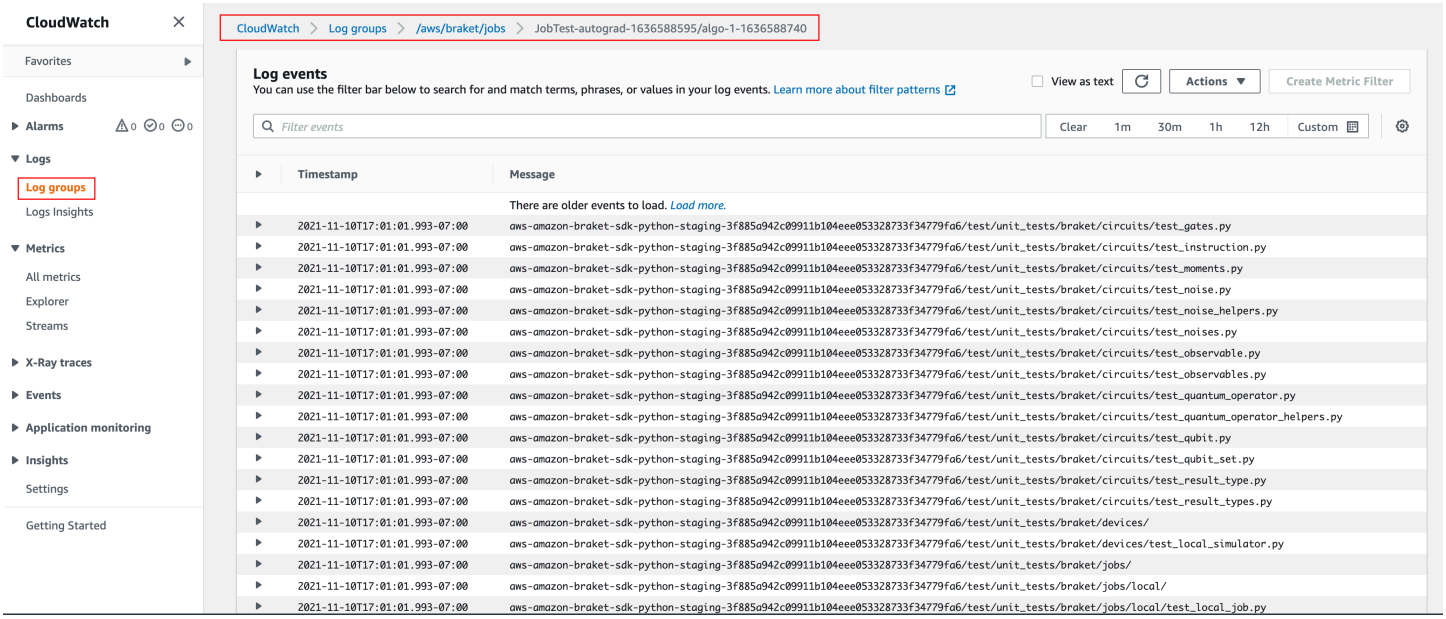
Der Parameter `wait_until_complete=True` legt einen ausführlichen Modus fest, sodass Ihr Job die Ausgabe des aktuellen Jobs ausgibt, während dieser ausgeführt wird. Sie sollten eine Ausgabe sehen, die dem folgenden Beispiel ähnelt.

```
Initializing Braket Job: arn:aws:braket:us-west-2:111122223333:job/braket-job-
default-123456789012
Job queue position: 1
Job queue position: 1
Job queue position: 1
..............
.
.
.
Beginning Setup
Checking for Additional Requirements
Additional Requirements Check Finished
Running Code As Process
Test job started!
Counter({'00': 58, '11': 42})
Counter({'00': 55, '11': 45})
Counter({'11': 51, '00': 49})
Counter({'00': 56, '11': 44})
Counter({'11': 56, '00': 44})
Test job completed!
Code Run Finished
2025-09-24 23:13:40,962 sagemaker-training-toolkit INFO     Reporting training SUCCESS
```

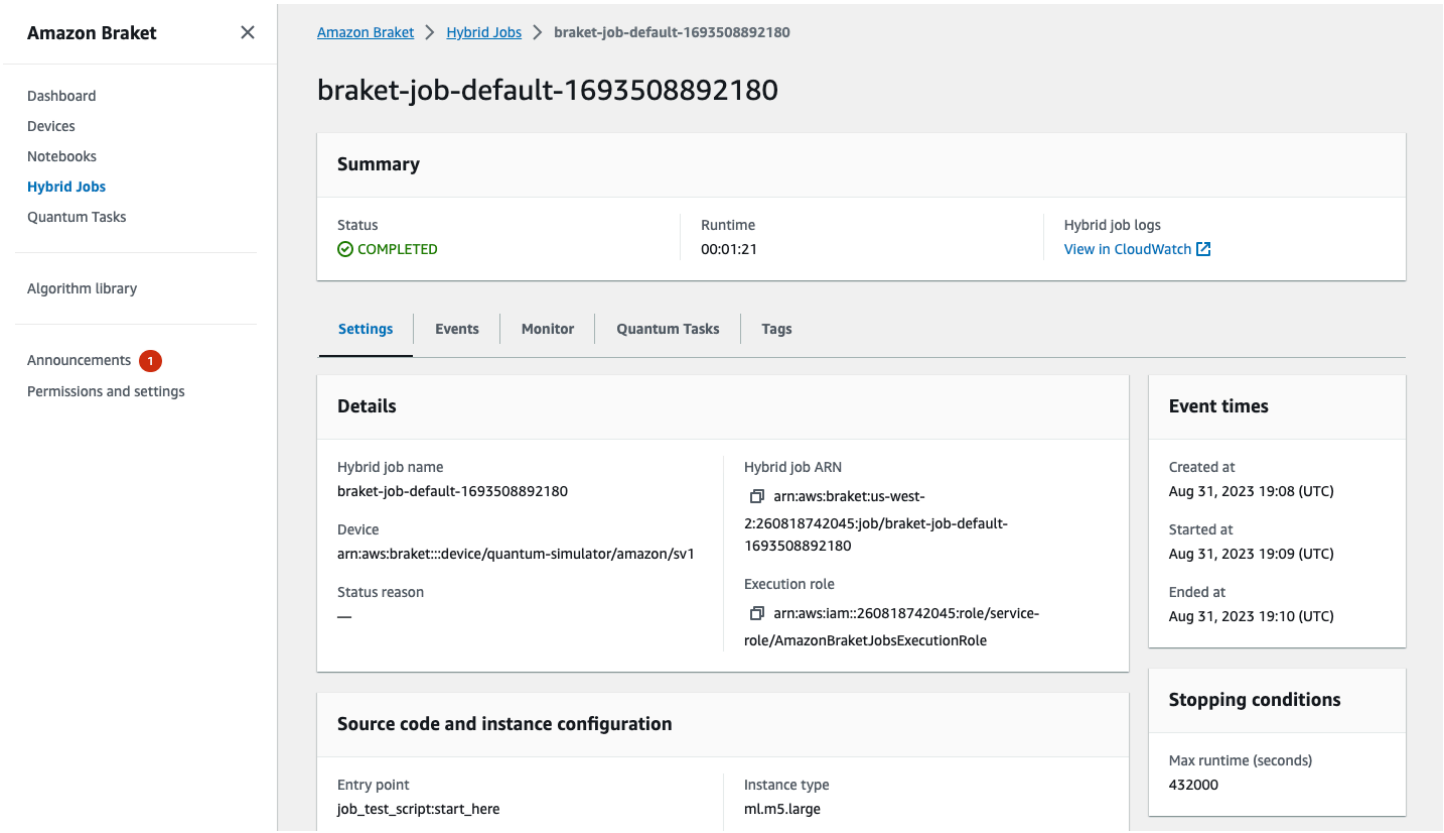
> **Note**
>
> Sie können Ihr maßgeschneidertes Modul auch mit der AwsQuantumJob.createMethode verwenden, indem Sie dessen Speicherort übergeben (entweder den Pfad zu einem lokalen Verzeichnis oder einer lokalen Datei oder einen S3-URI einer Datei tar.gz). Ein funktionierendes Beispiel finden Sie in der Parallelize_training_for_QMLDatei.ipynb im Ordner für hybride Jobs im Github-Repository für Amazon Braket-Beispiele.

Überwachen Sie Ihre Ergebnisse

Alternativ können Sie auf die Protokollausgabe von Amazon zugreifen CloudWatch. Gehen Sie dazu im linken Menü der Jobdetailseite zur Registerkarte Protokollgruppen, wählen Sie die Protokollgruppe und dann den Protokollstream aus`aws/braket/jobs`, der den Jobnamen enthält. Im obigen Beispiel ist dies `braket-job-default-1631915042705/algo-1-1631915190`.



Sie können den Status des Hybrid-Jobs auch in der Konsole einsehen, indem Sie die Seite Hybrid-Jobs und dann Einstellungen auswählen.



Ihr Hybrid-Job erzeugt einige Artefakte in Amazon S3, während er ausgeführt wird. Der Standardname für den S3-Bucket lautet `amazon-braket-<region>-<accountid>` und der Inhalt befindet sich im `jobs/<jobname>/<timestamp>` Verzeichnis. Sie können die S3-Speicherorte konfigurieren, an denen diese Artefakte gespeichert werden, indem Sie `code_location` bei der Erstellung des Hybrid-Jobs mit dem Braket Python SDK einen anderen angeben.

> **Note**
>
> Dieser S3-Bucket muss sich im selben Verzeichnis befinden AWS-Region wie Ihr Job-Skript.

Das `jobs/<jobname>/<timestamp>` Verzeichnis enthält einen Unterordner mit der Ausgabe des Einstiegspunktskripts in einer `model.tar.gz` Datei. Es gibt auch ein Verzeichnis namens`script`, das Ihre Algorithmus-Skriptartefakte in einer `source.tar.gz` Datei enthält. Die Ergebnisse Ihrer eigentlichen Quantenaufgaben befinden sich in dem Verzeichnis mit dem Namen`jobs/<jobname>/tasks`.

Speichern Sie Ihre Ergebnisse

Sie können die vom Algorithmus-Skript generierten Ergebnisse speichern, sodass sie sowohl im Hybrid-Job-Objekt im Hybrid-Job-Skript als auch im Ausgabeordner in Amazon S3 (in einer TAR-ZIP-Datei namens model.tar.gz) verfügbar sind.

Die Ausgabe muss in einer Datei im JSON-Format (JavaScript Object Notation) gespeichert werden. Wenn die Daten nicht ohne Weiteres in Text serialisiert werden können, wie im Fall eines Numpy-Arrays, können Sie eine Option zur Serialisierung mit einem ausgewählten Datenformat angeben. Weitere Informationen finden Sie im Modul braket.jobs.data_persistence.

Um die Ergebnisse der Hybridjobs zu speichern, fügen Sie der Datei algorithm_script.py die folgenden Zeilen hinzu, die mit #ADD kommentiert sind.

```
import os

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


def start_here():

    print("Test job started!")

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

    results = []  # ADD

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

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

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

Sie können dann die Ergebnisse des Jobs aus Ihrem Jobskript anzeigen, indem Sie die mit #ADD **`print(job.result())`**kommentierte Zeile anhängen.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
)

print(job.arn)
while job.state() not in AwsQuantumJob.TERMINAL_STATES:
    print(job.state())
    time.sleep(10)

print(job.state())
print(job.result())   # ADD
```

In diesem Beispiel haben wir die Option entfernt, `wait_until_complete=True` um ausführliche Ausgaben zu unterdrücken. Sie können es zum Debuggen wieder hinzufügen. Wenn Sie diesen Hybrid-Job ausführen, gibt er alle 10 Sekunden den Bezeichner und den`job-arn`, gefolgt vom Status des Hybrid-Jobs, aus`COMPLETED`, bis der Hybrid-Job fertig ist. Danach werden Ihnen die Ergebnisse der Glockenschaltung angezeigt. Sehen Sie sich das folgende Beispiel an.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-123456789012
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Checkpoints verwenden

Mithilfe von Checkpoints können Sie Zwischeniterationen Ihrer Hybrid-Jobs speichern. Im Beispiel für ein Algorithmus-Skript aus dem vorherigen Abschnitt würden Sie die folgenden Zeilen hinzufügen, die mit #ADD kommentiert sind, um Checkpoint-Dateien zu erstellen.

```
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!")
```

Wenn Sie den Hybrid-Job ausführen, erstellt er die Datei -checkpoint-1.json <jobname>in Ihren Hybrid-Job-Artefakten im Checkpoints-Verzeichnis mit einem Standardpfad. `/opt/jobs/checkpoints` Das Hybrid-Job-Skript bleibt unverändert, sofern Sie diesen Standardpfad nicht ändern möchten.

Wenn Sie einen Hybrid-Job von einem Checkpoint laden möchten, der durch einen früheren Hybrid-Job generiert wurde, verwendet `from braket.jobs import load_job_checkpoint` das Algorithmus-Skript. Die Logik zum Laden in Ihr Algorithmus-Skript lautet wie folgt.

```
from braket.jobs import load_job_checkpoint
```

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

Nachdem Sie diesen Checkpoint geladen haben, können Sie Ihre Logik auf der Grundlage des geladenen Inhalts fortsetzen. `checkpoint-1`

> **Note**
>
> Das checkpoint_file_suffix muss mit dem Suffix übereinstimmen, das zuvor bei der Erstellung des Checkpoints angegeben wurde.

Ihr Orchestrierungsskript muss den `job-arn` aus dem vorherigen Hybrid-Job stammenden Job mit der mit #ADD kommentierten Zeile angeben.

```
from braket.aws import AwsQuantumJob

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

Führen Sie Ihren lokalen Code als Hybrid-Job aus

Amazon Braket Hybrid Jobs bietet eine vollständig verwaltete Orchestrierung hybrider quantenklassischer Algorithmen und kombiniert Amazon EC2 EC2-Rechenressourcen mit dem Zugriff auf die Amazon Braket Quantum Processing Unit (QPU). Quantenaufgaben, die in einem Hybrid-Job erstellt wurden, haben Vorrang vor einzelnen Quantenaufgaben, sodass Ihre Algorithmen nicht durch Schwankungen in der Warteschlange für Quantenaufgaben unterbrochen werden. Jede QPU unterhält eine separate Warteschlange für Hybrid-Jobs, wodurch sichergestellt wird, dass jeweils nur ein Hybrid-Job ausgeführt werden kann.

In diesem Abschnitt:

- Erstellen Sie einen Hybrid-Job aus lokalem Python-Code
- Installieren Sie zusätzliche Python-Pakete und Quellcode

- Speichern und laden Sie Daten in eine Hybrid-Job-Instanz
- Bewährte Verfahren für hybride Jobdekorateure

Erstellen Sie einen Hybrid-Job aus lokalem Python-Code

Sie können Ihren lokalen Python-Code als Amazon Braket Hybrid Job ausführen. Sie können dies tun, indem Sie Ihren Code mit einem `@hybrid_job` Decorator annotieren, wie im folgenden Codebeispiel gezeigt. Für benutzerdefinierte Umgebungen können Sie sich dafür entscheiden, einen benutzerdefinierten Container aus Amazon Elastic Container Registry (ECR) zu verwenden.

> **Note**
>
> Standardmäßig wird nur Python 3.12 unterstützt.

Sie können den `@hybrid_job` Decorator verwenden, um eine Funktion mit Anmerkungen zu versehen. Braket wandelt den Code im Decorator in ein Braket-Hybrid-Job-Algorithmus-Skript um. Der Hybrid-Job ruft dann die Funktion im Decorator auf einer Amazon EC2 EC2-Instance auf. Sie können den Fortschritt des Jobs mit `job.state()` oder mit der Braket-Konsole überwachen. Das folgende Codebeispiel zeigt, wie eine Sequenz von fünf Zuständen auf dem State Vector Simulator (SV1) device ausgeführt wird.

```
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}
```

Sie erstellen den Hybrid-Job, indem Sie die Funktion wie normale Python-Funktionen aufrufen. Die Decorator-Funktion gibt jedoch eher das Hybrid-Job-Handle als das Ergebnis der Funktion zurück. Um die Ergebnisse nach Abschluss der Operation abzurufen, verwenden Sie`job.result()`.

```
job = run_hybrid_job(num_tasks=1)
result = job.result()
```

Das Geräteargument im `@hybrid_job` Decorator gibt das Gerät an, auf das der Hybrid-Job vorrangigen Zugriff hat — in diesem Fall den SV1 Simulator. Um die QPU-Priorität zu erhalten, müssen Sie sicherstellen, dass der in der Funktion verwendete Geräte-ARN mit dem im Decorator angegebenen übereinstimmt. Der Einfachheit halber können Sie die Hilfsfunktion verwenden, `get_job_device_arn()` um den in deklarierten Geräte-ARN zu erfassen`@hybrid_job`.

> **Note**
>
> Jeder Hybrid-Job hat mindestens eine Minute Startzeit, da er eine containerisierte Umgebung auf Amazon EC2 erstellt. Für sehr kurze Arbeitslasten, wie z. B. einen einzelnen Schaltkreis oder eine Reihe von Schaltungen, kann es daher ausreichend sein, Quantenaufgaben zu verwenden.

Hyperparameter

Die `run_hybrid_job()` Funktion verwendet das Argument`num_tasks`, um die Anzahl der erstellten Quantenaufgaben zu kontrollieren. Der Hybrid-Job erfasst dies automatisch als Hyperparameter.

> **Note**
>
> Hyperparameter werden in der Braket-Konsole als Zeichenketten angezeigt, die auf 2500 Zeichen begrenzt sind.

Metriken und Protokollierung

Innerhalb der `run_hybrid_job()` Funktion werden Metriken aus iterativen Algorithmen mit `log_metrics` aufgezeichnet. Metriken werden automatisch auf der Braket-Konsolenseite unter der Registerkarte Hybrid-Job dargestellt. Mit dem Braket-Kosten-Tracker können Sie die Kosten für Quantenaufgaben während der Ausführung des Hybrid-Jobs nahezu in Echtzeit verfolgen. Im obigen Beispiel wird der Metrikname „Wahrscheinlichkeit“ verwendet, der die erste Wahrscheinlichkeit aus dem Ergebnistyp aufzeichnet.

Ergebnisse werden abgerufen

Nach Abschluss des Hybrid-Jobs können Sie `job.result()` die Ergebnisse der Hybrid-Jobs abrufen. Alle Objekte in der Rückmeldung werden automatisch von Braket erfasst. Beachten Sie, dass die von der Funktion zurückgegebenen Objekte ein Tupel sein müssen, wobei jedes Element serialisierbar sein muss. Der folgende Code zeigt beispielsweise ein funktionierendes und ein fehlgeschlagenes Beispiel.

```
import numpy as np


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

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

Name des Berufs

Standardmäßig wird der Name für diesen Hybrid-Job aus dem Funktionsnamen abgeleitet. Sie können auch einen benutzerdefinierten Namen mit einer Länge von bis zu 50 Zeichen angeben. Im folgenden Code lautet der Jobname beispielsweise „my-job-name“.

```
@hybrid_job(device=Devices.Amazon.SV1, job_name="my-job-name")
def function():
    pass
```

Lokaler Modus

Lokale Jobs werden erstellt, indem das Argument `local=True` zum Decorator hinzugefügt wird. Dadurch wird der Hybrid-Job in einer containerisierten Umgebung in Ihrer lokalen Computerumgebung, z. B. Ihrem Laptop, ausgeführt. Lokale Jobs haben keine Priorität in der Warteschlange für Quantenaufgaben. In fortgeschrittenen Fällen, wie z. B. bei mehreren Knoten oder MPI, haben lokale Jobs möglicherweise Zugriff auf die erforderlichen Braket-Umgebungsvariablen. Der folgende Code erstellt einen lokalen Hybrid-Job mit dem Gerät als SV1-Simulator.

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

Alle anderen Hybrid-Job-Optionen werden unterstützt. Eine Liste der Optionen finden Sie im Modul braket.jobs.quantum_job_creation.

Installieren Sie zusätzliche Python-Pakete und Quellcode

Sie können Ihre Laufzeitumgebung so anpassen, dass sie Ihre bevorzugten Python-Pakete verwendet. Sie können entweder eine `requirements.txt` Datei, eine Liste von Paketnamen oder Ihren eigenen Container (BYOC) verwenden. Die `requirements.txt` Datei kann beispielsweise andere zu installierende Pakete enthalten.

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

Informationen zum Anpassen einer Laufzeitumgebung mithilfe einer `requirements.txt` Datei finden Sie im folgenden Codebeispiel.

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

```
    return ...
```

Alternativ können Sie die Paketnamen wie folgt als Python-Liste angeben.

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies=["qiskit", "pennylane>=0.31",
 "mitiq==0.29"])
def run_hybrid_job(num_tasks=1):
    return ...
```

Zusätzlicher Quellcode kann entweder als Liste von Modulen oder als einzelnes Modul wie im folgenden Codebeispiel angegeben werden.

```
@hybrid_job(device=Devices.Amazon.SV1, include_modules=["my_module1", "my_module2"])
def run_hybrid_job(num_tasks=1):
    return ...
```

Speichern und laden Sie Daten in eine Hybrid-Job-Instanz

Geben Sie die Eingabe-Trainingsdaten an

Wenn Sie einen Hybrid-Job erstellen, können Sie Eingabe-Trainingsdatensätze bereitstellen, indem Sie einen Amazon Simple Storage Service (Amazon S3) -Bucket angeben. Sie können auch einen lokalen Pfad angeben, dann lädt Braket die Daten automatisch auf Amazon S3 hoch unter. `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>` Wenn Sie einen lokalen Pfad angeben, ist der Kanalname standardmäßig auf „input“ eingestellt. Der folgende Code zeigt eine Numpy-Datei aus dem lokalen Pfad. `data/file.npy`

```
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1, input_data="data/file.npy")
def run_hybrid_job(num_tasks=1):
    data = np.load("data/file.npy")
    return ...
```

Für S3 müssen Sie die `get_input_data_dir()` Hilfsfunktion verwenden.

```
import numpy as np
from braket.jobs import get_input_data_dir

s3_path = "s3://amazon-braket-us-east-1-123456789012/job-data/file.npy"
```

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


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

Sie können mehrere Eingabedatenquellen angeben, indem Sie ein Wörterbuch mit Kanalwerten und S3-URIs oder lokalen Pfaden bereitstellen.

```
import numpy as np
from braket.jobs import get_input_data_dir

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


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

> **Note**
>
> Wenn die Eingabedaten groß sind (> 1 GB), dauert es lange, bis der Job erstellt wird. Dies liegt an den lokalen Eingabedaten, wenn sie zum ersten Mal in einen S3-Bucket hochgeladen werden. Anschließend wird der S3-Pfad zur Jobanforderung hinzugefügt. Schließlich wird die Jobanfrage an den Braket-Service übermittelt.

Ergebnisse auf S3 speichern

Um Ergebnisse zu speichern, die nicht in der Return-Anweisung der dekorierten Funktion enthalten sind, müssen Sie allen Schreibvorgängen von Dateien das richtige Verzeichnis anhängen. Das folgende Beispiel zeigt das Speichern eines Numpy-Arrays und einer Matplotlib-Figur.

```
import matplotlib.pyplot as plt
import numpy as np


@hybrid_job(device=Devices.Amazon.SV1)
def run_hybrid_job(num_tasks=1):
    result = np.random.rand(5)

    # Save a numpy array
    np.save("result.npy", result)

    # Save a matplotlib figure
    plt.plot(result)
    plt.savefig("fig.png")
    return ...
```

Alle Ergebnisse werden in einer Datei mit dem Namen komprimiert. `model.tar.gz` Sie können die Ergebnisse mit der Python-Funktion `job.result()` herunterladen oder indem Sie auf der Hybrid-Job-Seite in der Braket-Managementkonsole zum Ergebnisordner navigieren.

Speichern und von Checkpoints aus fortsetzen

Bei Hybrid-Jobs mit langer Laufzeit wird empfohlen, den Zwischenstatus des Algorithmus regelmäßig zu speichern. Sie können die integrierte `save_job_checkpoint()` Hilfsfunktion verwenden oder Dateien im `AMZN_BRAKET_JOB_RESULTS_DIR` Pfad speichern. Letzteres ist mit der Hilfsfunktion verfügbar`get_job_results_dir()`.

Im Folgenden finden Sie ein minimales funktionierendes Beispiel für das Speichern und Laden von Checkpoints mit einem hybriden Job Decorator:

```
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()
```

Im ersten Hybrid-Job `save_job_checkpoint()` wird ein Wörterbuch aufgerufen, das die Daten enthält, die wir speichern möchten. Standardmäßig muss jeder Wert als Text serialisierbar sein. Um komplexere Python-Objekte wie Numpy-Arrays zu überprüfen, können Sie festlegen. `data_format = PersistedJobDataFormat.PICKLED_V4` Dieser Code erstellt und überschreibt eine Checkpoint-Datei mit dem Standardnamen `<jobname>.json` in Ihren Hybrid-Job-Artefakten in einem Unterordner namens „Checkpoints“.

Um einen neuen Hybrid-Job zu erstellen, um vom Checkpoint aus fortzufahren, müssen wir angeben, `copy_checkpoints_from_job=job_arn` wo `job_arn` sich der Hybrid-Job-ARN des vorherigen Jobs befindet. Dann laden wir `load_job_checkpoint(job_name)` normalerweise vom Checkpoint aus.

Bewährte Verfahren für hybride Jobdekorateure

Machen Sie sich Asynchronität zu eigen

Hybrid-Jobs, die mit der Decorator-Annotation erstellt wurden, sind asynchron — sie werden ausgeführt, sobald die klassischen Ressourcen und die Quantenressourcen verfügbar sind. Sie überwachen den Fortschritt des Algorithmus mithilfe von Braket Management Console oder Amazon CloudWatch. Wenn Sie Ihren Algorithmus zur Ausführung einreichen, führt Braket Ihren Algorithmus in einer skalierbaren containerisierten Umgebung aus und die Ergebnisse werden abgerufen, wenn der Algorithmus abgeschlossen ist.

Führen Sie iterative Variationsalgorythmen aus

Hybrid-Jobs bieten Ihnen die Tools, um iterative quantenklassische Algorithmen auszuführen. Verwenden Sie für reine Quantenprobleme Quantenaufgaben oder eine Reihe von Quantenaufgaben. Der bevorzugte Zugriff auf bestimmte QPUs ist am vorteilhaftesten für lang andauernde Variationsalgorithmen, die mehrere iterative Aufrufe der QPUs mit klassischer Verarbeitung dazwischen erfordern.

Debuggen Sie im lokalen Modus

Bevor Sie einen Hybrid-Job auf einer QPU ausführen, wird empfohlen, ihn zunächst auf dem Simulator SV1 auszuführen, um sicherzustellen, dass er wie erwartet ausgeführt wird. Für Tests in kleinem Maßstab können Sie den lokalen Modus verwenden, um eine schnelle Iteration und ein schnelles Debuggen zu ermöglichen.

Verbessern Sie die Reproduzierbarkeit mit Bring Your Own Container (BYOC)

Erstellen Sie ein reproduzierbares Experiment, indem Sie Ihre Software und ihre Abhängigkeiten in einer containerisierten Umgebung kapseln. Indem Sie Ihren gesamten Code, Ihre Abhängigkeiten und Einstellungen in einem Container verpacken, verhindern Sie potenzielle Konflikte und Versionsprobleme.

Multi-instance verteilte Simulatoren

Um eine große Anzahl von Schaltungen auszuführen, sollten Sie erwägen, die integrierte MPI-Unterstützung zu verwenden, um lokale Simulatoren auf mehreren Instanzen innerhalb eines einzigen Hybrid-Jobs auszuführen. Weitere Informationen finden Sie unter Integrierte Simulatoren.

Verwenden Sie parametrische Schaltungen

Parametrische Schaltungen, die Sie über einen Hybrid-Job einreichen, werden automatisch auf bestimmten QPUs kompiliert. Dabei wird die parametrische Kompilierung verwendet, um die Laufzeiten Ihrer Algorithmen zu verbessern.

Regelmäßiger Checkpoint

Bei Hybrid-Jobs mit langer Laufzeit wird empfohlen, den Zwischenstatus des Algorithmus regelmäßig zu speichern.

Weitere Beispiele, Anwendungsfälle und bewährte Methoden finden Sie in den Amazon Braket-Beispielen. GitHub

Verwendung der API mit Hybrid-Jobs

Über den können Sie direkt auf Amazon Braket Hybrid Jobs zugreifen und mit diesen interagieren. API Standardwerte und praktische Methoden sind jedoch nicht verfügbar, wenn Sie den API direkt verwenden.

> **Note**
>
> Wir empfehlen dringend, dass Sie mit Amazon Braket Hybrid Jobs über das Amazon Braket Python SDK interagieren. Es bietet praktische Standardeinstellungen und Schutzmaßnahmen, die dazu beitragen, dass Ihre Hybrid-Jobs erfolgreich ausgeführt werden.

Dieses Thema behandelt die Grundlagen der Verwendung von. API Wenn Sie sich für die Verwendung der API entscheiden, denken Sie daran, dass dieser Ansatz komplexer sein kann, und bereiten Sie sich auf mehrere Iterationen vor, damit Ihr Hybrid-Job ausgeführt werden kann.

Um die API verwenden zu können, sollte Ihr Konto eine Rolle in der `AmazonBraketFullAccess` verwalteten Richtlinie haben.

> **Note**
>
> Weitere Informationen darüber, wie Sie eine Rolle mit der `AmazonBraketFullAccess` verwalteten Richtlinie erhalten, finden Sie auf der Seite Amazon Braket aktivieren.

Darüber hinaus benötigen Sie eine Ausführungsrolle. Diese Rolle wird an den Dienst übergeben. Sie können die Rolle mit der Amazon Braket-Konsole erstellen. Verwenden Sie die Registerkarte Ausführungsrollen auf der Seite „Berechtigungen und Einstellungen“, um eine Standardrolle für Hybrid-Jobs zu erstellen.

Das `CreateJob` API erfordert, dass Sie alle erforderlichen Parameter für den Hybrid-Job angeben. Um Python zu verwenden, komprimieren Sie Ihre Algorithmus-Skriptdateien in ein Tar-Bundle, z. B. eine Datei input.tar.gz, und führen Sie das folgende Skript aus. Aktualisieren Sie die Teile des Codes in den spitzen Klammern (<>) so, dass sie mit Ihren Kontoinformationen und dem Einstiegspunkt übereinstimmen, die den Pfad, die Datei und die Methode angeben, mit der Ihr Hybrid-Job beginnt.

```
from braket.aws import AwsDevice, AwsSession
import boto3
from datetime import datetime

s3_client = boto3.client("s3")
client = boto3.client("braket")

project_name = "job-test"
job_name = project_name + "-" + datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")
```

```
bucket = "amazon-braket-<your_bucket>"
s3_prefix = job_name

job_script = "input.tar.gz"
job_object = f"{s3_prefix}/script/{job_script}"
s3_client.upload_file(job_script, bucket, job_object)

input_data = "inputdata.csv"
input_object = f"{s3_prefix}/input/{input_data}"
s3_client.upload_file(input_data, bucket, input_object)

job = client.create_job(
    jobName=job_name,
    roleArn="arn:aws:iam::<your_account>:role/service-role/
AmazonBraketJobsExecutionRole",  # https://docs.aws.amazon.com/braket/latest/
developerguide/braket-manage-access.html#about-amazonbraketjobsexecution
    algorithmSpecification={
        "scriptModeConfig": {
            "entryPoint": "<your_execution_module>:<your_execution_method>",
            "containerImage": {"uri": "292282985366.dkr.ecr.us-west-1.amazonaws.com/
amazon-braket-base-jobs:1.0-cpu-py37-ubuntu18.04"},   # Change to the specific region
 you are using
            "s3Uri": f"s3://{bucket}/{job_object}",
            "compressionType": "GZIP"
        }
    },
    inputDataConfig=[
        {
            "channelName": "hellothere",
            "compressionType": "NONE",
            "dataSource": {
                "s3DataSource": {
                    "s3Uri": f"s3://{bucket}/{s3_prefix}/input",
                    "s3DataType": "S3_PREFIX"
                }
            }
        }
    ],
    outputDataConfig={
        "s3Path": f"s3://{bucket}/{s3_prefix}/output"
    },
    instanceConfig={
        "instanceType": "ml.m5.large",
        "instanceCount": 1,
```

```
        "volumeSizeInGb": 1
    },
    checkpointConfig={
        "s3Uri":  f"s3://{bucket}/{s3_prefix}/checkpoints",
        "localPath": "/opt/omega/checkpoints"
    },
    deviceConfig={
        "priorityAccess": {
            "devices": [
                "arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3"
            ]
        }
    },
    hyperParameters={
        "hyperparameter key you wish to pass": "<hyperparameter value you wish to
 pass>",
    },
    stoppingCondition={
        "maxRuntimeInSeconds": 1200,
        "maximumTaskLimit": 10
    },
)
```

Sobald Sie Ihren Hybrid-Job erstellt haben, können Sie über die `GetJob` API oder die Konsole auf die Details des Hybrid-Jobs zugreifen. Verwenden Sie den folgenden Python-Befehl, um die Hybrid-Jobdetails aus der Python-Sitzung abzurufen, in der Sie den `createJob` Code wie im vorherigen Beispiel ausgeführt haben.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

Um einen Hybridjob abzubrechen, rufen Sie den `CancelJob` API mit dem Amazon Resource Name Namen des Jobs ('JobArn') auf.

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

Sie können im Rahmen der `createJob` API Verwendung des `checkpointConfig` Parameters Checkpoints angeben.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
```

```
    },
```

> **Note**
>
> Der LocalPath von `checkpointConfig` darf nicht mit einem der folgenden reservierten Pfade beginnen:`/opt/ml`, `/opt/braket/tmp`, oder. `/usr/local/nvidia`

Erstellen und debuggen Sie einen Hybrid-Job im lokalen Modus

Wenn Sie einen neuen Hybridalgorithmus erstellen, hilft Ihnen der lokale Modus beim Debuggen und Testen Ihres Algorithmus-Skripts. Der lokale Modus ist eine Funktion, mit der Sie Code ausführen können, den Sie in Amazon Braket Hybrid Jobs verwenden möchten, ohne dass Braket die Infrastruktur für die Ausführung des Hybrid-Jobs verwalten muss. Führen Sie Hybrid-Jobs stattdessen lokal auf Ihrer Amazon Braket Notebook-Instance oder auf einem bevorzugten Client wie einem Laptop oder Desktop-Computer aus.

Im lokalen Modus können Sie weiterhin Quantenaufgaben an tatsächliche Geräte senden, aber Sie profitieren nicht von den Leistungsvorteilen, wenn Sie sie im lokalen Modus mit einer echten Quantenverarbeitungseinheit (QPU) ausführen.

Um den lokalen Modus zu verwenden, ändern Sie ihn `AwsQuantumJob` an der `LocalQuantumJob` Stelle, an der er in Ihrem Programm vorkommt. Um beispielsweise das Beispiel aus Create your first hybrid job (Erstellen Sie Ihren ersten Hybrid-Job) auszuführen, bearbeiten Sie das Hybrid-Job-Skript im Code wie folgt.

```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

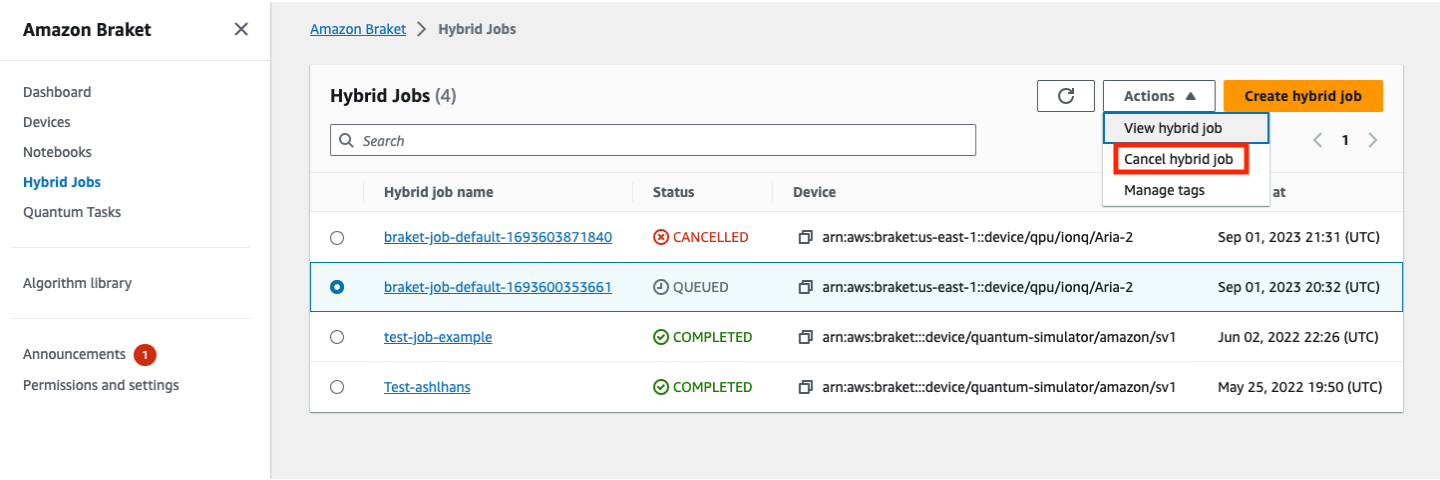
> **Note**
>
> Docker, das bereits in den Amazon Braket-Notebooks vorinstalliert ist, muss in Ihrer lokalen Umgebung installiert sein, um diese Funktion nutzen zu können. Anweisungen zur Installation

von Docker finden Sie auf der Seite Get Docker. Außerdem werden nicht alle Parameter im lokalen Modus unterstützt.

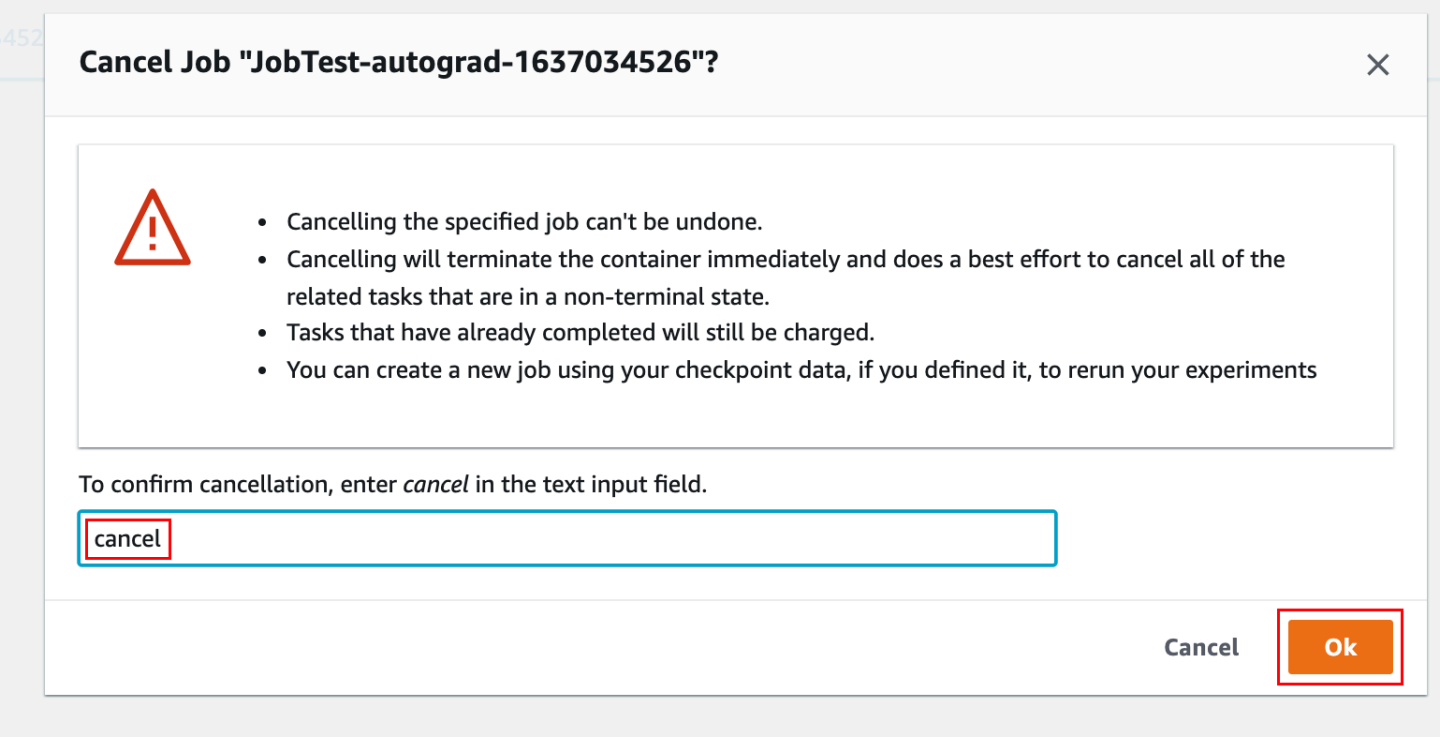
Einen Hybrid-Job stornieren

Möglicherweise müssen Sie einen Hybrid-Job stornieren, der sich nicht im Terminalmodus befindet. Dies kann entweder in der Konsole oder mit Code erfolgen.

Um Ihren Hybrid-Job in der Konsole zu stornieren, wählen Sie auf der Seite Hybrid-Jobs den Hybrid-Job aus, den Sie stornieren möchten, und wählen Sie dann im Drop-down-Menü Aktionen die Option Hybrid-Job stornieren aus.



Um die Stornierung zu bestätigen, geben Sie in das Eingabefeld Abbrechen ein, wenn Sie dazu aufgefordert werden, und wählen Sie dann OK.



Um Ihren Hybrid-Job mithilfe von Code aus dem Braket Python SDK abzubrechen, identifizieren Sie den `job_arn` Hybrid-Job mit und rufen Sie dann den entsprechenden `cancel` Befehl auf, wie im folgenden Code gezeigt.

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

Der `cancel` Befehl beendet den klassischen Hybrid-Job-Container sofort und bemüht sich nach besten Kräften, alle zugehörigen Quantenaufgaben abzubrechen, die sich noch in einem nicht-terminalen Zustand befinden.

Personalisieren Sie Ihren Hybrid-Job

Amazon Braket bietet mehrere Möglichkeiten, die Ausführung Ihrer Hybrid-Jobs anzupassen, sodass Sie die Umgebung an Ihre spezifischen Bedürfnisse anpassen können. In diesem Abschnitt werden Optionen für die Anpassung von Hybrid-Jobs untersucht, von der Definition der Algorithmus-Skriptumgebung bis hin zur Verwendung Ihres eigenen Containers. Sie erfahren, wie Sie Ihren Workflow mithilfe von Hyperparametern optimieren, Job-Instanzen konfigurieren und die parametrische Kompilierung nutzen können, um die Leistung zu verbessern. Diese Anpassungstechniken helfen Ihnen dabei, das Potenzial Ihrer hybriden Quantenberechnungen auf Amazon Braket zu maximieren.

In diesem Abschnitt:

- Definieren Sie die Umgebung für Ihr Algorithmus-Skript
- Hyperparameter verwenden
- Konfigurieren Sie Ihre Hybrid-Job-Instance
- Verwendung der parametrischen Kompilierung zur Beschleunigung von Hybrid-Jobs

Definieren Sie die Umgebung für Ihr Algorithmus-Skript

Amazon Braket unterstützt Umgebungen, die durch Container für Ihr Algorithmus-Skript definiert sind:

- Ein Basiscontainer (der Standard, wenn keiner angegeben `image_uri` ist)
- Ein Container mit CUDA-Q
- Ein Container mit Tensorflow und PennyLane
- Ein Container mit PyTorch, und PennyLane CUDA-Q

Die folgende Tabelle enthält Einzelheiten zu den Containern und den Bibliotheken, die sie enthalten.

Amazon Braket-Behälter

Typ	Base	CUDA-Q	TensorFlow	PyTorch
Bild-URI	292282985 366.dkr.ecr.us-west-2.amazo naws. com/ amazon-Braket-Base-Jobs: neueste	292282985 366.dkr.ecr.us-west-2.amazo naws. com/ amazon-Braket-Cudaq-Jobs: neueste	292282985 366.dkr.ecr.us-east-1.amazo naws. com/ amazon-Braket-TensorFlow-Jobs: neueste	292282985 366.dkr.ecr.us-west-2.amazo naws. com/ amazon-Jobs bei braket-pytorch-aktuell
Geerbte Bibliotheken		• Amazon-Br aket-Stan dardsimulator • Amazon-Br aket-Penn ylane-Plug-In	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy

Typ	Base	CUDA-Q	TensorFlow	PyTorch
		• Amazon-Braket-Schemas • Amazon-Braket-SDK • awscli • botocore • boto3 • Schreibtisch • Matplotlib • numpy • pandas • PennyLane • PennyLane-Lightning • Qiskit-Bracket-Anbieter • Anforderungen • Ausbildung zum Weisen • Scikit-learn • scipy		

Typ	Base	CUDA-Q	TensorFlow	PyTorch
Zusätzliche Bibliotheken	• Amazon-Braket-Standardsimulator • Amazon-Braket-Pennylane-Plug-In • Amazon-Braket-Schemas • Amazon-Braket-SDK • awscli • boto3 • ipykernel • Matplotlib • netzwerke • numpy • Openbabel • pandas • PennyLane • Protobug • PSI 4 • rsa • scipy	• Cudaq • cudaq-qec • Cudaq-Solver	• Standard-Simulator für Amazon-Brackets • Amazon-Braket-Pennylane-Plug-In • Amazon-Braket-Schemas • Amazon-Braket-SDK • ipykernel • Keras • Matplotlib • netzwerke • Openbabel • PennyLane • Protobug • PSI 4 • rsa • PennyLane-Lightning-gpu • Cu-Quantum	• Standards imulator für Amazon-Brackets • Amazon-Braket-Pennylane-Plug-In • Amazon-Braket-Schemas • Amazon-Braket-SDK • ipykernel • Keras • Matplotlib • netzwerke • Openbabel • PennyLane • Protobug • PSI 4 • rsa • PennyLane-Lightning-gpu • Cu-Quantum • Cudaq • cudaq-qec • Cudaq-Solver

Sie können die Open-Source-Container-Definitionen unter -braket-containers einsehen und darauf zugreifen. aws/amazon Wählen Sie den Container, der am besten zu Ihrem Anwendungsfall passt. Sie können jede der verfügbaren AWS Regionen in Braket verwenden (us-east-1, us-west-1, us-

west-2, eu-north-1, eu-west-2), aber die Container-Region muss mit der Region für Ihren Hybrid-Job übereinstimmen. Geben Sie das Container-Image an, wenn Sie einen Hybrid-Job erstellen, indem Sie Ihrem Aufruf im Hybrid-Job-Skript eines der folgenden drei Argumente hinzufügen. `create(…)` Sie können zur Laufzeit zusätzliche Abhängigkeiten in dem Container Ihrer Wahl installieren (auf Kosten des Starts oder der Laufzeit), da die Amazon Braket-Container über eine Internetverbindung verfügen. Das folgende Beispiel bezieht sich auf die Region us-west-2.

- Basisbild: image_uri="292282985366.dkr.ecr.us-west-2.amazonaws. com/amazon-braket-base-jobs: neueste“
- CUDA-Q bild: image_uri="292282985366.dkr.ecr.us-west-2.amazonaws. com/amazon-braket-cudaq-jobs: neueste“
- Tensorflow-Bild: image_uri="292282985366.dkr.ecr.us-west-2.amazonaws. com/amazon-braket-tensorflow-jobs: neueste“
- PyTorch bild: image_uri="292282985366.dkr.ecr.us-west-2.amazonaws. com/amazon-braket-pytorch-jobs: neueste“

`image-uris`Sie können auch mit der Funktion im Braket-SDK abgerufen werden. `retrieve_image()` Amazon Das folgende Beispiel zeigt, wie sie aus dem AWS-Region US-West-2 abgerufen werden können.

```
from braket.jobs.image_uris import retrieve_image, Framework

image_uri_base = retrieve_image(Framework.BASE, "us-west-2")
image_uri_cudaq = retrieve_image(Framework.CUDAQ, "us-west-2")
image_uri_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Verwendung Ihres eigenen Containers (BYOC)

Amazon Braket Hybrid Jobs bietet drei vorgefertigte Container für die Ausführung von Code in verschiedenen Umgebungen. Wenn einer dieser Container Ihren Anwendungsfall unterstützt, müssen Sie nur Ihr Algorithmus-Skript angeben, wenn Sie einen Hybrid-Job erstellen. Geringfügige fehlende Abhängigkeiten können mithilfe von Ihrem Algorithmus-Skript oder aus einer `requirements.txt` Datei hinzugefügt `pip` werden.

Falls keiner dieser Container Ihren Anwendungsfall unterstützt oder Sie diese erweitern möchten, unterstützt Braket Hybrid Jobs die Ausführung von Hybrid-Jobs mit Ihrem eigenen benutzerdefinierten

Docker Container-Image oder Bring Your Own Container (BYOC). Stellen Sie sicher, dass es die richtige Funktion für Ihren Anwendungsfall ist.

In diesem Abschnitt:

- Wann ist es die richtige Entscheidung, meinen eigenen Container mitzubringen?
- Rezept für das Mitbringen eines eigenen Containers
- Braket-Hybrid-Jobs in Ihrem eigenen Container ausführen

Wann ist es die richtige Entscheidung, meinen eigenen Container mitzubringen?

Bring Your Own Container (BYOC) zu Braket Hybrid Jobs bietet die Flexibilität, Ihre eigene Software zu verwenden, indem Sie sie in einer Paketumgebung installieren. Abhängig von Ihren spezifischen Anforderungen gibt es möglicherweise Möglichkeiten, dieselbe Flexibilität zu erreichen, ohne den gesamten URI-Zyklus BYOC Docker Build — Amazon ECR-Upload — benutzerdefiniertes Image durchlaufen zu müssen.

> **Note**
>
> BYOC ist möglicherweise nicht die richtige Wahl, wenn Sie eine kleine Anzahl zusätzlicher Python-Pakete (in der Regel weniger als 10) hinzufügen möchten, die öffentlich verfügbar sind. Zum Beispiel, wenn Sie verwenden. PyPi

In diesem Fall können Sie eines der vorgefertigten Braket-Images verwenden und dann bei der Einreichung des Jobs eine `requirements.txt` Datei in Ihr Quellverzeichnis aufnehmen. Die Datei wird automatisch gelesen und `pip` installiert die Pakete mit den angegebenen Versionen wie gewohnt. Wenn Sie eine große Anzahl von Paketen installieren, kann sich die Laufzeit Ihrer Jobs erheblich verlängern. Überprüfen Sie die Python- und gegebenenfalls die CUDA-Version des vorgefertigten Containers, den Sie verwenden möchten, um zu testen, ob Ihre Software funktioniert.

BYOC ist erforderlich, wenn Sie eine Nicht-Python-Sprache (wie C++ oder Rust) für Ihr Job-Skript verwenden möchten oder wenn Sie eine Python-Version verwenden möchten, die nicht über die vorgefertigten Braket-Container verfügbar ist. Es ist auch eine gute Wahl, wenn:

- Sie verwenden Software mit einem Lizenzschlüssel, und Sie müssen diesen Schlüssel auf einem Lizenzserver authentifizieren, um die Software ausführen zu können. Mit BYOC können Sie den Lizenzschlüssel in Ihr Docker Image einbetten und Code zur Authentifizierung hinzufügen.

- Sie verwenden Software, die nicht öffentlich verfügbar ist. Die Software wird beispielsweise in einem privaten GitHub Repository GitLab oder in einem Repository gehostet, für dessen Zugriff Sie einen bestimmten SSH-Schlüssel benötigen.
- Sie müssen eine große Softwaresuite installieren, die nicht in den von Braket bereitgestellten Containern verpackt ist. BYOC ermöglicht es Ihnen, lange Startzeiten für Ihre Hybrid-Job-Container aufgrund der Softwareinstallation zu vermeiden.

BYOC ermöglicht es Ihnen auch, Ihr benutzerdefiniertes SDK oder Ihren Algorithmus für Kunden verfügbar zu machen, indem Sie einen Docker Container mit Ihrer Software erstellen und diesen Ihren Benutzern zur Verfügung stellen. Sie können dies tun, indem Sie die entsprechenden Berechtigungen in Amazon ECR festlegen.

> **Note**
>
> Sie müssen alle geltenden Softwarelizenzen einhalten.

Rezept für das Mitbringen eines eigenen Containers

In diesem Abschnitt finden Sie eine schrittweise Anleitung bring your own container (BYOC) dazu, was Sie für Braket Hybrid Jobs benötigen — die Skripts, Dateien und Schritte, um sie zu kombinieren, um Ihre benutzerdefinierten Docker Images zum Laufen zu bringen. Die Rezepte für zwei häufige Fälle:

1. Installieren Sie zusätzliche Software in einem Docker Image und verwenden Sie in Ihren Jobs nur Python-Algorithmus-Skripte.
2. Verwenden Sie Algorithmusskripte, die in einer anderen Sprache als Python geschrieben wurden, mit Hybrid Jobs oder einer anderen CPU-Architektur als x86.

Die Definition des Container-Eintragsskripts ist für Fall 2 komplexer.

Wenn Braket Ihren Hybrid-Job ausführt, startet es die angeforderte Anzahl und Art von Amazon EC2 EC2-Instances und führt dann das Docker Image aus, das durch die Image-URI angegeben wurde, die zur Auftragserstellung eingegeben wurde. Wenn Sie die BYOC-Funktion verwenden, geben Sie eine Bild-URI an, die in einem privaten Amazon ECR-Repository gehostet wird, auf das Sie Lesezugriff haben. Braket Hybrid Jobs verwendet dieses benutzerdefinierte Image, um den Job auszuführen.

Die spezifischen Komponenten, die Sie benötigen, um ein Docker Image zu erstellen, das mit Hybrid-Jobs verwendet werden kann. Wenn Sie mit dem Schreiben und Erstellen nicht vertraut sind`Dockerfiles`, lesen Sie in der Dockerfile-Dokumentation und in der Dokumentation nach. Amazon ECR CLI

Voraussetzungen:

- Ein Basis-Image für Ihr Dockerfile
- (Optional) Ein modifiziertes Container-Einstiegspunktskript
- Installieren Sie die benötigte Software und das Container-Skript mit Dockerfile

Ein Basis-Image für Ihr Dockerfile

Wenn Sie Python verwenden und Software zusätzlich zu dem installieren möchten, was in den von Braket bereitgestellten Containern bereitgestellt wird, ist eine Option für ein Basis-Image eines der Braket-Container-Images, die in unserem GitHub Repo und auf Amazon ECR gehostet werden. Sie müssen sich bei Amazon ECR authentifizieren, um das Image abzurufen und darauf aufzubauen. Die erste Zeile Ihrer BYOC-Datei Docker könnte beispielsweise wie folgt lauten: `FROM [IMAGE_URI_HERE]`

Füllen Sie als Nächstes den Rest aus, Dockerfile um die Software zu installieren und einzurichten, die Sie dem Container hinzufügen möchten. Die vorgefertigten Braket-Images enthalten bereits das entsprechende Container-Einstiegspunkt-Skript, sodass Sie sich keine Gedanken darüber machen müssen, dieses einzubeziehen.

Wenn Sie eine Nicht-Python-Sprache wie C++, Rust oder Julia verwenden möchten, oder wenn Sie ein Image für eine Nicht-x86-CPU-Architektur wie ARM erstellen möchten, müssen Sie möglicherweise auf einem öffentlichen Barebone-Image aufbauen. Viele solcher Bilder finden Sie in der Amazon Elastic Container Registry Public Gallery. Stellen Sie sicher, dass Sie eine auswählen, die für die CPU-Architektur und gegebenenfalls für die GPU, die Sie verwenden möchten, geeignet ist.

(Optional) Ein modifiziertes Container-Einstiegspunktskript

> **Note**
>
> Wenn Sie einem vorgefertigten Braket-Image nur zusätzliche Software hinzufügen, können Sie diesen Abschnitt überspringen.

Um Nicht-Python-Code als Teil Ihres Hybrid-Jobs auszuführen, ändern Sie das Python-Skript, das den Container-Einstiegspunkt definiert. Zum Beispiel das `braket_container.py`Python-Skript auf dem Amazon Braket Github. Dies ist das Skript, das die von Braket vorgefertigten Images verwenden, um Ihr Algorithmus-Skript zu starten und die entsprechenden Umgebungsvariablen festzulegen. Das Container-Einstiegspunktskript selbst muss in Python sein, kann aber Nicht-Python-Skripte starten. In dem vorgefertigten Beispiel können Sie sehen, dass Python-Algorithmus-Skripts entweder als Python-Unterprozess oder als vollständig neuer Prozess gestartet werden. Indem Sie diese Logik ändern, können Sie das Einstiegspunktskript aktivieren, um Skripten zu starten, die keine Python-Algorithmen sind. Sie könnten beispielsweise die `thekick_off_customer_script()`Funktion so ändern, dass sie Rust-Prozesse in Abhängigkeit von der Dateinamenerweiterung startet.

Sie können sich auch dafür entscheiden, eine komplett neue zu schreiben`braket_container.py`. Es sollte Eingabedaten, Quellarchive und andere notwendige Dateien aus Amazon S3 in den Container kopieren und die entsprechenden Umgebungsvariablen definieren.

Installieren Sie die benötigte Software und das Container-Skript mit **`Dockerfile`**

> **Note**
>
> Wenn Sie ein vorgefertigtes Braket-Image als Docker Basis-Image verwenden, ist das Container-Skript bereits vorhanden.

Wenn Sie im vorherigen Schritt ein modifiziertes Container-Skript erstellt haben, müssen Sie es in den Container kopieren und die Umgebungsvariable `SAGEMAKER_PROGRAM` oder den Namen Ihres neuen Container-Einstiegspunktskripts definieren. `braket_container.py`

Im Folgenden finden Sie ein Beispiel für eine`Dockerfile`, mit der Sie Julia auf GPU-accelerated Jobs-Instanzen verwenden können:

```
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
```

In diesem Beispiel werden Skripte heruntergeladen und ausgeführt, die von bereitgestellt werden AWS , um die Einhaltung aller relevanten Open-Source Lizenzen sicherzustellen. Zum Beispiel durch die korrekte Zuweisung von installiertem Code, der von einem MIT license gesteuert wird.

Wenn Sie nicht-öffentlichen Code einbinden müssen, z. B. Code, der in einem privaten GitLab Speicher GitHub oder einem Repository gehostet wird, betten Sie keine SSH-Schlüssel in das Docker Image ein, um darauf zuzugreifen. Verwenden Sie stattdessen Docker Compose when you build, um den Zugriff auf SSH auf dem Host-Computer Docker zu ermöglichen, auf dem es basiert. Weitere Informationen finden Sie im Leitfaden Sichere Verwendung von SSH-Schlüsseln in Docker für den Zugriff auf private Github-Repositorys.

Ihr Image erstellen und hochladen Docker

Mit einem richtig definierten sind Sie nun bereit`Dockerfile`, die Schritte zum Erstellen eines privaten Amazon ECR-Repositorys zu befolgen, falls noch keines vorhanden ist. Sie können Ihr Container-Image auch erstellen, taggen und in das Repository hochladen.

Sie sind bereit, das Image zu erstellen, zu taggen und zu pushen. Eine vollständige Erklärung der Optionen `docker build` und einige Beispiele finden Sie in der Docker-Build-Dokumentation.

Für die oben definierte Beispieldatei könnten Sie Folgendes ausführen:

```
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
```

Zuweisen geeigneter Amazon ECR-Berechtigungen

Braket Hybrid Jobs DockerBilder müssen in privaten Amazon ECR-Repositorys gehostet werden. Standardmäßig gewährt ein privates Amazon ECR-Repo keinen Lesezugriff für die Braket Hybrid Jobs IAM role oder andere Benutzer, die Ihr Bild verwenden möchten, wie z. B. Mitarbeiter oder Schüler. Sie müssen eine Repository-Richtlinie festlegen, um die entsprechenden Berechtigungen zu gewähren. Im Allgemeinen sollten Sie nur den spezifischen Benutzern und IAM Rollen, die Sie für den Zugriff auf Ihre Bilder benötigen, die Erlaubnis erteilen, anstatt jedem, der über die Rechte verfügt, image URI zu erlauben, sie abzurufen.

Braket-Hybrid-Jobs in Ihrem eigenen Container ausführen

Um einen Hybrid-Job mit Ihrem eigenen Container zu erstellen, rufen Sie `AwsQuantumJob.create()` mit dem `image_uri` angegebenen Argument auf. Sie können eine QPU, einen On-Demand-Simulator, verwenden oder Ihren Code lokal auf dem klassischen Prozessor ausführen, der mit Braket Hybrid Jobs verfügbar ist. Wir empfehlen, Ihren Code auf einem Simulator wie SV1, DM1 oder TN1 zu testen, bevor Sie ihn auf einer echten QPU ausführen.

Um Ihren Code auf dem klassischen Prozessor auszuführen, geben Sie den `instanceType` und den an, den Sie verwenden, indem `instanceCount` Sie den aktualisieren. `InstanceConfig` Beachten Sie, dass Sie bei Angabe von `instance_count` > 1 sicherstellen müssen, dass Ihr Code auf mehreren Hosts ausgeführt werden kann. Die Obergrenze für die Anzahl der Instanzen, die Sie wählen können, ist 5. Beispiel:

```
job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    image_uri="111122223333.dkr.ecr.us-west-2.amazonaws.com/my-byoc-container:latest",
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

> **Note**
>
> Verwenden Sie den Geräte-ARN, um den Simulator zu verfolgen, den Sie als Metadaten für Hybrid-Jobs verwendet haben. Zulässige Werte müssen dem Format entsprechen`device = "local:<provider>/<simulator_name>"`. Denken Sie daran `<provider>` und `<simulator_name>` dürfen nur aus Buchstaben, Zahlen, `_-`, und bestehen`.`. Die Zeichenfolge ist auf 256 Zeichen begrenzt.
> Wenn Sie planen, BYOC zu verwenden, aber das Braket-SDK nicht zum Erstellen von Quantenaufgaben verwenden, sollten Sie den Wert der Umgebungsvariablen

AMZN_BRAKET_JOB_TOKEN an den jobToken Parameter in der Anfrage übergeben. CreateQuantumTask Wenn Sie dies nicht tun, erhalten die Quantenaufgaben keine Priorität und werden als reguläre eigenständige Quantenaufgaben abgerechnet.

Hyperparameter verwenden

Sie können Hyperparameter definieren, die Ihr Algorithmus benötigt, z. B. die Lernrate oder die Schrittgröße, wenn Sie einen Hybrid-Job erstellen. Hyperparameterwerte werden in der Regel zur Steuerung verschiedener Aspekte des Algorithmus verwendet und können häufig angepasst werden, um die Leistung des Algorithmus zu optimieren. Um Hyperparameter in einem Braket-Hybrid-Job zu verwenden, müssen Sie ihre Namen und Werte explizit als Wörterbuch angeben. Geben Sie die Hyperparameterwerte an, die bei der Suche nach dem optimalen Wertesatz getestet werden sollen. Der erste Schritt zur Verwendung von Hyperparametern besteht darin, die Hyperparameter als Wörterbuch einzurichten und zu definieren. Dies wird im folgenden Code beschrieben.

```
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

hyperparameters = {"shots": 1_000}
```

Übergeben Sie dann die im oben angegebenen Codeausschnitt definierten Hyperparameter, damit sie in dem Algorithmus Ihrer Wahl verwendet werden sollen. Um das folgende Codebeispiel auszuführen, erstellen Sie ein Verzeichnis mit dem Namen „src“ im selben Pfad wie Ihre Hyperparameterdatei. Fügen Sie innerhalb des Verzeichnisses „src“ die Codedateien 0_ Getting_started_papermill .ipynb, notebook_runner.py und requirements.txt hinzu.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="src",
    entry_point="src.notebook_runner:run_notebook",
    input_data="src/0_Getting_started_papermill.ipynb",
    hyperparameters=hyperparameters,
    job_name=f"papermill-job-demo-{int(time.time())}",
)
```

```
# Print job to record the ARN
print(job)
```

Um von Ihrem Hybrid-Job-Skript aus auf Ihre Hyperparameter zuzugreifen, sehen Sie sich die `load_jobs_hyperparams()` Funktion in der Python-Datei notebook_runner.py an. Führen Sie den folgenden Code aus, um außerhalb Ihres Hybrid-Job-Skripts auf Ihre Hyperparameter zuzugreifen.

```
from braket.aws import AwsQuantumJob

# Get the job using the ARN
job_arn = "arn:aws:braket:us-east-1:111122223333:job/5eabb790-d3ff-47cc-98ed-
b4025e9e296f"  # Replace with your job ARN
job = AwsQuantumJob(arn=job_arn)

# Access the hyperparameters
job_metadata = job.metadata()
hyperparameters = job_metadata.get("hyperParameters", {})
print(hyperparameters)
```

Weitere Informationen zur Verwendung von Hyperparametern finden Sie in den Tutorials QAOA mit Amazon Braket Hybrid Jobs PennyLane und Quantum Machine Learning in Amazon Braket Hybrid Jobs.

Konfigurieren Sie Ihre Hybrid-Job-Instance

Abhängig von Ihrem Algorithmus haben Sie möglicherweise unterschiedliche Anforderungen. Standardmäßig führt Amazon Braket Ihr Algorithmus-Skript auf einer `ml.m5.large` Instance aus. Sie können diesen Instance-Typ jedoch anpassen, wenn Sie einen Hybrid-Job mit dem folgenden Import- und Konfigurationsargument erstellen.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"), # Use NVIDIA T4
 instance with 4 GPUs.
    ...
    ),
```

Wenn Sie eine eingebettete Simulation ausführen und in der Gerätekonfiguration ein lokales Gerät angegeben haben, können Sie zusätzlich mehr als eine Instanz in der anfordern, `InstanceConfig`

indem Sie das angeben `instanceCount` und es auf mehr als eins setzen. Die Obergrenze ist 5. Sie können beispielsweise 3 Instanzen wie folgt auswählen.

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3), #
 Use 3 NVIDIA T4 instances
    ...
    ),
```

Wenn Sie mehrere Instanzen verwenden, sollten Sie erwägen, Ihren Hybrid-Job mithilfe der Datenparallelfunktion zu verteilen. Im folgenden Beispiel-Notizbuch finden Sie weitere Informationen dazu, wie Sie sich dieses Beispiel für das Parallelize-Training für QML ansehen können.

In den folgenden drei Tabellen sind die verfügbaren Instance-Typen und Spezifikationen für Standard-, Hochleistungs- und GPU-beschleunigte Instances aufgeführt.

> **Note**
>
> Die standardmäßigen klassischen Compute-Instance-Kontingente für Hybrid-Jobs finden Sie auf der Seite Amazon Braket-Kontingente.

Standard-Instances	vCPU	Arbeitsspeicher (GiB)
ml.t3.large	2	8
ml.t3.xlarge	4	16
ml.t3.2xlarge	8	32
ml.m5.large (Standard)	4	16
ml.m5.xlarge	4	16
ml.m5.2xlarge	8	32
ml.m5.4xlarge	16	64

Standard-Instances	vCPU	Arbeitsspeicher (GiB)
ml.m5.12xlarge	48	192
ml.m5.24xlarge	96	384

Hochleistungs-Instances	vCPU	Arbeitsspeicher (GiB)
ml.c5.xlarge	4	8
ml.c5.2xlarge	8	16
ml.c5.4xlarge	16	32
ml.c5.9xlarge	36	72
ml.c5.18xlarge	72	144
ml.c5n.xlarge	4	10.5
ml.c5n.2xlarge	8	21
ml.c5n.4xlarge	16	32
ml.c5n.9xlarge	36	72
ml.c5n.18xlarge	72	192

GPU-beschleunigte Instanzen	GPUs	vCPU	Arbeitsspeicher (GiB)	GPU-Speicher (GiB)
ml.p4d.24xlarge	8	96	1 152	320
ml.g4dn.xlarge	1	4	16	16
ml.g4dn.2xlarge	1	8	32	16

GPU-beschleunigte Instanzen	GPUs	vCPU	Arbeitsspeicher (GiB)	GPU-Speicher (GiB)
ml.g4dn.4xlarge	1	16	64	16
ml.g4dn.8xlarge	1	32	128	16
ml.g4dn.12xlarge	4	48	192	64
ml.g4dn.16xlarge	1	64	256	16
ml.g6.xlarge	1	4	16	24
ml.g6.2xlarge	1	8	32	24
ml.g6.4xlarge	1	16	64	24
ml.g6.8xlarge	1	32	128	24
ml.g6.12xlarge	4	48	192	96
ml.g6.16xlarge	1	64	256	24
ml.g6.24xlarge	4	96	384	96
ml.g6.48xlarge	8	192	768	192
ml.g6e.xlarge	1	4	32	48
ml.g6e.2xlarge	1	8	64	48
ml.g6e.4xlarge	1	16	128	48
ml.g6e.8xlarge	1	32	256	48
ml.g6e.12xlarge	4	48	384	192
ml.g6e.16xlarge	1	64	512	48
ml.g6e.24xlarge	4	96	768	192

GPU-beschleunigte Instanzen	GPUs	vCPU	Arbeitsspeicher (GiB)	GPU-Speicher (GiB)
ml.g6e.48xlarge	8	192	1536	384

Jede Instanz verwendet eine Standardkonfiguration des Datenspeichers (SSD) von 30 GB. Sie können den Speicher jedoch auf die gleiche Weise anpassen, wie Sie den konfigurieren`instanceType`. Das folgende Beispiel zeigt, wie der Gesamtspeicher auf 50 GB erhöht werden kann.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(
        instanceType="ml.g4dn.xlarge",
        volumeSizeInGb=50,
    ),
    ...
    ),
```

Konfigurieren Sie den Standard-Bucket in **`AwsSession`**

Die Verwendung Ihrer eigenen `AwsSession` Instance bietet Ihnen mehr Flexibilität, z. B. die Möglichkeit, einen benutzerdefinierten Speicherort für Ihren standardmäßigen Amazon S3 S3-Bucket anzugeben. Standardmäßig `AwsSession` hat an einen vorkonfigurierten Amazon S3 S3-Bucket-Standort von`"amazon-braket-{id}-{region}"`. Sie haben jedoch die Möglichkeit, den standardmäßigen Amazon S3 S3-Bucket-Speicherort zu überschreiben, wenn Sie einen erstellen`AwsSession`. Benutzer können optional ein `AwsSession` Objekt an die `AwsQuantumJob.create()` Methode übergeben, indem sie den `aws_session` Parameter angeben, wie im folgenden Codebeispiel gezeigt.

```
aws_session = AwsSession(default_bucket="amazon-braket-s3-demo-bucket")

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

```
)
```

Verwendung der parametrischen Kompilierung zur Beschleunigung von Hybrid-Jobs

Amazon Braket unterstützt die parametrische Kompilierung auf bestimmten QPUs. Auf diese Weise können Sie den mit dem rechenintensiven Kompilierungsschritt verbundenen Aufwand reduzieren, indem Sie eine Schaltung nur einmal kompilieren und nicht für jede Iteration in Ihrem Hybrid-Algorithmus. Dies kann die Laufzeiten von Hybrid-Jobs erheblich verbessern, da Sie vermeiden, dass Sie Ihre Schaltung bei jedem Schritt neu kompilieren müssen. Senden Sie einfach parametrisierte Schaltungen als Braket Hybrid Job an eine unserer unterstützten QPUs. Bei Hybrid-Jobs mit langer Laufzeit verwendet Braket bei der Kompilierung Ihrer Schaltung automatisch die aktualisierten Kalibrierungsdaten des Hardwareanbieters, um Ergebnisse von höchster Qualität zu gewährleisten.

Um eine parametrische Schaltung zu erstellen, müssen Sie zunächst Parameter als Eingaben in Ihrem Algorithmus-Skript angeben. In diesem Beispiel verwenden wir einen kleinen parametrischen Schaltkreis und ignorieren jegliche klassische Verarbeitung zwischen den einzelnen Iterationen. Bei typischen Workloads würden Sie viele Schaltungen stapelweise einreichen und eine klassische Verarbeitung durchführen, wie z. B. die Aktualisierung der Parameter in jeder Iteration.

```
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.")
```

Sie können das Algorithmus-Skript zur Ausführung als Hybrid-Job mit dem folgenden Job-Skript einreichen. Wenn der Hybrid-Job auf einer QPU ausgeführt wird, die parametrische Kompilierung unterstützt, wird die Schaltung nur beim ersten Lauf kompiliert. In nachfolgenden Läufen wird der kompilierte Schaltkreis wiederverwendet, wodurch die Laufzeitleistung des Hybrid-Jobs ohne zusätzliche Codezeilen erhöht wird.

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="algorithm_script.py",
)
```

> **Note**
>
> Die parametrische Kompilierung wird auf allen supraleitenden, Gate-basierten QPUs von unterstützt, Rigetti Computing mit Ausnahme von Pulspegelprogrammen.

Verwendung PennyLane mit Amazon Braket

Hybride Algorithmen sind Algorithmen, die sowohl klassische als auch Quantenbefehle enthalten. Die klassischen Befehle werden auf klassischer Hardware (einer EC2-Instanz oder Ihrem Laptop) ausgeführt, und die Quantenbefehle werden entweder auf einem Simulator oder auf einem Quantencomputer ausgeführt. Wir empfehlen, hybride Algorithmen mithilfe der Hybrid-Jobs-Funktion auszuführen. Weitere Informationen finden Sie unter Wann sollten Sie Amazon Braket Jobs verwenden?

Amazon Braket ermöglicht es Ihnen, hybride Quantenalgorithmen mithilfe des Amazon PennyLane Braket-Plug-ins oder mit dem Amazon Braket Python SDK und Beispiel-Notebook-Repositorys einzurichten und auszuführen. Amazon Braket-Beispiel-Notebooks, die auf dem SDK basieren, ermöglichen es Ihnen, bestimmte Hybrid-Algorithmen ohne das PennyLane Plugin einzurichten und auszuführen. Wir empfehlen dies jedoch, PennyLane da es eine umfassendere Benutzererfahrung bietet.

Über hybride Quantenalgorithmen

Hybride Quantenalgorithmen sind heute für die Industrie wichtig, da moderne Quantencomputer im Allgemeinen Rauschen und damit Fehler erzeugen. Jedes Quantengatter, das einer Berechnung

hinzugefügt wird, erhöht die Wahrscheinlichkeit, dass Rauschen entsteht. Daher können Algorithmen mit langer Laufzeit durch Rauschen überfordert werden, was zu fehlerhaften Berechnungen führt.

Reine Quantenalgorithmen wie der von Shor (Beispiel Quantum Phase Estimation) oder der von Grover (Grovers Beispiel) erfordern Tausende oder Millionen von Operationen. Aus diesem Grund können sie für bestehende Quantengeräte, die allgemein als Noisy Intermediate-Scale Quantum (NISQ) bezeichnet werden, unpraktisch sein.

In hybriden Quantenalgorithmen arbeiten Quantenverarbeitungseinheiten (QPUs) als Co-Prozessoren für klassische CPUs, insbesondere um bestimmte Berechnungen in einem klassischen Algorithmus zu beschleunigen. Die Ausführung von Schaltungen wird immer kürzer, was mit den Möglichkeiten heutiger Geräte möglich ist.

In diesem Abschnitt:

- Amazon Braket mit PennyLane
- Hybride Algorithmen in Amazon Braket-Beispielnotizbüchern
- Hybride Algorithmen mit eingebetteten Simulatoren PennyLane
- Adjoint Gradient aktiviert PennyLane mit Amazon Braket-Simulatoren
- Hybrid-Jobs verwenden und einen QAOA-Algorithmus ausführen PennyLane
- Führen Sie hybride Workloads mit eingebetteten Simulatoren aus PennyLane

Amazon Braket mit PennyLane

Amazon Braket bietet Unterstützung für PennyLane, ein Open-Source-Software-Framework, das auf dem Konzept der quantendifferenzierbaren Programmierung basiert. Sie können dieses Framework verwenden, um Quantenschaltkreise auf die gleiche Weise zu trainieren, wie Sie ein neuronales Netzwerk trainieren würden, um Lösungen für Rechenprobleme in den Bereichen Quantenchemie, Quantenmaschinenlernen und Optimierung zu finden.

Die PennyLane Bibliothek bietet Schnittstellen zu vertrauten Tools für maschinelles Lernen, einschließlich PyTorch und TensorFlow, um das Training von Quantenschaltkreisen schnell und intuitiv zu gestalten.

- Die PennyLane Bibliothek —— PennyLane ist in Amazon Braket-Notebooks vorinstalliert. Um von aus auf Amazon Braket-Geräte zuzugreifen PennyLane, öffnen Sie ein Notizbuch und importieren Sie die PennyLane Bibliothek mit dem folgenden Befehl.

```
import pennylane as qml
```

Tutorial-Notizbücher helfen Ihnen dabei, schnell loszulegen. Alternativ können Sie PennyLane On Amazon Braket von einer IDE Ihrer Wahl aus verwenden.

- Das Amazon PennyLane Braket-Plugin — Um Ihre eigene IDE zu verwenden, können Sie das Amazon PennyLane Braket-Plugin manuell installieren. Das Plugin stellt eine Verbindung PennyLane mit dem Amazon Braket Python SDK her, sodass Sie Schaltungen PennyLane auf Amazon Braket-Geräten ausführen können. Verwenden Sie den folgenden Befehl, um PennyLane das Plugin zu installieren.

```
pip install amazon-braket-pennylane-plugin
```

Das folgende Beispiel zeigt, wie Sie den Zugriff auf Amazon Braket-Geräte einrichten in 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)
```

Tutorial-Beispiele und weitere Informationen PennyLane dazu finden Sie im Amazon Braket-Beispiel-Repository.

Das Amazon PennyLane Braket-Plugin ermöglicht es Ihnen, PennyLane mit einer einzigen Amazon Codezeile zwischen Braket QPU und eingebetteten Simulatorgeräten zu wechseln. Es bietet zwei Amazon Braket-Quantengeräte, mit denen Sie arbeiten können: PennyLane

- `braket.aws.qubit`für den Betrieb mit den Quantengeräten des Amazon Braket-Dienstes, einschließlich QPUs und Simulatoren
- `braket.local.qubit`für die Ausführung mit dem lokalen Simulator des Amazon Braket-SDK

Das Amazon PennyLane Braket-Plugin ist Open Source. Sie können es aus dem PennyLane GitHub Plugin-Repository installieren.

Weitere Informationen PennyLane dazu finden Sie in der Dokumentation auf der PennyLane Website.

Hybride Algorithmen in Amazon Braket-Beispielnotizbüchern

Amazon Braket bietet eine Vielzahl von Beispiel-Notebooks, die nicht auf das PennyLane Plugin angewiesen sind, um Hybrid-Algorithmen auszuführen. Sie können mit jedem dieser Amazon Braket-Hybrid-Beispielnotizbücher beginnen, in denen Variationsmethoden wie der Quantum Approximate Optimization Algorithm (QAOA) oder Variational Quantum Eigensolver (VQE) veranschaulicht werden.

Die Amazon Braket-Beispielnotizbücher basieren auf dem Amazon Braket Python SDK. Das SDK bietet ein Framework für die Interaktion mit Quantencomputer-Hardwaregeräten über Amazon Braket. Es handelt sich um eine Open-Source-Bibliothek, die Sie beim Quantenanteil Ihres hybriden Workflows unterstützen soll.

Mit unseren Beispiel-Notizbüchern können Sie Amazon Braket weiter erkunden.

Hybride Algorithmen mit eingebetteten Simulatoren PennyLane

Amazon Braket Hybrid Jobs wird jetzt mit leistungsstarken CPU- und GPU-based Embedded-Simulatoren von geliefert. PennyLane Diese Familie eingebetteter Simulatoren kann direkt in Ihren Hybrid-Job-Container eingebettet werden und umfasst den schnellen `lightning.qubit` State-Vector-Simulator, den mit der CuQuantum-Bibliothek von NVIDIA beschleunigten `lightning.gpu` Simulator und andere. Diese eingebetteten Simulatoren eignen sich ideal für variationelle Algorithmen wie maschinelles Quantenlernen, die von fortschrittlichen Methoden wie der Methode der adjungierten Differenzierung profitieren können. Sie können diese eingebetteten Simulatoren auf einer oder mehreren CPU- oder GPU-Instanzen ausführen.

Mit Hybrid-Jobs können Sie jetzt Ihren variationellen Algorithmuscode mit einer Kombination aus einem klassischen Co-Prozessor und einer QPU, einem Amazon Braket-On-Demand-Simulator wieSV1, oder direkt mit dem eingebetteten Simulator von ausführen. PennyLane

Der eingebettete Simulator ist bereits mit dem Hybrid Jobs-Container verfügbar. Sie müssen Ihre Python-Hauptfunktion mit dem `@hybrid_job` Decorator dekorieren. Um den PennyLane `lightning.gpu` Simulator zu verwenden, müssen Sie außerdem eine GPU-Instanz angeben, `InstanceConfig` wie im folgenden Codeausschnitt gezeigt:

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


@hybrid_job(device="local:pennylane/lightning.gpu",
 instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"))
def function(wires):
    dev = qml.device("lightning.gpu", wires=wires)
    ...
```

Sehen Sie sich das Beispiel-Notizbuch an, um mit der Verwendung eines PennyLane eingebetteten Simulators mit Hybrid-Jobs zu beginnen.

Adjoint Gradient aktiviert PennyLane mit Amazon Braket-Simulatoren

Mit dem PennyLane Plug-in für Amazon Braket können Sie Gradienten mithilfe der Methode der adjungierten Differenzierung berechnen, wenn Sie es auf dem Local State Vector Simulator oder SV1 ausführen.

Hinweis: Um die Methode der adjungierten Differenzierung verwenden zu können, müssen Sie **`diff_method='device'`** in Ihrer und nicht angeben. **`qnode`** `diff_method='adjoint'` Sehen Sie sich das folgende Beispiel an.

```
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**
>
> Derzeit PennyLane berechnet ich Gruppierungsindizes für QAOA-Hamiltonianer und verwendet sie, um den Hamiltonian in mehrere Erwartungswerte aufzuteilen. Wenn Sie bei der Ausführung von QAOA die Fähigkeit zur adjungierten Differenzierung von SV1 verwenden möchten, müssen Sie den Hamiltonschen Wert rekonstruierenPennyLane, indem Sie die Gruppierungsindizes wie folgt entfernen: `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False) cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Hybrid-Jobs verwenden und einen QAOA-Algorithmus ausführen PennyLane

In diesem Abschnitt werden Sie das Gelernte anwenden, um ein echtes Hybridprogramm PennyLane mit parametrischer Kompilierung zu schreiben. Sie verwenden das Algorithmus-Skript, um ein Problem mit dem Quantum Approximate Optimization Algorithm (QAOA) zu lösen. Das Programm erstellt eine Kostenfunktion, die einem klassischen Max-Cut-Optimierungsproblem entspricht, spezifiziert einen parametrisierten Quantenschaltkreis und verwendet eine Gradientenabstiegsmethode, um die Parameter so zu optimieren, dass die Kostenfunktion minimiert wird. In diesem Beispiel generieren wir der Einfachheit halber das Problemdiagramm im Algorithmus-Skript. Für typischere Anwendungsfälle empfiehlt es sich jedoch, die Problemspezifikation über einen speziellen Kanal in der Eingabedatenkonfiguration bereitzustellen. Das Flag `parametrize_differentiable` ist standardmäßig auf voreingestellt, `True` sodass Sie automatisch die Vorteile einer verbesserten Laufzeitleistung durch die parametrische Kompilierung auf unterstützten QPUs nutzen können.

```
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**
>
> Die parametrische Kompilierung wird auf allen supraleitenden, Gate-basierten QPUs unterstützt, mit Ausnahme von Rigetti Computing Programmen auf Pulsebene.

Führen Sie hybride Workloads mit eingebetteten Simulatoren aus PennyLane

Schauen wir uns an, wie Sie eingebettete Simulatoren von PennyLane Amazon Braket Hybrid Jobs aus verwenden können, um Hybrid-Workloads auszuführen. Der GPU-based eingebettete Simulator von Pennylane verwendet die Nvidia `lightning.gpu` CuQuantum-Bibliothek, um Schaltungssimulationen zu beschleunigen. Der eingebettete GPU-Simulator ist in allen Braket-Jobcontainern vorkonfiguriert, die Benutzer sofort verwenden können. Auf dieser Seite zeigen wir Ihnen, wie Sie `lightning.gpu` damit Ihre Hybrid-Workloads beschleunigen können.

Verwenden von **`lightning.gpu`** für QAOA-Workloads

Sehen Sie sich die Beispiele des Quantum Approximate Optimization Algorithm (QAOA) aus diesem Notizbuch an. Um einen eingebetteten Simulator auszuwählen, geben Sie als `device` Argument eine Zeichenfolge der folgenden Form an:. `"local:<provider>/<simulator_name>"` Zum Beispiel würden Sie `"local:pennylane/lightning.gpu"` für festlegen`lightning.gpu`. Die Gerätezeichenfolge, die Sie dem Hybrid-Job beim Start geben, wird als Umgebungsvariable an den Job übergeben`"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)
```

Vergleichen Sie auf dieser Seite die beiden eingebetteten PennyLane Zustandsvektorsimulatoren `lightning.qubit` (welcher ist CPU-based) und `lightning.gpu` (welcher ist GPU-based). Stellen Sie den Simulatoren benutzerdefinierte Gate-Zerlegungen zur Verfügung, um verschiedene Gradienten zu berechnen.

Jetzt sind Sie bereit, das Skript zum Starten des Hybrid-Jobs vorzubereiten. Führen Sie den QAOA-Algorithmus mit zwei Instanztypen aus: `ml.m5.2xlarge` und. `ml.g4dn.xlarge` Der `ml.m5.2xlarge` Instanztyp ist vergleichbar mit einem Standard-Entwickler-Laptop. Dabei `ml.g4dn.xlarge` handelt es sich um eine beschleunigte Recheninstanz mit einer einzelnen NVIDIA T4-GPU mit 16 GB Arbeitsspeicher.

Um die GPU auszuführen, müssen wir zunächst ein kompatibles Image und die richtige Instanz angeben (die standardmäßig eine `ml.m5.2xlarge` Instanz ist).

```
from braket.aws import AwsSession
from braket.jobs.image_uris import Framework, retrieve_image

image_uri = retrieve_image(Framework.PL_PYTORCH, AwsSession().region)
instance_config = InstanceConfig(instanceType="ml.g4dn.xlarge")
```

Diese müssen wir dann zusammen mit den aktualisierten Geräteparametern sowohl in den System- als auch in den Hybrid-Job-Argumenten in den Hybrid-Job-Decorator eingeben.

```
@hybrid_job(
        device="local:pennylane/lightning.gpu",
        input_data=input_file_path,
        image_uri=image_uri,
        instance_config=instance_config)
def run_qaoa_hybrid_job_gpu(p=1, steps=10):
    params = np.random.rand(2, p)

    braket_task_tracker = Tracker()

    graph = nx.read_adjlist(input_file_path, nodetype=int)
    wires = list(graph.nodes)
    cost_h, _mixer_h = qaoa.maxcut(graph)
```

```
    device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
    prefix, device_name = device_string.split("/")
    dev= qml.device(simulator_name, wires=len(wires))
    ...
```

> **Note**
>
> Wenn Sie das mithilfe einer GPU-based Instanz `instance_config` als angeben, aber `device` den als eingebetteten CPU-based Simulator (`lightning.qubit`) wählen, wird die GPU nicht verwendet. Stellen Sie sicher, dass Sie den eingebetteten GPU-based Simulator verwenden, wenn Sie die GPU als Ziel verwenden möchten!

Die durchschnittliche Iterationszeit für die `m5.2xlarge` Instanz beträgt etwa 73 Sekunden, während sie für die `ml.g4dn.xlarge` Instanz etwa 0,6 Sekunden beträgt. Für diesen 21-Qubit-Workflow bietet uns die GPU-Instanz eine 100-fache Beschleunigung. Wenn Sie sich die Preisseite für Amazon Braket Hybrid Jobs ansehen, können Sie sehen, dass die Kosten pro Minute für eine `m5.2xlarge` Instance 0,00768 USD betragen, während sie für die `ml.g4dn.xlarge` Instance 0,01227 USD betragen. In diesem Fall ist die Ausführung auf der GPU-Instanz schneller und günstiger.

Maschinelles Quantenlernen und Datenparallelität

Wenn es sich bei Ihrem Workload-Typ um quantenmechanisches Lernen (QML) handelt, das anhand von Datensätzen trainiert, können Sie Ihren Workload mithilfe von Datenparallelität weiter beschleunigen. In QML enthält das Modell einen oder mehrere Quantenschaltkreise. Das Modell kann auch klassische neuronale Netze enthalten oder auch nicht. Beim Training des Modells mit dem Datensatz werden die Parameter im Modell aktualisiert, um die Verlustfunktion zu minimieren. Eine Verlustfunktion wird normalerweise für einen einzelnen Datenpunkt und der Gesamtverlust für den durchschnittlichen Verlust über den gesamten Datensatz definiert. In QML werden die Verluste normalerweise seriell berechnet, bevor bei Gradientenberechnungen der Durchschnitt zum Gesamtverlust berechnet wird. Dieses Verfahren ist zeitaufwändig, insbesondere wenn es Hunderte von Datenpunkten gibt.

Da der Verlust von einem Datenpunkt nicht von anderen Datenpunkten abhängt, können die Verluste parallel ausgewertet werden! Verluste und Gradienten, die mit verschiedenen Datenpunkten verbunden sind, können gleichzeitig ausgewertet werden. Dies wird als Datenparallelität bezeichnet. Mit SageMaker der verteilten Datenparallelbibliothek erleichtert Ihnen Amazon Braket Hybrid Jobs die Nutzung von Datenparallelität, um Ihr Training zu beschleunigen.

Stellen Sie sich den folgenden QML-Workload für Datenparallelität vor, der den Sonar-Datensatz aus dem bekannten UCI-Repository als Beispiel für die binäre Klassifizierung verwendet. Der Sonar-Datensatz enthält 208 Datenpunkte mit jeweils 60 Merkmalen, die anhand von Sonarsignalen erfasst wurden, die von Materialien abprallen. Jeder Datenpunkt ist entweder mit „M“ für Minen oder mit „R“ für Gesteine gekennzeichnet. Unser QML-Modell besteht aus einer Eingangsschicht, einem Quantenschaltkreis als versteckte Schicht und einer Ausgangsschicht. Die Eingabe- und Ausgabeschichten sind klassische neuronale Netze, die in PyTorch implementiert sind. Der Quantenschaltkreis ist mithilfe PennyLane des qml.qnn-Moduls in die PyTorch neuronalen Netze integriert. Weitere Informationen zur Arbeitslast finden Sie in unseren Beispiel-Notizbüchern. Wie im obigen QAOA-Beispiel können Sie die Leistung der GPU nutzen, indem Sie eingebettete GPU-based Simulatoren verwenden, um die Leistung gegenüber eingebetteten Simulatoren `lightning.gpu` zu verbessern. PennyLane CPU-based

Um einen Hybrid-Job zu erstellen, können Sie das Algorithmus-Skript, das Gerät `AwsQuantumJob.create` und andere Konfigurationen über seine Schlüsselwortargumente aufrufen und angeben.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.xlarge')

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

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

Um Datenparallelität zu verwenden, müssen Sie einige Codezeilen im Algorithmus-Skript für die SageMaker verteilte Bibliothek ändern, um das Training korrekt zu parallelisieren. Zunächst importieren Sie das `smdistributed` Paket, das den Großteil der Arbeit für die Verteilung Ihrer Workloads auf mehrere GPUs und mehrere Instanzen übernimmt. Dieses Paket ist im Braket und in den Containern vorkonfiguriert. PyTorch TensorFlow Das `dist` Modul teilt unserem Algorithmus-Skript mit, wie hoch die Gesamtzahl der GPUs für das Training (`world_size`) ist `rank` und wie hoch

der Wert eines `local_rank` GPU-Kerns ist. `rank`ist der absolute Index einer GPU für alle Instanzen, während `local_rank` es der Index einer GPU innerhalb einer Instanz ist. Wenn es beispielsweise vier Instanzen gibt, denen jeweils acht GPUs für das Training zugewiesen sind, `rank` liegen die Werte zwischen 0 und 31 und die `local_rank` Bereiche zwischen 0 und 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)
```

Als Nächstes definieren Sie a `DistributedSampler` entsprechend dem `world_size` und `rank` und übergeben es dann an den Datenlader. Dieser Sampler verhindert, dass GPUs auf denselben Ausschnitt eines Datensatzes zugreifen.

```
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,
)
```

Als Nächstes verwenden Sie die `DistributedDataParallel` Klasse, um Datenparallelität zu aktivieren.

```
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"])
```

Die oben genannten Änderungen sind erforderlich, um Datenparallelität zu verwenden. In QML möchten Sie häufig Ergebnisse speichern und den Trainingsfortschritt ausdrucken. Wenn jede GPU den Befehl zum Speichern und Drucken ausführt, wird das Protokoll mit den wiederholten Informationen überflutet und die Ergebnisse überschreiben sich gegenseitig. Um dies zu vermeiden, können Sie nur von der GPU aus speichern und drucken, die 0 hat`rank`.

```
if dp_info["rank"]==0:
    print('elapsed time: ', elapsed)
    torch.save(model.state_dict(), f"{output_dir}/test_local.pt")
    save_job_result({"last loss": loss_before})
```

Amazon Braket Hybrid Jobs unterstützt `ml.g4dn.12xlarge` Instance-Typen für die SageMaker Distributed Data Parallel Library. Sie konfigurieren den Instance-Typ über das `InstanceConfig` Argument in Hybrid Jobs. Damit die SageMaker Distributed Data Parallel Library weiß, dass Datenparallelität aktiviert ist, müssen Sie zwei zusätzliche Hyperparameter hinzufügen: `"sagemaker_distributed_dataparallel_enabled"` Einstellung auf `"true"` und `"sagemaker_instance_type"` Einstellung auf den Instanztyp, den Sie verwenden. Diese beiden Hyperparameter werden pro Paket verwendet. `smdistributed` Ihr Algorithmus-Skript muss sie nicht explizit verwenden. Im Amazon Braket SDK bietet es ein praktisches Schlüsselwortargument`distribution`. `distribution="data_parallel"`Bei der Erstellung hybrider Jobs fügt das Amazon Braket SDK die beiden Hyperparameter automatisch für Sie ein. Wenn Sie die Amazon Braket-API verwenden, müssen Sie diese beiden Hyperparameter einbeziehen.

Wenn die Instanz und die Datenparallelität konfiguriert sind, können Sie jetzt Ihren Hybrid-Job einreichen. Es gibt 4 GPUs in einer Instanz. `ml.g4dn.12xlarge` Wenn Sie festlegen`instanceCount=1`, wird die Arbeitslast auf die 8 GPUs in der Instanz verteilt. Wenn Sie `instanceCount` mehr als eine festlegen, wird die Arbeitslast auf die in allen Instanzen verfügbaren GPUs verteilt. Wenn Sie mehrere Instanzen verwenden, fällt für jede Instanz eine Gebühr an, die davon abhängt, wie lange Sie sie verwenden. Wenn Sie beispielsweise vier Instances verwenden, beträgt die abrechnungsfähige Zeit das Vierfache der Laufzeit pro Instance, da Ihre Workloads von vier Instances gleichzeitig ausgeführt werden.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.12xlarge',
                                 instanceCount=1,
```

```
)

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

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

> **Note**
>
> In der obigen Hybrid-Job-Erstellung `train_dp.py` ist das modifizierte Algorithmus-Skript für die Nutzung von Datenparallelität enthalten. Beachten Sie, dass Datenparallelität nur dann korrekt funktioniert, wenn Sie Ihr Algorithmus-Skript gemäß dem obigen Abschnitt ändern. Wenn die Option Datenparallelität ohne ein korrekt modifiziertes Algorithmus-Skript aktiviert ist, kann der Hybrid-Job Fehler auslösen, oder jede GPU kann wiederholt denselben Datenabschnitt verarbeiten, was ineffizient ist.

Bei richtiger Anwendung kann die Verwendung mehrerer Instanzen zu einer erheblichen Zeit- und Kostenreduzierung führen. Weitere Informationen finden Sie im Beispiel-Notizbuch.

Verwendung CUDA-Q mit Amazon Braket

NVIDIA's CUDA-Qist eine Softwarebibliothek, die für die Programmierung hybrider Quantenalgorithmen entwickelt wurde, die CPUs, GPUs und Quantenverarbeitungseinheiten (QPUs) kombinieren. Sie bietet ein einheitliches Programmiermodell, das es Entwicklern ermöglicht, sowohl klassische als auch Quantenbefehle in einem einzigen Programm auszudrücken und so Arbeitsabläufe zu rationalisieren. CUDA-Qbeschleunigt die Simulation und Laufzeit von Quantenprogrammen mit seinen integrierten CPU- und GPU-Simulatoren. CUDA-Qist mit nativen Braket-Notebook-Instances (NBIs) und Amazon Braket Hybrid Jobs verfügbar.

In diesem Abschnitt:

- CUDA-Q in NBIs
- CUDA-Q in hybriden Aufträgen

CUDA-Q in NBIs

CUDA-Qist standardmäßig in der Braket-NBI-Umgebung installiert. Sie können ein CUDA-Q Beispiel-Notizbuch öffnen, indem Sie auf der Jupyter Launcher-Seite die Kachel und Braket auswählen. CUDA-Q Dadurch wird das Beispiel-Notizbuch `0_Getting_started_with_CUDA-Q.ipynb` im Hauptfenster geöffnet. Weitere CUDA-Q Beispiele finden Sie im linken Bereich des `nvidia_cuda_q/` Verzeichnisses.

Sie können auch die Version von CUDA-Q oder einem anderen Paket eines Drittanbieters überprüfen, das in Ihrem NBI installiert ist. Sie können beispielsweise den folgenden Befehl in einer Notebook-Codezelle ausführenCUDA-Q, um die Versionen der Pakete Qiskit und Braket zu überprüfen PennyLane, die in der Umgebung installiert sind.

```
%pip freeze | grep -i -e cudaq -e qiskit -e pennylane -e braket
```

CUDA-Q in hybriden Aufträgen

Die Verwendung CUDA-Q auf Amazon Braket Hybrid Jobs bietet eine flexible On-Demand-Computerumgebung. Recheninstanzen werden nur für die Dauer Ihrer Arbeitslast ausgeführt, sodass sichergestellt ist, dass Sie nur für das bezahlen, was Sie tatsächlich nutzen. Amazon Braket Hybrid Jobs bietet auch ein skalierbares Erlebnis. Benutzer können mit kleineren Instances für das Prototyping und Testen beginnen und dann auf größere Instances skalieren, die größere Workloads für vollständige Experimente bewältigen können.

Amazon Braket Hybrid Jobs unterstützt GPUs, die für die Maximierung des CUDA-Q Potenzials unerlässlich sind. GPUs beschleunigen die Simulationen von Quantenprogrammen im Vergleich zu CPU-based Simulatoren erheblich, insbesondere bei der Arbeit mit Schaltungen mit hoher Qubitzahl. Die Parallelisierung wird bei der Verwendung CUDA-Q auf Amazon Braket Hybrid Jobs unkompliziert. Hybrid Jobs vereinfacht die Verteilung von Schaltkreisabtastungen und beobachtbaren Auswertungen auf mehrere Rechenknoten. Diese nahtlose Parallelisierung von CUDA-Q Workloads ermöglicht es Benutzern, sich mehr auf die Entwicklung ihrer Workloads zu konzentrieren, anstatt die Infrastruktur für groß angelegte Experimente einzurichten.

Sehen Sie sich zunächst das CUDA-QStarter-Beispiel auf dem Github für Amazon Braket-Beispiele an, um einen von Braket bereitgestellten CUDA-Q Hybrid-Job-Container zu verwenden.

Der folgende Codeausschnitt ist ein `hello-world` Beispiel für die Ausführung eines CUDA-Q Programms mit Amazon Braket Hybrid Jobs.

```
image_uri = retrieve_image(Framework.CUDAQ, AwsSession().region)

@hybrid_job(device='local:nvidia/qpp-cpu', image_uri=image_uri)
def hello_quantum():
    import cudaq

    # define the backend
    device=get_job_device_arn()
    cudaq.set_target(device.split('/')[-1])

    # define the Bell circuit
    kernel = cudaq.make_kernel()
    qubits = kernel.qalloc(2)
    kernel.h(qubits[0])
    kernel.cx(qubits[0], qubits[1])

    # sample the Bell circuit
    result = cudaq.sample(kernel, shots_count=1000)
    measurement_probabilities = dict(result.items())

    return measurement_probabilities
```

Das obige Beispiel simuliert eine Bell-Schaltung auf einem CPU-Simulator. Dieses Beispiel wird lokal auf Ihrem Laptop oder Braket Jupyter-Notebook ausgeführt. Aufgrund der `local=True` Einstellung wird bei der Ausführung dieses Skripts ein Container in Ihrer lokalen Umgebung gestartet, um das CUDA-Q Programm zum Testen und Debuggen auszuführen. Nachdem Sie den Test abgeschlossen haben, können Sie die `local=True` Markierung entfernen und Ihren Job AWS weiterführen. Weitere Informationen finden Sie unter Arbeiten mit Amazon Braket Hybrid Jobs.

Wenn Ihre Workloads eine hohe Qubit-Anzahl, eine große Anzahl von Schaltungen oder eine große Anzahl von Iterationen aufweisen, können Sie leistungsstärkere CPU-Rechenressourcen verwenden, indem Sie die Einstellung angeben. `instance_config` Der folgende Codeausschnitt zeigt, wie Sie die Einstellung im Decorator konfigurieren. `instance_config` `hybrid_job` Weitere Informationen zu den unterstützten Instanztypen finden Sie unter Konfiguration Ihrer Hybrid-Job-Instanz. Eine Liste der Instance-Typen finden Sie unter Amazon EC2 EC2-Instance-Typen.

```
@hybrid_job(
    device="local:nvidia/qpp-cpu",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.c5.2xlarge"),
)
def my_job_script():
    ...
```

Für anspruchsvollere Workloads können Sie Ihre Workloads auf einem CUDA-Q GPU-Simulator ausführen. Verwenden Sie den Backend-Namen, um einen GPU-Simulator zu aktivieren. `nvidia` Das `nvidia` Backend arbeitet als CUDA-Q GPU-Simulator. Wählen Sie als Nächstes einen Amazon EC2 EC2-Instance-Typ aus, der eine NVIDIA GPU unterstützt. Der folgende Codeausschnitt zeigt den Decorator. GPU-configured `hybrid_job`

```
@hybrid_job(
    device="local:nvidia/nvidia",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"),
)
def my_job_script():
    ...
```

Amazon Braket Hybrid Jobs und NBIs unterstützen parallel GPU-Simulationen mit. CUDA-Q Sie können die Auswertung mehrerer Observables oder mehrerer Schaltungen parallelisieren, um die Leistung Ihres Workloads zu steigern. Um mehrere Observables zu parallelisieren, nehmen Sie die folgenden Änderungen an Ihrem Algorithmus-Skript vor.

Stellen Sie die `mgpu` Option des Backends ein. `nvidia` Dies ist erforderlich, um die Observablen zu parallelisieren. Die Parallelisierung verwendet MPI für die Kommunikation zwischen GPUs, daher muss MPI vor der Ausführung initialisiert und danach finalisiert werden.

Geben Sie als Nächstes den Ausführungsmodus durch Einstellung an. `execution=cudaq.parallel.mpi` Der folgende Codeausschnitt zeigt diese Änderungen.

```
cudaq.set_target("nvidia", option="mqpu")
cudaq.mpi.initialize()
result = cudaq.observe(
    kernel, hamiltonian, shots_count=n_shots, execution=cudaq.parallel.mpi
)
cudaq.mpi.finalize()
```

Geben Sie im `hybrid_job` Decorator einen Instanztyp an, der mehrere GPUs hostet, wie im folgenden Codeausschnitt gezeigt.

```
@hybrid_job(
    device="local:nvidia/nvidia-mqpu",
    instance_config=InstanceConfig(instanceType="ml.g4dn.12xlarge", instanceCount=1),
    image_uri=image_uri,
)
def parallel_observables_gpu_job(sagemaker_mpi_enabled=True):
    ...
```

Das Notizbuch für parallel Simulationen im Github Amazon Braket-Beispiele bietet umfassende Beispiele, die zeigen, wie Quantenprogrammsimulationen auf GPU-Backends ausgeführt und parallel Simulationen von Observablen und Schaltkreisstapeln durchgeführt werden können.

Ihre Workloads auf Quantencomputern ausführen

Nach Abschluss der Simulatortests können Sie zur Ausführung von Experimenten auf QPUs übergehen. Stellen Sie das Ziel einfach auf eine Amazon Braket-QPU um, z. B. auf die IQM GeräteIonQ, oderRigetti. Der folgende Codeausschnitt veranschaulicht, wie Sie das Ziel für das Gerät festlegen. IQM Garnet Eine Liste der verfügbaren QPUs finden Sie in der Amazon Braket-Konsole.

```
device_arn = "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet"
cudaq.set_target("braket", machine=device_arn)
```

Weitere Informationen zu Hybrid-Jobs finden Sie unter Arbeiten mit Amazon Braket Hybrid Jobs im Entwicklerhandbuch. Weitere Informationen über CUDA-Q finden Sie in der NVIDIA CUDA-QDokumentation.

Problembehebung bei Amazon Braket

Verwenden Sie die Informationen und Lösungen zur Fehlerbehebung in diesem Abschnitt, um Probleme mit Amazon Braket zu lösen.

In diesem Abschnitt:

- AccessDeniedException
- Beim Aufrufen des Vorgangs ist ein Fehler aufgetreten (ValidationException) CreateQuantumTask
- Eine SDK-Funktion funktioniert nicht
- Der Hybrid-Job schlägt fehl aufgrund von ServiceQuotaExceededException
- Komponenten funktionieren in der Notebook-Instanz nicht mehr
- Problembehandlung beim Python 3.12-Upgrade
- Fehlerbehebung bei OpenQASM

AccessDeniedException

Wenn Sie AccessDeniedExceptionbei der Aktivierung oder Verwendung von Braket eine erhalten, versuchen Sie wahrscheinlich, Braket in einer Region zu aktivieren oder zu verwenden, auf die Ihre eingeschränkte Rolle keinen Zugriff hat.

Wenden Sie sich in solchen Fällen an Ihren internen AWS Administrator, um zu erfahren, welche der folgenden Bedingungen zutreffen:

- Wenn es Rolleneinschränkungen gibt, die den Zugriff auf eine Region verhindern.
- Wenn die Rolle, die Sie verwenden möchten, Braket verwenden darf.

Wenn Ihre Rolle bei der Verwendung von Braket keinen Zugriff auf eine bestimmte Region hat, können Sie keine Geräte in dieser bestimmten Region verwenden.

Beim Aufrufen des Vorgangs ist ein Fehler aufgetreten (ValidationException) CreateQuantumTask

Wenn Sie eine Fehlermeldung erhalten, die der folgenden ähnelt: `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller`

`doesn't have access to amazon-braket-…` Vergewissern Sie sich, dass Sie auf einen vorhandenen s3_folder verweisen. Braket erstellt nicht auto neue Amazon S3 S3-Buckets und -Präfixe für Sie.

Wenn Sie API direkt auf den zugreifen und eine Fehlermeldung ähnlich der folgenden erhalten: `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` Stellen Sie sicher, dass Sie den Bucket-Pfad nicht `s3://` in den Amazon S3 S3-Bucket-Pfad aufnehmen.

Eine SDK-Funktion funktioniert nicht

Ihre Python-Version muss 3.10 oder höher sein. Für Amazon Braket Hybrid Jobs empfehlen wir Python 3.12.

Stellen Sie sicher, dass Ihr SDK und Ihre Schemas es sind. up-to-date Führen Sie den folgenden Befehl aus, um das SDK über das Notebook oder Ihren Python-Editor zu aktualisieren:

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Führen Sie den folgenden Befehl aus, um die Schemas zu aktualisieren:

```
pip install amazon-braket-schemas --upgrade
```

Wenn Sie von Ihrem eigenen Kunden aus auf Amazon Braket zugreifen, stellen Sie sicher, dass Ihre AWS Region auf eine von Amazon Braket unterstützte Region eingestellt ist.

Der Hybrid-Job schlägt fehl aufgrund von ServiceQuotaExceededException

Ein Hybrid-Job, der Quantenaufgaben für die Amazon Braket-Simulatoren ausführt, kann nicht erstellt werden, wenn Sie das Limit für gleichzeitige Quantenaufgaben für das Simulatorgerät, auf das Sie abzielen, überschreiten. Weitere Informationen zu den Service-Limits finden Sie im Thema Kontingente.

Wenn Sie gleichzeitig Aufgaben auf einem Simulatorgerät in mehreren Hybridaufträgen von Ihrem Konto aus ausführen, kann dieser Fehler auftreten.

Um die Anzahl der gleichzeitigen Quantenaufgaben für ein bestimmtes Simulatorgerät zu sehen, verwenden Sie den `search-quantum-tasks`API, wie im folgenden Codebeispiel gezeigt.

```
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
```

Sie können die erstellten Quantenaufgaben auch anhand von CloudWatch Amazon-Metriken für ein Gerät anzeigen: Braket > Nach Gerät.

Um zu vermeiden, dass diese Fehler auftreten:

1. Beantragen Sie eine Erhöhung der Servicequote für die Anzahl gleichzeitiger Quantenaufgaben für das Simulatorgerät. Dies gilt nur für das SV1 Gerät.
2. Behandeln Sie `ServiceQuotaExceeded` Ausnahmen in Ihrem Code und versuchen Sie es erneut.

Komponenten funktionieren in der Notebook-Instanz nicht mehr

Wenn einige Komponenten Ihres Notebooks nicht mehr funktionieren, versuchen Sie Folgendes:

1. Laden Sie alle Notizbücher, die Sie erstellt oder geändert haben, auf ein lokales Laufwerk herunter.
2. Stoppen Sie Ihre Notebook-Instanz.
3. Löschen Sie Ihre Notebook-Instanz.
4. Erstellen Sie eine neue Notebook-Instanz mit einem anderen Namen.
5. Laden Sie die Notizbücher auf die neue Instanz hoch.

Problembehandlung beim Python 3.12-Upgrade

Datum des Inkrafttretens: 21. Januar 2026

-Übersicht

Mit Wirkung zum 21. Januar 2026 aktualisiert Amazon Braket die Python-Laufzeit von 3.10 auf 3.12 für alle Notebook-Instances und verwalteten Container-Images (Base, CUDA-Q und). TensorFlow PyTorch Dieses Handbuch bietet Lösungen für häufig auftretende Kompatibilitätsprobleme.

In diesem Abschnitt:

- Allgemeine Fehlermeldungen
- Von Braket verwaltete Notizbücher
- Hybrider Jobdekorateur
- Bring-Your-Own-Container (BYOC)
- Aktualisierung der Braket-Notebook-Instanz

Allgemeine Fehlermeldungen

Fehler: SDK-Python-Version stimmt nicht überein

Fehler:

```
RuntimeError: Python version must match between local environment and container. Client
 is running Python 3.10 locally, but container uses Python 3.12.
```

Ursache: Das Braket-SDK hat erkannt, dass auf Ihrem Notebook Python 3.10 ausgeführt wird, der Hybrid-Job-Container jedoch Python 3.12 ausführt.

Lösung: Führen Sie entweder ein Upgrade Ihres Notebooks auf Python 3.12 durch oder pinnen Sie es auf Python 3.10-Container.

Cloudpickle-Serialisierungsfehler

Fehler:

```
TypeError: code() argument 13 must be str, not int
```

Ursache: Wenn die SDK-Validierung umgangen wird, kann Cloudpickle aufgrund einer Konstruktoränderung in Python 3.12 den Code zwischen Python 3.10 und 3.12 nicht serialisieren. CodeType

Lösung: Stellen Sie sicher, dass Ihr Notebook und Ihr Container dieselbe Python-Version verwenden.

Von Braket verwaltete Notizbücher

Wenn Sie eine Braket-Notebook-Instanz auf Python 3.10 ausführen und Hybrid-Jobs einreichen, werden Sie auf Versionskonflikte stoßen, da die Job-Container jetzt standardmäßig Python 3.12 verwenden.

Sie haben zwei Optionen:

1. [Empfohlen] Erstellen Sie eine neue Notebook-Instanz mit Python 3.12 — siehe Braket Notebook Instance Upgrade
2. An Python 3.10-Container anheften — siehe Hybrid Job Decorator

Hybrider Jobdekorateur

Um den `@hybrid_job` Decorator verwenden zu können, muss die Python-Version Ihrer Umgebung mit der Python-Version des Containers übereinstimmen.

Option 1: Python 3.12-Container verwenden (empfohlen)

Wenn Sie Ihre Umgebung auf Python 3.12 aktualisiert haben, verwendet sie das neueste Tag (Standardverhalten).

Option 2: Verwenden Sie Python 3.10-Container

Wenn Sie bei Python 3.10 bleiben müssen, geben Sie den `image_uri` Parameter explizit im `@hybrid_job` Decorator an.

Python 3.10-Container-Tags:

Name des Images	Markierung
Base	1.0-cpu-py310-ubuntu22.04
CUDA-Q	0,12,0-cpu-py310-0,12.0
PyTorch	2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow	2.14.1-gpu-py310-cu118-ubuntu20.04

Das folgende Beispiel bezieht sich auf die Region us-west-2.

Vollständiges Bild: URIs

```
Base:       292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-
cpu-py310-ubuntu22.04
CUDA-Q:     292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-
jobs:0.12.0-cpu-py310-0.12.0
PyTorch:    292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-
jobs:2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow: 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-
jobs:2.14.1-gpu-py310-cu118-ubuntu20.04
```

Beispiel:

```
from braket.jobs.hybrid_job import hybrid_job
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

@hybrid_job(
    device=device_arn,
    image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-
jobs:1.0-cpu-py310-ubuntu22.04"
)
def my_job():
    pass
```

> **Note**
>
> - Python 3.10-Container bleiben verfügbar, erhalten aber keine Updates.
> - Siehe Definieren Sie die Umgebung für Ihr Algorithmus-Skript.

Bring-Your-Own-Container (BYOC)

Wenn Ihr Dockerfile ein von Braket verwaltetes Image mit dem neuesten Tag verwendet, werden bei einem Neustart nach dem 21. Januar 2026 von Python 3.12 unterstützte Images abgerufen.

Um auf den von Python 3.10 unterstützten Braket-verwalteten Images zu bleiben, aktualisieren Sie Ihr Dockerfile:

Vorher (erhält Python 3.12 nach dem Upgrade):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:latest
```

Danach (bleibt auf Python 3.10):

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-
py310-ubuntu22.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-
py310-0.12.0
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:2.2.0-gpu-
py310-cu121-ubuntu20.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:2.14.1-
gpu-py310-cu118-ubuntu20.04
```

Aktualisierung der Braket-Notebook-Instanz

Gehen Sie wie folgt vor, um auf Python 3.12 zu aktualisieren:

> **Important**
>
> Stellen Sie vor dem Löschen Ihrer Notebook-Instanz sicher, dass Sie alle Notizbücher und Dateien heruntergeladen haben, die Sie behalten möchten. Diese Daten können nach dem Löschen nicht wiederhergestellt werden.

1. Laden Sie alle Notizbücher, die Sie erstellt oder geändert haben, auf ein lokales Laufwerk herunter.
2. Stoppen Sie Ihre Notebook-Instanz.
3. Löschen Sie Ihre Notebook-Instanz.
4. Erstellen Sie eine neue Notebook-Instanz mit einem anderen Namen.
5. Laden Sie Ihre Notizbücher auf die neue Instanz hoch.

Fehlerbehebung bei OpenQASM

Dieser Abschnitt enthält Hinweise zur Fehlerbehebung, die nützlich sein können, wenn Fehler bei der Verwendung von OpenQASM 3.0 auftreten.

In diesem Abschnitt:

- Fehler in der Anweisung einschließen
- Nicht zusammenhängender Fehler qubits
- Mischen von qubits physischem und virtuellem qubits Fehler
- Fehler beim Abrufen von Ergebnistypen und Messen qubits im selben Programm
- Die klassischen Grenzwerte und die qubit Registergrenzen haben den Fehler überschritten
- Feld, dem kein wörtlicher Pragma-Fehler vorausgeht
- Fehler bei verbatim-Boxen, bei denen native Gates fehlen
- Bei den verbatim-Boxen fehlt ein physischer Fehler qubits
- Im wörtlichen Pragma fehlt der Fehler „Braket“
- Fehler „Single qubits kann nicht indexiert werden“
- Fehler: Die physikalischen qubits Verbindungen in zwei qubit Gates sind nicht verbunden
- Warnung zur Unterstützung des lokalen Simulators

Fehler in der Anweisung einschließen

Braket hat derzeit keine Standard-Gate-Bibliotheksdatei, die in OpenQASM-Programme aufgenommen werden könnte. Das folgende Beispiel löst beispielsweise einen Parser-Fehler aus.

```
OPENQASM 3;
include "standardlib.inc";
```

Dieser Code generiert die Fehlermeldung: `No terminal matches '"' in the current parser context, at line 2 col 17.`

Nicht zusammenhängender Fehler qubits

Die Verwendung von nicht zusammenhängend qubits auf Geräten, die `true` in der Gerätefunktion auf `requiresContiguousQubitIndices` eingestellt sind, führt zu einem Fehler.

Beim Ausführen von Quantenaufgaben auf Simulatoren und IonQ löst das folgende Programm den Fehler aus.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

Dieser Code generiert die Fehlermeldung: `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

Mischen von qubits physischem und virtuellem qubits Fehler

Das Mischen von physisch qubits und virtuell qubits in demselben Programm ist nicht zulässig und führt zu einem Fehler. Der folgende Code generiert den Fehler.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

Dieser Code generiert die Fehlermeldung: `[line 4] mixes physical qubits and qubits registers.`

Fehler beim Abrufen von Ergebnistypen und Messen qubits im selben Programm

Das Anfordern von qubits Ergebnistypen, die explizit im selben Programm gemessen wurden, führt zu einem Fehler. Der folgende Code generiert den Fehler.

```
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])
```

Dieser Code generiert die Fehlermeldung: `Qubits should not be explicitly measured when result types are requested.`

Die klassischen Grenzwerte und die qubit Registergrenzen haben den Fehler überschritten

Nur ein klassisches Register und ein qubit Register sind zulässig. Der folgende Code generiert den Fehler.

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

Dieser Code generiert die Fehlermeldung: `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

Feld, dem kein wörtlicher Pragma-Fehler vorausgeht

Allen Feldern muss ein wörtliches Pragma vorangestellt werden. Der folgende Code generiert den Fehler.

```
box{
rx(0.5) $0;
}
```

Dieser Code generiert die Fehlermeldung: `In verbatim boxes, native gates are required. x is not a device native gate.`

Fehler bei verbatim-Boxen, bei denen native Gates fehlen

Verbatim-Boxen sollten systemeigene Tore und physische Eingänge haben. qubits Der folgende Code generiert den Native-Gate-Fehler.

```
#pragma braket verbatim
box{
```

```
x $0;
}
```

Dieser Code generiert die Fehlermeldung: `In verbatim boxes, native gates are required. x is not a device native gate.`

Bei den verbatim-Boxen fehlt ein physischer Fehler qubits

Verbatim-Boxen müssen physisch sein. qubits Der folgende Code generiert den fehlenden physikalischen qubits Fehler.

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

Dieser Code generiert die Fehlermeldung: `Physical qubits are required in verbatim box.`

Im wörtlichen Pragma fehlt der Fehler „Braket“

Sie müssen „Braket“ in das wörtliche Pragma aufnehmen. Der folgende Code generiert den Fehler.

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

Dieser Code generiert die Fehlermeldung: `You must include “braket” in the verbatim pragma`

Fehler „Single qubits kann nicht indexiert werden“

Single qubits kann nicht indexiert werden. Der folgende Code generiert den Fehler.

```
OPENQASM 3;

qubit q;
h q[0];
```

Dieser Code generiert den Fehler: `[line 4] single qubit cannot be indexed.`

Einzelne qubit Arrays können jedoch wie folgt indexiert werden:

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

Fehler: Die physikalischen qubits Verbindungen in zwei qubit Gates sind nicht verbunden

Um physische Geräte zu verwendenqubits, überprüfen Sie zunächst, ob das Gerät physische qubits Geräte verwendet, `device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits` und überprüfen Sie dann das Verbindungsdiagramm, indem Sie auf `device.properties.paradigm.connectivity.connectivityGraph` oder klicken`device.properties.paradigm.connectivity.fullyConnected`.

```
OPENQASM 3;

cnot $0, $14;
```

Dieser Code generiert die Fehlermeldung: `[line 3] has disconnected qubits 0 and 14`

Warnung zur Unterstützung des lokalen Simulators

Das `LocalSimulator` unterstützt erweiterte Funktionen in OpenQASM, die möglicherweise nicht auf Simulatoren QPUs oder On-Demand-Simulatoren verfügbar sind. Wenn Ihr Programm Sprachfunktionen enthält, die nur für die spezifisch sind`LocalSimulator`, wie im folgenden Beispiel gezeigt, erhalten Sie eine Warnung.

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

Dieser Code generiert die Warnung: `Dieses Programm verwendet OpenQASM-Sprachfunktionen, die nur in der unterstützt werden. LocalSimulator Einige dieser Funktionen werden möglicherweise nicht auf Simulatoren QPUs oder On-Demand-Simulatoren unterstützt.

Weitere Informationen zu den unterstützten OpenQASM-Funktionen finden Sie auf der Seite Erweiterte Funktionsunterstützung für OpenQASM im Local Simulator.

Sicherheit in Amazon Braket

Cloud-Sicherheit hat AWS höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der AWS Dienste in der ausgeführt AWS Cloud werden. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Third-party Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den Compliance-Programmen, die für Amazon Braket gelten, finden Sie unter AWS Services im Umfang nach Compliance-Programm AWS .
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Service, den Sie nutzen. Sie sind auch für andere Faktoren verantwortlich, einschließlich der Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von Braket anwenden können. In den folgenden Themen erfahren Sie, wie Sie Braket so konfigurieren, dass es Ihre Sicherheits- und Compliance-Ziele erreicht. Sie erfahren auch, wie Sie andere AWS Dienste nutzen können, die Ihnen bei der Überwachung und Sicherung Ihrer Braket-Ressourcen helfen.

In diesem Abschnitt:

- Gemeinsame Verantwortung für die Sicherheit
- Datenschutz
- Datenaufbewahrung
- Zugriff auf Amazon Braket verwalten
- Rolle im Zusammenhang mit dem Service von Amazon Braket
- Konformitätsprüfung für Amazon Braket
- Infrastruktursicherheit in Amazon Braket
- Sicherheit von Amazon Braket-Hardwareanbietern
- Amazon VPC-Endpunkte für Amazon Braket

Gemeinsame Verantwortung für die Sicherheit

Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS-Services in der läuft AWS Cloud. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Third-party Auditoren testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS Compliance-Programme. Weitere Informationen zu den Compliance-Programmen, die für Amazon Braket gelten, finden Sie unter AWS Services in Scope by Compliance Program.
- Sicherheit in der Cloud — Sie sind dafür verantwortlich, die Kontrolle über Ihre Inhalte zu behalten, die auf dieser AWS Infrastruktur gehostet werden. Dieser Inhalt umfasst die Sicherheitskonfiguration und die Verwaltungsaufgaben für die AWS-Services , die Sie verwenden.

Datenschutz

Das AWS Modell der gilt für den Datenschutz in Amazon Braket. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der AWS Cloud alle Systeme laufen. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz . Informationen zum Datenschutz in Europa finden Sie im General Data Protection Regulation (GDPR) Center.

Aus Datenschutzgründen empfehlen wir, dass Sie Ihre AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.

- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit Amazon Braket oder anderen AWS-Services über die Konsole AWS CLI, API oder AWS SDKs arbeiten. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Datenaufbewahrung

Nach 90 Tagen entfernt Amazon Braket automatisch alle Quantenaufgaben-IDs und andere Metadaten, die mit Ihren Quantenaufgaben verknüpft sind. Aufgrund dieser Datenaufbewahrungsrichtlinie können diese Aufgaben und Ergebnisse nicht mehr per Suche in der Amazon Braket-Konsole abgerufen werden, obwohl sie in Ihrem S3-Bucket gespeichert bleiben.

Wenn Sie Zugriff auf historische Quantenaufgaben und -ergebnisse benötigen, die länger als 90 Tage in Ihrem S3-Bucket gespeichert sind, müssen Sie Ihre Aufgaben-ID und andere mit diesen Daten verknüpfte Metadaten separat aufzeichnen. Achten Sie darauf, die Informationen vor Ablauf von 90 Tagen zu speichern. Sie können diese gespeicherten Informationen verwenden, um die historischen Daten abzurufen.

Zugriff auf Amazon Braket verwalten

In diesem Kapitel werden die Berechtigungen beschrieben, die erforderlich sind, um Amazon Braket auszuführen oder den Zugriff bestimmter Benutzer und Rollen einzuschränken. Sie können jedem Benutzer oder jeder Rolle in Ihrem Konto die erforderlichen Berechtigungen gewähren (oder verweigern). Fügen Sie dazu diesem Benutzer oder dieser Rolle in Ihrem Konto die entsprechende Amazon Braket-Richtlinie bei, wie in den folgenden Abschnitten beschrieben.

Als Voraussetzung müssen Sie Amazon Braket aktivieren. Um Braket zu aktivieren, müssen Sie sich als Benutzer oder Rolle anmelden, der (1) Administratorrechte oder (2) die AmazonBraketFullAccessRichtlinie zugewiesen wurde und die Berechtigungen zum Erstellen von Amazon Simple Storage Service (Amazon S3) -Buckets besitzt.

In diesem Abschnitt:

- Ressourcen für Amazon Braket
- Notizbücher und Rollen
- AWS verwaltete Richtlinien für Amazon Braket
- Beschränken Sie den Benutzerzugriff auf bestimmte Geräte
- Beschränken Sie den Benutzerzugriff auf bestimmte Notebook-Instanzen
- Beschränken Sie den Benutzerzugriff auf bestimmte S3-Buckets

Ressourcen für Amazon Braket

Braket erstellt einen Ressourcentyp: die Quantum-Task-Ressource. Der AWS Ressourcenname (ARN) für diesen Ressourcentyp lautet wie folgt:

- Ressourcenname:: :Service AWS: :Braket
- ARN Regex: arn: $ {Partition} :braket: $ {Region} :$ {Account} :quantum-task/$ {} RandomId

Notizbücher und Rollen

Sie können den Ressourcentyp Notizbuch in Braket verwenden. Ein Notizbuch ist eine Amazon SageMaker AI-Ressource, die Braket gemeinsam nutzen kann. Um ein Notebook mit Braket zu verwenden, müssen Sie eine IAM-Rolle angeben, deren Name mit beginnt. `AmazonBraketServiceSageMakerNotebook`

Um ein Notizbuch zu erstellen, müssen Sie eine Rolle mit Administratorberechtigungen verwenden oder der die folgende Inline-Richtlinie zugeordnet ist.

JSON

```
{
  "Version":"2012-10-17",
```

```
  "Statement": [
    {
      "Sid": "CreateTheRole",
      "Effect": "Allow",
      "Action": "iam:CreateRole",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
    },
    {
      "Sid": "CreateThePolicy",
      "Effect": "Allow",
      "Action": "iam:CreatePolicy",
      "Resource": [
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
      ]
    },
    {
      "Sid": "AttachTheRolePolicy",
      "Effect": "Allow",
      "Action": "iam:AttachRolePolicy",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
      "Condition": {
        "ArnLike": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/AmazonBraketFullAccess",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
          ]
        }
      }
    }
  ]
}
```

Um die Rolle zu erstellen, folgen Sie den Schritten auf der Seite Notizbuch erstellen oder lassen Sie sie von Ihrem Administrator für Sie erstellen. Stellen Sie sicher, dass die AmazonBraketFullAccessRichtlinie angehängt ist.

Nachdem Sie die Rolle erstellt haben, können Sie diese Rolle für alle Notizbücher wiederverwenden, die Sie in future auf den Markt bringen.

AWS verwaltete Richtlinien für Amazon Braket

Eine AWS verwaltete Richtlinie ist eine eigenständige Richtlinie, die von erstellt und verwaltet wird AWS. AWS Verwaltete Richtlinien sind so konzipiert, dass sie Berechtigungen für viele gängige Anwendungsfälle bereitstellen, sodass Sie damit beginnen können, Benutzern, Gruppen und Rollen Berechtigungen zuzuweisen.

Beachten Sie, dass AWS verwaltete Richtlinien für Ihre speziellen Anwendungsfälle möglicherweise keine Berechtigungen mit den geringsten Rechten gewähren, da sie allen AWS Kunden zur Verfügung stehen. Wir empfehlen Ihnen, die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind.

Sie können die in AWS verwalteten Richtlinien definierten Berechtigungen nicht ändern. Wenn die in einer AWS verwalteten Richtlinie definierten Berechtigungen AWS aktualisiert werden, wirkt sich das Update auf alle Prinzidentitäten (Benutzer, Gruppen und Rollen) aus, denen die Richtlinie zugeordnet ist. AWS aktualisiert eine AWS verwaltete Richtlinie höchstwahrscheinlich, wenn eine neue Richtlinie eingeführt AWS-Service wird oder neue API-Operationen für bestehende Dienste verfügbar werden.

Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien im IAM-Benutzerhandbuch.

Themen

- AWS verwaltete Richtlinie: AmazonBraketFullAccess
- AWS verwaltete Richtlinie: AmazonBraketJobsExecutionPolicy
- AWS verwaltete Richtlinie: AmazonBraketServiceRolePolicy
- Amazon Braket-Updates für AWS verwaltete Richtlinien

AWS verwaltete Richtlinie: AmazonBraketFullAccess

Die AmazonBraketFullAccessRichtlinie gewährt Berechtigungen für Amazon Braket-Operationen, einschließlich Berechtigungen für die folgenden Aufgaben:

- Container von Amazon Elastic Container Registry herunterladen — Zum Lesen und Herunterladen von Container-Images, die für die Amazon Braket Hybrid Jobs-Funktion verwendet werden. Die Container müssen dem Format „arn:aws:ecr: ::repository/amazon-braket“ entsprechen.
- AWS CloudTrail Protokolle führen — Für alle Aktionen zum Beschreiben, Abrufen und Auflisten sowie für das Starten und Beenden von Abfragen, das Testen von Metrikfiltern und das Filtern von Protokollereignissen. Die AWS CloudTrail Protokolldatei enthält eine Aufzeichnung aller Amazon API Braket-Aktivitäten, die in Ihrem Konto stattfinden.
- Verwenden Sie Rollen zur Kontrolle von Ressourcen — Um eine dienstbezogene Rolle in Ihrem Konto zu erstellen. Die dienstbezogene Rolle hat in Ihrem Namen Zugriff auf AWS Ressourcen. Es kann nur vom Amazon Braket-Service verwendet werden. Außerdem, um IAM-Rollen an Amazon Braket zu übergeben und eine Rolle zu erstellen `CreateJob` API und der Rolle eine Richtlinie zuzuweisen, auf die sie beschränkt AmazonBraketFullAccess ist.
- Protokollgruppen, Protokollereignisse und Abfrageprotokollgruppen erstellen, um Nutzungsprotokolldateien für Ihr Konto zu verwalten — Um Protokollinformationen zur Nutzung von Amazon Braket in Ihrem Konto zu erstellen, zu speichern und einzusehen. Abfragen von Metriken zu Protokollgruppen für hybride Jobs Geben Sie den richtigen Braket-Pfad an und ermöglichen Sie das Einfügen von Protokolldaten. Geben Sie metrische Daten ein. CloudWatch
- Daten in Amazon S3 S3-Buckets erstellen und speichern und alle Buckets auflisten — Um S3-Buckets zu erstellen, listen Sie die S3-Buckets in Ihrem Konto auf und fügen Objekte in jeden Bucket in Ihrem Konto ein, dessen Name mit amazon-braket- beginnt, und holen Sie Objekte aus diesen ab. Diese Berechtigungen sind erforderlich, damit Braket Dateien mit Ergebnissen von verarbeiteten Quantenaufgaben in den Bucket legen und sie aus dem Bucket abrufen kann.
- IAM-Rollen weitergeben — Um IAM-Rollen an die zu übergeben. `CreateJob` API
- Amazon SageMaker AI Notebook — Zum Erstellen und Verwalten von Notebook-Instances, die auf die Ressource von „arn:aws:sagemaker: SageMaker ::notebook-instance/amazon-braket-“ beschränkt sind.
- Servicekontingente validieren — Um SageMaker KI-Notebooks und Amazon Braket Hybrid-Jobs zu erstellen, darf Ihre Ressourcenanzahl die Kontingente für Ihr Konto nicht überschreiten.
- Produktpreise anzeigen — Prüfen und planen Sie die Kosten für Quantenhardware, bevor Sie Ihre Workloads einreichen.

Die Berechtigungen für diese Richtlinie finden Sie AmazonBraketFullAccessin der AWS Managed Policy Reference.

AWS verwaltete Richtlinie: AmazonBraketJobsExecutionPolicy

Die AmazonBraketJobsExecutionPolicyRichtlinie gewährt Berechtigungen für Ausführungsrollen, die in Amazon Braket Hybrid Jobs verwendet werden, wie folgt:

- Container von Amazon Elastic Container Registry herunterladen — Berechtigungen zum Lesen und Herunterladen von Container-Images, die für die Amazon Braket Hybrid Jobs-Funktion verwendet werden. Container müssen dem Format „arn:aws:ecr: *:*:repository/amazon-braket*“ entsprechen.
- Erstellen Sie Protokollgruppen und protokollieren Sie Ereignisse und fragen Sie Protokollgruppen ab, um Nutzungsprotokolldateien für Ihr Konto zu verwalten — Protokollinformationen zur Nutzung von Amazon Braket in Ihrem Konto erstellen, speichern und anzeigen. Abfragen von Metriken zu Protokollgruppen für hybride Jobs Geben Sie den richtigen Braket-Pfad an und ermöglichen Sie das Einfügen von Protokolldaten. Geben Sie metrische Daten ein. CloudWatch
- Daten in Amazon S3 S3-Buckets speichern — Listet die S3-Buckets in Ihrem Konto auf, platziert Objekte und ruft Objekte aus jedem Bucket in Ihrem Konto ab, der mit amazon-braket - im Namen beginnt. Diese Berechtigungen sind erforderlich, damit Braket Dateien mit Ergebnissen von verarbeiteten Quantenaufgaben in den Bucket legen und sie aus dem Bucket abrufen kann.
- IAM-Rollen weitergeben — Weitergabe von IAM-Rollen an die. CreateJob API Rollen müssen dem Format arn:aws:iam: :* * entsprechen. :role/service-role/AmazonBraketJobsExecutionRole

Die Berechtigungen für diese Richtlinie finden Sie AmazonBraketJobsExecutionPolicyin der AWS Managed Policy Reference.

AWS verwaltete Richtlinie: AmazonBraketServiceRolePolicy

Die AmazonBraketServiceRolePolicyRichtlinie gewährt Berechtigungen für Amazon Braket-Operationen, einschließlich Berechtigungen für die folgenden Aufgaben:

- Amazon S3 — Berechtigungen zum Auflisten der Buckets in Ihrem Konto und zum Ablegen von Objekten in und Abrufen von Objekten aus jedem Bucket in Ihrem Konto, dessen Name mit `amazon-braket-` beginnt.
- Amazon CloudWatch Logs — Berechtigungen zum Auflisten und Erstellen von Protokollgruppen, zum Erstellen der zugehörigen Protokollstreams und zum Einfügen von Ereignissen in die für Amazon Braket erstellte Protokollgruppe.

Weitere Informationen zu serviceverknüpften Rollen finden Sie unter Servicebezogene Rollen in Amazon Braket.

Die Berechtigungen für diese Richtlinie finden Sie AmazonBraketServiceRolePolicyin der AWS Managed Policy Reference.

Amazon Braket-Updates für AWS verwaltete Richtlinien

Die folgende Tabelle enthält Einzelheiten zu den Aktualisierungen der AWS verwalteten Richtlinien für Amazon Braket seit Beginn der Nachverfolgung dieser Änderungen durch diesen Service.

Änderung	Beschreibung	Date (Datum)
AmazonBraketServiceRolePolicy- Richtlinie zur Ressourcenverwaltung	Der Bedingungsbereich „aws:ResourceAccount": „${aws:PrincipalAccount}“ wurde zu Amazon S3 hinzugefügt und CloudWatch protokolliert Aktionen.	11. Juli 2025
AmazonBraketFullAccess- Vollständige Zugriffsrichtlinie für Braket	Die Aktion „Preisgestaltung:GetProducts“ wurde hinzugefügt.	14. April 2025
AmazonBraketFullAccess- Vollständige Zugriffsrichtlinie für Braket	Der Bedingungsbereich „aws: ResourceAccount „: „${aws:PrincipalAccount}“ wurde zu S3-Aktionen hinzugefügt.	7. März 2025
AmazonBraketFullAccess- Vollständige Zugriffsrichtlinie für Braket	Die Aktionen servicequotas: GetServiceQuota und cloudwatch: hinzugefügt. GetMetricData	24. März 2023
AmazonBraketFullAccess- Vollständige Zugriffsrichtlinie für Braket	Die s3: ListAllMyBuckets - Berechtigungen zum Anzeigen und Prüfen der verwendeten Amazon S3 S3-Buckets wurden hinzugefügt.	31. März 2022

Änderung	Beschreibung	Date (Datum)
AmazonBraketFullAccess- Vollständige Zugriffsrichtlinie für Braket	Braket hat iam angepasst : PassRole Berechtigungen AmazonBraketFullAccess zum Einschließen des Pfads. `service-role/`	29. November 2021
AmazonBraketJobsExecutionPolicy- Richtlinie zur Ausführung von Hybrid-Jobs für Amazon Braket-Hybrid-Jobs	Braket hat den ARN für die Ausführung von Hybrid-Jobs aktualisiert, sodass er den `service-role/` Pfad enthält.	29. November 2021
Braket hat begonnen, Änderungen zu verfolgen	Braket begann, Änderungen an seinen AWS verwalteten Richtlinien nachzuverfolgen.	29. November 2021

Beschränken Sie den Benutzerzugriff auf bestimmte Geräte

Um den Benutzerzugriff für bestimmte Braket-Geräte einzuschränken, können Sie einer bestimmten IAM Rolle eine Richtlinie zum Verweigern von Berechtigungen hinzufügen.

Die folgenden Aktionen können eingeschränkt werden:

- `CreateQuantumTask`- um die Erstellung von Quantenaufgaben auf bestimmten Geräten zu verweigern.
- `CreateJob`- die Schaffung hybrider Arbeitsplätze auf bestimmten Geräten zu verweigern.
- `GetDevice`- um zu verhindern, dass Details zu bestimmten Geräten abgerufen werden.

Das folgende Beispiel schränkt den Zugriff auf alle QPUs für die AWS-Konto `123456789012` ein.

JSON

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

```
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ],
    "Condition": {
            "StringEquals": {
                "aws:PrincipalAccount": "123456789012"
            }
      }
    }
  ]
}
```

> **Note**
>
> Schließen Sie die `braket:GetDevice` Aktion aus der Richtlinie aus, um einem Benutzer Lesezugriff auf Geräteeigenschaften wie Geräteverfügbarkeit, Kalibrierungsdaten und Preise über die Braket-Konsole zu ermöglichen.

Um diesen Code anzupassen, ersetzen Sie die im vorherigen Beispiel gezeigte Zeichenfolge durch die Amazon Ressourcennummer (ARN) des eingeschränkten Geräts. Diese Zeichenfolge stellt den Ressourcenwert bereit. In Braket steht ein Gerät für eine QPU oder einen Simulator, den Sie aufrufen können, um Quantenaufgaben auszuführen. Die verfügbaren Geräte sind auf der Geräteseite aufgeführt. Es gibt zwei Schemas, die verwendet werden, um den Zugriff auf diese Geräte zu spezifizieren:

- `arn:aws:braket:<region>:*:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:*:device/quantum-simulator/<provider>/<device_id>`

Hier finden Sie Beispiele für verschiedene Arten des Gerätezugriffs

- Um alles aus QPUs allen Regionen auszuwählen: `arn:aws:braket:*:*:device/qpu/*`

- Um NUR alle QPUs in der Region US-West-2 auszuwählen: `arn:aws:braket:us-west-2:*:device/qpu/*`
- Entsprechend, um NUR alle QPUs in der Region US-West-2 auszuwählen (da Geräte eine Serviceressource und keine Kundenressource sind): `arn:aws:braket:us-west-2:*:device/qpu/*`
- So beschränken Sie den Zugriff auf alle On-Demand-Simulatorgeräte: `arn:aws:braket:*:*:device/quantum-simulator/*`
- So beschränken Sie den Zugriff auf Geräte eines bestimmten Anbieters (z. B. auf Rigetti QPU Geräte): `arn:aws:braket:*:*:device/qpu/rigetti/*`
- Um den Zugriff auf das TN1 Gerät einzuschränken: `arn:aws:braket:*:*:device/quantum-simulator/amazon/tn1`
- Um den Zugriff auf alle `Create` Aktionen einzuschränken: `braket:Create*`

Beschränken Sie den Benutzerzugriff auf bestimmte Notebook-Instanzen

Um den Zugriff bestimmter Benutzer auf bestimmte Braket-Notebook-Instanzen zu beschränken, können Sie einer bestimmten Rolle, einem bestimmten Benutzer oder einer bestimmten Gruppe eine Richtlinie zum Verweigern von Berechtigungen hinzufügen.

Im folgenden Beispiel werden Richtlinienvariablen verwendet, um die Berechtigungen zum Starten, Stoppen und Zugreifen auf bestimmte Notebook-Instanzen in der effizient einzuschränken AWS-Konto `123456789012`, die nach dem Benutzer benannt sind, der Zugriff haben soll (der Benutzer `Alice` hätte beispielsweise Zugriff auf eine Notebook-Instanz mit dem Namen`amazon-braket-Alice`).

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCreateDeleteUpdateNotebookInstances",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
```

```
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyDescribeStartStopNotebookInstances",
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Sid": "DenyNotebookInstanceUrl",
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

Beschränken Sie den Benutzerzugriff auf bestimmte S3-Buckets

Um den Zugriff bestimmter Benutzer auf bestimmte Amazon S3 S3-Buckets zu beschränken, können Sie einer bestimmten Rolle, einem bestimmten Benutzer oder einer bestimmten Gruppe eine Ablehnungsrichtlinie hinzufügen.

Das folgende Beispiel schränkt die Berechtigungen zum Abrufen und Platzieren von Objekten in einem bestimmten S3 Bucket (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) ein und schränkt auch die Auflistung dieser Objekte ein.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "s3:ListBucket"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "s3:GetObject"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice/*"
      ]
    }
  ]
}
```

Um den Zugriff auf den Bucket für eine bestimmte Notebook-Instanz einzuschränken, können Sie die vorherige Richtlinie zur Notebook-Ausführungsrolle hinzufügen.

Rolle im Zusammenhang mit dem Service von Amazon Braket

Wenn Sie Amazon Braket aktivieren, wird in Ihrem Konto eine serviceverknüpfte Rolle erstellt.

Eine serviceverknüpfte Rolle ist eine einzigartige Art von IAM-Rolle, die in diesem Fall direkt mit Amazon Braket verknüpft ist. Die mit dem Service verknüpfte Amazon Braket-Rolle ist so vordefiniert, dass sie alle Berechtigungen enthält, die Braket benötigt, wenn es in AWS-Services Ihrem Namen andere Personen anruft.

Eine servicebezogene Rolle erleichtert die Einrichtung von Amazon Braket, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Amazon Braket definiert die Berechtigungen

seiner serviceverknüpften Rollen. Sofern Sie diese Definitionen nicht ändern, kann nur Amazon Braket seine Rollen übernehmen. Zu den definierten Berechtigungen gehören die Vertrauensrichtlinie und die Berechtigungsrichtlinie. Die Berechtigungsrichtlinie kann an keine andere IAM-Entität angefügt werden.

Die serviceverknüpfte Rolle, die Amazon Braket einrichtet, ist Teil der AWS Identity and Access Management (IAM) -Funktion für serviceverknüpfte Rollen. Informationen zu anderen Rollen AWS-Services , die serviceverknüpfte Rollen unterstützen, finden Sie unter AWS Services That Work with IAM und suchen Sie in der Spalte Serviceverknüpfte Rolle nach den Services, für die Ja steht. Wählen Sie Ja mit einem Link aus, um die Dokumentation zur serviceverknüpften Rolle für diesen Dienst aufzurufen.

Weitere Informationen zur AWS verwalteten Richtlinie für dienstbezogene Rollen finden Sie unter. AmazonBraketServiceRolePolicy

Konformitätsprüfung für Amazon Braket

> **Note**
>
> AWS Die Compliance-Berichte beziehen sich nicht auf QPUs von Hardware-Drittanbietern, die sich dafür entscheiden können, ihre eigenen unabhängigen Audits zu durchlaufen.

Um zu erfahren, ob AWS-Service ein in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter AWS-Services Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Infrastruktursicherheit in Amazon Braket

Als verwalteter Service ist Amazon Braket durch AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf Amazon Braket zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Sie können diese API-Operationen von jedem Netzwerkstandort aus aufrufen, Braket unterstützt jedoch ressourcenbasierte Zugriffsrichtlinien, die Einschränkungen auf der Grundlage der Quell-IP-Adresse beinhalten können. Sie können auch Braket-Richtlinien verwenden, um den Zugriff von bestimmten Amazon Virtual Private Cloud (Amazon VPC) -Endpunkten oder bestimmten zu kontrollieren. VPCs Dadurch wird der Netzwerkzugriff auf eine bestimmte Braket-Ressource effektiv nur von der spezifischen VPC innerhalb des Netzwerks isoliert. AWS

Sicherheit von Amazon Braket-Hardwareanbietern

QPUs auf Amazon Braket werden von Hardware-Drittanbietern gehostet. Wenn Sie Ihre Quantenaufgabe auf einer QPU ausführen, verwendet Amazon Braket den DeviceARN als Kennung, wenn der Schaltkreis zur Verarbeitung an die angegebene QPU gesendet wird.

Wenn Sie Amazon Braket für den Zugriff auf Quantencomputer-Hardware verwenden, die von einem der Drittanbieter betrieben wird, werden Ihr Schaltkreis und die damit verbundenen Daten von Hardwareanbietern außerhalb der Einrichtungen verarbeitet, die von AWS betrieben werden. Informationen zum physischen Standort und zur AWS Region, in der jede QPU verfügbar ist, finden Sie im Abschnitt Gerätedetails der Amazon Braket-Konsole.

Ihr Inhalt ist anonymisiert. Nur der Inhalt, der für die Bearbeitung der Schaltung erforderlich ist, wird an Dritte gesendet. AWS-Konto Informationen werden nicht an Dritte weitergegeben.

Alle Daten werden im Ruhezustand und bei der Übertragung verschlüsselt. Daten werden nur zur Verarbeitung entschlüsselt. Drittanbieter von Amazon Braket dürfen Ihre Inhalte nicht für andere Zwecke als die Verarbeitung Ihrer Verbindung speichern oder verwenden. Sobald der Kreislauf abgeschlossen ist, werden die Ergebnisse an Amazon Braket zurückgegeben und in Ihrem S3-Bucket gespeichert.

Die Sicherheit der Drittanbieter von Quantenhardware von Amazon Braket wird regelmäßig überprüft, um sicherzustellen, dass die Standards für Netzwerksicherheit, Zugriffskontrolle, Datenschutz und physische Sicherheit eingehalten werden.

Amazon VPC-Endpunkte für Amazon Braket

Sie können eine private Verbindung zwischen Ihrer VPC und Amazon Braket herstellen, indem Sie einen VPC-Schnittstellen-Endpunkt erstellen. Schnittstellenendpunkte basieren auf einer Technologie AWS PrivateLink, die den Zugriff auf Braket-APIs ohne Internet-Gateway, NAT-Gerät, VPN-Verbindung oder Verbindung ermöglicht. Direct Connect Instances in Ihrer VPC benötigen keine öffentlichen IP-Adressen, um mit Braket-APIs zu kommunizieren.

Jeder Schnittstellenendpunkt wird durch eine oder mehrere Elastic-Network-Schnittstellen in Ihren Subnetzen dargestellt.

Damit AWS PrivateLink verlässt der Datenverkehr zwischen Ihrer VPC und Braket das Amazon Netzwerk nicht, was die Sicherheit der Daten, die Sie mit Cloud-basierten Anwendungen teilen, erhöht, da Ihre Daten weniger dem öffentlichen Internet ausgesetzt sind. Weitere Informationen finden Sie unter Zugreifen auf einen AWS Service über einen Schnittstellen-VPC-Endpunkt im Amazon VPC-Benutzerhandbuch.

In diesem Abschnitt:

- Überlegungen zu Amazon Braket VPC-Endpunkten
- Richten Sie Braket ein und PrivateLink
- Zusätzliche Informationen zum Erstellen eines Endpunkts
- Steuern Sie den Zugriff mit Amazon VPC-Endpunktrichtlinien

Überlegungen zu Amazon Braket VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für Braket einrichten, stellen Sie sicher, dass Sie die Voraussetzungen für Schnittstellen-Endpunkte im Amazon VPC-Benutzerhandbuch lesen.

Braket unterstützt Aufrufe all seiner API-Aktionen von Ihrer VPC aus.

Standardmäßig ist der vollständige Zugriff auf Braket über den VPC-Endpunkt zulässig. Sie können den Zugriff kontrollieren, wenn Sie VPC-Endpunktrichtlinien angeben. Weitere Informationen finden Sie unter Steuern des Zugriffs auf VPC-Endpunkte mithilfe von Endpunktrichtlinien im Amazon-VPC-Benutzerhandbuch.

Richten Sie Braket ein und PrivateLink

Für die Verwendung AWS PrivateLink mit Amazon Braket müssen Sie einen Amazon Virtual Private Cloud (Amazon VPC) -Endpunkt als Schnittstelle erstellen und dann über den Amazon API Braket-Service eine Verbindung zum Endpunkt herstellen.

Hier sind die allgemeinen Schritte dieses Prozesses, die in späteren Abschnitten ausführlich erläutert werden.

- Konfigurieren und starten Sie eine Amazon VPC zum Hosten Ihrer AWS Ressourcen. Wenn bereits eine VPC vorhanden ist, können Sie diesen Schritt überspringen.
- Erstellen Sie einen Amazon VPC-Endpunkt für Braket
- Connect Braket-Quantenaufgaben und führen Sie sie über Ihren Endpunkt aus

Schritt 1: Starten Sie bei Bedarf eine Amazon VPC

Denken Sie daran, dass Sie diesen Schritt überspringen können, wenn in Ihrem Konto bereits eine VPC in Betrieb ist.

Eine VPC steuert Ihre Netzwerkeinstellungen wie den IP-Adressbereich, Subnetze, Routing-Tabellen und Netzwerk-Gateways. Im Wesentlichen starten Sie Ihre AWS Ressourcen in einem benutzerdefinierten virtuellen Netzwerk. Weitere Informationen zu VPCs finden Sie im Amazon-VPC-Benutzerhandbuch.

Öffnen Sie die Amazon VPC-Konsole und erstellen Sie eine neue VPC mit Subnetzen, Sicherheitsgruppen und Netzwerk-Gateways.

Schritt 2: Erstellen Sie einen VPC-Schnittstellen-Endpunkt für Braket

Sie können einen VPC-Endpunkt für den Braket-Service entweder mit der Amazon VPC-Konsole oder mit () erstellen. AWS Command Line Interface AWS CLI Weitere Informationen finden Sie unter Erstellen eines VPC-Endpunkts im Amazon VPC-Benutzerhandbuch.

Um einen VPC-Endpunkt in der Konsole zu erstellen, öffnen Sie die Amazon VPC-Konsole, öffnen Sie die Seite Endpoints und fahren Sie mit der Erstellung des neuen Endpoints fort. Notieren Sie sich die Endpunkt-ID, um später darauf zurückgreifen zu können. Sie ist als Teil der `–endpoint-url` Kennzeichnung erforderlich, wenn Sie bestimmte Anrufe an das Braket API tätigen.

Erstellen Sie den VPC-Endpunkt für Braket mit dem folgenden Dienstnamen:

- `com.amazonaws.substitute_your_region.braket`

Weitere Informationen finden Sie unter Zugreifen auf einen AWS Service über einen Schnittstellen-VPC-Endpunkt im Amazon VPC-Benutzerhandbuch.

Schritt 3: Connect und führen Sie Braket-Quantenaufgaben über Ihren Endpunkt aus

Nachdem Sie einen VPC-Endpunkt erstellt haben, können Sie CLI-Befehle ausführen, die den `endpoint-url` Parameter zur Angabe von Schnittstellenendpunkten für die Laufzeit API oder enthalten, wie das folgende Beispiel:

```
aws braket search-quantum-tasks --endpoint-url
 VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

Wenn Sie private DNS-Hostnamen für Ihren VPC-Endpunkt aktivieren, müssen Sie den Endpunkt in Ihren CLI-Befehlen nicht als URL angeben. Stattdessen wird der Amazon API Braket-DNS-Hostname, den die CLI und das Braket-SDK standardmäßig verwenden, zu Ihrem VPC-Endpunkt aufgelöst. Er hat die im folgenden Beispiel gezeigte Form:

```
https://braket.substituteYourRegionHere.amazonaws.com
```

Der Blogbeitrag mit dem Titel Direkter Zugriff auf Amazon SageMaker AI-Notebooks von Amazon VPC über einen AWS PrivateLink Endpunkt bietet ein Beispiel dafür, wie ein Endpunkt eingerichtet wird, um sichere Verbindungen zu SageMaker Notebooks herzustellen, die Amazon Braket-Notebooks ähneln.

Wenn Sie die Schritte im Blogbeitrag befolgen, denken Sie daran, Amazon SageMaker AI durch den Namen AmazonBraket zu ersetzen. Geben Sie für Service Name Ihren korrekten AWS-Region Namen ein `com.amazonaws.us-east-1.braket` oder ersetzen Sie ihn, wenn Ihre Region nicht us-east-1 ist.

Zusätzliche Informationen zum Erstellen eines Endpunkts

- Informationen zum Erstellen einer VPC mit privaten Subnetzen finden Sie unter Erstellen einer VPC mit privaten Subnetzen.
- Informationen zum Erstellen und Konfigurieren eines Endpunkts mithilfe der Amazon VPC-Konsole oder der AWS CLI finden Sie unter Erstellen eines VPC-Endpunkts im Amazon VPC-Benutzerhandbuch.
- Informationen zum Erstellen und Konfigurieren eines Endpunkts mithilfe finden Sie in der CloudFormation Ressource AWS: :EC2: :VPCEnpoint im Benutzerhandbuch.CloudFormation

Steuern Sie den Zugriff mit Amazon VPC-Endpunktrichtlinien

Um den Konnektivitätszugriff auf Amazon Braket zu kontrollieren, können Sie Ihrem Amazon VPC-Endpunkt eine AWS Identity and Access Management (IAM-) Endpunktrichtlinie hinzufügen. Die Richtlinie gibt die folgenden Informationen an:

- Der Principal (Benutzer oder Rolle), der Aktionen ausführen kann.
- Aktionen, die ausgeführt werden können
- Die Ressourcen, für die Aktionen ausgeführt werden können.

Weitere Informationen finden Sie unter Steuern des Zugriffs auf VPC-Endpunkte mithilfe von Endpunktrichtlinien im Amazon-VPC-Benutzerhandbuch.

Beispiel: VPC-Endpunktrichtlinie für Braket-Aktionen

Das folgende Beispiel zeigt eine Endpunktrichtlinie für Braket. Wenn diese Richtlinie an einen Endpunkt angehängt ist, gewährt sie allen Prinzipalen auf allen Ressourcen Zugriff auf die aufgelisteten Braket-Aktionen.

```
{
 "Statement":[
 {
   "Principal":"*",
   "Effect":"Allow",
   "Action":[
     “braket:action-1",
     “braket:action-2",
     “braket:action-3”
```

```
      ],
    "Resource":"*"
    }
  ]
}
```

Sie können komplexe IAM-Regeln erstellen, indem Sie mehrere Endpunktrichtlinien anhängen. Weitere Informationen und Beispiele finden Sie unter:

- Amazon Virtual Private Cloud-Endpunktrichtlinien für Step Functions
- Erstellung detaillierter IAM-Berechtigungen für Benutzer Non-Admin
- Steuern Sie den Zugriff auf VPC-Endpunkte mithilfe von Endpunktrichtlinien

Protokollierung und Überwachung

Nachdem Sie eine Quantenaufgabe über den Amazon Braket-Service eingereicht haben, können Sie den Status und den Fortschritt dieser Aufgabe über das Amazon Braket-SDK und die Konsole genau verfolgen. Dies bietet Ihnen eine zentrale Oberfläche, über die Sie die Implementierung Ihrer Workloads verfolgen, potenzielle Engpässe oder Probleme identifizieren und geeignete Maßnahmen ergreifen können, um die Leistung und Zuverlässigkeit Ihrer Quantenanwendungen zu optimieren. Wenn die Quantenaufgabe abgeschlossen ist, speichert Braket die Ergebnisse an Ihrem angegebenen Amazon S3 S3-Standort. Die Bearbeitungszeit für Quantenaufgaben kann variieren, insbesondere für Aufgaben, die auf Geräten mit Quantenverarbeitungseinheiten (QPU) ausgeführt werden. Dies ist größtenteils auf die Länge der Ausführungswarteschlange zurückzuführen, da Quantenhardwareressourcen von mehreren Benutzern gemeinsam genutzt werden.

Liste der Statustypen:

- `CREATED`— Amazon Braket hat Ihre Quantenaufgabe erhalten.
- `QUEUED`— Amazon Braket hat Ihre Quantenaufgabe bearbeitet und wartet nun darauf, auf dem Gerät ausgeführt zu werden.
- `RUNNING`— Ihre Quantenaufgabe läuft auf einer QPU oder einem On-Demand-Simulator.
- `COMPLETED`— Ihre Quantenaufgabe wurde auf der QPU oder dem On-Demand-Simulator ausgeführt.
- `FAILED`— Ihre Quantenaufgabe wurde ausgeführt und ist fehlgeschlagen. Je nachdem, warum Ihre Quantenaufgabe fehlgeschlagen ist, versuchen Sie erneut, Ihre Quantenaufgabe einzureichen.
- `CANCELLED`— Sie haben die Quantenaufgabe abgesagt. Die Quantenaufgabe wurde nicht ausgeführt.

In diesem Abschnitt:

- Verfolgung von Quantenaufgaben mit dem Amazon Braket SDK
- Überwachung von Quantenaufgaben über die Amazon Braket-Konsole
- Taggen von Amazon Braket-Ressourcen
- Überwachen Sie Ihre Quantenaufgaben mit EventBridge
- Überwachen Sie Ihre Metriken mit CloudWatch
- Protokollieren Sie Ihre Quantenaufgaben mit CloudTrail

- Erweiterte Protokollierung mit Amazon Braket

Verfolgung von Quantenaufgaben mit dem Amazon Braket SDK

Der Befehl `device.run(…)` definiert eine Quantenaufgabe mit einer eindeutigen Quantenaufgaben-ID. Sie können den Status abfragen und verfolgen, `task.state()` wie im folgenden Beispiel gezeigt.

Hinweis: `task = device.run()` ist eine asynchrone Operation, was bedeutet, dass Sie weiterarbeiten können, während das System Ihre Quantenaufgabe im Hintergrund bearbeitet.

Rufen Sie ein Ergebnis ab

Wenn Sie anrufen`task.result()`, fragt das SDK Amazon Braket ab, um festzustellen, ob die Quantenaufgabe abgeschlossen ist. Das SDK verwendet die Abfrageparameter, die Sie in definiert haben. `.run()` Nach Abschluss der Quantenaufgabe ruft das SDK das Ergebnis aus dem S3-Bucket ab und gibt es als `QuantumTaskResult` Objekt zurück.

```
# 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
```

Brechen Sie eine Quantenaufgabe ab

Um eine Quantenaufgabe abzubrechen, rufen Sie die `cancel()` Methode auf, wie im folgenden Beispiel gezeigt.

```
# cancel quantum task
task.cancel()
status = task.state()
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

Überprüfen Sie die Metadaten

Sie können die Metadaten der fertigen Quantenaufgabe überprüfen, wie im folgenden Beispiel gezeigt.

```
# 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
```

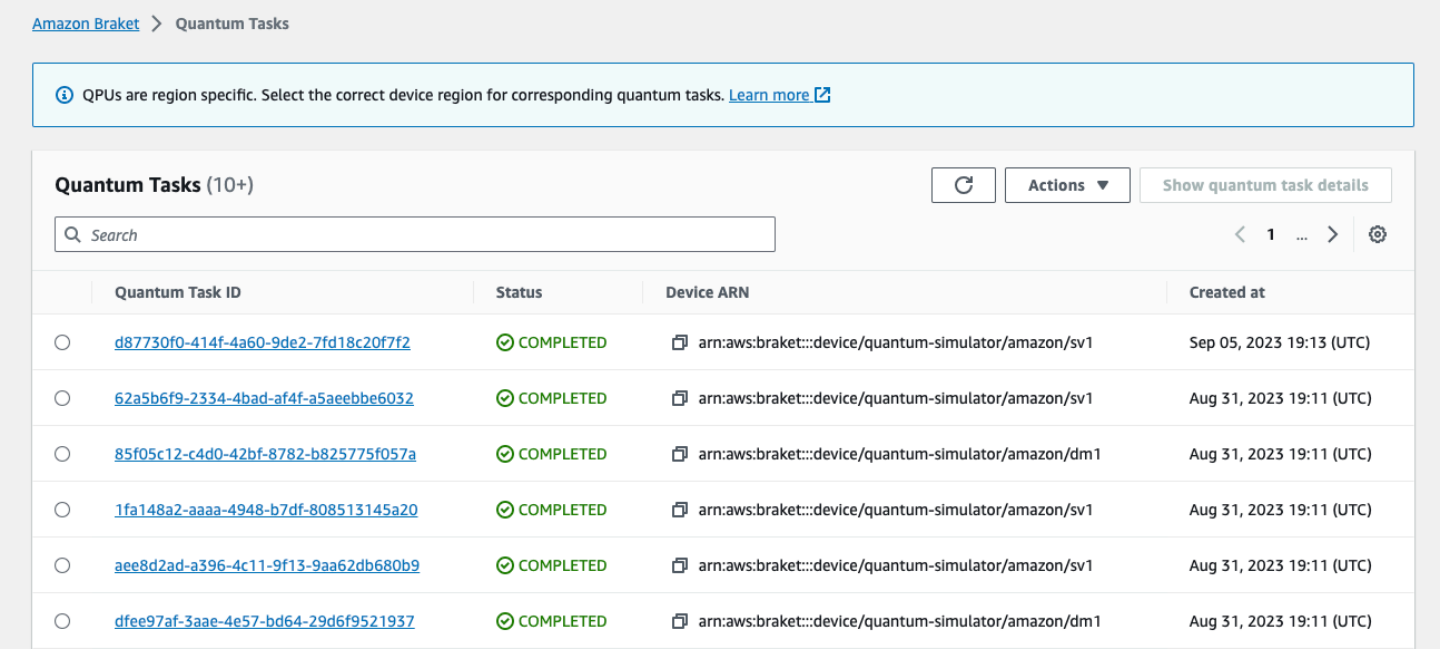
Rufen Sie eine Quantenaufgabe oder ein Ergebnis ab

Wenn Ihr Kernel stirbt, nachdem Sie die Quantenaufgabe eingereicht haben oder wenn Sie Ihr Notebook oder Ihren Computer schließen, können Sie das `task` Objekt mit seiner eindeutigen ARN (Quantenaufgaben-ID) rekonstruieren. Anschließend können Sie aufrufen`task.result()`, um das Ergebnis aus dem S3-Bucket abzurufen, in dem es gespeichert ist.

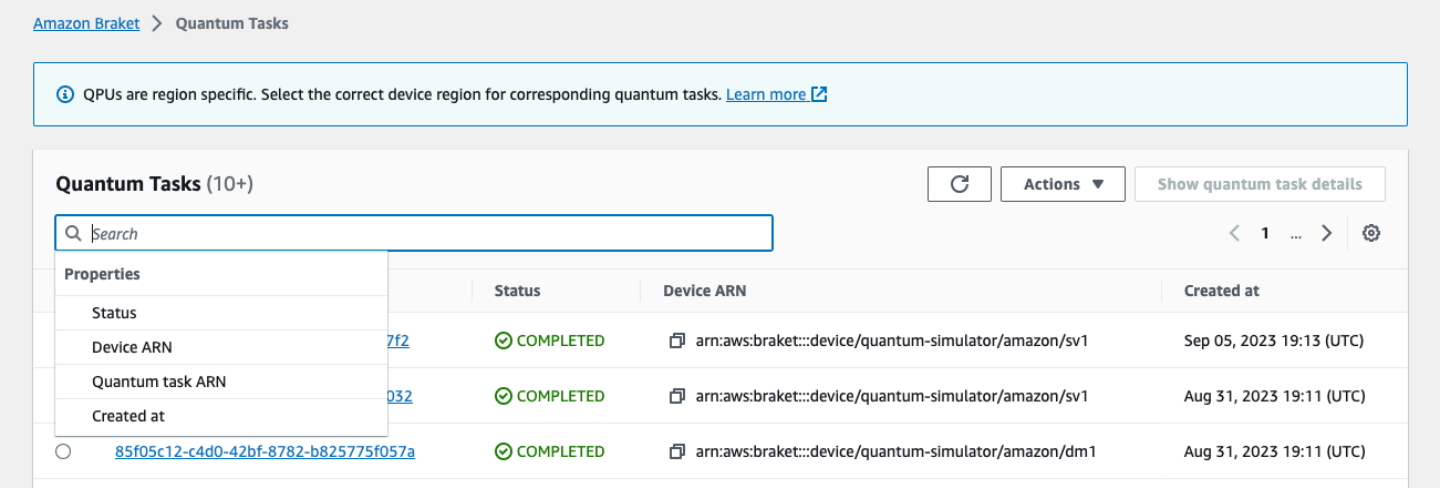
```
from braket.aws import AwsSession, AwsQuantumTask

# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Überwachung von Quantenaufgaben über die Amazon Braket-Konsole

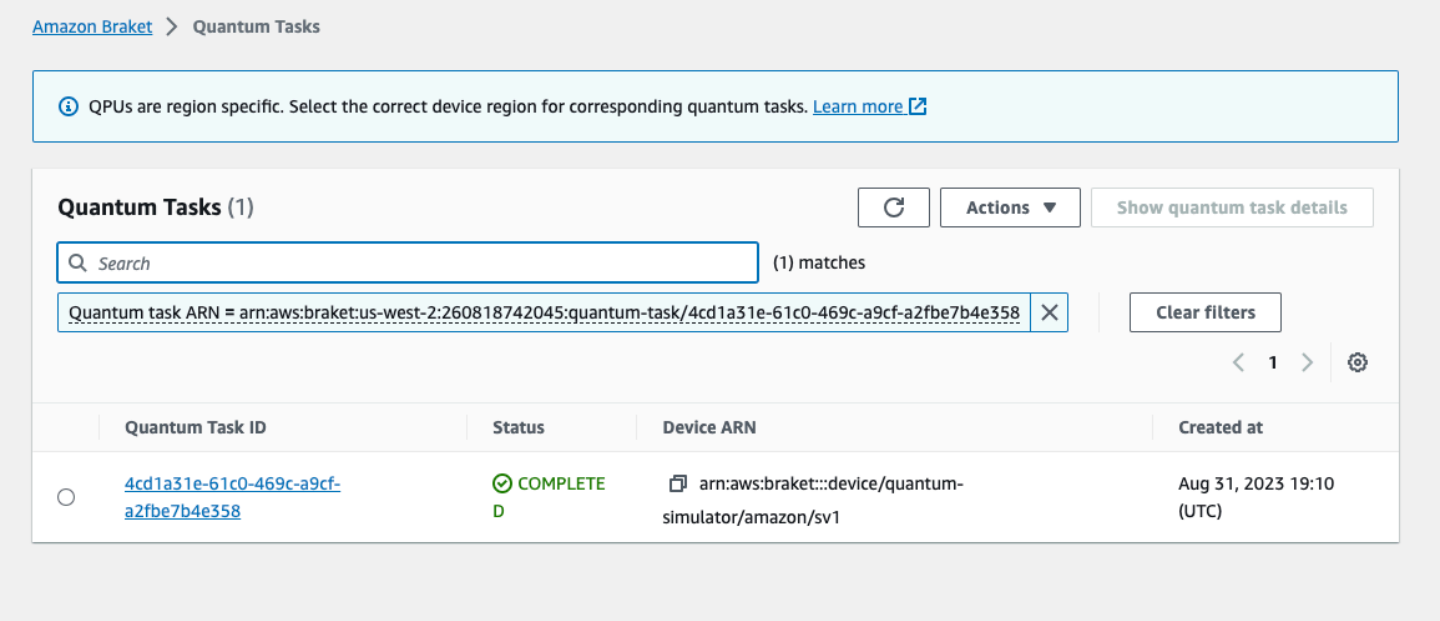
Amazon Braket bietet eine bequeme Möglichkeit, die Quantenaufgabe über die Amazon Braket-Konsole zu überwachen. Alle eingereichten Quantenaufgaben werden im Feld Quantenaufgaben aufgeführt, wie in der folgenden Abbildung dargestellt. Dieser Dienst ist regionsspezifisch, was bedeutet, dass Sie sich nur die Quantenaufgaben ansehen können, die in der jeweiligen Region erstellt wurden. AWS-Region



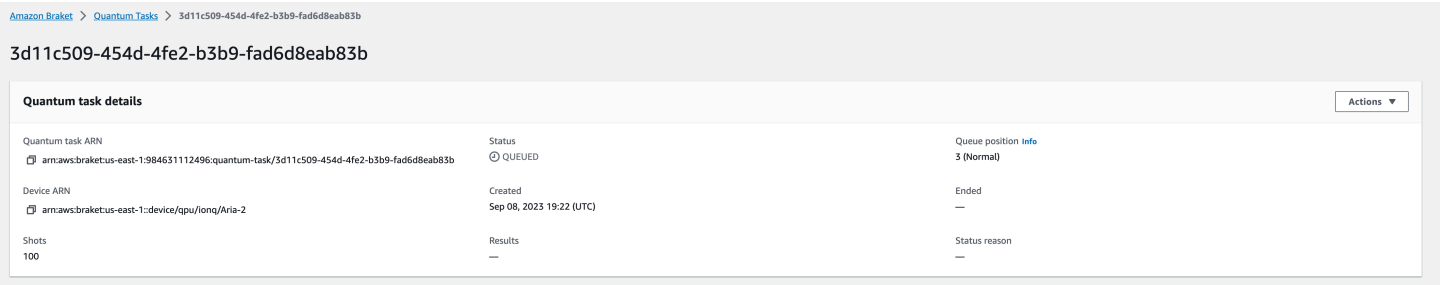
Sie können über die Navigationsleiste nach bestimmten Quantenaufgaben suchen. Die Suche kann auf Quantum Task ARN (ID), Status, Gerät und Erstellungszeit basieren. Die Optionen werden automatisch angezeigt, wenn Sie die Navigationsleiste auswählen, wie im folgenden Beispiel gezeigt.



Die folgende Abbildung zeigt ein Beispiel für die Suche nach einer Quantenaufgabe anhand ihrer eindeutigen Quantenaufgaben-ID, die durch einen Anruf `task.id` abgerufen werden kann.



Darüber hinaus kann, wie in der Abbildung unten zu sehen ist, der Status einer Quantenaufgabe überwacht werden, während sie sich in einem bestimmten QUEUED Zustand befindet. Wenn Sie auf die Quantenaufgaben-ID klicken, wird die Detailseite angezeigt. Auf dieser Seite wird die dynamische Warteschlangenposition für Ihre Quantenaufgabe im Verhältnis zu dem Gerät angezeigt, auf dem sie verarbeitet wird.



Quantenaufgaben, die im Rahmen eines Hybrid-Jobs eingereicht werden, haben Priorität, wenn sie sich in der Warteschlange befinden. Quantenaufgaben, die außerhalb eines Hybridauftrags eingereicht werden, haben die normale Priorität in der Warteschlange.

Kunden, die das Braket-SDK abfragen möchten, können ihre Warteschlangenpositionen für Quantenaufgaben und Hybrid-Jobs programmgesteuert abrufen. Weitere Informationen finden Sie auf der Seite Wann wird meine Aufgabe ausgeführt.

Taggen von Amazon Braket-Ressourcen

Ein Tag ist eine benutzerdefinierte Attributbezeichnung, die Sie einer Ressource zuweisen oder die sie einer AWS AWS Ressource zuweist. Bei einem Tag handelt es sich um Metadaten, die mehr über Ihre Ressource aussagen. Jedes Tag besteht aus einem Schlüssel und einem Wert. Dies werden als Schlüssel-Wert-Paare bezeichnet. Für Tags, die Sie zuweisen, definieren Sie einen Schlüssel und einen Wert.

In der Amazon Braket-Konsole können Sie zu einer Quantenaufgabe oder einem Notizbuch navigieren und sich die Liste der zugehörigen Tags ansehen. Sie können ein Tag hinzufügen, ein Tag entfernen oder ein Tag ändern. Sie können eine Quantenaufgabe oder ein Notizbuch bei der Erstellung taggen und dann die zugehörigen Tags über die Konsole verwalten, AWS CLI, oderAPI.

Weitere Informationen zu AWS und Tags

- Allgemeine Informationen zum Tagging, einschließlich Benennungs- und Verwendungskonventionen, finden Sie unter Was ist der Tag-Editor? im Benutzerhandbuch für AWS Tagging-Ressourcen und Tag-Editor.
- Informationen zu Einschränkungen beim Tagging finden Sie unter Beschränkungen und Anforderungen für die Benennung von Tags im AWS Tagging-Ressourcen- und Tag-Editor-Benutzerhandbuch.
- Bewährte Methoden und Tagging-Strategien finden Sie unter Bewährte Methoden für das Taggen von AWS-Ressourcen.
- Eine Liste der Services, die die Verwendung von Tags unterstützen, finden Sie in der Resource Groups Tagging API-Referenz.

Die folgenden Abschnitte enthalten genauere Informationen zu Tags für Amazon Braket.

In diesem Abschnitt:

- Verwenden von Markierungen
- Unterstützte Ressourcen für das Tagging in Amazon Braket
- Markierung mit der Amazon Braket API
- Tagging-Einschränkungen
- Verwaltung von Tags in Amazon Braket
- Beispiel für AWS CLI Tagging in Amazon Braket

Verwenden von Markierungen

Mithilfe von Tags können Sie Ihre Ressourcen in Kategorien einteilen, die für Sie nützlich sind. Sie können beispielsweise das Tag „Abteilung“ zuweisen, um die Abteilung anzugeben, der diese Ressource gehört.

Jedes -Tag besteht aus zwei Teilen:

- Ein Tag-Schlüssel (zum Beispiel CostCenter„Umgebung“ oder „Projekt“). Bei Tag-Schlüsseln wird zwischen Groß- und Kleinschreibung unterschieden.
- Ein optionales Feld, das als Tag-Wert bezeichnet wird (z. B. 111122223333 oder Production). Ein nicht angegebener Tag-Wert entspricht einer leeren Zeichenfolge. Wie bei Tag-Schlüsseln wird auch bei Tag-Werten zwischen Groß- und Kleinschreibung unterschieden.

Mithilfe von Tags können Sie die folgenden Dinge tun:

- Identifizieren und organisieren Sie Ihre AWS Ressourcen. Viele AWS-Services unterstützen Tagging, sodass Sie Ressourcen aus verschiedenen Diensten dasselbe Tag zuweisen können, um anzuzeigen, dass die Ressourcen miteinander verknüpft sind.
- Behalten Sie Ihre AWS Kosten im Blick. Sie aktivieren diese Tags auf dem AWS Fakturierung und Kostenmanagement Dashboard. AWS verwendet die Tags, um Ihre Kosten zu kategorisieren und Ihnen einen monatlichen Kostenverteilungsbericht zu senden. Weitere Informationen finden Sie unter Use cost allocation tags (Verwendung von Kostenzuordnungs-Tags) im AWS Fakturierung und Kostenmanagement -Benutzerhandbuch.
- Kontrollieren Sie den Zugriff auf Ihre AWS Ressourcen. Weitere Informationen finden Sie unter Steuern des Zugriffs mithilfe von Tags.

Unterstützte Ressourcen für das Tagging in Amazon Braket

Der folgende Ressourcentyp in Amazon Braket unterstützt Tagging:

- `quantum-task`-Ressource
- Name der Ressource: `AWS::Service::Braket`
- ARM-Regex: `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

Hinweis: Sie können Tags für Ihre Amazon Braket-Notizbücher in der Amazon Braket-Konsole anwenden und verwalten, indem Sie die Konsole verwenden, um zur Notizbuchressource zu navigieren, obwohl es sich bei den Notizbüchern tatsächlich um Amazon SageMaker AI-Ressourcen handelt. Weitere Informationen finden Sie in der Dokumentation unter Notebook-Instanz-Metadaten. SageMaker

Markierung mit der Amazon Braket API

- Wenn Sie Amazon Braket verwenden, API um Tags für eine Ressource einzurichten, rufen Sie die `TagResource`APIan.

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

- Um Tags aus einer Ressource zu entfernen, rufen Sie den auf `UntagResource`API.

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

- Um alle Tags aufzulisten, die mit einer bestimmten Ressource verknüpft sind, rufen Sie den auf `ListTagsForResource`API.

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

Tagging-Einschränkungen

Die folgenden grundlegenden Einschränkungen gelten für Tags in Amazon Braket-Ressourcen:

- Maximale Anzahl von Tags, die Sie einer Ressource zuweisen können: 50
- Maximale Schlüssellänge: 128 Unicode-Zeichen
- Maximale Wertlänge: 256 Unicode-Zeichen
- Gültige Zeichen für Schlüssel und Wert: `a-z, A-Z, 0-9, space` und diese Zeichen: `_ . : / = + -` und `@`
- Bei Schlüsseln und Werten wird die Groß-/Kleinschreibung berücksichtigt.
- Nicht `aws` als Präfix für Schlüssel verwenden; es ist für die AWS Verwendung reserviert.

Verwaltung von Tags in Amazon Braket

Sie legen Tags als Eigenschaften für eine Ressource fest. Sie können Tags über die Amazon Braket-Konsole, Amazon Braket oder die anzeigen, hinzufügen, ändernAPI, auflisten und löschen. AWS CLI Weitere Informationen finden Sie in der Amazon Braket-API-Referenz.

In diesem Abschnitt:

- Hinzufügen von -Tags
- Anzeigen von Tags
- Tags bearbeiten
- Entfernen von Tags

Hinzufügen von -Tags

Sie können zu folgenden Zeiten Tags zu taggbaren Ressourcen hinzufügen:

- Wenn Sie die Ressource erstellen: Verwenden Sie die Konsole oder fügen Sie den `Tags` Parameter in die `Create` Operation in die AWS API ein.
- Nachdem Sie die Ressource erstellt haben: Verwenden Sie die Konsole, um zur Quantentask- oder Notebook-Ressource zu navigieren, oder rufen Sie den `TagResource` Vorgang in der AWS API auf.

Um einer Ressource bei der Erstellung Tags hinzuzufügen, benötigen Sie außerdem die Berechtigung, eine Ressource des angegebenen Typs zu erstellen.

Anzeigen von Tags

Sie können die Tags auf allen markierbaren Ressourcen in Amazon Braket anzeigen, indem Sie die Konsole verwenden, um zu der Aufgaben- oder Notizbuchressource zu navigieren, oder indem Sie den Vorgang aufrufen. AWS `ListTagsForResource` API

Sie können den folgenden AWS API Befehl verwenden, um Tags auf einer Ressource anzuzeigen:

- AWS API: `ListTagsForResource`

Tags bearbeiten

Sie können Tags bearbeiten, indem Sie die Konsole verwenden, um zur Quantum-Aufgabe- oder Notizbuchressource zu navigieren, oder Sie können den folgenden Befehl verwenden, um den Wert für ein Tag zu ändern, das an eine taggable Ressource angehängt ist. Wenn Sie einen Tag-Schlüssel angeben, der bereits existiert, wird der Wert für diesen Schlüssel überschrieben:

- AWS API: `TagResource`

Entfernen von Tags

Sie können Tags aus einer Ressource entfernen, indem Sie die zu entfernenden Schlüssel angeben, indem Sie die Konsole verwenden, um zur Quantentask- oder Notebook-Ressource zu navigieren, oder wenn Sie den `UntagResource` Vorgang aufrufen.

- AWS API: `UntagResource`

Beispiel für AWS CLI Tagging in Amazon Braket

Wenn Sie mit dem AWS Command Line Interface (AWS CLI) arbeiten, um mit Amazon Braket zu interagieren, ist der folgende Code ein Beispielbefehl, der zeigt, wie Sie ein Tag erstellen, das für eine von Ihnen erstellte Quantenaufgabe gilt. In diesem Beispiel wird die Aufgabe auf dem SV1 Quantensimulator ausgeführt, wobei die Parametereinstellungen für die Rigetti Quantenverarbeitungseinheit (QPU) spezifiziert sind. Es ist wichtig, dass im Beispielbefehl das Tag ganz am Ende angegeben wird, nach allen anderen erforderlichen Parametern. In diesem Fall hat das Tag einen Schlüssel von `state` und einen Wert von`Washington`. Diese Tags könnten verwendet werden, um diese spezielle Quantenaufgabe zu kategorisieren oder zu identifizieren.

```
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\"}
```

Dieses Beispiel zeigt, wie Sie Tags auf Ihre Quantenaufgaben anwenden können AWS CLI, wenn Sie diese ausführen. Dies ist hilfreich für die Organisation und Nachverfolgung Ihrer Braket-Ressourcen.

Überwachen Sie Ihre Quantenaufgaben mit EventBridge

Amazon EventBridge überwacht Statusänderungsereignisse in Amazon Braket-Quantenaufgaben. Ereignisse von Amazon Braket werden fast in Echtzeit EventBridge zugestellt. Sie können Regeln schreiben, die angeben, welche Ereignisse Sie interessieren, einschließlich automatisierter Aktionen, die ergriffen werden, wenn ein Ereignis einer Regel entspricht. Zu den automatischen Aktionen, die ausgelöst werden können, gehören:

- Eine AWS Lambda Funktion aufrufen
- Aktivierung einer AWS Step Functions Zustandsmaschine
- Benachrichtigen eines Amazon SNS-Themas

EventBridge überwacht diese Amazon Braket-Statusänderungsereignisse:

- Der Status der Quantenaufgabe ändert sich

Amazon Braket garantiert die Lieferung von Ereignissen zur Änderung des Status von Quantenaufgaben. Diese Ereignisse werden mindestens einmal zugestellt, aber möglicherweise nicht in der richtigen Reihenfolge.

Weitere Informationen finden Sie unter Events in Amazon EventBridge.

In diesem Abschnitt:

- Überwachen Sie den Status von Quantenaufgaben mit EventBridge
- Beispiel für eine Amazon EventBridge Braket-Veranstaltung

Überwachen Sie den Status von Quantenaufgaben mit EventBridge

Mit können Sie Regeln erstellen EventBridge, die Aktionen definieren, die ergriffen werden sollen, wenn Amazon Braket eine Benachrichtigung über eine Statusänderung in Bezug auf eine Braket-Quantenaufgabe sendet. Sie können beispielsweise eine Regel erstellen, die Ihnen jedes Mal eine E-Mail-Nachricht sendet, wenn sich der Status einer Quantenaufgabe ändert.

1. Melden Sie sich AWS mit einem Konto an, das über Nutzungsberechtigungen EventBridge und Amazon Braket verfügt.
2. Öffnen Sie die EventBridge Amazon-Konsole.
3. Erstellen Sie mit den folgenden Werten eine EventBridge Regel:
   - Bei Regeltyp wählen Sie Regel mit einem Ereignismuster aus.
   - Wählen Sie für Event source (Ereignisquelle) Other (Andere) aus.
   - Wählen Sie im Abschnitt Ereignismuster die Option Benutzerdefinierte Muster (JSON-Editor) aus, und fügen Sie dann das folgende Ereignismuster in den Textbereich ein:

     ```
     {
       "source": [
         "aws.braket"
       ],
       "detail-type": [
         "Braket Task State Change"
       ]
     }
     ```

     Um alle Ereignisse aus Amazon Braket zu erfassen, schließen Sie den `detail-type` Abschnitt aus, wie im folgenden Code gezeigt:

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

   - Wählen AWS-ServiceSie für Zieltypen und für Ziel auswählen ein Ziel aus, z. B. ein Amazon SNS-Thema oder eine Amazon AWS Lambda SNS-Funktion. Das Ziel wird ausgelöst, wenn ein Ereignis zur Änderung des Status einer Quantenaufgabe von Amazon Braket empfangen wird.

Verwenden Sie beispielsweise ein Amazon Simple Notification Service (SNS) -Thema, um eine E-Mail oder Textnachricht zu senden, wenn ein Ereignis eintritt. Erstellen Sie dazu zunächst mit der Amazon SNS SNS-Konsole ein Amazon SNS SNS-Thema. Weitere Informationen finden Sie unter Verwenden von Amazon SNS für Benutzerbenachrichtigungen.

Einzelheiten zum Erstellen von Regeln finden Sie unter EventBridge Amazon-Regeln erstellen, die auf Ereignisse reagieren.

Beispiel für eine Amazon EventBridge Braket-Veranstaltung

Informationen zu den Feldern für ein Amazon Braket Quantum Task Status Change-Ereignis finden Sie unter Ereignisse in Amazon EventBridge.

Die folgenden Attribute werden im JSON-Feld „Detail“ angezeigt.

- **`quantumTaskArn`**(str): Die Quantenaufgabe, für die dieses Ereignis generiert wurde.
- **`status`**(Optional [str]): Der Status, in den die Quantenaufgabe übergegangen ist.
- **`deviceArn`**(str): Das vom Benutzer angegebene Gerät, für das diese Quantenaufgabe erstellt wurde.
- shots(int): Die Anzahl der vom Benutzer shots angeforderten.
- **`outputS3Bucket`**(str): Der vom Benutzer angegebene Ausgabe-Bucket.
- **`outputS3Directory`**(str): Das vom Benutzer angegebene Ausgabeschlüsselpräfix.
- **`createdAt`**(str): Die Erstellungszeit der Quantenaufgabe als ISO-8601-Zeichenfolge.
- **`endedAt`**(Optional [str]): Der Zeitpunkt, zu dem die Quantenaufgabe einen Endzustand erreicht hat. Dieses Feld ist nur vorhanden, wenn die Quantenaufgabe in einen Endzustand übergegangen ist.

Der folgende JSON-Code zeigt ein Beispiel für ein Amazon Braket Quantum Task Status Change-Ereignis.

```
{
    "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"
      }
}
```


Überwachen Sie Ihre Metriken mit CloudWatch

Sie können Amazon Braket mithilfe von Amazon überwachen. Amazon CloudWatch sammelt Rohdaten und verarbeitet sie zu lesbaren Metriken, die nahezu in Echtzeit ablaufen. Sie können historische Informationen, die vor bis zu 15 Monaten generiert wurden, oder Suchkennzahlen, die in den letzten zwei Wochen aktualisiert wurden, in der CloudWatch Amazon-Konsole anzeigen, um einen besseren Überblick über die Leistung von Amazon Braket zu erhalten. Weitere Informationen finden Sie unter CloudWatch Metriken verwenden.

> **Note**
>
> Sie können die CloudWatch Log-Streams für Amazon Braket-Notebooks anzeigen, indem Sie auf der Amazon SageMaker AI-Konsole zur Notebook-Detailseite navigieren. Zusätzliche Amazon Braket-Notebook-Einstellungen sind über die SageMaker Konsole verfügbar.

In diesem Abschnitt:

- Amazon Braket-Metriken und -Dimensionen

Amazon Braket-Metriken und -Dimensionen

Metriken sind das grundlegende Konzept von. CloudWatch Eine Metrik stellt einen zeitlich geordneten Satz von Datenpunkten dar, die veröffentlicht werden. CloudWatch Jede Metrik ist durch eine Reihe von Dimensionen gekennzeichnet. Weitere Informationen zu den Dimensionen von Metriken finden Sie unter CloudWatch Dimensionen. CloudWatch

Amazon Braket sendet die folgenden, für Amazon Braket spezifischen Metrikdaten an die CloudWatch Amazon-Metriken:

Metriken für Quantenaufgaben

Metriken sind verfügbar, wenn Quantenaufgaben existieren. Sie werden in der Konsole unter AWS/Braket/By Device angezeigt. CloudWatch

Metrik	Description
Anzahl	Anzahl der Quantenaufgaben.
Latenz	Diese Metrik wird ausgegeben, wenn eine Quantenaufgabe abgeschlossen ist. Sie stellt die Gesamtzeit von der Initialisierung bis zur Fertigstellung einer Quantenaufgabe dar.

Dimensionen für Quanten-Task-Metriken

Die Quanten-Task-Metriken werden mit einer Dimension veröffentlicht, die auf dem `deviceArn` Parameter basiert und die Form arn:aws:braket: ::device/xxx hat.

Protokollieren Sie Ihre Quantenaufgaben mit CloudTrail

Amazon Braket ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS-Service in Amazon Braket ausgeführt wurden. CloudTrail erfasst alle API Aufrufe von Amazon Braket als Ereignisse. Zu den erfassten Anrufen gehören Anrufe von der Amazon Braket-Konsole und Code-Aufrufe an den Amazon Braket-Betrieb. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Übermittlung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für Amazon Braket. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem

in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Amazon Braket gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

In diesem Abschnitt:

- Informationen zu Amazon Braket in CloudTrail
- Grundlegendes zu Amazon Braket-Protokolldateieinträgen

Informationen zu Amazon Braket in CloudTrail

CloudTrail ist auf Ihrem aktiviert AWS-Konto , wenn Sie das Konto erstellen. Wenn in Amazon Braket eine Aktivität auftritt, wird diese Aktivität zusammen mit anderen AWS-Service Ereignissen im CloudTrail Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS-Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich Ereignissen für Amazon Braket, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole anlegen, gilt dieser für alle AWS-Regionen-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere konfigurieren, AWS-Services um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail Unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Alle Amazon Braket-Aktionen werden von CloudTrail protokolliert. Beispielsweise generieren Aufrufe von `GetQuantumTask` oder `GetDevice` -Aktionen Einträge in den CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anforderung aus einem anderen AWS-Service gesendet wurde.

Weitere Informationen finden Sie unter CloudTrail userIdentity-Element.

Grundlegendes zu Amazon Braket-Protokolldateieinträgen

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel ist ein Protokolleintrag für die `GetQuantumTask` Aktion, der die Details einer Quantenaufgabe abruft.


```
{
  "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"
}
```

Das Folgende zeigt einen Protokolleintrag für die `GetDevice` Aktion, der die Details eines Geräteereignisses zurückgibt.

```
{
  "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"
}
```

Erweiterte Protokollierung mit Amazon Braket

Sie können den gesamten Prozess der Aufgabenverarbeitung mit einem Logger aufzeichnen. Diese fortgeschrittenen Protokollierungstechniken ermöglichen es Ihnen, die Hintergrundabfrage zu verfolgen und einen Datensatz für das spätere Debuggen zu erstellen.

Um den Logger zu verwenden, empfehlen wir, die `poll_interval_seconds` Parameter `poll_timeout_seconds` und zu ändern, sodass eine Quantenaufgabe lange andauern kann und der Status der Quantenaufgabe kontinuierlich protokolliert und die Ergebnisse in einer Datei gespeichert werden. Sie können diesen Code in ein Python-Skript statt in ein Jupyter-Notebook übertragen, sodass das Skript als Prozess im Hintergrund ausgeführt werden kann.

Konfigurieren Sie den Logger

Konfigurieren Sie zunächst den Logger so, dass alle Protokolle automatisch in eine Textdatei geschrieben werden, wie in den folgenden Beispielzeilen gezeigt.

```
# 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
```

Erstellen Sie die Schaltung und führen Sie sie aus

Jetzt können Sie eine Schaltung erstellen, sie zur Ausführung an ein Gerät senden und sehen, was passiert, wie in diesem Beispiel gezeigt.

```
# 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
)
```

Prüfen Sie die Protokolldatei

Sie können überprüfen, was in die Datei geschrieben wurde, indem Sie den folgenden Befehl eingeben.

```
# 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})
```

Holen Sie sich den ARN aus der Protokolldatei

Aus der zurückgegebenen Protokolldateiausgabe können Sie, wie im vorherigen Beispiel gezeigt, die ARN-Informationen abrufen. Mit der ARN-ID können Sie das Ergebnis der abgeschlossenen Quantenaufgabe abrufen.

```
# 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()
```

Amazon Braket-Kontingente

In der folgenden Tabelle sind die Servicekontingente für Amazon Braket aufgeführt. Service Quotas, auch als Limits bezeichnet, sind die maximale Anzahl von Serviceressourcen oder -vorgängen für Ihr AWS-Konto.

Einige Kontingente können erhöht werden. Weitere Informationen finden Sie unter AWS-Service - Kontingente.

- Die Burst-Rate-Kontingente können nicht erhöht werden.
- Die maximale Erhöhung der Rate für einstellbare Kontingente (mit Ausnahme der Burst-Rate, die nicht angepasst werden kann) beträgt das Zweifache des angegebenen Standardratenlimits. Beispielsweise kann ein Standardkontingent von 60 auf ein Maximum von 120 angepasst werden.
- Das einstellbare Kontingent für gleichzeitige SV1 (DM1) Quantenaufgaben erlaubt ein Maximum von 60 pro AWS-Region.
- Die maximal zulässige Anzahl von Recheninstanzen für einen Hybrid-Job ist 1, und die Kontingente sind anpassbar.

Ressource	Description	Einschränkungen	Einstellbar
Rate der API Anfragen	Die maximale Anzahl von Anforderungen pro Sekunde, die Sie in diesem Konto in der aktuellen Region senden können.	140	Ja
Burst-Rate der API Anfragen	Die maximale Anzahl von zusätzlichen Anforderungen pro Sekunde (RPS), die Sie in einem Burst in diesem Konto in der aktuellen Region senden können.	600	Nein

Ressource	Description	Einschränkungen	Einstellbar
Rate der `CreateQuantumTask` Anfragen	Die maximale Anzahl von `CreateQuantumTask` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	20 pro Sekunde	Ja
Burst-Rate der `CreateQuantumTask` Anfragen	Die maximale Anzahl zusätzlicher `CreateQuantumTask` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	40	Nein
Rate der Anfragen `SearchQuantumTasks`	Die maximale Anzahl von `SearchQuantumTasks` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5 pro Sekunde	Ja

Ressource	Description	Einschränkungen	Einstellbar
Burst-Rate der `SearchQuantumTasks` Anfragen	Die maximale Anzahl zusätzlicher `SearchQuantumTasks` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	50	Nein
Rate der Anfragen `GetQuantumTask`	Die maximale Anzahl von `GetQuantumTask` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	100 pro Sekunde	Ja
Burst-Rate der `GetQuantumTask` Anfragen	Die maximale Anzahl zusätzlicher `GetQuantumTask` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	500	Nein

Ressource	Description	Einschränkungen	Einstellbar
Rate der Anfragen `CancelQuantumTask`	Die maximale Anzahl von `CancelQuantumTask` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	2 pro Sekunde	Ja
Burst-Rate der `CancelQuantumTask` Anfragen	Die maximale Anzahl zusätzlicher `CancelQuantumTask` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	20	Nein
Rate der Anfragen `GetDevice`	Die maximale Anzahl von `GetDevice` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5 pro Sekunde	Ja
Burst-Rate der `GetDevice` Anfragen	Die maximale Anzahl zusätzlicher `GetDevice` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	50	Nein

Ressource	Description	Einschränkungen	Einstellbar
Rate der Anfragen `SearchDevices`	Die maximale Anzahl von `SearchDevices` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5 pro Sekunde	Ja
Burst-Rate der `SearchDevices` Anfragen	Die maximale Anzahl zusätzlicher `SearchDevices` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	50	Nein
Rate der Anfragen `CreateJob`	Die maximale Anzahl von `CreateJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	1 pro Sekunde	Ja
Burst-Rate der `CreateJob` Anfragen	Die maximale Anzahl zusätzlicher `CreateJob` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	5	Nein

Ressource	Description	Einschränkungen	Einstellbar
Rate der Anfragen `SearchJobs`	Die maximale Anzahl von `SearchJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5 pro Sekunde	Ja
Burst-Rate der `SearchJobs` Anfragen	Die maximale Anzahl zusätzlic her `SearchJob` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	50	Nein
Rate der Anfragen `GetJob`	Die maximale Anzahl von `GetJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5 pro Sekunde	Ja
Burst-Rate der `GetJob` Anfragen	Die maximale Anzahl zusätzlicher `GetJob` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	25	Nein

Ressource	Description	Einschränkungen	Einstellbar
Rate der Anfragen `CancelJob`	Die maximale Anzahl von `CancelJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	2 pro Sekunde	Ja
Burst-Rate der `CancelJob` Anfragen	Die maximale Anzahl zusätzlic her `CancelJob` Anfragen pro Sekunde (RPS), die Sie in einem Burst auf diesem Konto in der aktuellen Region senden können.	5	Nein
Anzahl gleichzeitiger Quantenaufgaben SV1	Die maximale Anzahl gleichzeitiger Quantenaufgaben, die auf dem Zustandsv ektorsimulator (SV1) in der aktuellen Region ausgeführt werden.	100 US-Ost-1, 50 US-West1, 100 US-West-2, 50 eu-west-2	Nein
Anzahl gleichzeitiger Quantenaufgaben DM1	Die maximale Anzahl gleichzeitiger Quantenaufgaben, die auf dem Dichtemat rixsimulator (DM1) in der aktuellen Region ausgeführt werden.	100 US-Ost-1, 50 US-West1, 100 US-West-2, 50 eu-west-2	Nein

Ressource	Description	Einschränkungen	Einstellbar
Anzahl gleichzeitiger Quantenaufgaben TN1	Die maximale Anzahl gleichzeitiger Quantenaufgaben, die auf dem Tensor-Netzwerksimulator (TN1) in der aktuellen Region ausgeführt werden.	10 US-Ost-1, 10 US-West-2, 5 eu-west-2,	Ja
Anzahl gleichzeitiger hybrider Jobs	Die maximale Anzahl gleichzeitiger Hybridjobs in der aktuellen Region.	3	Ja
Laufzeitlimit für hybride Jobs	Die maximale Zeit in Tagen, während der ein Hybridjob ausgeführt werden kann.	5	Nein

Im Folgenden sind die klassischen Standardkontingente für Recheninstanzen für Hybrid-Jobs aufgeführt. Um diese Kontingente zu erhöhen, wenden Sie sich an Support. Darüber hinaus werden die verfügbaren Regionen für jede Instanz angegeben.

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c4.xlarge für	Die maximal zulässige Anzahl von Instances des Typs	5	Ja	Ja	Ja	Ja	Ja	Nein

Ressourc	Descripti on	Einschrän kungen	Einstellb ar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Hybrid-Jobs	ml.c4.xla rge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.							
Maximale Anzahl von Instanzen von ml.c4.2xl arge für Hybridauf träge	Die maximal zulässige Anzahl von Instances des Typs ml.c4.2xl arge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c4.4xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c4.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c4.8xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c4.8xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5.xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5.4xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	1	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5.9xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.c5.9xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	1	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5.18xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.c5.18xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressourc	Descripti on	Einschrän kungen	Einstellb ar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5n.xl arge für Hybridauf träge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.xl arge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5n.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5n.4xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5n.9xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.9xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Nein	Nein

Ressourc	Descripti on	Einschrän kungen	Einstellb ar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.c5n.18 xlarge für Hybridauf träge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.18 xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.4xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.8xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.8xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.12xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.12xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g4dn.16xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.16xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g6.xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.2xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.2xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.4xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.4xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.8xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.8xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.12xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.12xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.16xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.16xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.24xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.24xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6.48xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6.48xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.g6e.xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.2xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.2xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.4xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.4xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.8xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.8xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.12xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.12xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.16xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.16xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.24xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.24xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instances von ml.g6e.48xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.g6e.48xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	1	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m4.xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m4.2xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m4.4xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	2	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m4.10› large für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.10› large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m4.16› large für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.16› large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.large für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressourc	Descripti on	Einschrän kungen	Einstellb ar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.xla rge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.xla rge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.2xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.4xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.4xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.12xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.12xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.m5.24› large für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.24› large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.p2.xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.p2.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.p2.8xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.p2.8xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.p2.16xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.p2.16xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.p4d.24xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.p4d.24xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja	Ja	Nein	Ja	Nein	Nein

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.t3.large für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.t3.large, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressource	Description	Einschränkungen	Einstellbar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.t3.xlarge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.t3.xlarge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Ja	Ja	Ja	Ja

Ressourc	Descripti on	Einschrän kungen	Einstellb ar	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Maximale Anzahl von Instanzen von ml.t3.2xl arge für Hybrid-Jobs	Die maximale Anzahl von Instances des Typs ml.t3.2xl arge, die für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region zulässig ist.	5	Ja	Ja	Ja	Ja	Ja	Ja

Limit-Aktualisierungen anfordern

Wenn Sie für einen Instance-Typ eine ServiceQuotaExceeded Ausnahme erhalten und nicht genügend Instances dafür verfügbar sind, können Sie auf der Seite Service Quotas in der AWS Konsole eine Limiterhöhung beantragen und unter AWS Services nach Amazon Braket suchen.

> **Note**
>
> Wenn Ihr Hybrid-Job die angeforderte ML-Rechenkapazität nicht bereitstellen kann, verwenden Sie eine andere Region. Wenn Sie eine Instanz in der Tabelle nicht sehen, ist sie außerdem nicht für Hybrid-Jobs verfügbar.

Zusätzliche Kontingente und Limits

- Die Amazon Braket-Quanten-Task-Aktion ist auf eine Größe von 5 MB begrenzt.
- Für SV1 beträgt die maximale Betriebsdauer 3 Stunden für Schaltungen mit bis zu 31 Qubits und 11 Stunden für Schaltungen über 31 Qubits.
- Die maximal zulässige Anzahl von Schüssen pro Aufgabe für Rigetti Geräte SV1DM1, und beträgt 50.000.
- Die maximal zulässige Anzahl von Schüssen pro Aufgabe TN1 beträgt 1000.
- Für AQT das IBEX-Q1 Gerät sind maximal 2000 Schüsse pro Aufgabe zulässig.
- Für alle IonQ Geräte gilt: Die Mindestanzahl von Schüssen pro Aufgabe beträgt 100. Bei Verwendung eines On-Demand-Modells gilt ein Limit von 1 Million Gateshots und ein Minimum von 2500 Schüssen für Aufgaben zur Fehlerminimierung. Bei einer direkten Reservierung gibt es kein Gateshot-Limit und ein Minimum von 500 Schüssen für Aufgaben zur Fehlerminimierung.
- Für QuEra das Aquila-Gerät liegt das Maximum bei 1.000 Schüssen pro Aufgabe.
- Für IQM Emerald Geräte Garnet und Geräte liegt das Maximum bei 20.000 Schüssen pro Aufgabe.
- Für TN1 und die QPU-Geräte müssen die Schüsse pro Aufgabe > 0 sein.

Dokumentenverlauf für den Amazon Braket Developer Guide

In der folgenden Tabelle werden die Dokumentationsversionen für Amazon Braket beschrieben.

- Neuestes API Referenz-Update: 20. November 2025
- Letzte Aktualisierung der Dokumentation: 14. Mai 2026

Änderung	Beschreibung	Date (Datum)
Neue Kontingente vom Typ Hybrid-Jobinstanz	Service-Kontingente für ml.g6ml.g6e, und ml.t3 Instance-Typen für Amazon Braket Hybrid Jobs hinzugefügt.	14. Mai 2026
Neues Gerät Rigetti Cepheus-1-108Q	Unterstützung für das Rigetti Cepheus-1-108Q Gerät hinzugefügt. Ein 108-Qubit-Gerät, das skalierbare Multi-Chip-Technologie verwendet.	7. April 2026
Außerbetriebnahme von IonQ-Geräten Aria-1	Die Unterstützung für das Aria-1 IonQ-Gerät wurde entfernt.	2. März 2026
Die Seiten „Mit Reservierungen arbeiten“ aktualisieren	Die Übersichtlichkeit der Seiten „Mit Reservierungen arbeiten“ wurde verbessert	3. Februar 2026
Support Python Version 3.12 für Braket-Notebooks und verwaltete Container	Unterstützung für Python Version 3.12 für Amazon Braket-Notebooks und verwaltete Container (Base, CUDA-Q PennyLane, und Tensorflow) hinzugefügt. Enthält eine Anleitung zur	21. Januar 2026

	Fehlerbehebung für das Python 3.12-Upgrade.	
P3-Beispiele entfernen	SageMaker stellt ihre `ml.p3` Instance-Familie ein. Beispiele wurden durch die empfohlene `ml.g4dn` Instance-Familie ersetzt.	19. Dezember 2025
Neue Funktion zum Ausgabenlimit	Unterstützung für die Amazon Braket-Ausgabenlimit-Funktion hinzugefügt, mit der optionale Budgetobergrenzen für einzelne QPUs festgelegt werden können, die Aufgaben, die den konfigurierten Ausgabengrenzwert überschreiten, automatisch validieren und ablehnen.	20. November 2025
Neues Braket-Gerät AQT IBEX-Q1	Unterstützung für ein AQT IBEX-Q1Gerät hinzugefügt. Dieses Gerät basiert auf einem Kristall aus $^{40}Ca^{+}$-Ionen in einer makroskopischen Hochfrequenzfalle, die sich in einer Ultrahochvakuumkammer befindet.	18. November 2025
Neue native Unterstützung für CUDA-Q NBIs auf Amazon Braket	Native Unterstützung für Notebook-Instances CUDA-Q in Amazon Braket wurde hinzugefügt. Weitere Informationen finden Sie unter CUDA-Q NBIs.	10. November 2025

IonQ Aria-2Außerbetrieb nahme von Geräten	Die Unterstützung für das IonQ Aria-2 Gerät wurde entfernt.	27. Oktober 2025
Konsolidierte Dokumentation zu Hybrid-Jobs	Die Abschnitte für Hybridjobs wurden so konsolidiert, dass sie unter Arbeiten mit Amazon Braket Hybrid-Jobs angezeigt werden.	21. Oktober 2025
Neuer von Braket bereitges tellter Behälter CUDA-Q	Unterstützung für einen bereitgestellten CUDA-Q Hybrid-Job-Container wurde hinzugefügt. Weitere Informationen finden Sie unter Definieren Sie die Umgebung für Ihr Algorithmus-Skript.	2. September 2025
Neue Emulatorfunktion für lokale Geräte	Es wurde Unterstützung für ein Emulator-Tool für lokale Quantengeräte hinzugefügt, mit dem Sie Ihre wörtliche n Programme emulieren können, bevor Sie sie an Quantengeräte senden.	25. August 2025
Pennylane und Seiten wurden in den Bereich Build verschoben CUDA-Q	Pennylane und die CUDA-QSeiten wurden so verschobe n, dass sie jetzt im Inhaltsve rzeichnis unter dem Abschnitt Build angezeigt werden.	15. August 2025

Neue Funktion `ProgramSet`	Unterstützung für Programmsätze hinzugefügt, eine Operation zum Ausführen mehrerer Quantenschaltkreise in einer einzigen Quantenaufgabe.	14. August 2025
Neues Gerät IQM Emerald	Unterstützung für das IQM Emerald Gerät hinzugefügt. Ein 54-Quibit-Gerät mit einer quadratischen (Kristall-) Gittertopologie.	21. Juli 2025
AmazonBraketServiceRolePolicyDie Richtlinie wurde aktualisiert	AmazonBraketServiceRolePolicybietet jetzt nur noch die Aktionen s3: * und logs: * für aws: PrincipalAccount. Dadurch wird der Zugriff nur auf die Buckets und Protokollgruppen des Anforderers beschränkt.	11. Juli 2025
Neue Funktion „Experimentelle Fähigkeiten“: Dynamische Schaltungen	Mid-circuit Mess- und Feed-Forward-Operationen sind als experimentelle Funktionen verfügbar, siehe Zugriff auf dynamische Schaltungen auf IQM-Geräten.	26. Juni 2025
Die Richtlinie wurde aktualisiert AmazonBraketFullAccess	AmazonBraketFullAccessbeinhaltet jetzt die Preisgestaltung: GetProducts zur Anzeige der Hardwarekosten auf der Konsole.	14. April 2025

Neues Gerät IonQ Forte-Enterprise-1	Unterstützung für das Forte-Enterprise-1 IonQ-Gerät hinzugefügt. Ein 36-Quibit-Gerät, das die Technologie „Trapped Ion“ nutzt.	17. März 2025
Die Berechtigungen für S3-Bedingungen wurden verbessert	Um die Sicherheit zu verbessern, bietet es `AmazonBraketFullAccess` jetzt nur noch `s3:*` Aktionen für `aws:PrincipalAccount`. Dadurch wird nur der Zugriff auf die eigenen Buckets des Anforderers eingeschränkt.	7. März 2025
Neues Gerät Rigetti Ankaa-3	Unterstützung für das Rigetti Ankaa-3 Gerät hinzugefügt. Ein 84-Quibit-Gerät, das skalierbare Multi-Chip-Technologie verwendet.	14. Januar 2025
Rigetti Ankaa-2Außerbetriebnahme von Geräten	Die Unterstützung für das Rigetti Ankaa-2 Gerät wurde entfernt.	14. Januar 2025
Unterstützung für IPv6-Datenverkehr	Amazon Braket unterstützt jetzt IPv6-Verkehr über den Dualstack-Endpunkt. `braket.{region}.api.aws`	12. Dezember 2024
Support für NVIDIA's CUDA-Qauf Amazon Braket	Kunden können jetzt Quantenprogramme mithilfe des NVIDIA's CUDA-Q Developer Frameworks auf Amazon Braket ausführen.	6. Dezember 2024

IonQ Forte-1Das Gerät ist sofort verfügbar	IonQ Forte-1Das Gerät ist nicht mehr nur für Reservier ungen verfügbar und steht unseren Kunden jetzt problemlos zur Verfügung.	22. November 2024
Rigetti Aspen-M-3Außerbetr iebnahme von Geräten	Die Unterstützung für das Rigetti Aspen-M-3 Gerät wurde entfernt.	27. September 2024
IonQ HarmonyAußerbetrie bnahme von Geräten	Die Unterstützung für das IonQ Harmony Gerät wurde entfernt.	29. August 2024
Neues Gerät Rigetti Ankaa-2	Unterstützung für das Rigetti Ankaa-2 Gerät hinzugefü gt. Ein 84-Quibit-Gerät, das skalierbare Multi-Chip-Technol ogie verwendet.	26. August 2024
Neuorganisation des Entwickle rhandbuchs	Das neue Entwicklerhandbuch nimmt die bestehende Build, Test, Run-Kundenreise auf und führt Benutzer auf diesem Weg mit Amazon Braket.	23. August 2024
OQC LucyAußerbetriebnahme von Geräten	Die Unterstützung für das OQC Lucy Gerät wurde entfernt.	28. Juni 2024
Neues Gerät IQM Garnet und neue Region Europe North 1	Unterstützung für das IQM Garnet-Gerät hinzugefügt. Ein 20-Qubit-Gerät mit einer quadratischen Gittertopologie. Erweiterung der von Braket unterstützten Regionen auf Europa Nord 1 (Stockholm).	22. Mai 2024

Lokale Verstimmung veröffentlicht	Zu den experimentellen Funktionen gehört jetzt die Funktion zur lokalen Feinabstimmung der Aquila QuEra QPU.	11. April 2024
Der Inaktivitätsmanager für Notebooks wurde veröffentlicht	Wenn Sie eine Notebook-Instanz erstellen, aktivieren Sie den Inaktivitätsmanager und legen Sie eine Leerlaufzeit fest, damit die Braket-Notebook-Instanz automatisch zurückgesetzt wird.	27. März 2024
Überarbeitung des Inhaltsverzeichnisses	Das Inhaltsverzeichnis von Amazon Braket wurde neu organisiert, um den Anforderungen des AWS Styleguides zu entsprechen und den Inhaltsfluss für das Kundenerlebnis zu verbessern.	12. Dezember 2023
Braket Direct veröffentlicht	Unterstützung für Braket Direct-Funktionen hinzugefügt, darunter: • Mit Reservierungen arbeiten • Expertenrat einholen • Erkunden Sie experimentelle Möglichkeiten	27. November 2023
Erstellen Sie eine Amazon Braket-Notebook-Instance aktualisiert	Die Dokumentation wurde aktualisiert, um Informationen zur Erstellung einer Notebook-Instance für neue und bestehende Amazon Braket-Kunden hinzuzufügen.	27. November 2023

Verwendung Ihres eigenen Containers (BYOC) aktualisiert	Die Dokumentation wurde aktualisiert und enthält nun Informationen darüber, wann BYOC verwendet wird, wie das Rezept für BYOC ist und wie Braket-Hybrid-Jobs auf dem Container ausgeführt werden.	18. Oktober 2023
Hybrid Jobs Decorator veröffentlicht	Führen Sie Ihren lokalen Code als Hybrid-Job ausSeite hinzugefügt. Enthält Beispiele: • Erstellen Sie einen Hybrid-Job aus lokalem Python-Code • Installieren Sie zusätzliche Python-Pakete und Quellcode • Speichern und laden Sie Daten in eine Hybrid-Job-Instanz • Bewährte Methoden für hybride Jobdekorateure	16. Oktober 2023
Sichtbarkeit der Warteschlange wurde hinzugefügt	Die Dokumentation im Developer's Guide wurde um queue depth und aktualisiertqueue position. Die API-Dokumentation wurde aktualisiert, um die neuen API-Änderungen im Hinblick auf die Sichtbarkeit von Warteschlangen widerzuspiegeln.	25. September 2023

Standardisieren Sie die Benennung in der Dokumentation	Die Dokumentation wurde aktualisiert, um alle Instanzen von „Job“ in „Hybrid-Job“ und „Task“ in „Quantenaufgabe“ umzuwandeln	11. September 2023
Neues Gerät IonQ Aria 2	Unterstützung für das IonQ Aria 2 Gerät hinzugefügt	8. September 2023
Native Gates wurde aktualisiert	Die Dokumentation wurde aktualisiert, um Informationen über den programmatischen Zugriff auf native Gates von Rigetti hinzuzufügen.	16. August 2023
XanaduAbfahrt	Die Dokumentation wurde aktualisiert, um alle Xanadu Geräte zu entfernen	02. Juni 2023
Neues Gerät IonQ Aria	Unterstützung für das IonQ Aria Gerät hinzugefügt	16. Mai 2023
RigettiGerät im Ruhestand	Der Support für wurde eingestellt Rigetti Aspen-M-2	2. Mai 2023
Aktualisierte AmazonBraketFullAccessRichtlinieninformationen	Das Skript, das den Inhalt der AmazonBraketFullAccessRichtlinie definiert, wurde aktualisiert und umfasst nun die GetMetricData Aktionen servicequotas: GetServiceQuota und cloudwatch: sowie Informationen zu Einschränkungen in Bezug auf Kontingente.	19. April 2023

Einführung von Guided Journeys	Die Dokumentation wurde geändert, um die aktuellere und einfachere Methode für das Braket-Onboarding widerzuspiegeln.	5. April 2023
Neues Gerät Rigetti Aspen-M-3	Unterstützung für das Rigetti Aspen-M-3 Gerät hinzugefügt	17. Januar 2023
Neue Funktion für adjungierte Farbverläufe	Es wurden Informationen über die Funktion „Adjungierter Farbverlauf“ hinzugefügt, die von angeboten wird SV1	7. Dezember 2022
Neue Funktion zur Algorithmus-Bibliothek	Es wurden Informationen zur Braket-Algorithmusbibliothek hinzugefügt, die einen Katalog vorgefertigter Quantenalgorithmen enthält	28. November 2022
D-WaveAbfahrt	Die Dokumentation wurde aktualisiert, um das Entfernen aller D-Wave Geräte zu ermöglichen	17. November 2022
Neues Gerät QuEra Aquila	Unterstützung für das QuEra Aquila Gerät hinzugefügt	31. Oktober 2022
Support für Braket Pulse	Es wurde Unterstützung für Braket Pulse hinzugefügt, wodurch die Impulssteuerung auf Rigetti allen Geräten verwendet werden kann OQC	20. Oktober 2022
Support für native IonQ-Gates	Unterstützung für das vom IonQ-Gerät angebotene native Gate-Set wurde hinzugefügt	13. September 2022

Neue Instanzkontingente	Die standardmäßigen klassischen Compute-Instance-Kontingente für Hybrid-Jobs wurden aktualisiert	22. August 2022
Neues Service-Dashboard	Die Screenshots der Konsole wurden aktualisiert und enthalten nun auch das Service-Dashboard	17. August 2022
Neues Gerät Rigetti Aspen-M-2	Unterstützung für das Rigetti Aspen-M-2 Gerät hinzugefügt	12. August 2022
Neue OpenQASM-Funktionen	Unterstützung für OpenQASM-Funktionen für die lokalen Simulatoren (braket_sv und braket_dm) hinzugefügt	04. August 2022
Neue Verfahren zur Kostenverfolgung	Es wurde hinzugefügt, wie maximale Kostenschätzungen für Simulatoren und Hardware-Workloads nahezu in Echtzeit abgerufen werden können	18. Juli 2022
Neues Gerät Xanadu Borealis	Unterstützung für das Xanadu Borealis Gerät hinzugefügt	2. Juni 2022
Neue Verfahren zur Vereinfachung des Onboardings	Es wurden Informationen zur Funktionsweise der neuen und vereinfachten Onboarding-Verfahren hinzugefügt	16. Mai 2022
Neues Gerät D-Wave Advantage_system6.1	Unterstützung für das D-Wave Advantage_system6.1 Gerät hinzugefügt	12. Mai 2022

Support für eingebettete Simulatoren	Es wurde hinzugefügt, wie eingebettete Simulationen mit Hybridjobs ausgeführt werden und wie der PennyLane Blitzsimulator verwendet wird	4. Mai 2022
AmazonBraketFullAccess - Vollständige Zugriffsrichtlinie für Amazon Braket	s3: ListAllMyBuckets Berechtigungen hinzugefügt, die es Benutzern ermöglichen, die für Amazon Braket erstellten und verwendeten Buckets anzusehen und zu überprüfen	31. März 2022
Support für OpenQASM	OpenQASM 3.0-Unterstützung für Gate-basierte Quantengeräte und Simulatoren hinzugefügt	7. März 2022
Neuer Quantenhardwareanbieter Oxford Quantum Circuits und neue Region, eu-west-2	Unterstützung für OQC und eu-west-2 hinzugefügt	28. Februar 2022
Neues Gerät Rigetti	Unterstützung für Rigetti Aspen M-1 hinzugefügt	15. Februar 2022
Neue Ressourcenlimits	Die maximale Anzahl gleichzeitiger SV1 Aufgaben wurde von 55 auf 100 erhöht DM1	5. Januar 2022
Neues Gerät Rigetti	Unterstützung für Rigetti Aspen-11 hinzugefügt	20. Dezember 2021
RigettiGerät im Ruhestand	Die Unterstützung für Rigetti Aspen-10 das Gerät wurde eingestellt	20. Dezember 2021

Neuer Ergebnistyp	Der Ergebnistyp mit reduzierter Dichte wird vom Simulator und DM1 Geräten mit lokaler Dichtematrix unterstützt	20. Dezember 2021
Die Beschreibung der Richtlinie wurde aktualisiert	Amazon Braket hat den Rollen-ARN aktualisiert, sodass er den `servicerole/` Pfad enthält. Informationen zu Richtlinienaktualisierungen finden Sie in der Tabelle Amazon Braket-Aktualisierungen für AWS verwaltete Richtlinien.	29. November 2021
Stellenangebote bei Amazon Braket	Benutzerhandbuch für Amazon Braket Hybrid Jobs und hinzugefügt API	29. November 2021
Neues Gerät Rigetti	Unterstützung für Rigetti Aspen-10 hinzugefügt	20. November 2021
D-WaveGerät im Ruhestand	Die Unterstützung für D-Wave QPU wurde eingestellt, Advantage_system1	4. November 2021
Neues Gerät D-Wave	Unterstützung für eine zusätzliche D-Wave QPU hinzugefügt, Advantage_system4	5. Oktober 2021
Neue Geräuschsimulatoren	Unterstützung für einen Dichtematrixsimulator (DM1), der Schaltungen von bis zu 17 simulieren kann, qubits und für einen lokalen Geräuschsimulator braket_dm hinzugefügt	25. Mai 2021

PennyLane Unterstützung	Unterstützung für PennyLane auf Amazon Braket hinzugefügt	08. Dezember 2020
Neuer Simulator	Unterstützung für einen Tensor-Netzwerksimulator (TN1) hinzugefügt, der größere Schaltungen ermöglicht	08. Dezember 2020
Batching von Aufgaben	Braket unterstützt das Batching von Kundenaufgaben	24. November 2020
Manuelle Zuordnung qubit	Braket unterstützt die manuelle qubit Zuordnung auf dem Rigetti Gerät	24. November 2020
Anpassbare Kontingente	Braket unterstützt per Self-Service einstellbare Kontingente für Ihre Aufgabenressourcen	30. Oktober 2020
Support für PrivateLink	Sie können private VPC-Endpunkte für Ihre Braket-Jobs einrichten	30. Oktober 2020
Unterstützung für Tags	Braket unterstützt API basierte Tags für die Quantum-Task-Ressource	30. Oktober 2020
Neues Gerät D-Wave	Unterstützung für eine zusätzliche D-Wave QPU hinzugefügt, Advantage_system1	29. September 2020
Erstversion	Erste Version der Amazon Braket-Dokumentation	12. August 2020

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.