aws

Entwicklerhandbuch

Amazon Braket

Amazon Braket: Entwicklerhandbuch

Die Handelsmarken und 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 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

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 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.

Was ist Amazon Braket?

AmazonBraket ist ein vollständig verwaltetes ProgrammAWS-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 Braket-Entwicklung besteht aus drei Phasen: Erstellen, Testen und Ausführen:

Build — Braket bietet vollständig verwaltete Jupyter-Notebook-Umgebungen, die den Einstieg erleichtern. Auf Braket-Notebooks sind Beispielalgorithmen, Ressourcen und Entwicklertools, einschließlich des Braket-SDK, vorinstalliert. Amazon Mit dem Amazon Braket-SDK können Sie Quantenalgorithmen erstellen und sie 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 vonIonQ, und OQCRigetti, 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.

Begriffe und Konzepte von Amazon Braket

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.

Hybrid-Job in Braket

AmazonBraket verfügt über eine Funktion namens Amazon Braket Hybrid Jobs, die vollständig verwaltete Ausführungen hybrider Algorithmen ermöglicht. Ein Braket-Hybridjob 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.

Gate-basiertes Quantencomputing

Beim Gate-basierten Quantencomputing (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.

Hamiltonisch

Die Quantendynamik eines physikalischen Systems wird durch seinen Hamilton-Operator bestimmt, der alle Informationen über die Wechselwirkungen zwischen Systembestandteilen und die Auswirkungen exogener Antriebskräfte kodiert. Der Hamilton-Operator eines N-Qubit-Systems wird auf klassischen Maschinen üblicherweise als eine $2^{N \text{ mal } 2^N}$ 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 über benutzerdefinierte Wellenformen aktiviert, 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 reversible 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 das 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 vom QPU-Gerät akzeptierten Gates. Diese Gates können möglicherweise nicht direkt auf der QPU ausgeführt werden, 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 dershots) und andere Anforderungsmetadaten. Sie können Quantenaufgaben über das Amazon Braket-SDK oder direkt mithilfe der Operation erstellen. CreateQuantumTask API 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 Operationen GetQuantumTask oder SearchQuantumTasks API einsehen.

Qubit

Die grundlegende Informationseinheit in einem Quantencomputer wird als qubit (Quantenbit) bezeichnet, ähnlich wie ein Bit in der klassischen Datenverarbeitung. A qubit ist ein Quantensystem mit zwei Ebenen, 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 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. 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.

AWSTerminologie 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 erhält. 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 und Benutzerrichtlinien in der Amazon S3 S3-Dokumentation.

Amazon Braket-Preise

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. Um mehr über die Preisgestaltung zu erfahren, besuchen Sie bitte unsere Preisseite.

Kostenverfolgung nahezu in Echtzeit

Das Braket SDK bietet Ihnen die Möglichkeit, Ihren Quanten-Workloads eine Kostenverfolgung nahezu in Echtzeit hinzuzufügen. 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)
```

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.

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

tracker.simulator_tasks_cost()
$0.00375
```

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

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

> **Note**
>
> 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 folgendeAWS 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 QHPs, 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

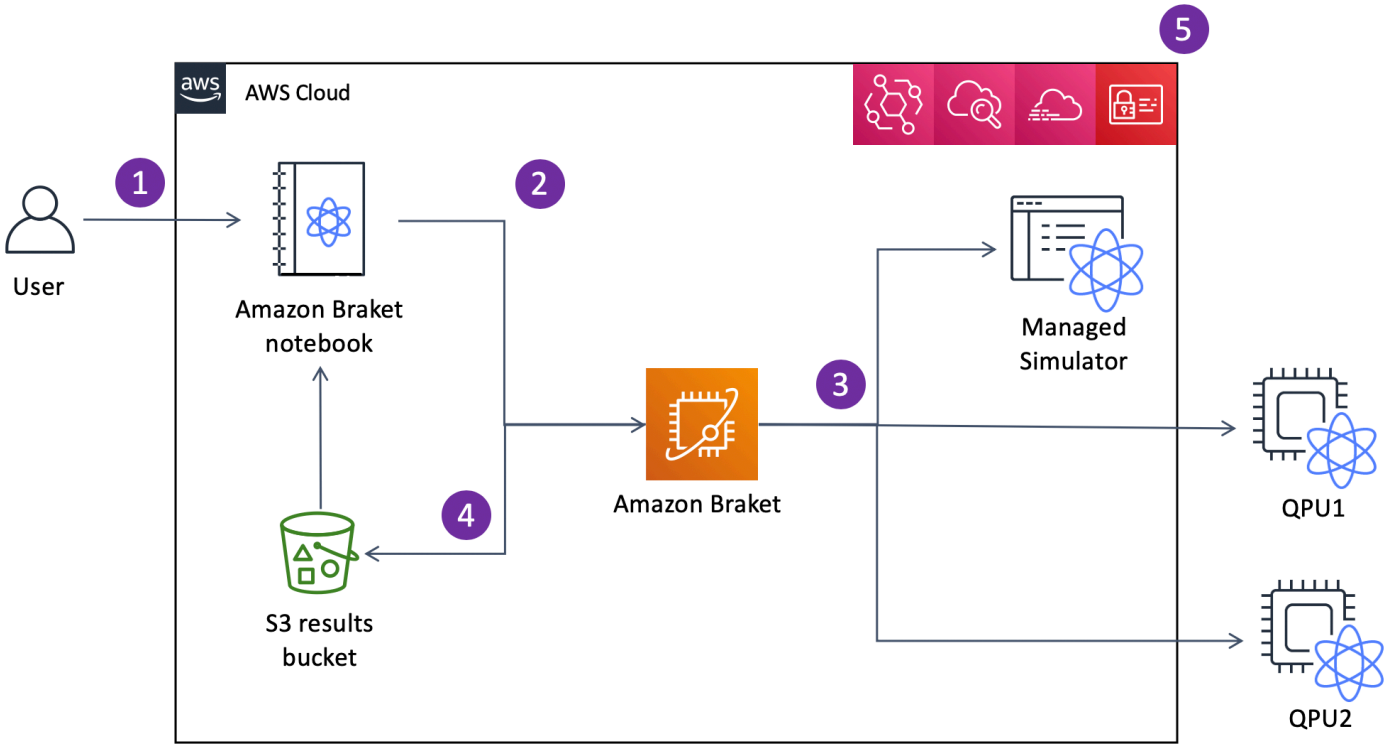
- In der Konsole werden Quantenaufgaben nur für Ihre aktuellen Aufgaben angezeigtAWS-Region. Wenn Sie nach abrechnungsfähigen Quantenaufgaben suchen, die eingereicht wurden, achten Sie darauf, alle Regionen zu überprüfen.
- Eine Liste der Geräte und der zugehörigen Regionen finden Sie auf der Dokumentationsseite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

So funktioniert Amazon Braket

Amazon Braket bietet On-Demand-Zugriff auf Quantencomputergeräte, darunter On-Demand-Schaltungssimulatoren und verschiedene Arten von QPUs. In Amazon Braket ist die atomare Anfrage an ein Gerät eine Quantenaufgabe. Bei Gate-basierten QC-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.

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

Amazon Braket-Quanten-Taskflow



Mit Jupyter Notizbüchern können Sie Ihre Quantenaufgaben bequem 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. 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 verschiedene Arten von QPUs (IonQ,,,Rigetti) Oxford Quantum Circuits (OQC)QuEra, drei On-Demand-Simulatoren (SV1,,TN1)DM1, zwei lokale Simulatoren und einen eingebetteten Simulator. Weitere Informationen finden Sie unter Von Amazon Braket unterstützte Geräte.

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 Braket-Konsole oder mithilfe der `GetQuantumTask` Bedienung von Amazon Braket anzeigen und verwalten. Amazon API

AmazonBraket 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 (5) integriert.

Datenverarbeitung durch Dritte

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.

Kern-Repositorys und Plugins für Braket

Kern-Repositorien

Im Folgenden wird eine Liste von 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).

Plug-ins

Dann gibt es noch die verschiedenen Plugins, die zusammen mit verschiedenen Geräten und Programmierwerkzeugen verwendet werden. Dazu gehören von Braket unterstützte Plugins sowie Plugins, die von Drittanbietern unterstützt werden, 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.
- PennyLane Braket-Plugin — Wird PennyLane als QML-Framework auf Braket verwendet.

Drittanbieter (das Braket-Team überwacht und trägt dazu bei):

- Qiskit-Braket-Anbieter — Verwenden Sie das SDK, um auf Braket-Ressourcen zuzugreifenQiskit.
- Braket-Julia SDK — (EXPERIMENTELL) Eine native Julia-Version des Braket-SDK

Von Amazon Braket unterstützte Geräte

In Amazon Braket steht ein Gerät für eine QPU oder einen Simulator, den Sie aufrufen können, um Quantenaufgaben auszuführen. Amazon Braket bietet Zugriff auf QPU-Geräte vonIonQ,, und Oxford Quantum Circuits QuEraRigetti, drei On-Demand-Simulatoren, drei lokalen Simulatoren und einem eingebetteten Simulator. 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 derzeit, dass Sie zusammenhängende Qubits oder Indizes verwenden. Wenn Sie mit dem Amazon Braket SDK arbeiten, haben Sie Zugriff auf die Geräteeigenschaften, wie im folgenden Codebeispiel gezeigt.

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

```
device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/sv1')
 #SV1
# device = LocalSimulator()
 #Local State Vector Simulator
# device = LocalSimulator("default")
 #Local State Vector Simulator
# device = LocalSimulator(backend="default")
 #Local State Vector Simulator
# device = LocalSimulator(backend="braket_sv")
 #Local State Vector Simulator
# device = LocalSimulator(backend="braket_dm")
 #Local Density Matrix Simulator
# device = LocalSimulator(backend="braket_ahs")
 #Local Analog Hamiltonian Simulation
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/tn1')
 #TN1
# device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/dm1')
 #DM1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Harmony')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
 #IonQ
# device = AwsDevice('arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3')
 #Rigetti Aspen-M-3
# device = AwsDevice('arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy')
 #OQC Lucy
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/quera/Aquila')
 #QuEra Aquila

# get device properties
device.properties
```

Unterstützte Anbieter von Quantenhardware

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

Unterstützte Simulatoren

- Vektorsimulator für lokale Zustände (braket_sv) ('Standardsimulator')
- Simulator für lokale Dichtematrix () `braket_dm`
- Lokaler AHS-Simulator
- Zustandsvektorsimulator (SV1)
- Dichtematrixsimulator (DM1)
- Tensor-Netzwerksimulator () TN1
- PennyLaneDie Blitzsimulatoren

Wählen Sie den besten Simulator für Ihre Quantenaufgabe

- Vergleichen Sie Simulatoren

> **Note**
>
> Scrollen Sie in der folgenden Tabelle nach rechts, um die AWS-Regionen für jedes Gerät verfügbaren Optionen anzuzeigen.

Amazon Braket-Geräte

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
IonQ	Aria 1	Gate-basiert	QPU	arn:aws:braket:us-east-1: :device/qpu/ionq/aria-1	us-east-1
IonQ	Aria 2	Gate-basiert	QPU	arn:aws:braket:us-east-1: :device/qpu/ionq/aria-2	us-east-1
IonQ	Forte 1	Gate-basiert	QPU (nur Reservierung)	arn:aws:braket:us-east-1: :device/qpu/ionq/forte-1	us-east-1

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
IonQ	Harmony	Gate-basiert	QPU	arn:aws:braket:us-east-1: :device/qpu/ionq/harmony	us-east-1
Oxford Quantum Circuits	Lucy	Gate-basiert	QPU	arn:aws:braket:eu-west-2: :device/qpu/oqc/Lucy	eu-west-2
QuEra	Aquila	Analoge Hamiltonsche Simulation	QPU	arn:aws:braket:us-east-1: :device/qpu/quera/Aquila	us-east-1
Rigetti	Aspen M-3	Gate-basiert	QPU	arn:aws:braket:us-west-1: :device/qpu/rigetti/Aspen-M-3	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	SV1	Gate-basiert	Simulator auf Abruf	arn:aws:braket: ::device/quantensimulator/amazon/sv1	Alle Regionen, in denen Braket verfügbar ist. Amazon

Anbieter	Gerätename	Paradigma	Typ	Geräte-ARN	Region
AWS	DM1	Gate-basiert	Simulator auf Abruf	arn:aws:braket: ::device/quantensimulator/amazon/dm1	Alle Regionen, in denen Braket verfügbar ist. Amazon
AWS	TN1	Gate-basiert	Simulator auf Abruf	arn:aws:braket: ::device/quantensimulator/amazon/tn1	us-west-2, us-ost-1 und eu-west-2

> **Note**
>
> Auf bestimmte QPUs kann nur über Reservierungen über Braket Direct zugegriffen werden, siehe Reservierungen.

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

IonQ

IonQbietet Gate-basierte QPUs, die auf der Ionenfall-Technologie 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.

Rigetti

RigettiQuantenprozessoren sind universell einsetzbare Maschinen im Gate-Modell, die auf vollständig einstellbarer Supraleitung basieren. qubits Das Aspen-M-3 79-Qubit-Gerät nutzt die firmeneigene Multi-Chip-Technologie und besteht aus zwei 40-Qubit-Prozessoren.

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'
```

Bei wörtlicher Kompilierung unterstützen die Rigetti-Geräte die folgenden nativen Gatter.

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

RigettiSupraleitende Quantenprozessoren können das Rx-Gate nur mit den Winkeln ±π /2 oder ±π betreiben.

Die Rigetti Geräte verfügen über eine Pulssteuerung, die eine Reihe von vordefinierten Frames der folgenden Typen unterstützt:

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

Weitere Informationen zu diesen Frames finden Sie unter Rollen von Frames und Ports.

Oxford Quantum Circuits (OQC)

OQCQuantenprozessoren sind universelle Maschinen nach dem Gate-Modell, die mit skalierbarer Coaxmon-Technologie gebaut wurden. Das OQC Lucy System ist ein 8-qubit Gerät mit der Topologie eines Rings, bei dem jeder Ring mit seinen beiden nächsten qubit Nachbarn verbunden ist.

Das Lucy Gerät unterstützt die folgenden Quantengatter.

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

Bei wörtlicher Kompilierung unterstützt das OQC-Gerät die folgenden systemeigenen Gatter.

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

Die Steuerung auf Pulsebene ist auf den Geräten verfügbar. OQC Die OQC Geräte unterstützen eine Reihe vordefinierter Frames der folgenden Typen:

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

OQCGeräte unterstützen die dynamische Deklaration von Frames, sofern Sie eine gültige Port-ID angeben. Weitere Informationen zu diesen Frames und Ports finden Sie unter Rollen von Frames und Ports.

> **Note**
>
> Wenn Sie die Pulssteuerung mit verwendenOQC, darf die Länge Ihrer Programme ein Maximum von 90 Mikrosekunden nicht überschreiten. Die maximale Dauer beträgt ungefähr 50 Nanosekunden für Single-Qubit-Gates und 1 Mikrosekunde für Zwei-Qubit-Gates. Diese Zahlen können je nach den verwendeten Qubits, der aktuellen Kalibrierung des Geräts und der Zusammenstellung der Schaltungen variieren.

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.

Vektorsimulator mit lokalem Zustand () **`braket_sv`**

Der lokale Zustandsvektorsimulator (`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-Instanz 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, es führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel betreiben. 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, es führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel betreiben. 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 leistungsstarker Tensor-Netzwerksimulator auf Abruf. TN1kann bestimmte Schaltungstypen mit bis zu 50 qubits und einer Schaltungstiefe von 1.000 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, es führt Ihre Schaltungen bei Bedarf aus und kann mehrere Schaltungen parallel betreiben. 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-Notebook, 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“ der Schaltung zu ermittelnTN1.
- Teilen Sie große shot Simulationen auf mehrere Quantenaufgaben auf.

Eingebettete Simulatoren

Bei eingebetteten Simulatoren wird die Simulation mit dem Algorithmuscode in denselben Container eingebettet und die Simulation wird direkt auf der Hybrid-Job-Instanz ausgeführt. Dies kann nützlich sein, um Engpässe zu beseitigen, die mit der Kommunikation der Simulation mit einem entfernten Gerät verbunden sind. Dies kann zu einem deutlich geringeren Speicherverbrauch, einer geringeren Anzahl von Schaltungsausführungen zur Erzielung des gewünschten Ergebnisses und einer um das Zehnfache oder mehr verbesserten Leistung führen. Weitere Informationen zu eingebetteten Simulatoren finden Sie auf der Seite Einen Hybrid-Job mit Amazon Braket-Hybridjobs ausführen.

PennyLaneDie Blitzsimulatoren

Sie können die Blitzsimulatoren als eingebettete Simulatoren auf Braket verwenden PennyLane. Mit PennyLane den Blitzsimulatoren können Sie fortschrittliche Methoden zur Berechnung von Gradienten nutzen, wie z. B. die Differenzierung von Adjungten, um Gradienten schneller zu bewerten. 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 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.

Aufgrund 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 zuerst 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 Gatter 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 größte Teil der Struktur in einfachere Beziehungen wie diese zerlegt werden kann, die in mehr, kleineren oder einheitlicheren Matrizen dargestellt werden können, ist die TN1 Auswertung einfach.

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. Deshalb versucht Amazon Braket keine extrem komplexen Kontraktionen.

Concurrency (Nebenläufigkeit)

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

Amazon Braket-Regionen und Endpunkte

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

Regionale Verfügbarkeit von Amazon Braket

Name der Region	Region	Endpunkt Braket	QPU
USA Ost (Nord-Virginia)	us-east-1	braket.us-east-1.amazonaws.com	Auf Q
USA Ost (Nord-Virginia)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
USA West (Nordkalifornien)	us-west-1	braket.us-west-1.amazonaws.com	Rigetti
EU West 2 (London)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Sie können Amazon Braket von jeder Region aus ausführen, in der es verfügbar ist, aber jede QPU ist nur in einer einzigen Region verfügbar. 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.

Allgemeine Informationen zur AWS Funktionsweise mit Regionen und Endpunkten finden Sie unter AWS-Service Endpunkte in der AWS Allgemeinen Referenz.

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.

Benachrichtigungen über Statusänderungen per E-Mail oder SMS

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 den Status von Geräten und Quantenaufgaben 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.

Benachrichtigungen über den Abschluss von Quantum-Aufgaben

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 — zum Beispiel, wenn Sie eine umfangreiche Aufgabe einreichen oder wenn Sie eine Aufgabe außerhalb des Verfügbarkeitsfensters eines Geräts einreichen. Wenn Sie nicht warten möchten, bis die Aufgabe abgeschlossen ist, können Sie eine SNS-Benachrichtigung einrichten.

Ein Amazon Braket-Notizbuch führt Sie durch die Einrichtungsschritte. Weitere Informationen finden Sie im Amazon Braket-Beispielnotizbuch zum Einrichten von Benachrichtigungen.

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 Hybrid-Job einreichen, können Sie anhand der Warteschlangentiefe des Geräts überprüfen, wie viele Quantenaufgaben oder Hybrid-Jobs 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 anzeigen 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/Harmony")

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


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

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 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 Position in der Warteschlange 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:us-east-1::device/qpu/ionq/Harmony")

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

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

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


from braket.aws import AwsQuantumJob

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

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

Erste Schritte mit Amazon Braket

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
- Führen Sie Ihre erste Schaltung mit dem Amazon Braket Python SDK aus
- Führe deine ersten Quantenalgorithmen aus

Amazon Braket aktivieren

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

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 ein 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

Amazon Braket bietet vollständig verwaltete Jupyter-Notebooks für den Einstieg. Die Amazon Braket-Notebook-Instances basieren auf SageMaker Amazon-Notebook-Instances. In den folgenden Anweisungen werden die Schritte beschrieben, die zum Erstellen einer neuen Notebook-Instance für neue und bestehende Kunden erforderlich sind.

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, das sich in der Mitte Ihrer Dashboard-Seite befindet, auf Erste Schritte, um einen Notizbuchnamen einzugeben. Dadurch wird ein Standard-Jupyter-Notizbuch erstellt.
3. Die Erstellung Ihres Notizbuches kann mehrere Minuten dauern. Ihr Notizbuch wird auf der Seite Notizbücher mit dem Status Ausstehend aufgeführt. Wenn Ihre Notebook-Instanz einsatzbereit ist,

ändert sich der Status auf InService. Möglicherweise müssen Sie die Seite aktualisieren, um den aktualisierten Status des Notebooks anzuzeigen.

Bestehende Amazon Braket-Kunden

1. Öffnen Sie die Amazon Braket-Konsole, wählen Sie im linken Bereich Notebooks und dann Notebook-Instance erstellen aus. Wenn Sie keine Notebooks haben, wählen Sie das Standard-Setup aus, um ein Standard-Jupyter-Notizbuch zu erstellen, geben Sie einen Notebook-Instanznamen ein, der nur alphanumerische Zeichen und Bindestriche enthält, und wählen Sie Ihren bevorzugten visuellen Modus aus. Aktivieren oder deaktivieren Sie dann den 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 Rechengebühren mehr an, die Speichergebühren bleiben jedoch bestehen.
   b. Um die verbleibende Leerlaufzeit in Ihrer Notebook-Instanz anzuzeigen, navigieren Sie zur Befehlsleiste und wählen Sie die Registerkarte Braket und anschließend die Registerkarte Inactivity Manager aus.

   > **Note**
   >
   > Um zu verhindern, dass Ihre Arbeit verloren geht, sollten Sie erwägen, Ihre SageMaker Notebook-Instanz in ein Git-Repository zu integrieren. Alternativ verhindert das Verschieben Ihrer Arbeit außerhalb der `/Braket Examples` Ordner `/Braket Algorithms` und, dass die Datei beim Neustart der Notebook-Instanz überschrieben wird.

2. (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. Standardmäßig wird der kostengünstige Standard-Instance-Typ ml.t3.medium ausgewählt. Weitere Informationen zu Instance-Preisen finden Sie unter SageMaker Amazon-Preise. Wenn Sie Ihrer Notebook-Instanz ein öffentliches Github-Repository zuordnen möchten, klicken Sie auf das Drop-down-Menü Git-Repository und wählen Sie 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.
   b. Konfigurieren Sie unter Berechtigungen alle optionalen IAM-Rollen, Root-Zugriff und Verschlüsselungsschlüssel.

c. Konfigurieren Sie unter Netzwerk benutzerdefinierte Netzwerk- und Zugriffseinstellungen für Ihre Jupyter Notebook Instance.

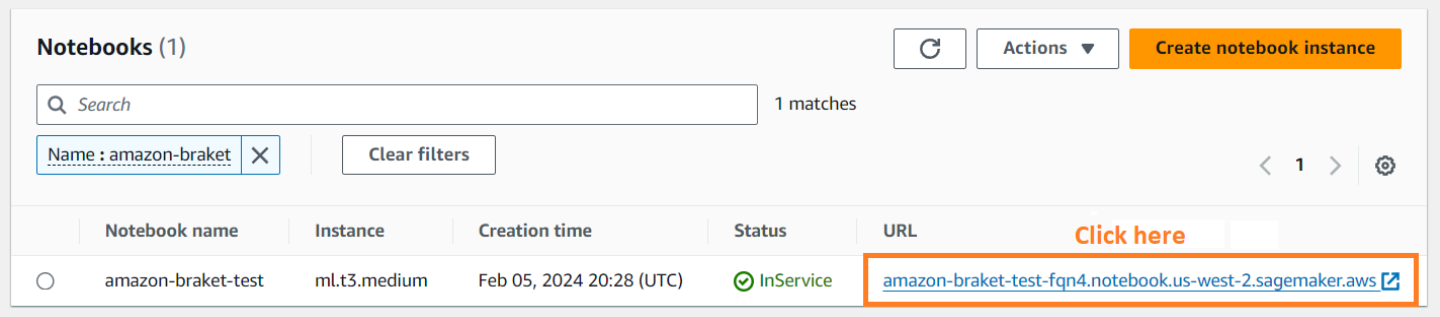
3. Überprüfen Sie Ihre Einstellungen, legen Sie alle Tags fest, um Ihre Notebook-Instance zu identifizieren, und klicken Sie auf Launch.

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

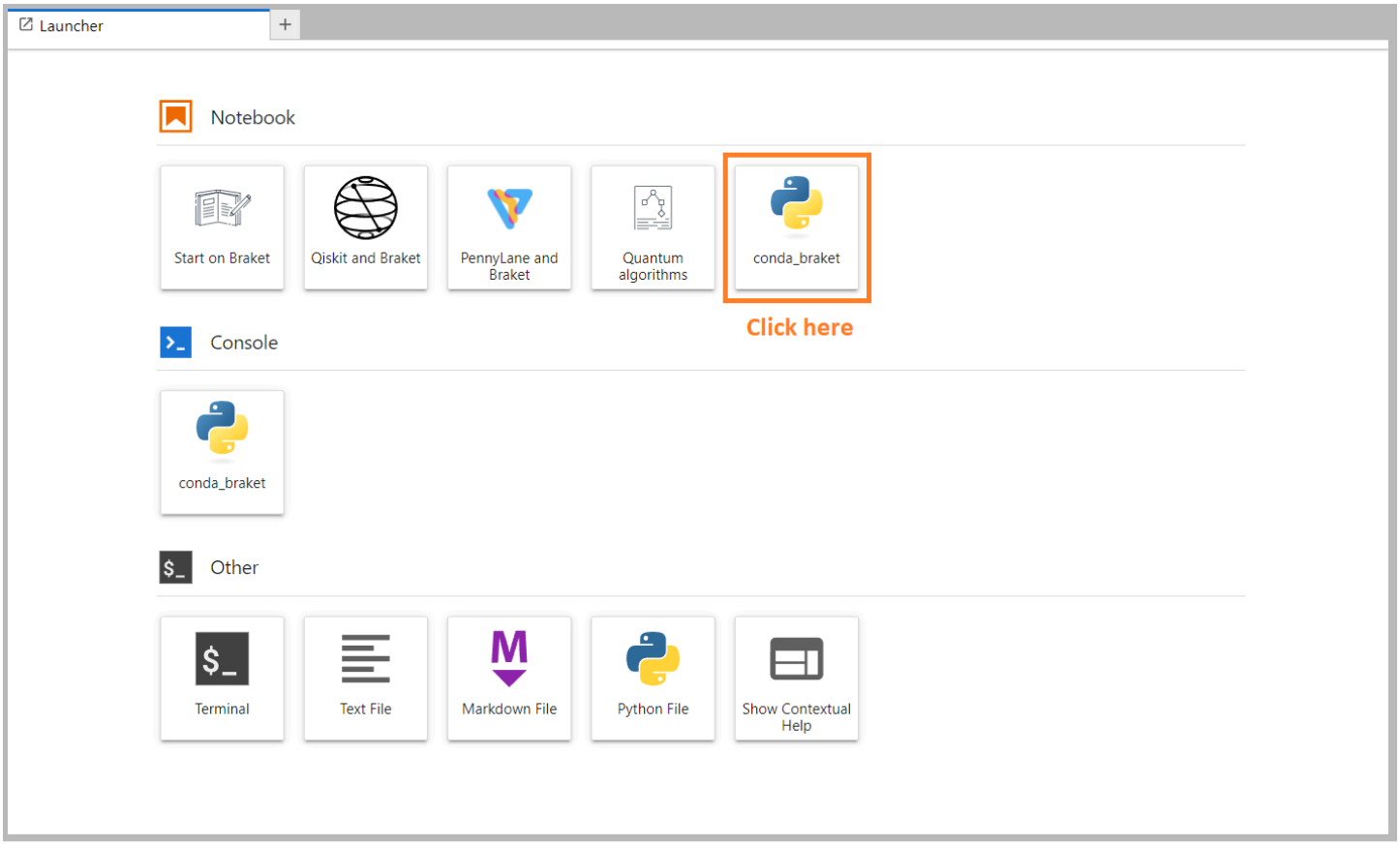
Wenn Sie in der Amazon Braket-Konsole innerhalb AWS des Amazon Braket-SDK arbeiten und Plugins in den von Ihnen erstellten Notizbüchern vorinstalliert sind. Wenn Sie es auf Ihrem eigenen Computer ausführen möchten, können Sie das SDK und die Plugins installieren, wenn Sie den Befehl ausführen `pip install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

Führen Sie Ihre erste Schaltung mit dem Amazon Braket Python SDK aus

Nachdem Ihre Notebook-Instance gestartet wurde, öffnen Sie die Instance mit einer Standard-Jupyter-Schnittstelle, indem Sie das Notebook auswählen, das Sie gerade erstellt haben.



AmazonAuf den Braket-Notebook-Instanzen ist das Amazon Braket-SDK mit all seinen Abhängigkeiten vorinstalliert. Erstellen Sie zunächst ein neues Notebook 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 Amazon Braket-SDK-Module und definieren Sie einen einfachen Bell-State-Schaltkreis.

```
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)
```

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).

So instanziieren Sie den lokalen Simulator:

```
# 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, ist eine gleichmäßige Überlagerung von |00> und |11>, und Sie werden erwartungsgemäß eine ungefähr gleiche (bis zum shot Rauschen) Verteilung von 00 und 11 als Messergebnisse feststellen.

Führen Sie Ihre Schaltung auf einem On-Demand-Simulator aus

AmazonBraket 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 `example-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 = "example-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. Navigiere in der Konsole zur Registerkarte Quantenaufgaben und deine Quantenaufgabe sollte ganz oben in 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 QPU Geräte vonIonQ, Oxford Quantum CircuitsQuEra, undRigetti. 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 Rigetti Gerät instanziiert wird.

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

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/Harmony")
```

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)
'2' # returns the number of hybrid jobs queued on the device
```

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()`

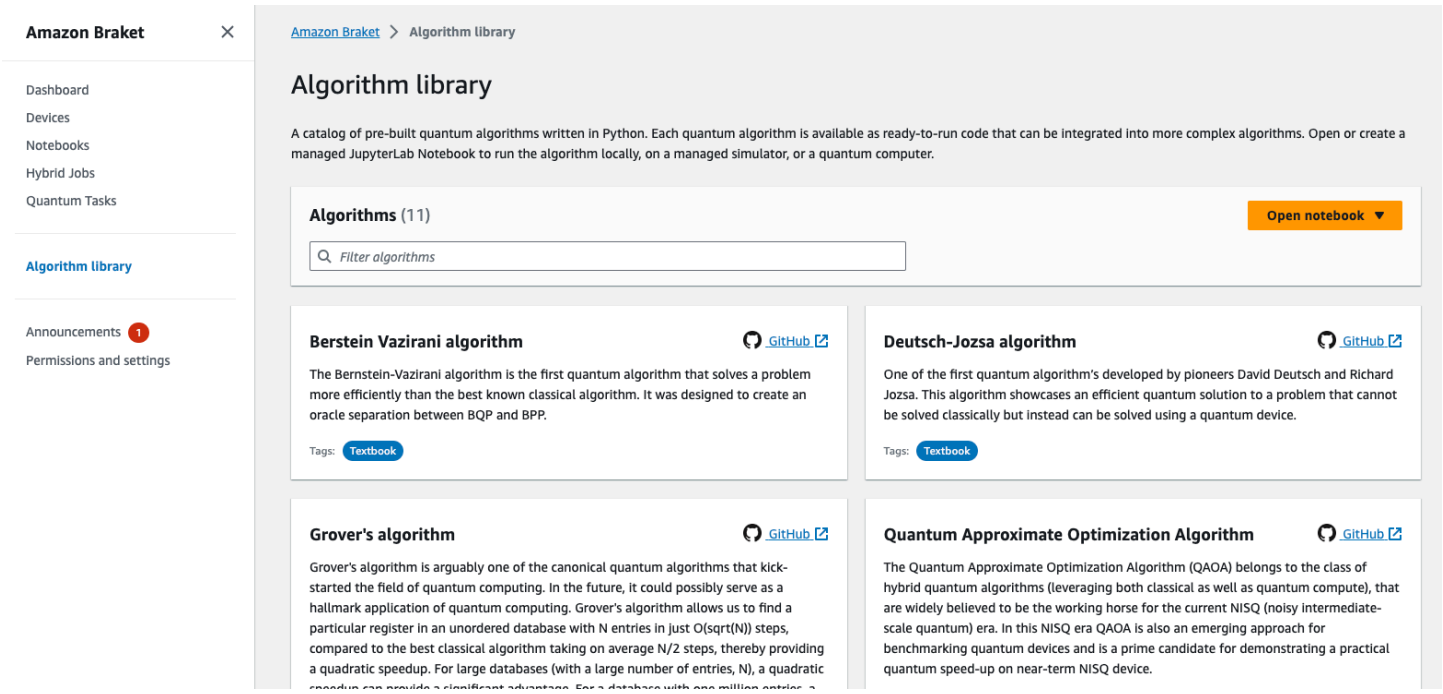
```
# define quantum task with 1 day polling timeout
task = device.run(bell, s3_folder, poll_timeout_seconds=24*60*60)
print(task.result().measurement_counts)
```

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'
```

Führe deine ersten Quantenalgorithmen aus

Die Amazon Braket-Algorithmusbibliothek ist ein Katalog vorgefertigter Quantenalgorithmen, die in Python geschrieben wurden. Sie können diese Algorithmen unverändert ausführen oder sie als Ausgangspunkt für die Erstellung komplexerer Algorithmen verwenden. Sie können von der Braket-Konsole aus auf die Algorithmusbibliothek zugreifen. Sie können auch auf Github auf die Braket-Algorithmusbibliothek zugreifen: https://github.com/aws-samples/. amazon-braket-algorithm-library



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.

Mit Amazon Braket zusammenarbeiten

In diesem Abschnitt erfahren Sie, wie Sie Quantenschaltkreise entwerfen, diese Probleme als Quantenaufgaben an Geräte senden und die Quantenaufgaben mit dem Amazon Braket SDK überwachen.

Im Folgenden sind die wichtigsten Möglichkeiten zur Interaktion mit Ressourcen auf Amazon Braket aufgeführt.

- 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 Braket-Python-SDK zusammen mit dem Amazon 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:

Hallo AHS: Führen Sie Ihre erste analoge Hamiltonsche Simulation aus

AHS

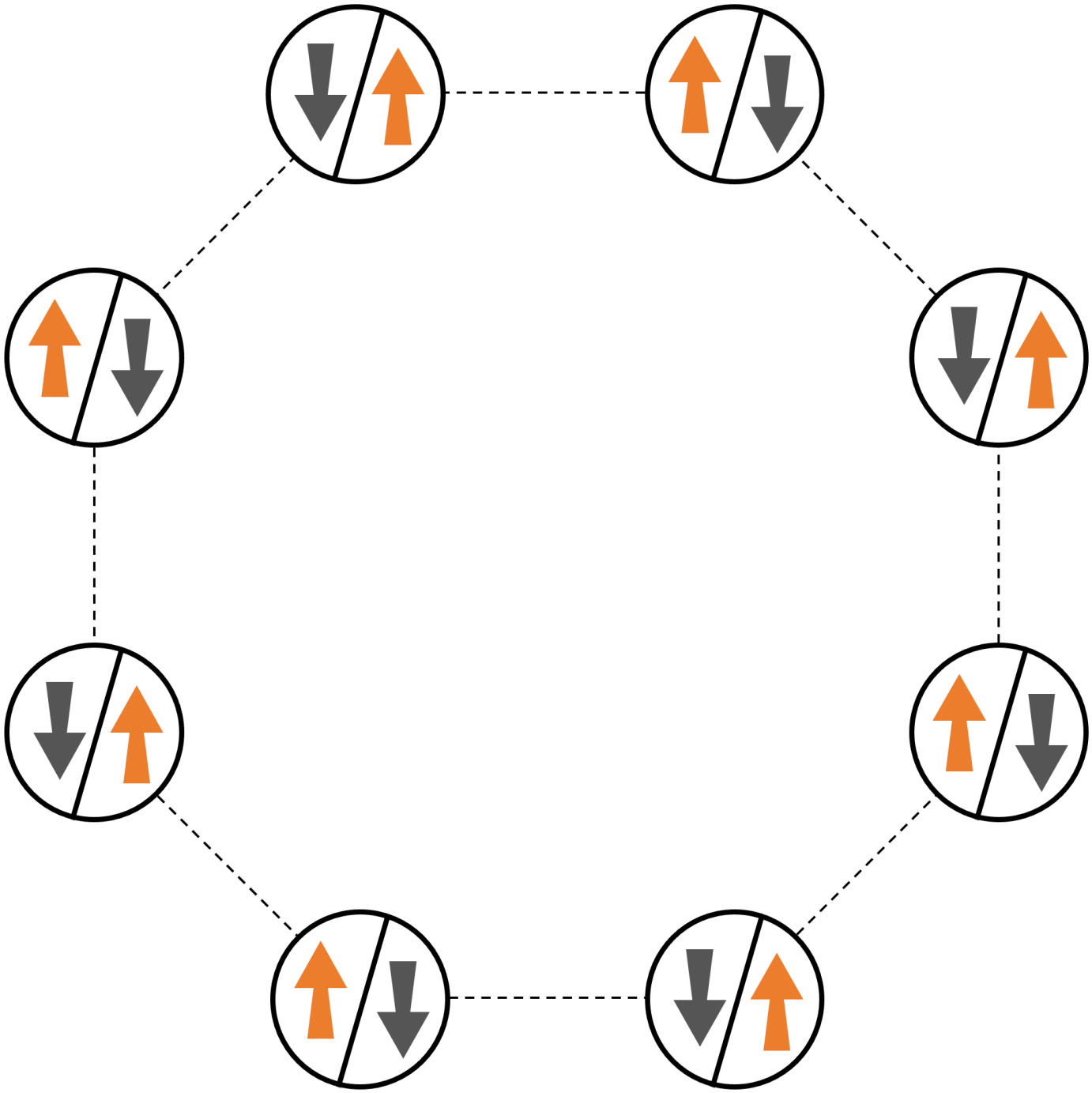
Die analoge Hamiltonsche Simulation (AHS) ist ein Paradigma des Quantencomputers, das sich von Quantenschaltkreisen unterscheidet: Statt einer Abfolge von Gattern, die jeweils nur auf ein paar Qubits gleichzeitig wirken, wird ein AHS-Programm durch die zeit- und raumabhängigen Parameter des betreffenden Hamiltonschen Parameters definiert. Der Hamiltonsche Wert eines Systems kodiert

seine Energieniveaus und die Auswirkungen äußerer Kräfte, die zusammen die zeitliche Entwicklung seiner Zustände bestimmen.
Bei einem N-Qubit-System kann der Hamiltonsche Wert durch eine quadratische Matrix komplexer Zahlen mit 2 N x 2 N dargestellt werden.

Quantengeräte, die AHS ausführen können, stimmen ihre Parameter (z. B. Amplitude und Verstimmung eines kohärenten Antriebsfeldes) so ab, dass sie der zeitlichen Entwicklung des Quantensystems unter dem benutzerdefinierten Hamiltonschen Wert sehr nahe kommen. Das AHS-Paradigma eignet sich zur Simulation der statischen und dynamischen Eigenschaften von Quantensystemen vieler interagierender Teilchen. Speziell entwickelte QPUs, wie das Aquila-Gerät von, QuEra können die zeitliche Entwicklung von Systemen mit Größen simulieren, die sonst auf klassischer Hardware nicht realisierbar wären.

Interagierende Spin-Kette

Als kanonisches Beispiel für ein System aus vielen interagierenden Teilchen betrachten wir einen Ring mit acht Spins (von denen sich jeder im Zustand „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

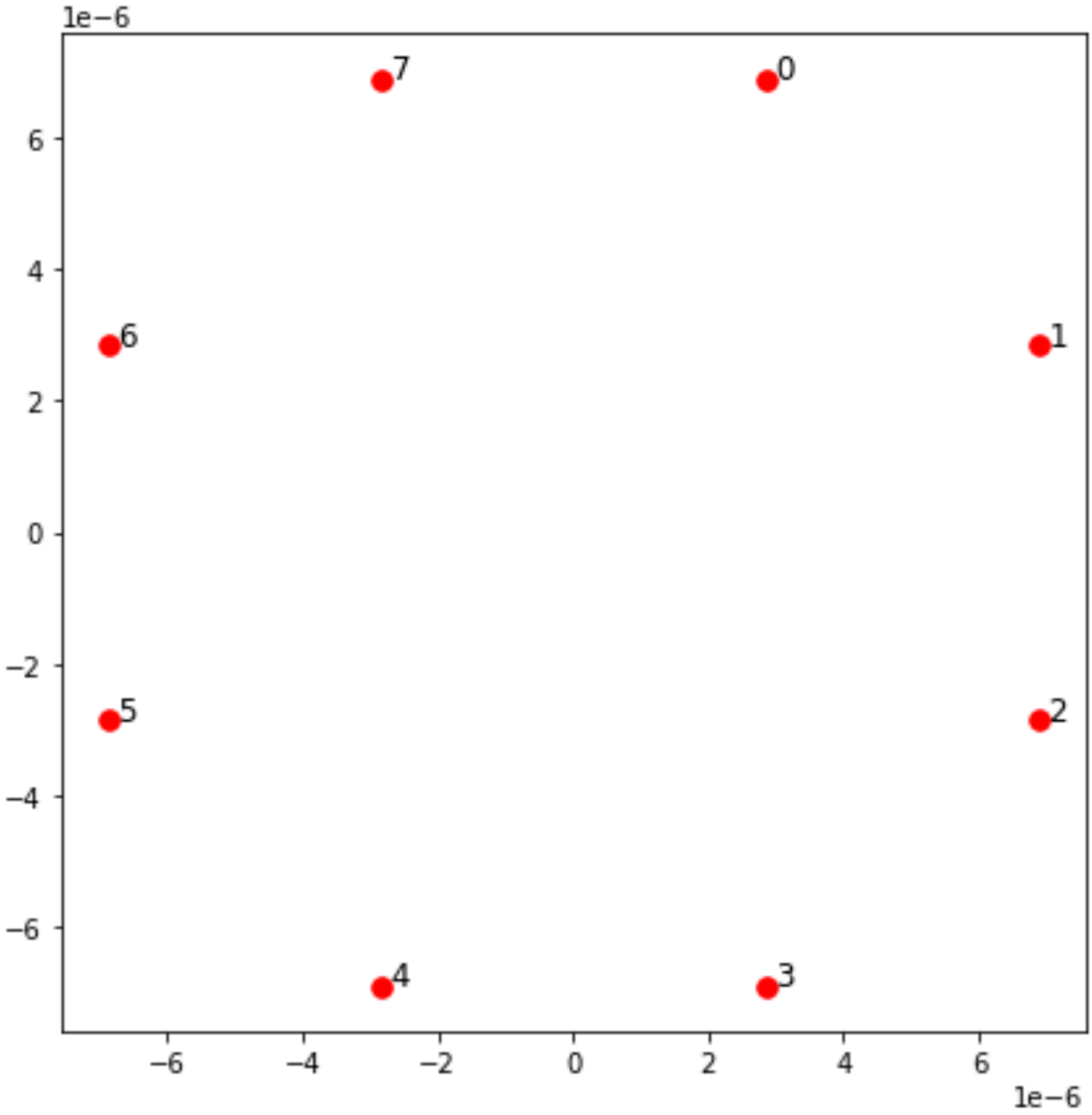
Wir werden für jeden Spin ein neutrales Atom verwenden, und die Spinzustände „hoch“ und „runter“ werden jeweils im angeregten Rydberg-Zustand und im Grundzustand der Atome kodiert. Zuerst erstellen wir die 2-D-Anordnung. Wir können den obigen Ring von Spins mit dem folgenden Code programmieren.

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`

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

from braket.ahs.atom_arrangement import AtomArrangement

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

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

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=↑ j czech-↑ j ein Operator, der den Wert 1 nur annimmt, wenn sich Spin j im Zustand „up“ befindet, andernfalls 0. Die Stärke ist V j,k =C6/(dj,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 und 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.

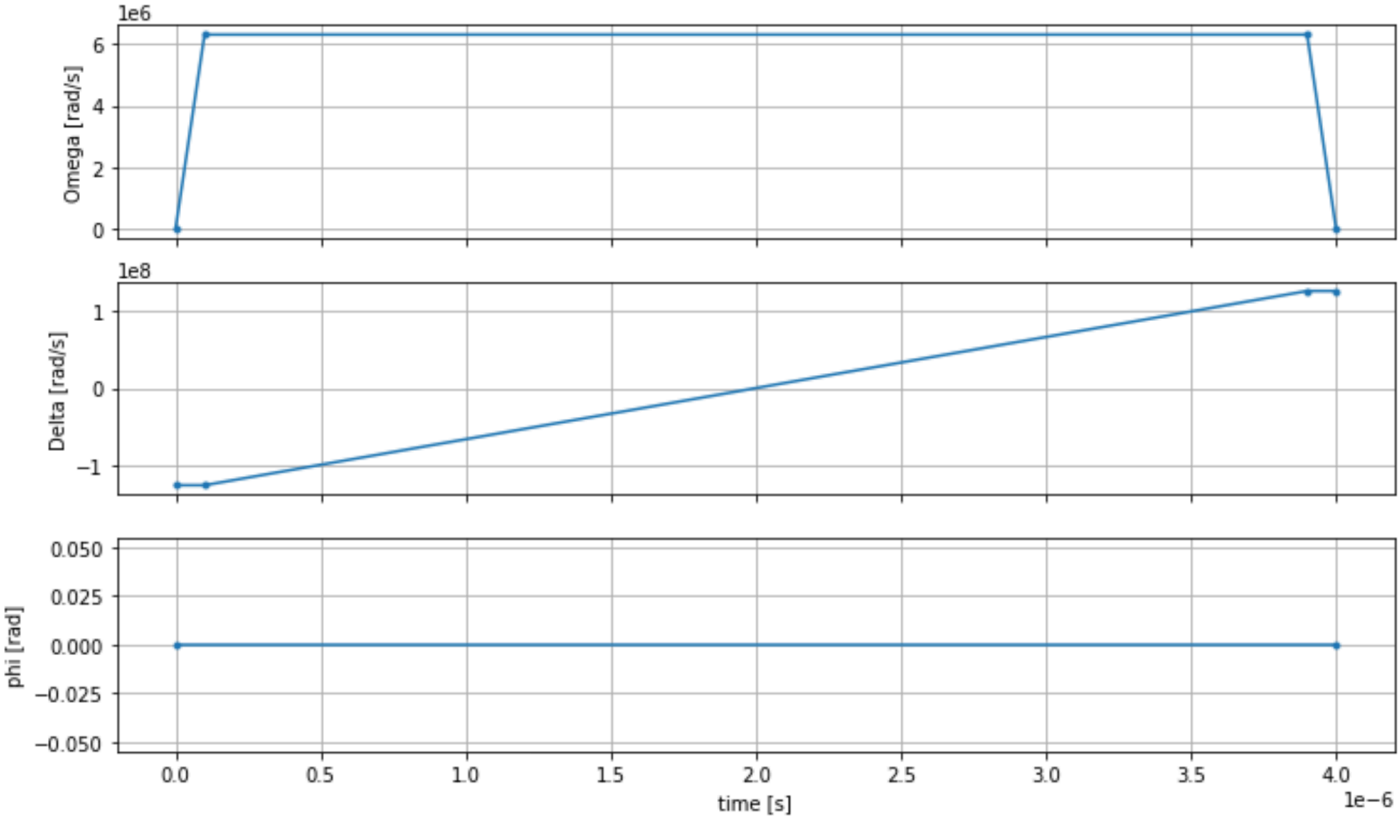
```
from braket.timings.time_series import TimeSeries
```

```
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 Fahrfeld (und die impliziten Van-der-Waals-Interaktionen) bilden das Programm Analog Hamiltonian 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-kompatiblen QPU auf dem lokalen AHS-Simulator ausführen, der mit dem Braket-SDK geliefert wird. Da der lokale Simulator kostenlos mit dem Braket-SDK verfügbar ist, ist dies eine 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 „down“, „u“ für „up“ oder „e“ für leere Seite 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
```

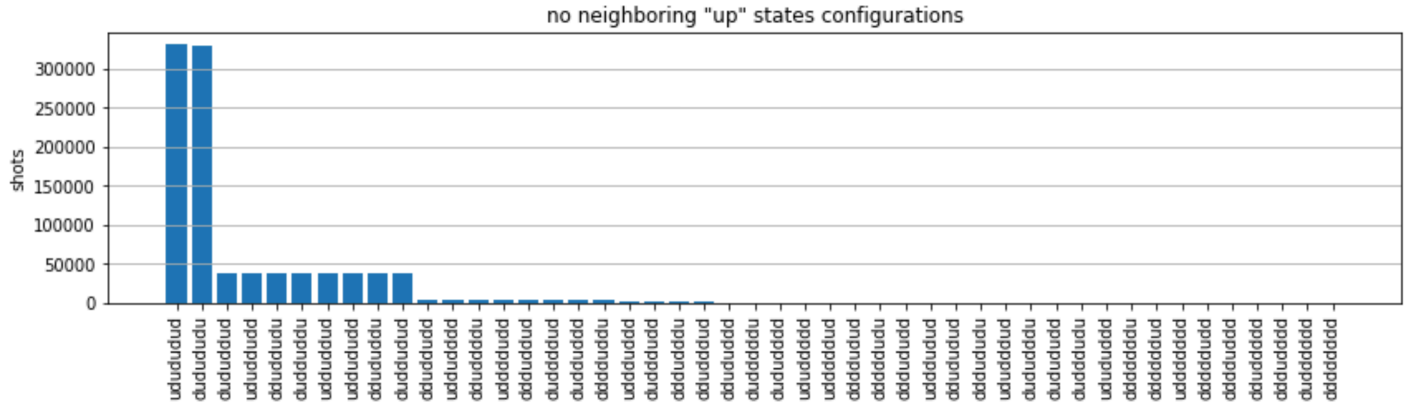
```
    Args:
       result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

    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)
```

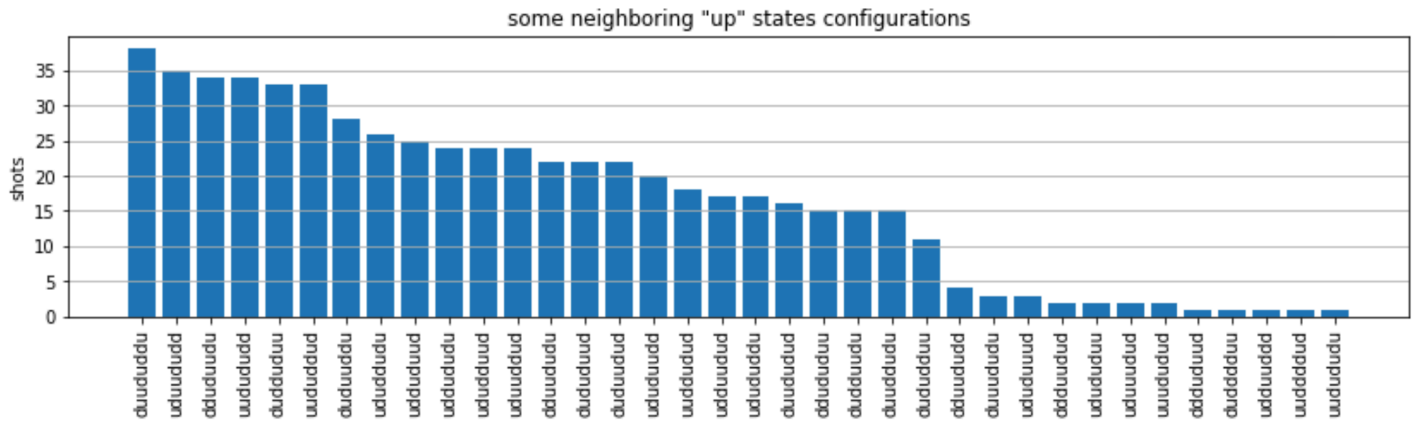
```
{'udududud': 330944, 'dudududu': 329576, 'dududdud': 38033, ...}
```

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 präpariert 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 „Erste Schritte“ 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 beinhaltet 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}")
```

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

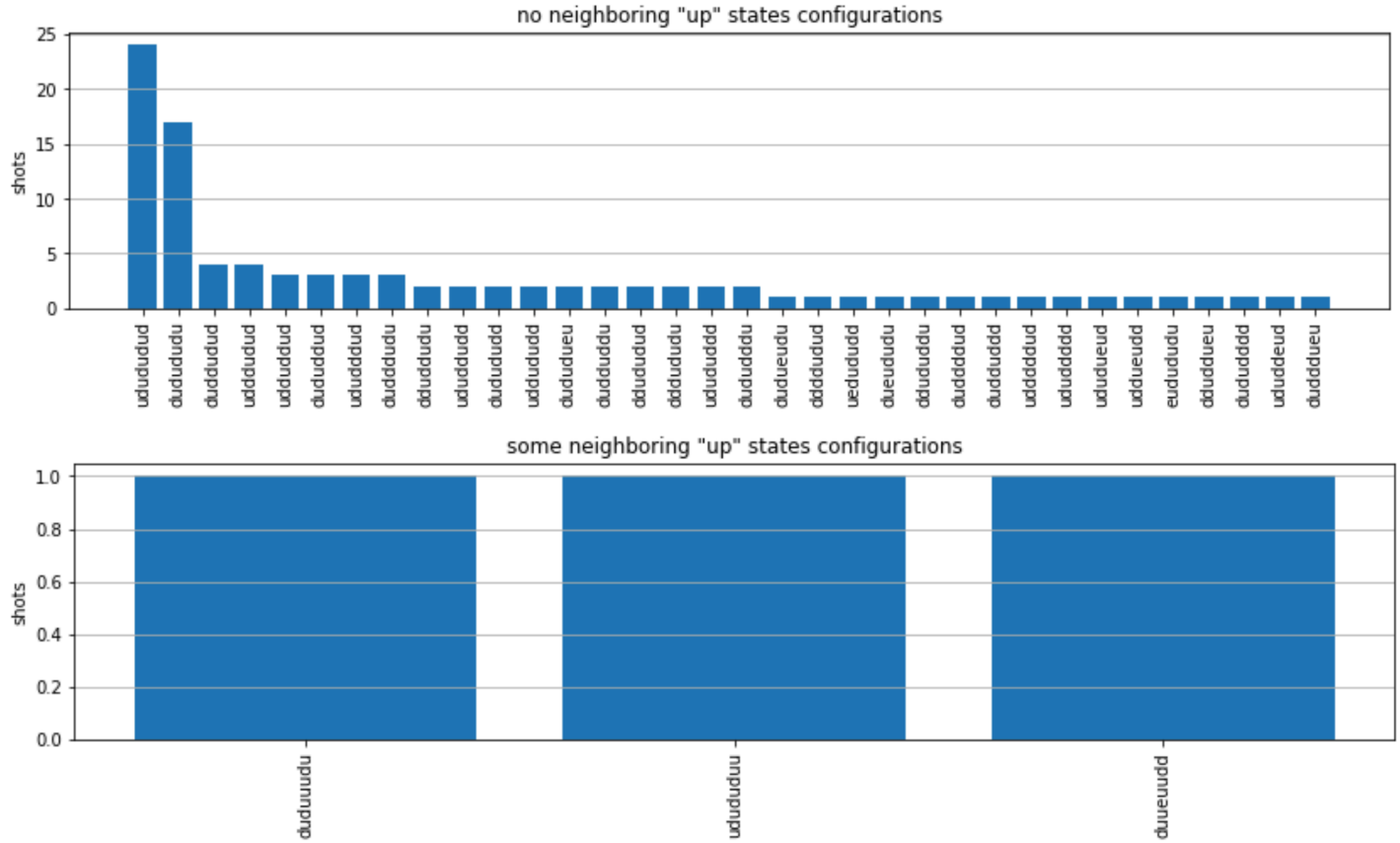
Mit den gleichen `get_counts` Funktionen wie zuvor können wir die Anzahl berechnen:

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

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

und plotten Sie sie mit`plot_counts`:

```
plot_counts(counts_aquila)
```



Beachten Sie, dass ein kleiner Teil der Aufnahmen leere Seiten 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.

Next

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.

Konstruieren Sie Schaltkreise im SDK

Dieser Abschnitt enthält Beispiele für das Definieren eines Schaltkreises, das Anzeigen verfügbarer Gates, 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

Quantengatter und Schaltkreise sind in der `braket.circuits`Klasse des Amazon Braket Python SDK definiert. Im 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 beschriftet`q3`) `q0` `q1`q2, 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-C---
        |
q1 : -H-|-C-
        | |
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)
```

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 auf dem 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', 'GPi', 'GPi2', 'H', 'I', 'ISwap', 'MS', 'PSwap',
 'PhaseShift', 'PulseGate', 'Rx', 'Ry', 'Rz', 'S', 'Si', 'Swap', 'T', 'Ti', 'Unitary',
 'V', 'Vi', 'X', 'XX', 'XY', 'Y', 'YY', 'Z', 'ZZ']
```

Jedes dieser Gatter kann an einen Schaltkreis angehängt werden, indem die Methode für diesen Schaltungstyp aufgerufen wird. Sie würden beispielsweise aufrufen`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])
new_instr = instr.copy(target_mapping={0: 5})
```

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/Harmony")

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

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

Unterstützte Gates müssen möglicherweise zu systemeigenen 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/Aspen-M-3")

#specify the qubits
a=10
b=113
```

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

Manuelle Zuordnung qubit

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.

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

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

> **Note**
>
> Der OQC Compiler unterstützt keine Einstellungen. `disable_qubit_rewiring=True` Wenn Sie dieses Flag auf setzen`True`, wird der folgende Fehler angezeigt:`An error occurred (ValidationException) when calling the CreateQuantumTask operation: Device arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy does not support disabled qubit rewiring`.

Wörtliche Zusammenstellung

Wenn Sie einen Quantenschaltkreis auf Quantencomputern vonRigetti,, IonQ oder Oxford Quantum Circuits (OQC) aus 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 angegeben beibehalten wird (unterstützt von RigettiIonQ, undOQC) oder dass nur bestimmte Teile davon erhalten bleiben (nur unterstützt vonRigetti). Bei der Entwicklung von Algorithmen für Hardware-Benchmarking- oder Fehlerminimierungsprotokolle müssen Sie die Möglichkeit haben, die Gates und Schaltungslayouts, die Sie auf der Hardware ausführen, 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 derzeit auf Geräten RigettiIonQ, und Oxford Quantum Circuits (OQC) unterstützt und erfordert die Verwendung systemeigener Gatter. Bei der verbatim-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/Aspen-M-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 OQC 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 vollständig ignoriert wird.

Simulation von Geräuschen

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

```
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 eine einfache Schaltung mit depolarisierendem Rauschen und ein benutzerdefinierter Kraus-Kanal verwendet werden.

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

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

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

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

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

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

Beispiel 1

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

Oder

Beispiel 2

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

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

Der Stromkreis wird überprüft

Quantenschaltkreise in Amazon Braket haben ein Pseudozeitkonzept namens. `Moments` Jeder qubit kann pro ein einzelnes Tor 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 Tiefe des Schaltkreises anzeigen, indem Sie die Methode aufrufen, `circuit.depth` wie im folgenden Beispiel gezeigt.

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

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

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

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

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

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)]))
Instruction('operator': Rx('angle': 0.15, 'qubit_count': 1), 'target':
 QubitSet([Qubit(0)]))

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

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

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

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

Sie können einem Schaltkreis auch Tore hinzufügen`Moments`.

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

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

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

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

Arten von Ergebnissen

AmazonBraket kann verschiedene Arten von Ergebnissen zurückgeben, wenn ein Stromkreis mit `ResultType` gemessen wird. Ein Schaltkreis 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.
- `Reduced density matrix`gibt eine Dichtematrix für ein Subsystem eines angegebenen Ziels 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 Geräte:

	Lokale SIM	SV1	DM1	TN1	Rigetti	IonQ	OQC
Adjungierter Gradient	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
Erwartung	Y	Y	Y	Y	Y	Y	Y

Probability (Wahrscheinlichkeit)	Y	Y	Y	N	Y*	Y	Y
Matrix mit reduzierter Dichte	Y	N	Y	N	N	N	N
Zustandsvektor	Y	N	N	N	N	N	N
Beispiel	Y	Y	Y	Y	Y	Y	Y
Varianz	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> * unterstützt Rigetti nur Wahrscheinlichkeitsergebnistypen von bis zu qubits 40.

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

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

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

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

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

```
from braket.circuits import Observable

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

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

> **Note**
>
> Einige Geräte liefern Messungen (zum BeispielRigetti) als Ergebnisse und andere liefern Wahrscheinlichkeiten als Ergebnisse (zum Beispiel IonQ undOQC). Das SDK bietet eine Messeigenschaft für Ergebnisse, aber für Geräte, die Wahrscheinlichkeiten zurückgeben, wird diese Eigenschaft nachberechnet. Somit werden bei Geräten wie denen, die von bereitgestellt OQC wurden IonQ und deren Messergebnisse durch die Wahrscheinlichkeit bestimmt werden, nicht zurückgegeben, da Messungen pro Schuss nicht zurückgegeben werden. Sie können überprüfen, ob ein Ergebnis nachberechnet wurde, indem Sie sich das Objekt `measurements_copied_from_device` auf dem Ergebnis ansehen, wie in dieser Datei dargestellt.

Observablen

AmazonBraket umfasst eine `Observable` Klasse, die verwendet werden kann, um eine zu messende Observable zu spezifizieren.

Sie können auf jedes Objekt höchstens ein eindeutiges beobachtbares Objekt ohne Identität anwenden. qubit Wenn Sie zwei oder mehr verschiedene Observablen ohne Identität für dasselbe angebenqubit, wird ein Fehler angezeigt. Zu diesem Zweck zählt jeder Faktor eines Tensorprodukts als einzelne beobachtbare Größe. Es ist also zulässig, dass mehrere Tensorprodukte auf dasselbe einwirkenqubit, vorausgesetzt, dass der Faktor, der darauf einwirkt, derselbe ist. qubit

Sie können auch eine Observable skalieren und Observablen hinzufügen (skaliert oder nicht). Dadurch wird eine erstelltSum, die im Ergebnistyp verwendet werden kann. `AdjointGradient`

Die `Observable` Klasse umfasst die folgenden Observablen.

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

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

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

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

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

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

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

```
Sum of other (scaled) observables: Sum(TensorProduct(X('qubit_count': 1),
 X('qubit_count': 1)), TensorProduct(Z('qubit_count': 1), Z('qubit_count': 1)))
```

Parameter

Schaltungen können freie Parameter enthalten, die Sie nach dem Prinzip „einmal konstruieren — viele Male ausführen“ und zur Berechnung von Gradienten verwenden können. Freie Parameter haben einen in einer Zeichenkette codierten Namen, anhand dessen Sie ihre Werte angeben oder festlegen können, ob in Bezug auf sie unterschieden werden soll.

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

Geben Sie die Parameter, die Sie unterscheiden möchten, entweder mithilfe ihres Namens (als Zeichenfolge) oder durch direkten Verweis an. Beachten Sie, dass die Berechnung des Gradienten anhand des `AdjointGradient` Ergebnistyps unter Berücksichtigung des Erwartungswerts der beobachtbaren Größe erfolgt.

Hinweis: Wenn Sie die Werte freier Parameter festgelegt haben, indem Sie sie als Argumente an den parametrisierten Schaltkreis übergeben haben, führt das Ausführen einer Schaltung mit dem `AdjointGradient` Ergebnistyp und den angegebenen Parametern zu einem Fehler. Das liegt daran, dass die Parameter, die wir zur Differenzierung verwenden, nicht mehr vorhanden sind. Sehen Sie sich das folgende -Beispiel an.

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

Einreichung von Quantenaufgaben an QPUs und Simulatoren

AmazonBraket bietet Zugriff auf mehrere Geräte, die Quantenaufgaben ausführen können. Sie können Quantenaufgaben einzeln einreichen oder die Batchverarbeitung von Quantenaufgaben einrichten.

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.

- IonQ Aria 1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1`
- IonQ Harmony : `arn:aws:braket:us-east-1::device/qpu/ionq/Harmony`
- OQC Lucy : `arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Aspen-M-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3`

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 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:

- Batching von Quantenaufgaben
- Richten Sie SNS-Benachrichtigungen ein (optional)
- Kompilierte Schaltkreise untersuchen

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

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

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

Reichen Sie ein Beispiel für eine Quantenaufgabe ein

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

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

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

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

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

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 3 MB begrenzt.

Reichen Sie eine parametrisierte Aufgabe ein

Amazon Braket On-Demand- und lokale Simulatoren und QPUs unterstützen 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 auf bestimmten QPUs verbessern. 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 QPUs von Rigetti Computing und Oxford Quantum Circuits mit Ausnahme von Pulspegelprogrammen unterstützt.

```
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})
```

Angeben der shots

Das shots Argument bezieht sich auf die Anzahl der gewünschten Messungen. shots 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 verwendet der Simulator Stichproben aus der Ausgangsverteilungshots, um das shot Rauschen realer QPUs zu emulieren. QPU-Geräte erlauben nur > 0. shots

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

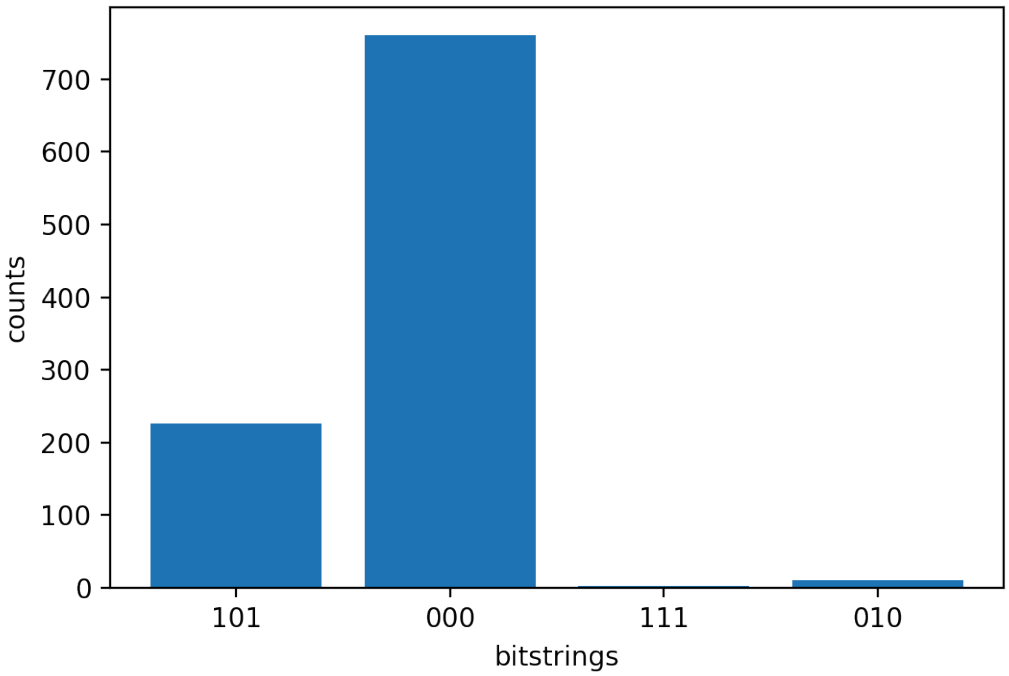
Sehen Sie sich die Beispielergebnisse an

Da Sie auch die angegeben habenResultType, können Sie sich die zurückgegebenen Ergebnisse ansehen. 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])

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

```
Result types include:
```

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



Quantenaufgaben an eine QPU senden

AmazonMit Braket können Sie eine Quantenschaltung auf einem QPU-Gerät ausführen. Das folgende Beispiel zeigt, wie eine Quantenaufgabe an unsere Geräte gesendet Rigetti wird. IonQ

Wählen Sie das Rigetti Aspen-M-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/Aspen-M-3")

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

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

Das vorherige Wörterbuch `connectivityGraph` enthält Informationen zur Konnektivität des aktuellen Rigetti Geräts.

Wählen Sie das IonQ Harmony Gerät

Für das IonQ Harmony Gerät `connectivityGraph` ist der leer, wie im folgenden Beispiel gezeigt, weil das Gerät all-to-allKonnektivität bietet. Daher `connectivityGraph` ist eine detaillierte Angabe nicht erforderlich.

```
# or choose the IonQ Harmony device
```

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

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

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

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.

AmazonBraket kann QPU-Quantenaufgaben innerhalb bestimmter Verfügbarkeitsfenster ausführen, aber Sie können Quantenaufgaben trotzdem jederzeit (rund um die Uhr) einreichen, 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.

Eine Quantenaufgabe mit dem lokalen Simulator ausführen

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 Instanztyp des Amazon Braket-Notebooks oder von Ihren lokalen Hardwarespezifikationen ab.

Die folgenden Befehle sind alle identisch und instanziieren den lokalen State-Vector-Simulator (rauschfrei).

```
# 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")
```

Batching von Quantenaufgaben

Die Batchverarbeitung von Quantenaufgaben ist auf jedem Amazon Braket-Gerät verfügbar, mit Ausnahme des lokalen Simulators. Batching ist besonders nützlich für Quantenaufgaben, die Sie auf den On-Demand-Simulatoren (TN1oderSV1) ausführen, da sie mehrere Quantenaufgaben parallel verarbeiten können. Um Ihnen bei der Einrichtung verschiedener Quantenaufgaben zu helfen, bietet Amazon Braket Beispielnotizbücher.

Durch Batching können Sie Quantenaufgaben parallel starten. Wenn Sie beispielsweise eine Berechnung durchführen möchten, für die 10 Quantenaufgaben erforderlich sind und die Schaltkreise

in diesen Quantenaufgaben unabhängig voneinander sind, empfiehlt es sich, Batching zu verwenden. Auf diese Weise müssen Sie nicht warten, bis eine Quantenaufgabe abgeschlossen ist, bevor eine andere Aufgabe beginnt.

Das folgende Beispiel zeigt, wie ein Stapel von Quantenaufgaben ausgeführt wird:

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

Weitere Informationen finden Sie in den Amazon Braket-Beispielen zu GitHub Quantum Task Batching, die genauere Informationen zum Batching enthalten.

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 Vorteile der Batchverarbeitung, wenn Sie PennyLane on Amazon Braket verwenden, indem Sie festlegen, `parallel = True` wann Sie ein Amazon Braket-Gerät instanziieren, wie im folgenden Beispiel gezeigt.

```
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 angeben, das für alle Quantenaufgaben im Stapel verwendet wird, oder ein `list` Eingabewörterbuch. In diesem Fall wird das -te Wörterbuch mit `i` der -ten Aufgabe verknüpft, wie im folgenden Beispiel `i` gezeigt.

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

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

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

circ_b = Circuit().rx(0, alpha).rz(1, alpha).cnot(0,2).zz(0, 2, beta)
circ_b.expectation(observable=Observable.Z(), target=2)

# use the same inputs for both circuits in one batch

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

# or provide each task its own set of inputs

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

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

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)
```

Richten Sie SNS-Benachrichtigungen ein (optional)

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.

Kompilierte Schaltkreise untersuchen

Wenn eine Schaltung auf einem Hardwaregerät ausgeführt wird, muss sie in einem akzeptablen Format kompiliert werden, z. B. das Transpilieren der Schaltung bis zu den systemeigenen Gates, die von der QPU unterstützt werden. Die Überprüfung der tatsächlich kompilierten Ausgabe kann für Debugging-Zwecke sehr nützlich sein. Mit dem folgenden Code können Sie sich diese Schaltung Rigetti sowohl für OQC Geräte als auch für Geräte ansehen.

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

> **Note**
>
> Heute können Sie Ihre kompilierte Schaltung nicht für IonQ Geräte anzeigen.

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:

- Berechnung von Gradienten 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 einfache 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 Braket zum ersten Mal verwenden, müssen Sie Braket aktivieren. Amazon Amazon 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
- OpenQASM-Pragmas in Klammern
- Erweiterte Funktionsunterstützung für OpenQASM auf dem Local Simulator
- Unterstützte Operationen und Grammatik mit OpenPulse

Unterstützte OpenQASM-Datentypen

Die folgenden OpenQASM-Datentypen werden von Braket unterstützt. Amazon

- Nichtnegative Ganzzahlen werden für (virtuelle und physische) Qubit-Indizes verwendet:
  - `cnot q[0], q[1];`
  - `h $0;`
- Fließkommazahlen oder Konstanten können für Gate-Drehwinkel 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 Braket unterstützt. Amazon

- `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;`

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

OpenQASM-Pragmas in Klammern

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

- 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
- Benutzerdefinierte 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 Braket-Gerät zu senden. Amazon

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 die, um OpenQASM 3.0-Aufgaben zu erstellen AWS CLI

Ein Beispiel für ein OpenQASM 3.0-Programm

Um eine OpenQASM 3.0-Aufgabe zu erstellen, können Sie mit einem einfachen OpenQASM 3.0-Programm (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.

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

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

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

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

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 = "amazon-braket-my-bucket"
s3_prefix = "openqasm-tasks"

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

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

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

Verwenden Sie die, um OpenQASM 3.0-Aufgaben zu erstellen AWS CLI

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/Aspen-M-3" \
    --shots 100 \
    --output-s3-bucket "amazon-braket-my-bucket" \
    --output-s3-key-prefix "openqasm-tasks" \
```

```
    --action '{
        "braketSchemaHeader": {
            "name": "braket.ir.openqasm.program",
            "version": "1"
        },
        "source": $(cat ghz.qasm)
    }'
```

Support 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 dargestellt. 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. Die folgenden Beispiele zeigen dieses `pulse` Feld für die OQC Geräte Rigetti und.

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

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

```
      "PERMITTED_FREQUENCY_DIFFERENCE": 400000000
    }
  }
}

// OQC

{
  "pulse": {
    "braketSchemaHeader": {
      "name": "braket.device_schema.pulse.pulse_device_action_properties",
      "version": "1"
    },
    "supportedQhpTemplateWaveforms": {
      "gaussian": {
        "functionName": "gaussian",
        "arguments": [
          {
            "name": "length",
            "type": "float",
            "optional": false
          },
          {
            "name": "sigma",
            "type": "float",
            "optional": false
          },
          {
            "name": "amplitude",
            "type": "float",
            "optional": true
          },
          {
            "name": "zero_at_edges",
            "type": "bool",
            "optional": true
          }
        ]
      },
      ...
    },
    "ports": {
      "channel_1": {
        "portId": "channel_1",
```

```
        "direction": "tx",
        "portType": "port_type_1",
        "dt": 5e-10,
        "qubitMappings": [
          0
        ]
      },
      ...
    },
    "supportedFunctions": {
      "new_frame": {
        "functionName": "new_frame",
        "arguments": [
          {
            "name": "port",
            "type": "port",
            "optional": false
          },
          {
            "name": "frequency",
            "type": "float",
            "optional": false
          },
          {
            "name": "phase",
            "type": "float",
            "optional": true
          }
        ]
      },
      ...
    },
    "frames": {
      "q0_drive": {
        "frameId": "q0_drive",
        "portId": "channel_1",
        "frequency": 5500000000,
        "centerFrequency": 5500000000,
        "phase": 0,
        "qubitMappings": [
          0
        ]
      },
     ...
```

```
    },
    "supportsLocalPulseElements": false,
    "supportsDynamicFrames": true,
    "supportsNonNativeGatesWithPulses": true,
    "validationParameters": {
      "MAX_SCALE": 1,
      "MAX_AMPLITUDE": 1,
      "PERMITTED_FREQUENCY_DIFFERENCE": 1,
      "MIN_PULSE_LENGTH": 8e-9,
      "MAX_PULSE_LENGTH": 0.00012
    }
  }
}
```

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 () qhpSpecificProperties
  - 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 Hardware-Port 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 () qhpSpecificProperties
  - 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, können nur Frames, die in der Frame-Struktur aufgeführt sind, 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
- 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)
```

Gaußsche

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

τist die Länge der Wellenform, σ ist die Breite der Gaußkurve und 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, σ ist die Breite der Gaußkurve, β ist ein freier Parameter und A ist die Amplitude. 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)
```

SupportsLocalPulseElements:

Beschreibt, ob Impulselemente wie Ports, Frames und Wellenformen lokal in Blöcken definiert werden können. defcal Wenn der Wert istfalse, 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 wir ein nicht systemeigenes Gate wie ein Gate in einem Programm verwenden, ohne zuerst das H 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
- Minimale Pulslänge/Dauer in Sekunden
- Maximale Pulslänge/Dauer 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 Rauschen 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 $^{2*num\_qubits-Matrizen}$. 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.

QubitNeuverkabelung mit OpenQASM 3.0

AmazonBraket unterstützt die physikalische 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 einen Quantenschaltkreis auf Quantencomputern vonRigetti, und aus ausführenOQC, können Sie den Compiler anweisenIonQ, Ihre Schaltungen exakt so auszuführen, wie sie definiert sind, ohne Änderungen vornehmen zu müssen. 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 konservierten Bereiche verwenden. Derzeit IonQ wird OQC nur die wörtliche Kompilierung für die gesamte Schaltung unterstützt, sodass jede Anweisung in der Schaltung in einem wörtlichen Feld enthalten sein muss.

Mit OpenQASM können Sie ein wörtliches Pragma für eine Codebox angeben, die von der Low-Level-Kompilierungsroutine der Hardware unberührt und nicht optimiert ist. Das folgende Codebeispiel zeigt, wie Sie den verwenden. `#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;
```

Weitere Informationen zur wörtlichen Kompilierung finden Sie im Beispielnotizbuch zur Verbatim-Kompilierung.

Die Braket-Konsole

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

Weitere -Quellen

OpenQASM ist in allen Braket-Regionen verfügbar. Amazon

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 im `shots=0` (exakten) Modus unter Verwendung der Methode der adjungierten Differenzierung. Sie können das entsprechende Pragma angeben, um den Gradienten anzugeben, den Sie berechnen möchten, wie im folgenden Beispiel gezeigt.

```
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 Parameter einzeln aufzulisten, können Sie sie auch `all` im Pragma angeben. Dadurch wird der Gradient in Bezug auf alle aufgelisteten `input` Parameter berechnet. Dies kann praktisch sein, wenn die Anzahl der Parameter sehr groß ist. In diesem Fall sieht das Pragma wie das folgende Beispiel aus.

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

Alle beobachtbaren Typen werden unterstützt, einschließlich einzelner Operatoren, Tensorprodukte, hermiteschen Observablen und. `Sum` Der Operator, den Sie zur Berechnung des Gradienten verwenden möchten, muss in den `expectation()` Kapselungsgenerator eingeschlossen werden, und die Qubits, auf die jeder Term wirkt, müssen angegeben werden.

Reichen Sie ein analoges Programm mit's 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-Algorithmus, simuliert durchAquila, 2) AHS-Programmparameter, 3) AHS-Ergebnisinhalt, 4) Fähigkeitsparameter. Aquila Wir empfehlen, die Textsuche Strg+F zu verwenden, um Parameter zu finden, die für Ihre Fragen relevant sind.

Hamiltonisch

Die Aquila Maschine von QuEra simuliert nativ den folgenden (zeitabhängigen) Hamiltonian:

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

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

where

- $H_{\text{drive},k}(t) = (\frac{1}{2}\Omega(t)\, e^{i(\varphi(t)S_{-,k} + \frac{1}{2}\Omega(t)\, e^{-i\varphi(t)S_{+,k}} + (-\delta(t_{+,k})n)}, \text{global } k$
  - 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 =|g389 ¤ 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 (d. h. n=|r¤ ¤ r|)
- $H(t) = -\Delta(t)\, h\, n_{\text{local detuning},k\ \text{local}\ k\ k}$
  - 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
- $V_{\text{vdw},k,l} = C_6/(d_{k,l})\, 6\, n\, n,_{k\ l}$
  - 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-Wechselwirkungskoeffizienten (C). [6]

AHS-Programmschema in Braket

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

```
Program(
    braketSchemaHeader=BraketSchemaHeader(
        name='braket.ir.ahs.program',
        version='1'
    ),
    setup=Setup(
        ahs_register=AtomArrangement(
            sites=[
                [Decimal('0'), Decimal('0')],
                [Decimal('0'), Decimal('4e-6')],
                [Decimal('4e-6'), Decimal('0')],
            ],
```

```
            filling=[1, 1, 1]
        )
    ),
    hamiltonian=Hamiltonian(
        drivingFields=[
            DrivingField(
                amplitude=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('15700000.0'),
Decimal('15700000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
                    ),
                    pattern='uniform'
                ),
                phase=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001')]
                    ),
                    pattern='uniform'
                ),
                detuning=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('-54000000.0'), Decimal('54000000.0')],
                        times=[Decimal('0'), Decimal('0.000001')]
                    ),
                    pattern='uniform'
                )
            )
        ],
        localDetuning=[
            LocalDetuning(
                magnitude=PhysicalField(
                    times_series=TimeSeries(
                        values=[Decimal('0'), Decimal('25000000.0'),
Decimal('25000000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
                    ),
                    pattern=Pattern([Decimal('0.8'), Decimal('1.0'), Decimal('0.9')])
                )
            )
        ]
```

```
    )
)
```

JSON (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, 0.000001000]
                    },
                    "pattern": "uniform"
                },
                "detuning": {
                    "time_series": {
                        "values": [-54000000.0, 54000000.0],
                        "times": [0E-9, 0.000001000]
                    },
                    "pattern": "uniform"
```

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

Hauptfelder

Feld Programm	Typ	description
setup.ahs_register.sites	Liste [Liste [Dezimal]]	Liste der 2D-Koordinaten, an denen die Pinzette Atome einfängt
setup.ahs_register.filling	Liste [int]	Markiert Atome, die die Fallenstellen besetzen, mit 1 und leere Stellen mit 0
Hamiltonian.DrivingFields [] .amplitude.time_series.times	Liste [Dezimal]	Zeitpunkte der Antriebsamplitude, Omega (t)
Hamiltonian.DrivingFields [] .amplitude.time_series.values	Liste [Dezimal]	Werte der Antriebsamplitude, Omega (t)

Feld Programm	Typ	description
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
Hamiltonian.DrivingFields [] .detuning.time_ser ies.times	Liste [Dezimal]	Zeitpunkte der Fahrverstimmung, Delta_Global (t)
Hamiltonian.DrivingFields [] .detuning.time_ser ies.values	Liste [Dezimal]	Werte der Fahrverstimmung, Delta_Global (t)
Hamiltonian.DrivingFields [] .detuning.pattern	str	das räumliche Muster der Fahrverstimmung, delta_Global (t); muss „einheitlich“ sein

Feld Programm	Typ	description
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)
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“	Typ	description
braketSchemaHeader.name	str	Name des Schemas; muss 'braket.ir.ahs.program' sein
braketSchemaHeader.version	str	Version des Schemas

Ergebnisschema der 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	Typ	description
Messungen [] .shotResult.Presequence	Liste [int]	Vorsequenz-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 [] .shotResult.postSequence	Liste [int]	Bits für die Messung nach der Sequenz für jeden Schuss: 0, wenn sich das Atom im Rydberg-Zustand befindet oder die Stelle leer ist, 1, wenn sich das Atom im Grundzustand befindet, gemessen am Ende der Impulssequenzen, die die Quantenentwicklung bestimmen

Metadaten-Felder

Feld für das Ergebnis der Aufgabe	Typ	description
braketSchemaHeader.name	str	Name des Schemas; muss 'braket.t ask_resul t.analog_ hamiltoni an_simula tion_task _result' sein

Feld für das Ergebnis der Aufgabe	Typ	description
braketSchemaHeader.version	str	Version des Schemas
Metadaten der Aufgabe. braketSchemaHeader.name	str	Name des Schemas; muss 'braket.task_result.task_metadata' sein
Aufgaben-Metadaten. braketSchemaHeader. Version	str	Version des Schemas
TaskMetadata.id	str	Die ID der Quantenaufgabe. Für AWS Quantenaufgaben ist dies die Quantenaufgabe ARN.
TaskMetadata.shots	int	Die Anzahl der Schüsse für die Quantenaufgabe

Feld für das Ergebnis der Aufgabe	Typ	description
taskMetadata.Shots.DeviceID	str	Die ID des Geräts, auf dem die Quantenaufgabe ausgeführt wurde. Für AWS Geräte ist dies der Geräte-ARN.
TaskMetadata.Shots.CreatedAt	str	Der Zeitstempel der Erstellung; das Format muss im ISO-8601/RFC3339-String-Format yyyy-MM-DDTHH:mm:ss.sssz sein. Die Standardeinstellung ist None.

Feld für das Ergebnis der Aufgabe	Typ	description
taskMetadata.shots.endedat	str	Der Zeitstempel, zu dem die Quantenaufgabe beendet wurde. Das Format muss im ISO-8601/RFC3339-String-Format yyyy-MM-DDTHH:mm:ss.sssz sein. Die Standardeinstellung ist None.
taskMetadata.shots.Status	str	Der Status der Quantenaufgabe (CREATED, QUEUED, RUNNING, COMPLETED, FAILED). Die Standardeinstellung ist Keine.
TaskMetadata.Shots.FailureReason	str	Der Fehlergrund der Quantenaufgabe. Die Standardeinstellung ist Keine.

Feld für das Ergebnis der Aufgabe	Typ	description
Zusätzliche Metadaten.Action	Braket.IR.AHS.Program_V1.Program	(Siehe den Abschnitt zum Braket AHS-Programmschema)
Zusätzliche Metadaten.Aktion. braketSchemaHeader.Metadaten.Name abfragen	str	Name des Schemas; muss 'braket.task_result.quera_metadata' sein
Zusätzliche Metadaten.Action. braketSchemaHeader.querametadata.version	str	Version des Schemas
Zusätzliche Metadaten.Aktion. numSuccessfulShots	int	Anzahl der vollständig erfolgreichen Schüsse; muss der angeforderten Anzahl von Schüssen entsprechen
Messungen [] .shotMetadata.shotStatus	int	Der Status des Schusses (Erfolg, Teilerfolg, Fehlschlag) muss „Success“ 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“	Typ	description
service.executionWindows [] .executionDay	ExecutionDay	Tage des Ausführun gsfensters; muss 'Jeden Tag', 'Wochentage', 'Wochenende', 'Montag', 'Dienstag', 'Mittwoch', Donnerstag ',' Freitag ',' Samstag 'oder' Sonntag 'sein

Feld „Serviceeigenschaften“	Typ	description
Dienst.AusführungWindows []. windowStartHour	Datetime.time	UTC-24-Stunden-Format der Uhrzeit, zu der das Ausführungsfenster gestartet wird
Service.ExecutionWindows []. windowEndHour	Datetime.time	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.: 'Minute', 'Stunde', 'Schuss', 'Aufgabe'

Metadaten-Felder

Metadaten-Feld	Typ	description
Aktion [] .version	str	Version des AHS-Programmschemas
Aktion [] .actionType	ActionType	Name des AHS-Programmschemas; muss 'braket.ir.ahs.program' sein
Dienst. braketSchemaHeader.name	str	Name des Schemas; muss 'braket.device_schema.devic

Metadaten-Feld	Typ	description
		e_service_properties' sein
Dienst. braketSchemaHeader. Ausführung	str	Version des Schemas
service.deviceDocumentation.imageUrl	str	URL für das Bild des Geräts
Service.DeviceDocumentation.Summary	str	kurze Beschreibung auf dem Gerät
Service. Gerätedokumentation. externalDocumentationUrl	str	URL der externen Dokumentation
service.deviceLocation	str	geografischer Standort des Geräts
Service.AktualisiertAt	datetime	Zeitpunkt, an dem die Geräteeigenschaften zuletzt aktualisiert wurden

Arbeitet mit 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 einfachen Zugriff auf API Braket. Amazon

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

Boto3 bietet die Kernfunktionalität, die zusammen mit dem Amazon Braket Python SDK funktioniert, 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 aufrufen`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
- AWS CLI Profile für Boto3 und das Amazon Braket SDK konfigurieren

Schalten Sie den Amazon Braket Boto3-Client ein

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

> **Note**
>
> Aus Gründen der Abwärtskompatibilität mit älteren Versionen von BraketSchemas wurden OpenQASM-Informationen in Aufrufen weggelassen. `GetDevice` API Um diese Informationen zu erhalten, muss der Benutzeragent eine aktuelle Version von BraketSchemas (1.8.0 oder höher) vorlegen. Das Braket SDK meldet dies automatisch für Sie. Wenn Sie bei der Verwendung eines Braket-SDK keine OpenQASM-Ergebnisse in der `GetDevice` Antwort sehen, müssen Sie möglicherweise die `AWS_EXECUTION_ENV` Umgebungsvariable setzen, um den Benutzeragenten zu konfigurieren. Wie das für die SDKs, Boto3 und Go, Java und/GetDevice funktioniert AWS CLI, finden Sie in den Codebeispielen im Thema OpenQASM-Ergebnisse nicht zurück. JavaScript TypeScript

```
import boto3
import botocore

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

Da Sie nun einen `braket` Client eingerichtet haben, können Sie Anfragen stellen und Antworten vom Braket-Service aus verarbeiten. Amazon 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

- `search_devices(**kwargs)`

Suchen Sie 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']}")
```

Erstelle 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']}")
```

AWS CLI Profile für Boto3 und das Amazon Braket SDK konfigurieren

Das Amazon Braket-SDK stützt sich auf die AWS CLI Standardanmeldedaten, sofern Sie nicht ausdrücklich etwas anderes angeben. Wir empfehlen, bei der Ausführung auf einem verwalteten

Amazon Braket-Notebook die Standardeinstellung beizubehalten, da Sie eine IAM-Rolle angeben müssen, die über Berechtigungen zum Starten der Notebook-Instanz 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 CLIprofile
- 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**`profile`**

Es würde den Rahmen dieses Dokuments sprengen, zu erklären, wie man einen Benutzer erstellt und wie man ein nicht standardmäßiges Profil konfiguriert. 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 Anrufe regionsbasierte 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 angegebene Argument durch einen Wert`region`, der einem der Werte entspricht, AWS-Regionen in denen Amazon Braket verfügbar ist, z. B.`us-east-1`, usw. `us-west-1`

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

Das folgende Beispiel zeigt, wie Sie eine Boto3-Braket-Sitzung initialisieren und ein Gerät in dieser Sitzung instanziieren.

```
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 nutzen, die dem CLI-Profil zugeordnet sind, das Sie zuvor als `profile` angegeben haben.

Pulssteuerung auf Amazon Braket

In diesem Abschnitt wird erklärt, wie Sie die Impulssteuerung auf verschiedenen QPUs in Amazon Braket verwenden.

In diesem Abschnitt:

Braket Pulse

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. Lassen Sie uns zunächst einige Schlüsselkonzepte für die Impulssteuerung in Braket vorstellen.

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

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

Ports

Ein Port ist eine Softwareabstraktion, die jede Eingabe-/Ausgabe-Hardwarekomponente darstellt, die Qubits steuert. Es hilft Hardwareanbietern, eine Schnittstelle bereitzustellen, über die 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)
```

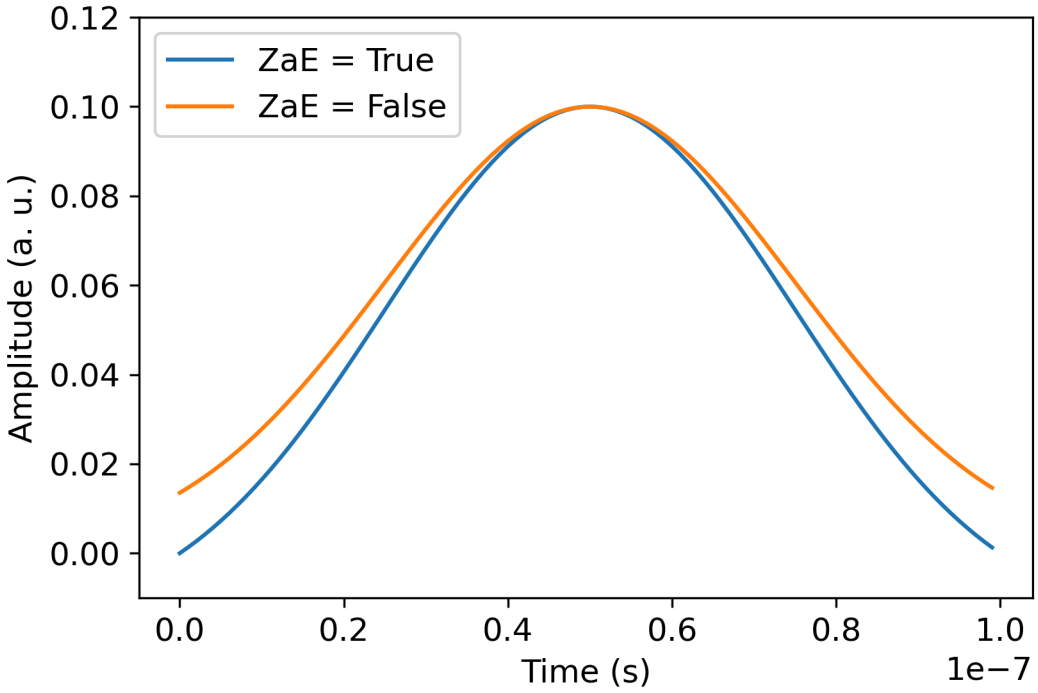
Wellenformen

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
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`.

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



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`

Nachdem wir uns nun mit den grundlegenden Konzepten für den Zugriff auf Impulsebene befasst haben, werden wir uns als Nächstes ansehen, wie eine Schaltung mithilfe von Gates und Impulsen aufgebaut wird.

Die Rollen von Frames und Ports

In diesem Abschnitt werden die vordefinierten Frames und Ports beschrieben, die für jedes Gerät verfügbar sind. Wir werden auch kurz auf die Mechanismen eingehen, die beim Abspielen von Impulsen auf bestimmten Frames eine Rolle spielen.

Rigetti

Frames (Frames)

RigettiGeräte unterstützen vordefinierte Frames, deren Frequenz und Phase so kalibriert sind, dass sie mit dem zugehörigen Qubit in Resonanz stehen. Gemäß der Benennungskonvention `{i}` bezieht `q{i}[_q{j}]_{role}_frame` sich der Begriff auf die erste Qubit-Nummer, `{j}` bezieht sich auf die zweite Qubit-Nummer, falls der Frame dazu dient, eine Wechselwirkung zwischen zwei Qubits zu aktivieren, und `{role}` bezieht sich auf die Rolle des Frames. Die Rollen lauten wie folgt:

- `rf`ist der Frame, der den 0-1-Übergang des Qubits steuert. Impulse werden als transiente Mikrowellensignale mit Frequenz und Phase übertragen, die zuvor über die `set` Funktionen und bereitgestellt wurden. `shift` Die zeitabhängige Amplitude des Signals ergibt sich aus der Wellenform, die auf dem Frame wiedergegeben wird. Der Frame verbindet eine Wechselwirkung mit einem Qubit außerhalb der Diagonale. Weitere Informationen finden Sie bei Krantz et al. und Rahamim et al. .
- `rf_f12`ist dem Übergang von 1 bis 2 ähnlich `rf` und seine Parameter zielen darauf ab.
- `ro_rx`wird verwendet, um ein dispersives Auslesen des Qubits über einen gekoppelten koplanaren Wellenleiter zu erreichen. Die Frequenz, die Phase und der gesamte Parametersatz für die Auslesewellenform sind vorkalibriert. Es wird derzeit über die verwendet`capture_v0`, wofür außer der Frame-ID kein Argument erforderlich ist.
- `ro_tx`dient zur Übertragung von Signalen vom Resonator. Es ist derzeit unbenutzt.
- `cz`ist ein Rahmen, der so kalibriert ist, dass er das `cz` Two-Qubit-Gate aktiviert. Wie bei allen Frames, die einem `ff` Port zugeordnet sind, löst er eine verwirrende Wechselwirkung durch die Flusslinie aus, indem er das einstellbare Qubit des Paares in Resonanz mit seinem Nachbarn moduliert. Weitere Informationen zum Verschränkungsmechanismus finden Sie bei Reagor et al. , Caldwell et al. , und Didier et al. .
- `cphase`ist ein Frame, der so kalibriert ist, dass er das `cphaseshift` Two-Qubit-Gate aktiviert und mit einem Port verbunden ist. `ff` Weitere Informationen zum Verschränkungsmechanismus finden Sie in der Beschreibung des Frames. `cz`
- `xy`ist ein Frame, der so kalibriert ist, dass er die XY (θ) -Gates mit zwei Qubits aktiviert, und der mit einem Port verbunden ist. `ff` Weitere Informationen zum Verschränkungsmechanismus und zur Herstellung von XY-Gattern finden Sie in der Beschreibung des `cz` Frames und bei Abrams et al. .

Da Frames, die auf dem `ff` Port basieren, die Frequenz des einstellbaren Qubits verschieben, werden alle anderen treibenden Frames, die sich auf das Qubit beziehen, um einen Betrag dephasiert, der mit der Amplitude und der Dauer der Frequenzverschiebung zusammenhängt. Folglich müssen Sie diesen Effekt ausgleichen, indem Sie den Frames der benachbarten Qubits eine entsprechende Phasenverschiebung hinzufügen.

Ports

Die Rigetti Geräte bieten eine Liste von Anschlüssen, die Sie anhand der Gerätefunktionen überprüfen können. Die Portnamen folgen der Konvention, die `q{i}_{type}` `{i}` sich auf die Qubit-Nummer und den Typ des Anschlusses `{type}` bezieht. Beachten Sie, dass nicht alle Qubits über einen vollständigen Satz von Anschlüssen verfügen. Die Arten von Anschlüssen sind wie folgt:

- `rf`stellt die Hauptschnittstelle zur Steuerung des Single-Qubit-Übergangs dar. Sie ist mit den Frames `rf` und `rf_f12` D verknüpft. Es ist kapazitiv mit dem Qubit gekoppelt und ermöglicht so Mikrowellenbetrieb im Gigahertz-Bereich.
- `ro_tx`dient zur Übertragung von Signalen an den kapazitiv mit dem Qubit gekoppelten Ausleseresonator. Die Übertragung des Auslesesignals erfolgt achtfach nach Oktagon.
- `ro_rx`dient zum Empfang von Signalen aus dem an das Qubit gekoppelten Ausleseresonator.
- `ff`stellt die induktiv mit dem Qubit gekoppelte Fast-Flux-Leitung dar. Damit können wir die Frequenz des Transmons einstellen. Nur Qubits, die so konzipiert sind, dass sie sehr gut abstimmbar sind, haben einen Port. `ff` Dieser Anschluss dient der Aktivierung der Qubit-Qubit-Interaktion, da zwischen jedem Paar benachbarter Transmonen eine statische kapazitive Kopplung besteht.

Weitere Informationen zur Architektur finden Sie bei Valery et al. .

OQC

Frames (Frames)

OQCGeräte unterstützen vordefinierte Frames, deren Frequenz und Phase so kalibriert sind, dass sie mit dem zugehörigen Qubit in Resonanz stehen. Die Benennungskonvention für diese Frames lautet wie folgt:

- treibender Frame: `{i}` bezieht `q{i}[_q{j}]_{role}` sich dabei auf die erste Qubit-Nummer, `{j}` bezieht sich auf die zweite Qubit-Nummer, falls der Frame dazu dient, eine Zwei-Qubit-Interaktion zu aktivieren, und `{role}` bezieht sich auf die Rolle des Frames, wie unten beschrieben.
- Qubit-Ausleserahmen: `{i}` bezieht `r{i}_{role}` sich dabei auf die Qubit-Nummer und `{role}` bezieht sich auf die Rolle des Frames, wie unten beschrieben.

Wir empfehlen, jeden Frame für seine vorgesehene Rolle wie folgt zu verwenden:

- `drive`wird als Hauptrahmen verwendet, um den 0-1-Übergang des Qubits voranzutreiben. Impulse werden als transiente Mikrowellensignale mit Frequenz und Phase übertragen, die zuvor über die `set` UND-Funktionen bereitgestellt wurden. `shift` Die zeitabhängige Amplitude des Signals ergibt sich aus der Wellenform, die auf dem Frame wiedergegeben wird. Der Frame verbindet eine Wechselwirkung mit einem Qubit außerhalb der Diagonale. Weitere Informationen finden Sie bei Krantz et al. und Rahamim et al. .
- `second_state`entspricht dem `drive` Frame, aber seine Frequenz ist auf die Resonanz beim Übergang von 1 bis 2 abgestimmt.
- `measure`dient zum Auslesen. Die Frequenz, die Phase und der gesamte Parametersatz für die Auslesewellenform sind vorkalibriert. Es wird derzeit über die verwendet`capture_v0`, wofür außer der Frame-ID kein Argument erforderlich ist.
- `acquire`dient zur Erfassung von Signalen aus dem Resonator. Es ist derzeit unbenutzt.
- `cross_resonance`aktiviert die Kreuzresonanz-Wechselwirkung zwischen den Qubits `i` und `j` steuert das Kontrollqubit mit `i` der Übergangsfrequenz des Ziel-Qubits. `j` Folglich wird die Bildfrequenz anhand der Frequenz des Zielqubits festgelegt. Die Wechselwirkung erfolgt mit einer Geschwindigkeit, die proportional zur Amplitude dieses Kreuzresonanzantriebs ist. Verschiedene Arten von Übersprechen führen zu unerwünschten Effekten, die Korrekturen erfordern. Siehe Patterson et al. für weitere Informationen über die Kreuzresonanz-Wechselwirkung mit koaxial geformten Transmon-Qubits („Coaxmonen“).
- `cross_resonance_cancellation`hilft Ihnen dabei, Korrekturen vorzunehmen, um schädliche Effekte zu unterdrücken, die durch Übersprechen hervorgerufen werden, wenn die Kreuzresonanz-Wechselwirkung aktiviert ist. Die anfängliche Bildfrequenz ist auf die Übergangsfrequenz des Kontrollqubits eingestellt. `i` Weitere Informationen zur Stornierungsmethode finden Sie bei Patterson et al. .

Ports

Die OQC Geräte bieten eine Liste von Anschlüssen, die Sie anhand der Gerätefunktionen überprüfen können. Die zuvor beschriebenen Frames sind Ports zugeordnet, die anhand ihrer ID identifiziert werden, `channel_{N}` wobei `{N}` es sich um eine Ganzzahl handelt. Ports sind die Schnittstelle zu Steuerleitungen (Richtung`tx`) und Ausleseresonatoren (Richtung`rx`), die mit Koaxmons verbunden sind. Jedes Qubit ist einer Steuerleitung und einem Ausleseresonator zugeordnet. Der Übertragungsport ist die Schnittstelle für die Manipulation von Einzelqubits und Zwei-Qubits. Der Empfangsport dient zum Auslesen von Qubits.

Hallo Pulse

Hier erfahren Sie, wie Sie ein einfaches Glockenpaar direkt mit Impulsen konstruieren und dieses Impulsprogramm auf dem Rigetti Gerät ausführen. Ein Glockenpaar ist ein Zwei-Qubit-Schaltkreis, der aus einem Hadamard-Gate auf dem ersten Qubit und einem `cnot` Gate zwischen dem ersten und dem zweiten Qubit besteht. Um verschränkte Zustände mit Impulsen zu erzeugen, sind spezifische Mechanismen erforderlich, die vom Hardwaretyp und der Gerätearchitektur abhängen. Wir werden keinen systemeigenen Mechanismus verwenden, um das `cnot` Gate zu erzeugen. Stattdessen werden wir spezifische Wellenformen und Frames verwenden, die das `cz` Gate nativ aktivieren. In diesem Beispiel werden wir mithilfe der nativen Single-Qubit-Gates ein Hadamard-Gate erstellen `rx` und `rz` das Gate mithilfe von Impulsen ausdrücken. `cz`

Lassen Sie uns zunächst die erforderlichen Bibliotheken importieren. Zusätzlich zur `Circuit` Klasse müssen Sie jetzt auch die `PulseSequence` Klasse importieren.

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

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

Als Nächstes instanziieren Sie ein neues Braket-Gerät mit dem Amazon-Ressourcennamen (ARN) des Geräts. Rigetti Aspen-M-3 Das Layout des Rigetti Aspen-M-3 Geräts finden Sie auf der Seite Geräte in der Amazon Braket-Konsole.

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

Da das Hadamard-Gate kein systemeigenes Gate des Rigetti Geräts ist, kann es nicht in Kombination mit Impulsen verwendet werden. Sie müssen es daher in eine Abfolge von Gates `rx` und `rz` nativen Gates zerlegen.

```
import numpy as np
import matplotlib.pyplot as plt
@circuit.subroutine(register=True)
def rigetti_native_h(q0):
    return (
        Circuit()
```

```
        .rz(q0, np.pi)
        .rx(q0, np.pi/2)
        .rz(q0, np.pi/2)
        .rx(q0, -np.pi/2)
    )
```

Für das `cz` Gate verwenden wir eine beliebige Wellenform mit Parametern (Amplitude, Anstiegs-/Fallzeit und Dauer), die vom Hardwareanbieter während einer Kalibrierungsphase vorgegeben wurden. Diese Wellenform wird auf den angewendet. `q10_q113_cz_frame` Eine neuere Version der hier verwendeten Arbiträrwellenform finden Sie unter QCS auf der Website. Rigetti Möglicherweise müssen Sie ein QCS-Konto erstellen.

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

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

Dies sollte Folgendes zurückgeben:

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

Jetzt können wir das `cz` Gate mit der Wellenform konstruieren, die wir gerade definiert haben. Denken Sie daran, dass das `cz` Gate aus einem Phasenwechsel des Ziel-Qubits besteht, wenn sich das Kontrollqubit im Status befindet. |1>

```
phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

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

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

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

Die `a_b_cz_wfm` Wellenform wird auf einem Frame wiedergegeben, der einem Fast-Flux-Port zugeordnet ist. Ihre Aufgabe besteht darin, die Qubit-Frequenz zu verschieben, um eine Qubit-Qubit-Wechselwirkung zu aktivieren. Weitere Informationen finden Sie unter Rollen von Frames und Ports. Da die Frequenz variiert, rotieren die Qubit-Frames mit anderen Geschwindigkeiten als die `rf` Einzel-Qubit-Frames, die unverändert bleiben: Letztere werden dephasiert. Diese Phasenverschiebungen wurden zuvor anhand von Ramsey Sequenzen kalibriert und werden hier als hartcodierte Informationen durch und (volle Periode) bereitgestellt. `phase_shift_a` `phase_shift_b` Wir korrigieren diese Dephasierung, indem wir Anweisungen auf Frames verwendenshift_phase. `rf` Beachten Sie, dass diese Sequenz nur in Programmen funktioniert, in

denen kein XY Frame mit Qubit verwandt b ist a und verwendet wird, da wir die Phasenverschiebung, die bei diesen Frames auftritt, nicht kompensieren. Dies ist bei diesem einzelnen Bell-Paar-Programm der Fall, das nur `rf cz` UND-Frames verwendet. Weitere Informationen finden Sie unter Caldwell et al. .

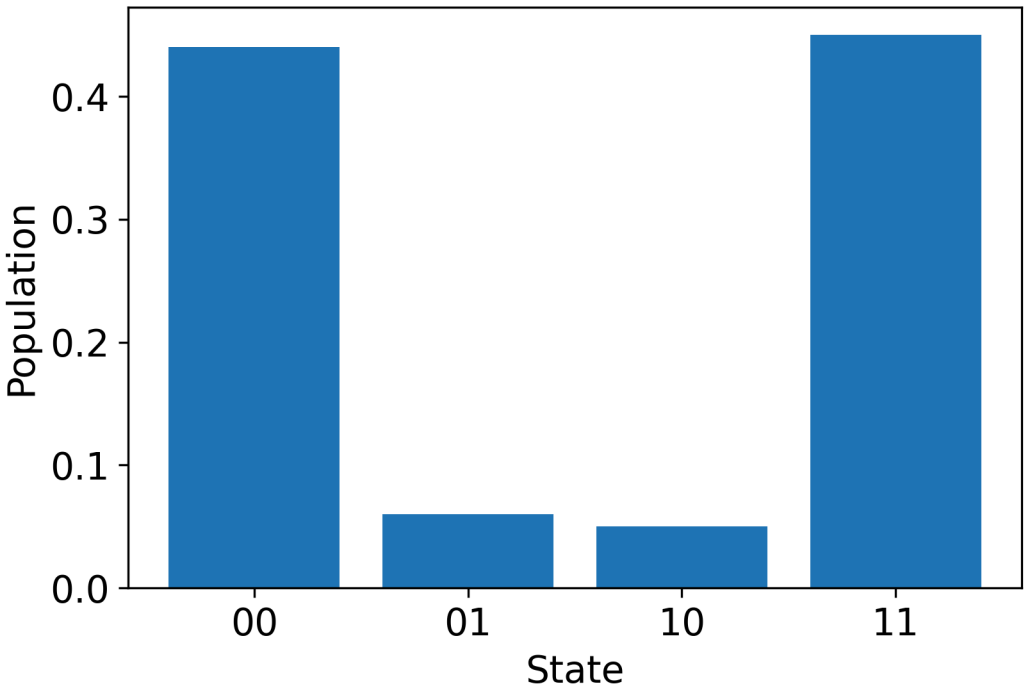
Jetzt sind wir bereit, ein Bell-Paar mit Impulsen zu erstellen.

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

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

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

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

Lassen Sie uns dieses Bell-Paar auf dem Rigetti Gerät ausführen. Beachten Sie, dass für die Ausführung dieses Codeblocks eine Gebühr anfällt. Weitere Informationen zu diesen Kosten finden Sie auf der Seite mit den Amazon Braket-Preisen. Wir empfehlen Ihnen, Ihre Schaltungen mit einer kleinen Anzahl von Schüssen zu testen, um sicherzustellen, dass sie auf dem Gerät ausgeführt werden können, bevor Sie die Anzahl der Schüsse erhöhen.

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

counts = task.result().measurement_counts

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



Hallo Pulse mit OpenPulse

OpenPulseist eine Sprache zur Spezifikation der Pulssteuerung 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.

In diesem Beispiel konstruieren wir eine Bell-Schaltung mit OpenQASM 3.0 und OpenPulse auf einem Gerät, das frequenzabstimmbare Transmonen verwendet. Denken Sie daran, dass ein Bell-Schaltkreis ein Zwei-Qubit-Schaltkreis ist, der aus einem Hadamard-Gate auf dem ersten Qubit besteht, gefolgt von einem Gate zwischen den beiden Qubits. `cnot` Da sich `cnot` Gates von `cz` Gates nur durch eine Basistransformation unterscheiden, definieren wir hier ein Bell-Paar, indem wir stattdessen Hadamard und `cz` Gates verwenden, da das Gerät für diese Demonstration eine einfachere Möglichkeit bietet, Gatter zu erzeugen. `cz`

Beginnen wir mit der Definition des Hadamard-Gates mithilfe systemeigener Gates des Geräts.

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

Für das CZ-Gatter verwenden wir eine beliebige Wellenform mit zuvor festgelegten Parametern (Amplitude, Anstiegs-/Fallzeit und Dauer). Diese Wellenform wird auf die angewendet. `q10_q113_cz_frame`

```
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.4844321964728096,
```

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

Die Dauer der `q10_q113_cz_wfm` Wellenform beträgt 124 Samples, was 124 ns entspricht, da das minimale `dt` Zeitinkrement 1 ns beträgt.

Die `q10_q113_cz_wfm` Wellenform wird auf einem Frame abgespielt, der an einen Fast-Flux-Port gebunden ist. Ihre Aufgabe besteht darin, die Qubit-Frequenz zu verschieben, um eine Qubit-Qubit-Wechselwirkung zu aktivieren. Weitere Informationen finden Sie unter Rollen von Frames und Ports. Da die Frequenz variiert, rotieren die Qubit-Frames unterschiedlich schnell im Vergleich zu `rf` Einzel-Qubit-Frames, die unverändert bleiben: Letztere werden dephasiert. Diese Dephasierung kann mit Ramsey Sequenzen während einer Kalibrierungsphase gemessen und durch Anweisungen auf und Frames kompensiert werden. `shift_phase` `rf` `xy` Weitere Informationen finden Sie bei Caldwell et al. .

Wir können jetzt den Bell-Pair-Schaltkreis ausführen, bei dem wir das `cnot` Gate mit ein paar Hadamard- und Gates zerlegt haben. `cz`

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

Die vollständige OpenQASM 3.0-Darstellung für den Bell-Schaltkreis, der mit einer Kombination aus nativen Gates und Impulsen konstruiert wurde, sieht wie folgt aus.

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.1640330394224107, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.461213110959685 9, 0.4753811409710734, 0.4814117075406603,
 0.4835753359644072 7, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

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

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

Sie können jetzt das Braket-SDK verwenden, um dieses OpenQASM 3.0-Programm mit dem folgenden Code auf dem Rigetti Gerät auszuführen.

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

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

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

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

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

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

```
)
```

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/Aspen-M-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)
.cz(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)
```

Amazon Braket Hybrid Jobs-Benutzerhandbuch

Dieser Abschnitt enthält Anweisungen zum Einrichten und Verwalten von Hybrid-Jobs in Amazon Braket.

Sie können auf Hybridjobs in Braket zugreifen, indem Sie:

- Das Amazon Braket Python SDK.
- Die Amazon Braket-Konsole.
- Die Amazon API Braket.

In diesem Abschnitt:

- Interagieren Sie direkt mit Hybrid-Jobs über API

Was ist ein Hybrid-Job?

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 iterative Algorithmen mit langer Laufzeit, bei denen sowohl klassische als auch Quantenressourcen verwendet werden. Sie senden Ihren Algorithmus zur Ausführung, Braket führt ihn in einer skalierbaren containerisierten Umgebung aus und Sie rufen die Ergebnisse ab, wenn der Algorithmus abgeschlossen ist.

Darüber hinaus profitieren Quantenaufgaben, die im Rahmen eines Hybrid-Jobs erstellt wurden, von einer Warteschlange mit höherer Priorität zu einer Ziel-QPU. Dadurch wird sichergestellt, dass Ihre Quantenaufgaben vor anderen in der Warteschlange verarbeitet und ausgeführt werden. Dies ist besonders vorteilhaft für iterative Hybridalgorithmen, bei denen nachfolgende Aufgaben 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.

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 on-the-fly 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.

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

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 3.10 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 innerhalb des Decorators 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.

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

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

Name des Job

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. 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 ...
```

Die `requirements.txt` Datei kann beispielsweise andere zu installierende Pakete enthalten.

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

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

Angabe der Eingabe-Trainingsdaten

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`

```
@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.

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

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

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

Sie können mehrere Eingabedatenquellen angeben, indem Sie ein Wörterbuch mit Kanalwerten und S3-URIs oder lokalen Pfaden bereitstellen.

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


@hybrid_job(device=None, input_data=input_data)
```

```
def multiple_input_job():
    np.load(get_input_data_dir("input") + "/file.npy")
    np.load(get_input_data_dir("input_2") + "/data.json")
```

> **Note**
>
> 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 werden auf S3 gespeichert

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.

```
@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 Wiederaufnahme von Checkpoints aus

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 übergeben, `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 zuerst 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.

Verteilte Simulatoren mit mehreren Instanzen

Wenn Sie eine große Anzahl von Schaltungen ausführen möchten, 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

Führen Sie einen Hybrid-Job mit Amazon Braket Hybrid Jobs aus

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-basierten Simulatoren verwenden, die mit den eingebetteten Simulatoren auf Braket verfügbar sind (z. B.). `lightning.gpu` Wenn Sie sich für einen der CPU-basierten eingebetteten Simulatoren entscheiden (z. B., oder`braket:default-simulator`)`lightning.qubit`, wird die GPU nicht verwendet, und es können Ihnen unnötige Kosten entstehen.

Erstelle deinen ersten Hybrid-Job

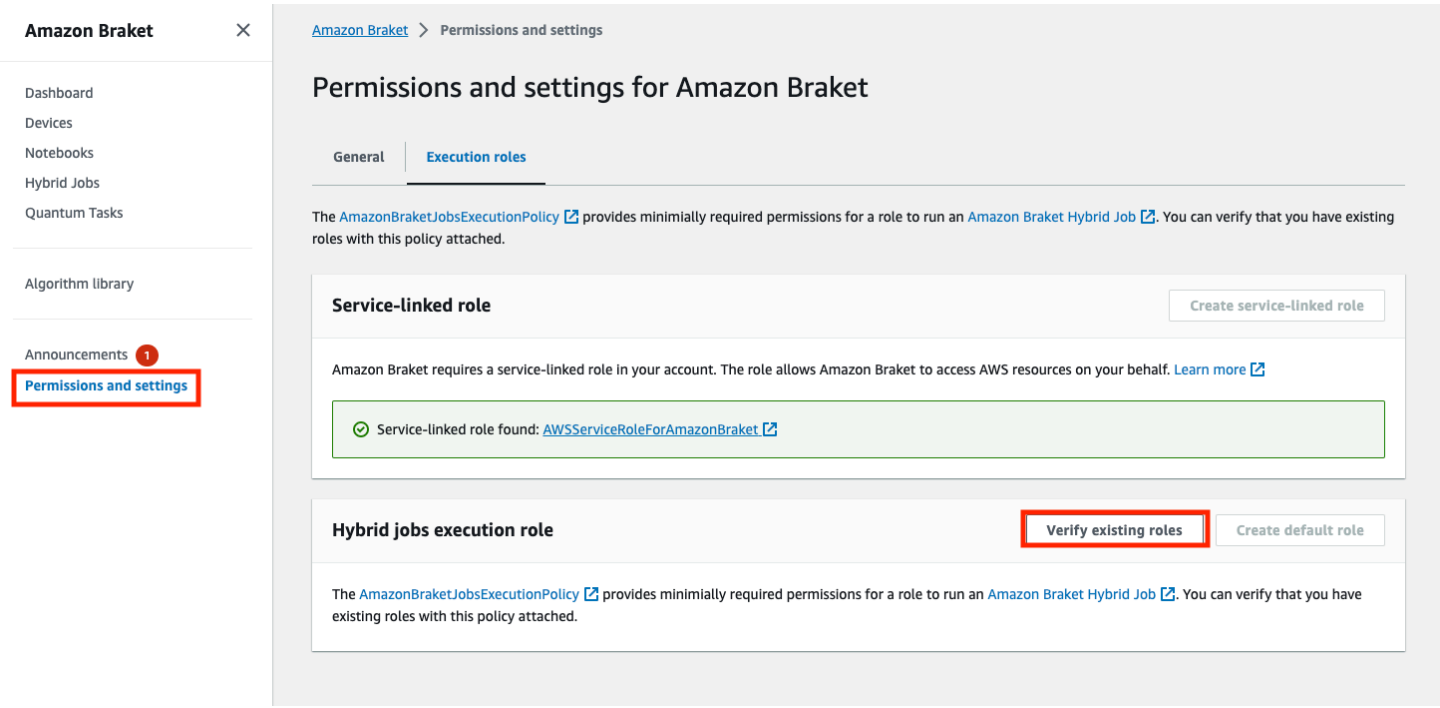
In diesem Abschnitt erfahren Sie, wie Sie mit einem Python-Skript einen Hybrid-Job erstellen. Informationen zum Erstellen eines Hybrid-Jobs aus lokalem Python-Code, z. B. Ihrer bevorzugten integrierten Entwicklungsumgebung (IDE) oder einem Braket-Notizbuch, finden Sie alternativ unterFühren Sie Ihren lokalen Code als Hybrid-Job aus.

In diesem Abschnitt:

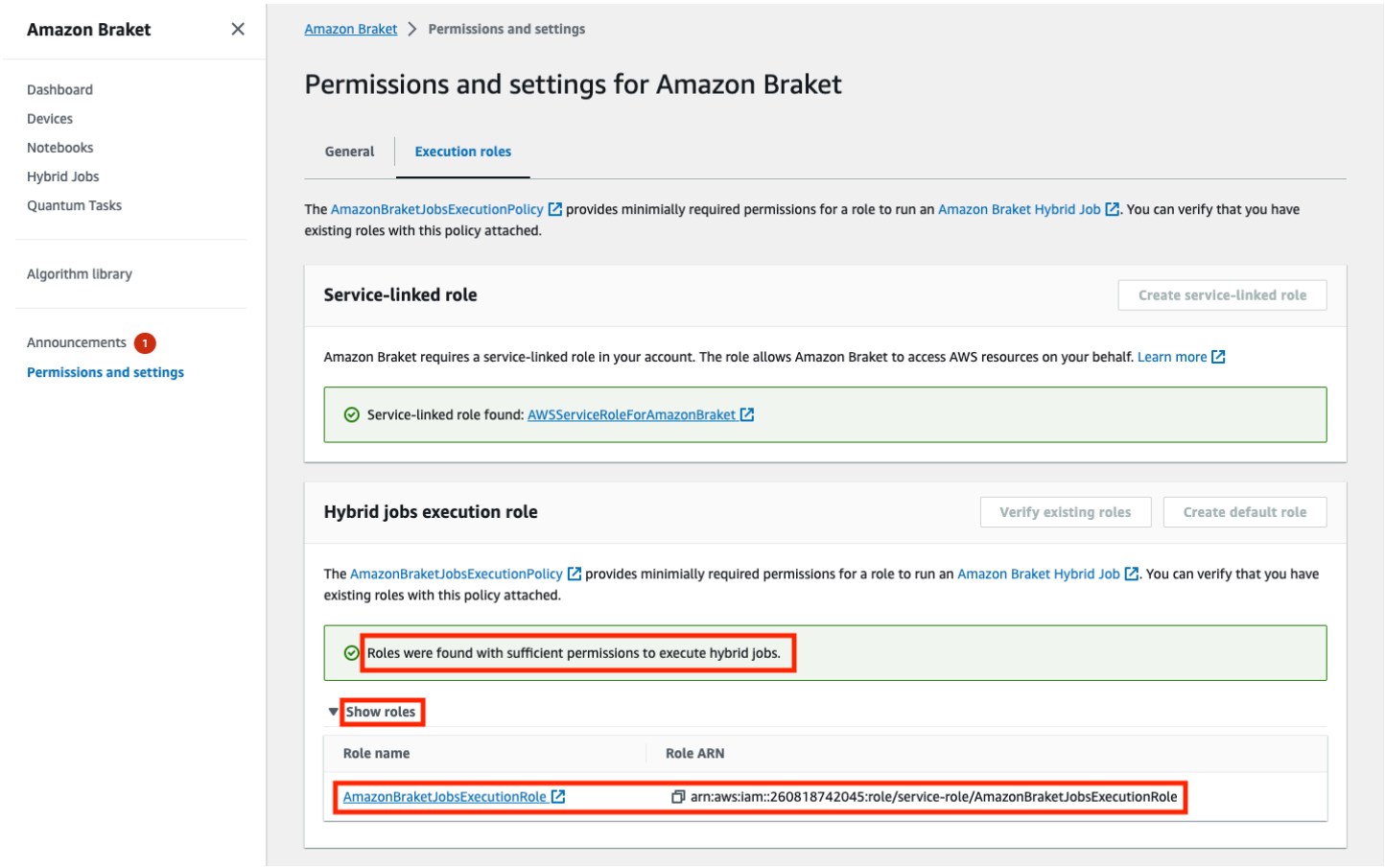
- Legen Sie Berechtigungen fest
- Erstellen und ausführen
- Überwachen Sie die Ergebnisse

Legen Sie Berechtigungen fest

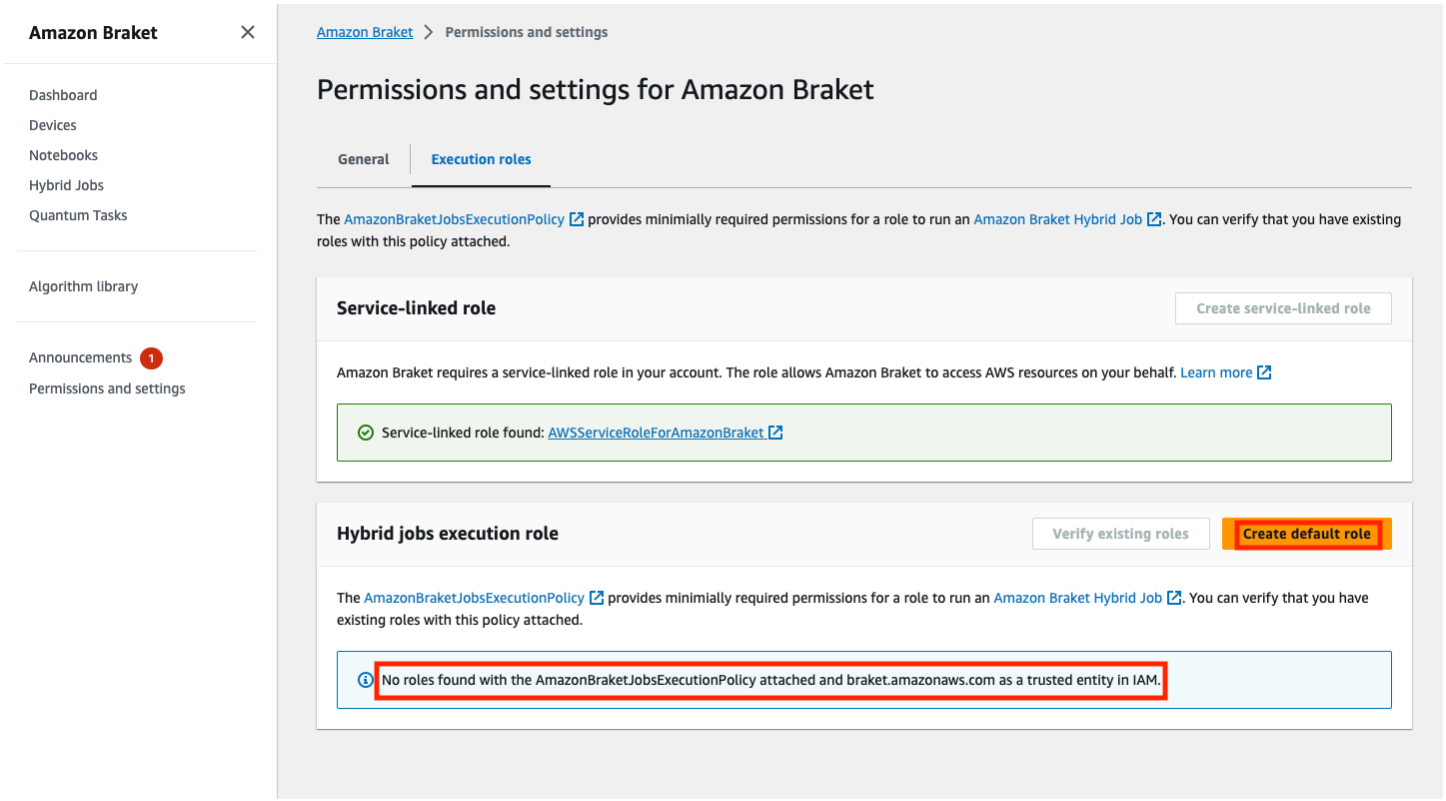
Bevor Sie Ihren ersten Hybridauftrag ausführen, müssen Sie sicherstellen, dass Sie über ausreichende Berechtigungen verfügen, um mit dieser Aufgabe fortzufahren. Um festzustellen, ob Sie über die richtigen Berechtigungen verfügen, wählen Sie im Menü auf der linken Seite der Braket-Konsole die Option 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.



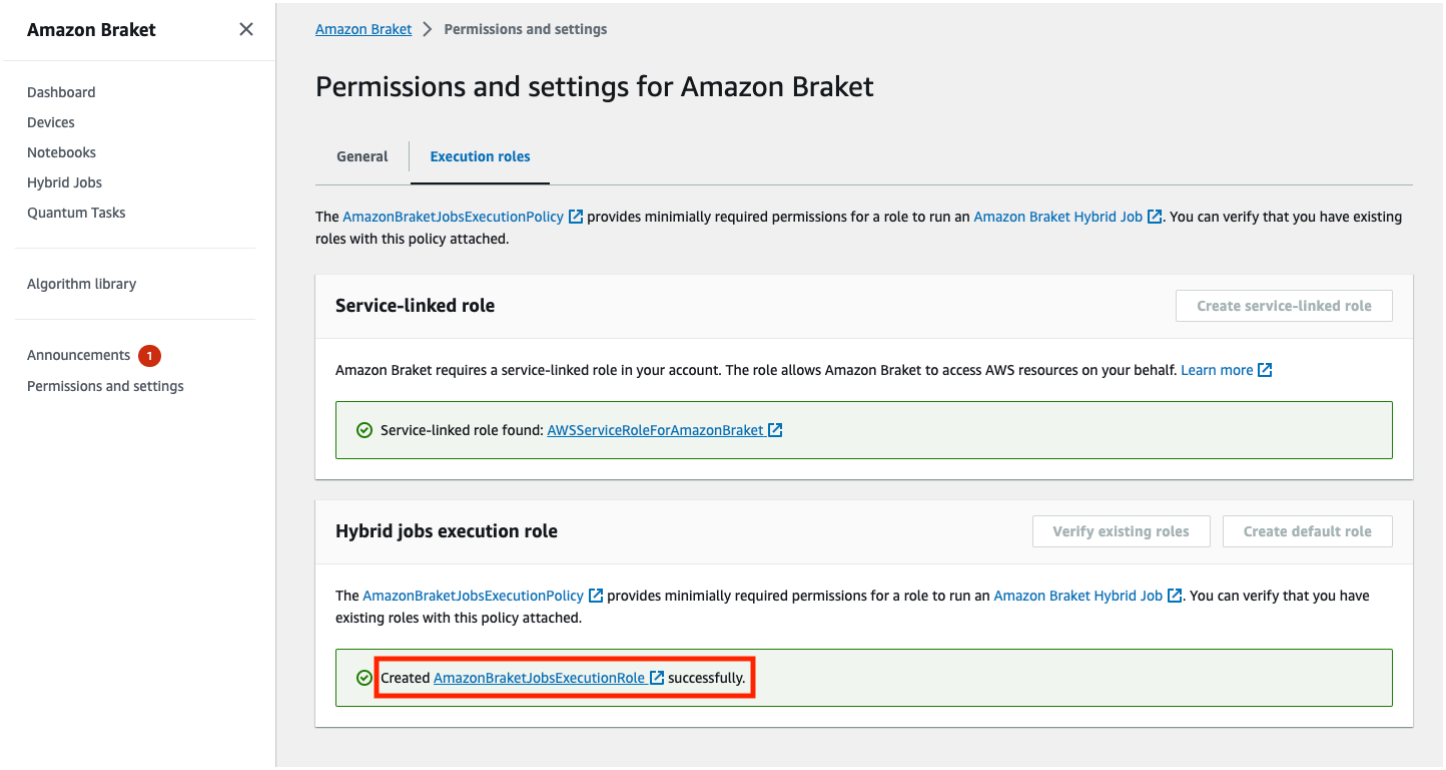
Um zu überprüfen, ob Sie über Rollen mit ausreichenden Berechtigungen für die Ausführung eines Hybrid-Jobs verfügen, klicken Sie auf die Schaltfläche „Bestehende Rolle überprüfen“. Wenn Sie dies tun, erhalten Sie eine Meldung, dass die Rollen gefunden wurden. Um die Namen der Rollen und ihre Rollen-ARNs zu sehen, klicken Sie auf die Schaltfläche Rollen anzeigen.



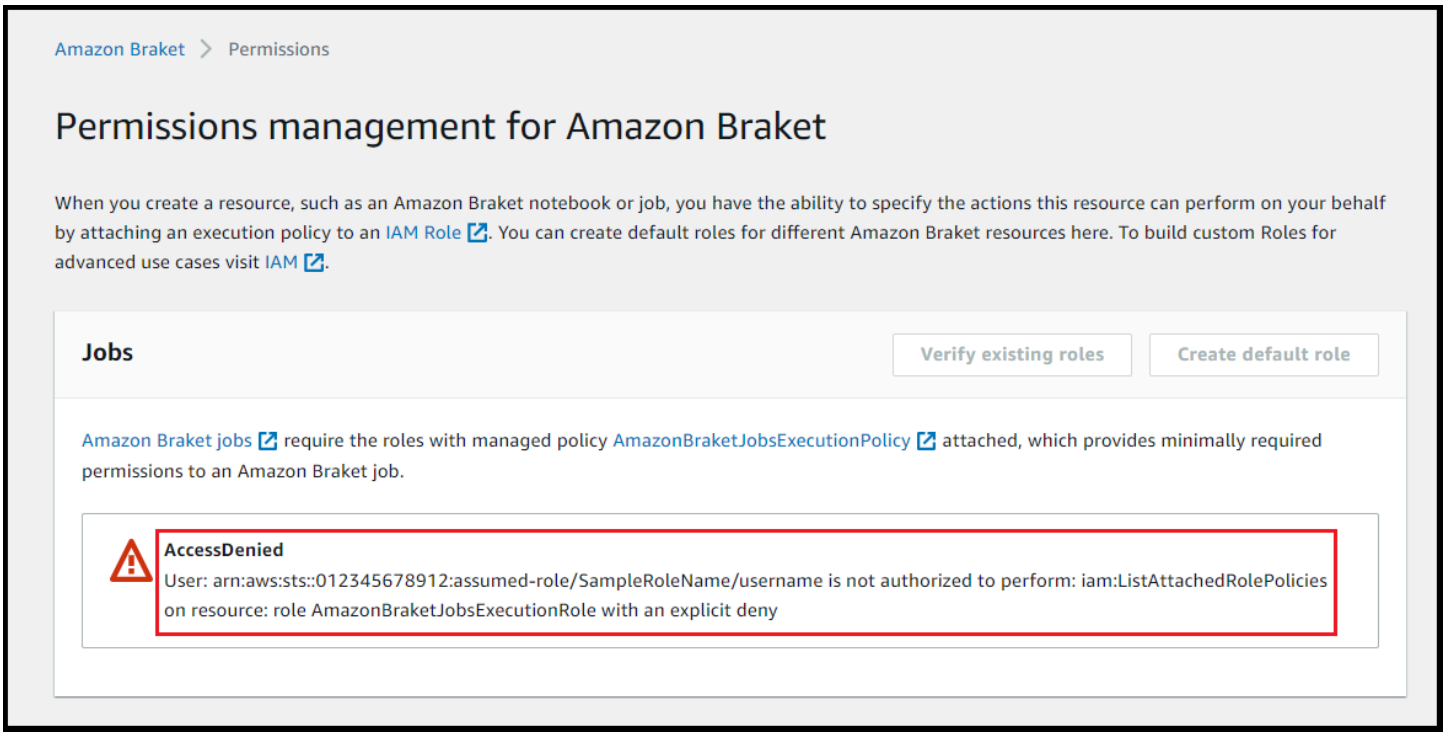
Wenn Sie nicht über eine Rolle mit ausreichenden Berechtigungen verfügen, um einen Hybridjob auszuführen, erhalten Sie eine Meldung, dass keine solche Rolle gefunden wurde. Wählen Sie die Schaltfläche Standardrolle erstellen, um eine Rolle mit ausreichenden Berechtigungen zu erhalten.



Wenn die Rolle erfolgreich erstellt wurde, erhalten Sie eine Bestätigungsmeldung.

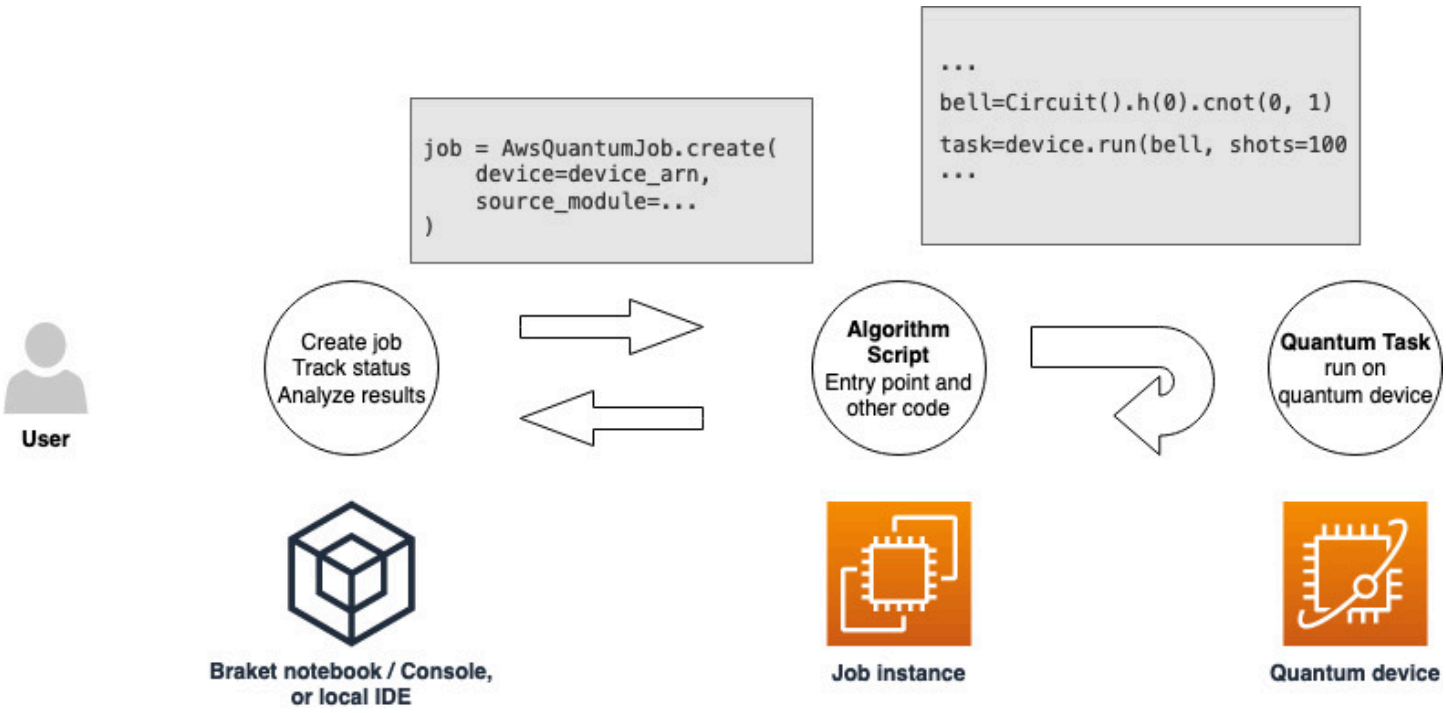


Wenn Sie nicht berechtigt sind, diese Anfrage zu stellen, wird Ihnen der Zugriff verweigert. Wenden Sie sich in diesem Fall an Ihren internen AWS Administrator.



Erstellen und ausführen

Sobald Sie über eine Rolle mit 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 GerätelonQ,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.

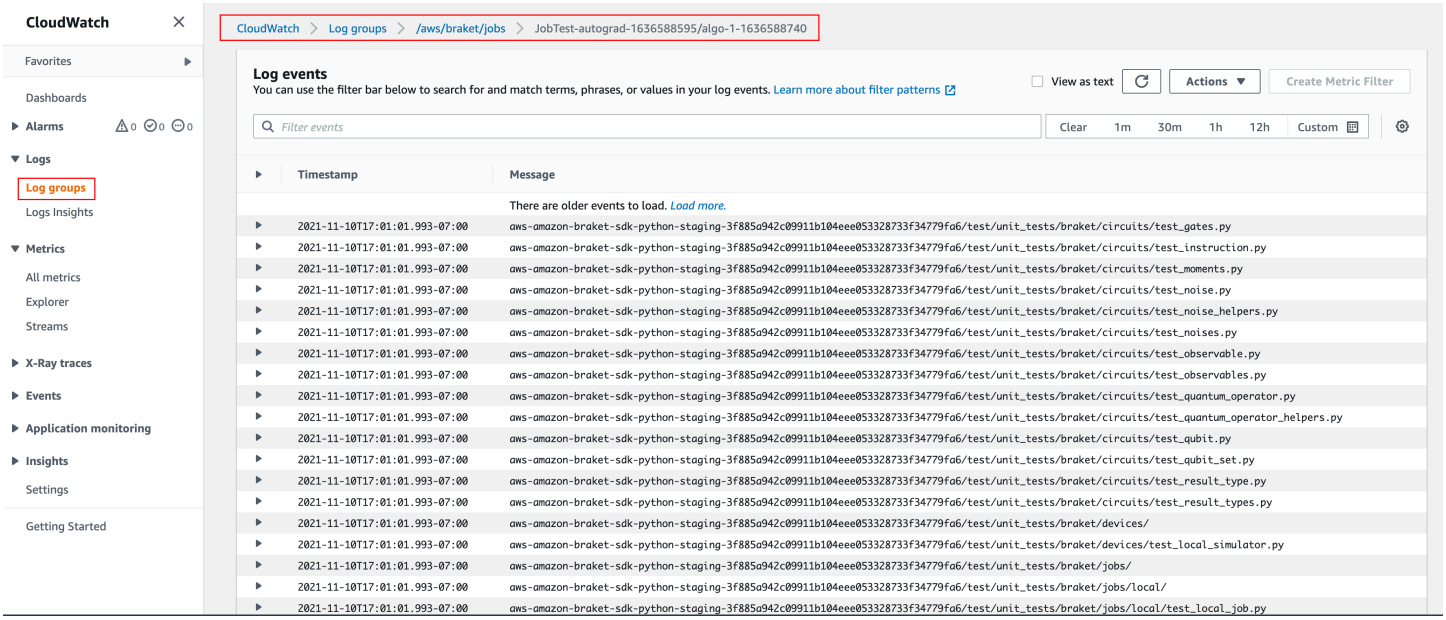
```
job = AwsQuantumJob.create(
    Devices.Amazon.SV1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    wait_until_complete=True,
)
Initializing Braket Job: arn:aws:braket:us-west-2:<accountid>:job/<UUID>
.....................................
.
.
.

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

> **Note**
>
> Sie können Ihr maßgeschneidertes Modul auch mit der Methode AwsQuantumJob.create 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

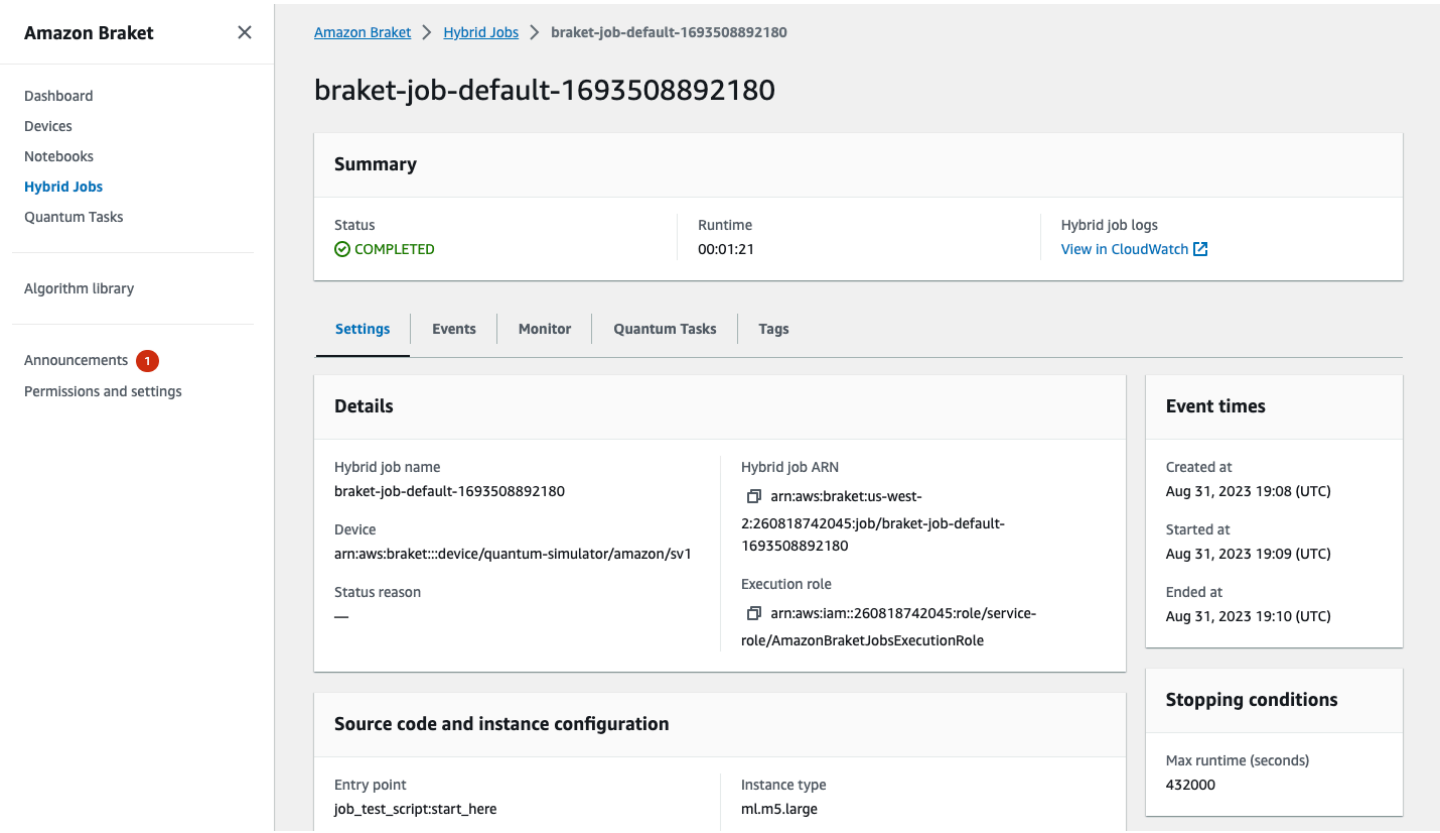
funktionierendes Beispiel finden Sie in der Datei Parallelize_Training_for_Qml.ipynb im Ordner für hybride Jobs im Github-Repository für Amazon Braket-Beispiele.

Überwachen Sie die 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 anzeigen, 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`.

Eingaben, Ausgaben, Umgebungsvariablen und Hilfsfunktionen

Zusätzlich zu der Datei oder den Dateien, aus denen Ihr komplettes Algorithmus-Skript besteht, kann Ihr Hybrid-Job zusätzliche Eingaben und Ausgaben enthalten. Wenn Ihr Hybrid-Job gestartet wird, kopiert Amazon Braket die Eingaben, die im Rahmen der Hybrid-Job-Erstellung bereitgestellt wurden, 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.

AmazonBraket bietet auch mehrere Umgebungsvariablen und Hilfsfunktionen, um die Interaktionen mit Container-Eingaben und -Ausgaben zu vereinfachen.

In diesem Abschnitt werden die wichtigsten Konzepte der vom Amazon Braket Python SDK bereitgestellten `AwsQuantumJob.create` Funktion und ihre Zuordnung zur Container-Dateistruktur erläutert.

In diesem Abschnitt:

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 weitergebenhyperparameters, 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 anzugebenjob-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 istNone, 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 kopiertoutput_data_config. Wenn Sie diesen Wert nicht angeben, 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.

Umgebungsvariablen

AmazonBraket bietet mehrere Umgebungsvariablen, um die Interaktionen mit Container-Eingaben und -Ausgaben zu vereinfachen. Der folgende Code listet die Umgebungsvariablen auf, die Braket verwendet.

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

```
os.environ["AMZN_BRAKET_TASK_RESULTS_S3_URI"]
# the S3 location where the job results would be stored, as specified in CreateJob
 request's OutputDataConfig
os.environ["AMZN_BRAKET_JOB_RESULTS_S3_PATH"]
# the string that should be passed to CreateQuantumTask's jobToken parameter for
 quantum tasks created in the job container
os.environ["AMZN_BRAKET_JOB_TOKEN"]
```

Hilfsfunktionen

AmazonBraket 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.

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

Speichern Sie die Auftragsergebnisse

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 Hybrid-Jobs zu speichern, fügen Sie dem Algorithmus-Skript die folgenden Zeilen hinzu, die mit #ADD kommentiert sind.

```
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!!!!!")
```

Anschließend können Sie sich die Ergebnisse des Jobs aus Ihrem Jobskript anzeigen lassen, 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="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-1234567890123
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Speichern und starten Sie Hybridaufträge mithilfe von Checkpoints neu

Sie können Zwischeniterationen Ihrer Hybrid-Jobs mithilfe von Checkpoints 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, es sei denn, Sie möchten diesen Standardpfad ändern.

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.

```
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.

```
job = AwsQuantumJob.create(
    source_module="source_dir",
```

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

Definieren Sie die Umgebung für Ihr Algorithmus-Skript

AmazonBraket unterstützt drei Umgebungen, die durch Container für Ihr Algorithmus-Skript definiert sind:

- Ein Basiscontainer (der Standard, wenn keiner angegeben `image_uri` ist)
- Ein Container mit Tensorflow und PennyLane
- Ein Container mit und PyTorch PennyLane

Die folgende Tabelle enthält Einzelheiten zu den Containern und den darin enthaltenen Bibliotheken.

Amazon Braket-Behälter

Typ	PennyLane mit TensorFlow	PennyLane mit PyTorch	Pennylane
Basis	292282985366.dkr.ecr.us-east-1.amazonaws.com /:latest amazon-braket-tensorflow-jobs	292282985366.dkr.ecr.us-west-2.amazonaws.com /:aktuell amazon-braket-pytorch-jobs	292282985366.dkr.ecr.us-west-2.amazonaws.com /:aktuell amazon-braket-base-jobs
Geerbte Bibliotheken	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
Zusätzliche Bibliotheken	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas

Typ	PennyLane mit TensorFlow	PennyLane mit PyTorch	Pennylane
	• amazon-braket-schemas • amazon-braket-sdk • ipykernel • Keras • Matplotlib • netzwerke • Openbabel • PennyLane • Protobug • PSI 4 • rsa • PennyLane-Blitzing-GPU • Cu-Quantum	• amazon-braket-schemas • amazon-braket-sdk • ipy Kernel • Keras • Matplotlib • netzwerke • Openbabel • PennyLane • Protobug • PSI 4 • rsa • PennyLane-Blitzing-GPU • Cu-Quantum	• amazon-braket-sdk • awscli • boto3 • ipy Kernel • Matplotlib • netzwerke • numpy • Openbabel • pandas • PennyLane • Protobug • PSI 4 • rsa • scipy

Sie können die Open-Source-Container-Definitionen unter aws/ einsehen und darauf zugreifen. amazon-braket-containers Wählen Sie den Container, der am besten zu Ihrem Anwendungsfall passt. Der Container muss sich in dem befinden, AWS-Region von dem aus Sie Ihren Hybrid-Job aufrufen. Sie geben das Container-Image an, wenn Sie einen Hybrid-Job erstellen, indem Sie Ihrem `create(…)` Aufruf im Hybrid-Job-Skript eines der folgenden drei Argumente hinzufügen. 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.

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

`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_tf = retrieve_image(Framework.PL_TENSORFLOW, "us-west-2")
image_uri_pytorch = retrieve_image(Framework.PL_PYTORCH, "us-west-2")
```

Verwenden von Hyperparametern

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. Beachten Sie, dass die Werte vom Datentyp „Zeichenfolge“ sein müssen. Sie geben die Hyperparameterwerte an, die Sie testen möchten, wenn Sie nach dem optimalen Wertesatz suchen. Der erste Schritt zur Verwendung von Hyperparametern besteht darin, die Hyperparameter als Wörterbuch einzurichten und zu definieren, was im folgenden Code zu sehen ist:

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

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

Anschließend würden Sie die im oben angegebenen Codeausschnitt definierten Hyperparameter zur Verwendung im Algorithmus Ihrer Wahl mit etwas übergeben, das wie folgt aussieht:

```
import time
```

```
from braket.aws import AwsQuantumJob

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

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

> **Note**
>
> Weitere Informationen zu Eingabedaten finden Sie im Abschnitt Eingaben.

Die Hyperparameter würden dann mit dem folgenden Code in das Hybrid-Job-Skript geladen:

```
import json
import os

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

> **Note**
>
> Weitere Informationen darüber, wie Sie Informationen wie die Eingabedaten und den Geräte-ARN an das Hybrid-Job-Skript übergeben, finden Sie auf dieser Github-Seite.

Einige Anleitungen, die sehr nützlich sind, um mehr über die Verwendung von Hyperparametern zu erfahren, finden Sie in den Tutorials QAOA mit Amazon Braket Hybrid Jobs PennyLane und Quantum Machine Learning in Amazon Braket Hybrid Jobs.

Konfigurieren Sie die Hybrid-Job-Instance so, dass sie Ihr Algorithmus-Skript ausführt

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 Instanztyp 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.p3.8xlarge"), # Use NVIDIA Tesla
 V100 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 den InstanceCount angeben und ihn auf mehr als eins setzen. Die Obergrenze liegt bei 5. Sie können beispielsweise 3 Instanzen wie folgt auswählen.

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.p3.8xlarge", instanceCount=3), #
 Use 3 NVIDIA Tesla V100
    ...
    ),
```

Wenn Sie mehrere Instanzen verwenden, sollten Sie erwägen, Ihren Hybrid-Job mithilfe der Datenparallelfunktion zu verteilen. Weitere Informationen zur Verwendung dieses Braket-Beispiels finden Sie im folgenden Beispiel-Notizbuch.

In den folgenden drei Tabellen sind die verfügbaren Instanztypen und Spezifikationen für standardmäßige, rechenoptimierte und beschleunigte Recheninstanzen aufgeführt.

> **Note**
>
> Die standardmäßigen klassischen Compute-Instance-Kontingente für Hybrid-Jobs finden Sie auf dieser Seite.

Standard-Instances	vCPU	Arbeitsspeicher
ml.m5.large (Standard)	2	8 GiB
ml.m5.xlarge	4	16 GiB
ml.m5.2xlarge	8	32 GiB
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB
ml.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

Für Datenverarbeitung optimierte Instances	vCPU	Arbeitsspeicher
ml.c4.xlarge	4	7,5 GiB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB

Für Datenverarbeitung optimierte Instances	vCPU	Arbeitsspeicher
ml.c4.8xlarge	36	192 GiB
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.x groß	4	10,5 GiB
ml.c5n.2 x groß	8	21 GiB
ml.c5n.4x groß	16	42 GiB
ml.c5n.9x groß	36	96 GiB
ml.c5n.18x groß	72	192 GiB

Beschleunigte Rechenins tanzen	vCPU	Arbeitsspeicher
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB

Beschleunigte Rechenins tanzen	vCPU	Arbeitsspeicher
ml.g4dn.xlarge	4	16 GiB
ml.g4dn.2xlarge	8	32 GiB
ml.g4dn.4xlarge	16	64 GiB
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

> **Note**
>
> P3-Instances sind in us-west-1 nicht verfügbar. Wenn Ihr Hybrid-Job die angeforderte ML-Rechenkapazität nicht bereitstellen kann, verwenden Sie eine andere Region.

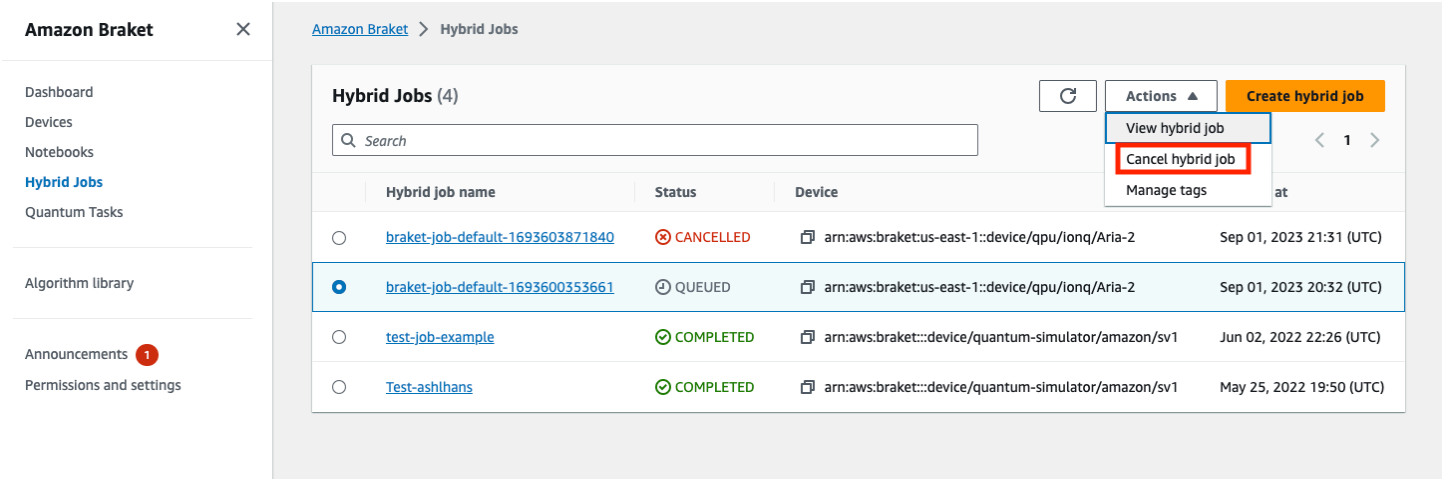
Jede Instanz verwendet eine Standardkonfiguration des Datenspeichers (SSD) von 30 GB. Sie können den Speicher jedoch genauso anpassen, wie Sie den konfigurieren`instanceType`. Das folgende Beispiel zeigt, wie der Gesamtspeicher auf 50 GB erhöht werden kann.

```
from braket.jobs.config import InstanceConfig

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

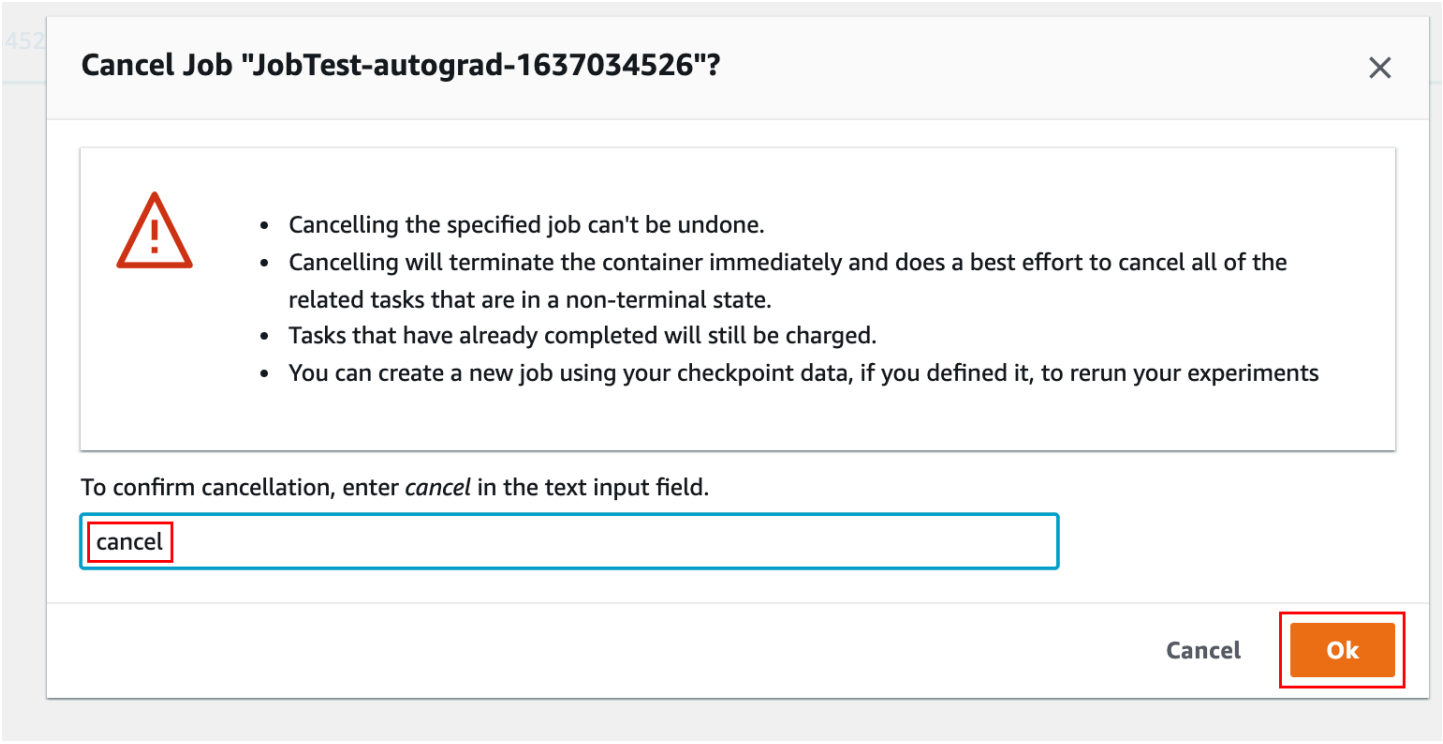
Einen Hybrid-Job stornieren

Möglicherweise müssen Sie einen Hybrid-Job in einem Zustand stornieren, in dem es sich nicht um ein Terminal handelt. 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 den Abbruch zu bestätigen, geben Sie in das Eingabefeld Cancel 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.

Verwendung von parametrischer 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 hybriden 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 Rigetti Computing und Oxford Quantum Circuits mit Ausnahme von Pulspegelprogrammen unterstützt.

PennyLane Mit Amazon Braket verwenden

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?

AmazonBraket ermöglicht es Ihnen, hybride Quantenalgorithmen mit Hilfe des Amazon PennyLane Braket-Plugins 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.

Amazon Braket mit PennyLane

AmazonBraket 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

AmazonBraket bietet eine Vielzahl von Beispiel-Notebooks, für die das Ausführen hybrider Algorithmen nicht auf das PennyLane Plugin angewiesen ist. 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

AmazonBraket Hybrid Jobs bietet jetzt leistungsstarke CPU- und GPU-basierte eingebettete Simulatoren von. 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 lediglich 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 hybird_job
from braket.jobs.config import InstanceConfig


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

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)`

Verwenden Sie Amazon Braket Hybrid Jobs und führen Sie PennyLane einen QAOA-Algorithmus aus

In diesem Abschnitt verwenden Sie das, was Sie gelernt haben, 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 einfache Gradientenabstiegsmethode, um die Parameter so zu optimieren, dass die Kostenfunktion minimiert wird. In diesem Beispiel wird der Einfachheit halber das Problemdiagramm im Algorithmus-Skript generiert. 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 von Rigetti Computing und Oxford Quantum Circuits mit Ausnahme von Pulslevel-Programmen unterstützt.

Beschleunigen Sie Ihre hybriden Workloads mit eingebetteten Simulatoren von PennyLane

Schauen wir uns an, wie Sie eingebettete Simulatoren von PennyLane Amazon Braket Hybrid Jobs aus verwenden können, um Hybrid-Workloads auszuführen. Der GPU-basierte eingebettete Simulator von Pennylane verwendet die Nvidia CuQuantum-Bibliothek`lightning.gpu`, 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.

Verwendung **`lightning.gpu`** für Workloads mit dem Quantum Approximate Optimization Algorithm

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)
```

Lassen Sie uns auf dieser Seite die beiden eingebetteten PennyLane Zustandsvektorsimulatoren `lightning.qubit` (der CPU-basiert) und `lightning.gpu` (der GPU-basiert ist) vergleichen. Sie müssen den Simulatoren einige benutzerdefinierte Gate-Zerlegungen zur Verfügung stellen, um verschiedene Gradienten zu berechnen.

Jetzt sind Sie bereit, das Skript zum Starten des Hybrid-Jobs vorzubereiten. Sie führen den QAOA-Algorithmus mit zwei Instance-Typen aus: `m5.2xlarge` und. `p3.2xlarge` Der `m5.2xlarge` Instanztyp ist vergleichbar mit einem Standard-Entwickler-Laptop. Dabei `p3.2xlarge` handelt es sich um eine beschleunigte Recheninstanz mit einer einzelnen NVIDIA Volta-GPU mit 16 GB Arbeitsspeicher.

Das wird `hyperparameters` für all Ihre Hybrid-Jobs gleich sein. Um verschiedene Instanzen und Simulatoren auszuprobieren, müssen Sie lediglich zwei Zeilen wie folgt ändern.

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

oder:

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

> **Note**
>
> Wenn Sie das mithilfe einer GPU-basierten Instanz **`instance_config`** als angeben, aber den **`device`** eingebetteten CPU-basierten Simulator (**`lightning.qubit`**) wählen, wird

die GPU nicht verwendet. Stellen Sie sicher, dass Sie den eingebetteten GPU-basierten Simulator verwenden, wenn Sie die GPU als Ziel verwenden möchten!

Zunächst können Sie zwei Hybrid-Jobs erstellen und Max-Cut mit QAOA in einem Diagramm mit 18 Eckpunkten lösen. Das entspricht einer 18-Qubit-Schaltung, die relativ klein ist und schnell auf Ihrem Laptop oder der Instanz ausgeführt werden kann. `m5.2xlarge`

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

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

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

Die durchschnittliche Iterationszeit für die `m5.2xlarge` Instance beträgt etwa 25 Sekunden, während sie für die `p3.2xlarge` Instance etwa 12 Sekunden beträgt. Für diesen 18-Qubit-Workflow bietet uns die GPU-Instanz eine 2-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 `p3.2xlarge` Instance 0,06375 USD betragen. Das Ausführen von insgesamt 5 Iterationen, wie Sie es hier getan haben, würde 0,016$ mit der CPU-Instance oder 0,06375$ mit der GPU-Instance kosten — beides ziemlich günstig!

Lassen Sie uns nun das Problem noch schwieriger machen und versuchen, ein Max-Cut-Problem in einem Diagramm mit 24 Eckpunkten zu lösen, was 24 Qubits ergibt. Führen Sie die Hybrid-Jobs erneut auf denselben beiden Instanzen aus und vergleichen Sie die Kosten.

> **Note**
>
> Sie werden feststellen, dass die Ausführung dieses Hybrid-Jobs auf der CPU-Instanz etwa fünf Stunden dauern kann!

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

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

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

Die durchschnittliche Iterationszeit für die `m5.2xlarge` Instanz beträgt ungefähr eine Stunde, während sie für die `p3.2xlarge` Instanz ungefähr zwei Minuten beträgt. Bei diesem größeren Problem ist die GPU-Instanz um eine Größenordnung schneller! Um von dieser Beschleunigung zu profitieren, mussten Sie lediglich zwei Codezeilen ändern und dabei den Instanztyp und den verwendeten lokalen Simulator austauschen. Die Ausführung von insgesamt 5 Iterationen, wie es hier der Fall war, würde etwa 2,27072$ mit der CPU-Instanz oder etwa 0,775625$ mit der GPU-Instanz kosten. Die CPU-Auslastung ist nicht nur teurer, sondern nimmt auch mehr Zeit in Anspruch. Wenn Sie diesen Workflow mit einer GPU-Instanz beschleunigen AWS, auf PennyLane der der eingebettete Simulator von NVIDIA verfügbar ist CuQuantum, können Sie Workflows mit mittleren Qubit-Zahlen (zwischen 20 und 30) zu geringeren Gesamtkosten und in kürzerer Zeit ausführen. Das bedeutet, dass Sie mit Quantencomputing auch bei Problemen experimentieren können, die zu groß sind, um sie schnell auf Ihrem Laptop oder einer Instanz ähnlicher Größe auszuführen.

Maschinelles Quantenlernen und Datenparallelität

Wenn es sich bei Ihrem Workload-Typ um quantenmaschinelles Lernen (QML) handelt, das auf 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-basierte Simulatoren wie PennyLane diese verwenden, um die Leistung gegenüber eingebetteten CPU-basierten Simulatoren `lightning.gpu` zu verbessern.

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.p3.2xlarge')

hyperparameters={"nwires": "10",
```

```
                "ndata": "32",
                ...
}

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

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.p3.16xlarge` 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 8 GPUs in einer Instanz. `ml.p3.16xlarge` 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.p3.16xlarge',
                                 instanceCount=1,
)

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

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

> **Note**
>
> 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.

Vergleichen wir die Laufzeit und die Kosten anhand eines Beispiels, bei dem ein Modell mit einem 26-Qubit-Quantenschaltkreis trainiert wird, um das oben erwähnte Problem der binären Klassifikation zu lösen. Die in diesem `ml.p3.16xlarge` Beispiel verwendete Instanz kostet 0,4692$ pro Minute. Ohne Datenparallelität benötigt der Simulator etwa 45 Minuten, um das Modell für eine Epoche (d. h. über 208 Datenpunkte) zu trainieren, und es kostet etwa 20$. Bei Datenparallelität für 1 Instanz und 4 Instanzen dauert es nur 6 Minuten bzw. 1,5 Minuten, was ungefähr 2,8$ für beide entspricht. Durch die Verwendung von Datenparallelität über 4 Instanzen hinweg verbessern Sie nicht nur die Laufzeit um das 30-fache, sondern reduzieren auch die Kosten um eine Größenordnung!

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. Stattdessen führen Sie Hybrid-Jobs lokal auf Ihrer Braket Notebook-Instanz 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 QPU ausführen.

Um den lokalen Modus zu verwenden, ändern Sie ihn an der Stelle, `AwsQuantumJob` `LocalQuantumJob` an der er auftritt. Um beispielsweise das Beispiel aus Create your first hybrid job (Erstellen Sie Ihren ersten Hybrid-Job) auszuführen, bearbeiten Sie das Hybrid-Job-Skript 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 hier. Außerdem werden im lokalen Modus nicht alle Parameter unterstützt.

Bringen Sie Ihren eigenen Container mit (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). Aber bevor wir uns damit befassen, sollten wir sicherstellen, dass es sich tatsächlich um die richtige Funktion für Ihren Anwendungsfall handelt.

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 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 step-by-step Anleitung dazubring your own container (BYOC), was Sie für Braket Hybrid Jobs benötigen — die Skripts, Dateien und Schritte, um sie zu kombinieren, damit Sie mit Ihren benutzerdefinierten Docker Images loslegen können. Wir bieten 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`, empfehlen wir Ihnen, beim Lesen dieser Anweisungen die Amazon ECR CLI Dockerfile-Dokumentation und gegebenenfalls die Dokumentation zu lesen.

Hier ist ein Überblick darüber, was Sie benötigen:

- Ein Basis-Image für Ihr Dockerfile
- (Optional) Ein modifiziertes Container-Einstiegspunktskript
- ADockerfile, das die erforderliche Software installiert und das Container-Skript enthält

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 nur zusätzliche Software zu einem vorgefertigten Braket-Image hinzufügen, können Sie diesen Abschnitt überspringen.

Um Nicht-Python-Code als Teil Ihres Hybrid-Jobs auszuführen, müssen Sie das Python-Skript ändern, 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.

A`Dockerfile`, das die erforderliche Software installiert und das Container-Skript enthält

> **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-beschleunigten 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, indem jeder installierte Code, der von einem gesteuert wird, ordnungsgemäß zugewiesen wird. MIT license

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 diese 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. Beispielsweise:

```
job = AwsQuantumJob.create(
    source_module="source_dir",
    entry_point="source_dir.algorithm_script:start_here",
    image_uri="111122223333.dkr.ecr.us-west-2.amazonaws.com/my-byoc-container:latest",
    instance_config=InstanceConfig(instanceType="ml.p3.8xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

> **Note**
>
> 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.

Konfigurieren Sie den Standard-Bucket in **`AwsSession`**

Wenn Sie Ihren eigenen Bucket angeben, haben Sie mehr Flexibilität, z. B. `AwsSession` was den Speicherort Ihres Standard-Buckets angeht. Standardmäßig `AwsSession` hat ein den Standard-Bucket-Speicherort von`f"amazon-braket-{id}-{region}"`. Sie können diese Standardeinstellung jedoch überschreiben, wenn Sie eine erstellen`AwsSession`. Benutzer können optional ein `AwsSession` Objekt `AwsQuantumJob.create` mit dem Parameternamen übergeben, `aws_session` wie im folgenden Codebeispiel gezeigt.

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

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

Interagieren Sie direkt mit Hybrid-Jobs über API

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/Aspen-M-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`

Fehlerminimierung

Die Abschwächung von Quantum-Fehlern ist eine Reihe von Techniken, die darauf abzielen, die Auswirkungen von Fehlern auf Quantencomputer zu reduzieren.

Quantum-Geräte unterliegen Umgebungsrauschen, die die Qualität der durchgeführten Berechnungen beeinträchtigen. Fehlertolerantes Quanten-Computing bietet zwar eine Lösung für dieses Problem, aktuelle Quantengeräte sind jedoch durch die Anzahl der Qubits und relativ hohe Fehlerraten begrenzt. Um dies kurzfristig zu beheben, untersuchen Forscher Methoden zur Verbesserung der Genauigkeit verrauschter Quantenberechnungen. Dieser Ansatz, bekannt als Quantenfehlerminderung, beinhaltet die Verwendung verschiedener Techniken, um das beste Signal aus verrauschten Messungsdaten zu extrahieren.

Fehlerminimierung auf IonQ Aria

Die Fehlerminimierung umfasst die Ausführung mehrerer physischer Verbindungen und die Kombination ihrer Messungen, um ein verbessertes Ergebnis zu erzielen. Das IonQ Aria Gerät verfügt über eine Methode zur Fehlerminderung, die als Entzerrung bezeichnet wird.

Durch die Entzerrung wird eine Verbindung mehreren Varianten zugeordnet, die auf verschiedene Qubit-Permutationen oder mit unterschiedlichen Torzerlegungen wirken. Dadurch werden die Auswirkungen von systematischen Fehlern wie Torüberrotationen oder einem einzelnen fehlerhaften Qubit reduziert, indem verschiedene Implementierungen einer Verbindung verwendet werden, die andernfalls die Messungsergebnisse verzerren könnten. Dies geht auf Kosten von zusätzlichem Aufwand für die Anpassung mehrerer Qubits und Toren ein.

Weitere Informationen zur Entzerrung finden Sie unter Verbesserung der Leistung von Quantencomputern durch Symmetrisierung.

> **Note**
>
> Die Entzerrung erfordert mindestens 2 500 Bilder.

Sie können eine Quantenaufgabe mit Entzerrung auf einem -IoQ AriaGerät mit dem folgenden Code ausführen:

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

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

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

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

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

Wenn die Quantenaufgabe abgeschlossen ist, können Sie die Messwahrscheinlichkeiten und alle Ergebnistypen aus der Quantenaufgabe sehen. Die Messwahrscheinlichkeiten und Zählungen aller Varianten werden in einer einzigen Verteilung zusammengefasst. Alle Ergebnistypen, die in der Verbindung angegeben sind, z. B. Erwartungswerte, werden anhand der aggregierten Messungsanzahl berechnet.

Scharfzeichnen

Sie können auch auf Messwahrscheinlichkeiten zugreifen, die mit einer anderen Nachbearbeitungsstrategie namens Verbesserung berechnet wurden. Beim Schärfen werden die Ergebnisse jeder Variante verglichen und inkonsistente Bilder verworfen, wodurch das wahrscheinlichste Messergebnis für alle Varianten bevorzugt wird. Weitere Informationen finden Sie unter Verbesserung der Leistung von Quantencomputern durch Symmetrisierung.

Wichtig ist, dass die Verbesserung die Form der Ausgabeverteilung spärlich ist, mit wenigen Hochwahrscheinlichkeitszuständen und vielen Nullwahrscheinlichkeitszuständen. Es kann die Wahrscheinlichkeitsverteilung verzerren, wenn diese Annahme nicht gültig ist.

Sie können auf die Wahrscheinlichkeiten über eine geschärfte Verteilung im `additional_metadata` Feld auf dem `GateModelTaskResult` im Braket Python SDK zugreifen. Beachten Sie, dass die Erhöhung nicht die Messungsanzahl zurückgibt, sondern eine neu normalisierte Wahrscheinlichkeitsverteilung zurückgibt. Der folgende Codeausschnitt zeigt, wie Sie nach dem Schärfen auf die Verteilung zugreifen.

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

Braket Direct

Mit Braket Direct können Sie dedizierten Zugriff auf verschiedene Quantengeräte Ihrer Wahl reservieren, sich mit Experten für Quantencomputer 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.

In diesem Abschnitt:

Reservierungen

Mit Reservierungen erhalten Sie 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 in Abständen von 1 Stunde möglich und können bis zu 48 Stunden im Voraus ohne zusätzliche Kosten storniert werden. Sie können wählen, ob Sie Quantenaufgaben und Hybridaufträge für eine bevorstehende Reservierung im Voraus in die Warteschlange stellen oder Workloads während Ihrer Reservierung einreichen möchten.

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 Quantum Processing Unit (QPU) ausführen.

Die folgenden Quantencomputer stehen für Reservierungen zur Verfügung:

- Aria von IonQ
- QuEraist Aquila
- Rigetti ist Aspen-M-3

Wann sollte man eine Reservierung nutzen

Wenn Sie den dedizierten Gerätezugriff mit Reservierungen nutzen, können Sie bequem und vorhersehbar genau wissen, wann Ihre Quanten-Workload beginnt und wann die Ausführung 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:

Erstellen Sie 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.
6. Wenn Sie nach Bestätigung Ihrer Reservierung an einem Braket-Experten für eine Sitzung zur Reservierungsvorbereitung interessiert sind, wählen Sie optional Ich bin an einer Vorbereitungssitzung interessiert.

Sie können uns auch kontaktieren, um eine Reservierung vorzunehmen, indem Sie die folgenden Schritte ausführen:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Bereich Geräte und wählen Sie das Gerät aus, das Sie reservieren möchten.
3. Wählen Sie im Abschnitt Zusammenfassung die Option Gerät reservieren aus.
4. Folgen Sie den Schritten 4-6 des vorherigen Verfahrens.

Nachdem Sie das Formular abgeschickt haben, erhalten Sie vom Braket-Team eine E-Mail mit den nächsten Schritten zur Erstellung Ihrer Reservierung. Sobald Ihre Reservierung bestätigt ist, erhalten Sie den Reservierungs-ARN per E-Mail.

> **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.

Wenn Sie Interesse an einer Sitzung zur Reservierungsvorbereitung bekundet haben, kontaktiert Sie das Braket-Team per E-Mail, um eine 30-minütige Sitzung mit einem Braket-Experten zu vereinbaren.

Erledigen Sie Ihr Arbeitspensum mit einer Reservierung

Während einer Reservierung laufen nur Ihre Workloads auf dem Gerät. Um die Quantenaufgaben und Hybridjobs festzulegen, die während einer Gerätereservierung ausgeführt werden sollen, müssen Sie einen gültigen Reservierungs-ARN verwenden.

> **Note**
>
> Reservierungen sind AWS konto- und gerätespezifisch. Nur das AWS Konto, das die Reservierung erstellt hat, kann Ihren Reservierungs-ARN verwenden. Darüber hinaus ist der Reservierungs-ARN nur auf dem reservierten Gerät zu den ausgewählten Start- und Endzeiten gültig.

Um Ihre reservierte Zeit optimal zu nutzen, können Sie sich dafür entscheiden, Aufgaben und Jobs vor Ihrer Reservierung in die Warteschlange zu stellen. Diese Workloads behalten ihren `QUEUED` Status, bis die Reservierung beginnt. Wenn die Reservierung beginnt, werden alle Workloads in der Warteschlange in der Reihenfolge ausgeführt, in der sie eingereicht wurden. Arbeitsaufgaben werden vor eigenständigen Quantenaufgaben priorisiert.

> **Note**
>
> Da während Ihrer Reservierung nur Ihre Workloads ausgeführt werden, ist die Warteschlange für Aufgaben und Jobs, die mit einem Reservierungs-ARN eingereicht wurden, nicht sichtbar.

Codebeispiele für die Erstellung einer Quantenaufgabe für eine Reservierung:

1. Definieren Sie eine Schaltung zur Vorbereitung des GHZ-Zustands im OpenQASM-Format.

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

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

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

c = measure q;
```

2. Erstellen Sie eine Quantenaufgabe mit Ihrer Schaltung und dem Reservierungs-ARN.

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

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

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

program = Program(source=ghz_qasm_string)
```

```
# Reservation ARN will be of the form arn:aws:braket:us-
east-1:<AccountId>:reservation/<ReservationId>
# Example: arn:aws:braket:us-east-1:123456789012:reservation/f17cc20b-1ba4-461f-8854-
de4bb2aa64c1
###############################################################################
# IMPORTANT: If the reservation ARN is not specified, the created task
# queues and runs outside of the reservation.
# (The only exception is when the task is created by the script of a hybrid
#  job that had the reservation ARN passed at the time of its creation.
#  See “Code example for creating a hybrid job for a Braket Direct reservation:”
#  in the following section.)
###############################################################################
my_task = device.run(
    program,
    reservation_arn="arn:aws:braket:us-east-1:<AccountId>:reservation/
<ReservationId>"
)

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

Codebeispiel für die Erstellung eines Hybrid-Jobs für eine Braket Direct-Reservierung:

1. Definieren Sie Ihr Algorithmus-Skript.

```
//algorithm_script.py

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

```
def start_here():

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

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

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

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

2. Erstellen Sie den Hybrid-Job mit Ihrem Algorithmus-Skript und dem Reservierungs-ARN.

```
from braket.aws import AwsQuantumJob

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

3. Erstellen Sie den Hybrid-Job mit dem Remote Decorator..

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

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

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 back-to-back 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 Reservierungsbeginn 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.

Fachkundige Beratung

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 Bereich Expertenberatung. Es stehen die folgenden Optionen für Expertenberatung zur Verfügung:

- 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 use-case-to-device Eignung prüfen, Optionen identifizieren, wie Sie Braket am besten für Ihren Algorithmus nutzen können, und Empfehlungen zur 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 Quantenhardwareanbietern: ionQ, Oxford Quantum Circuits QuEra, und Rigetti bieten jeweils 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.
- AmazonQuantum Solutions Lab (QSL): Das QSL 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 QSL zu kontaktieren, wählen Sie Connect aus und geben Sie Kontaktinformationen und Anwendungsfalldetails ein.
  - Das QSL-Team wird sich mit den nächsten Schritten per E-Mail mit Ihnen in Verbindung setzen.

Experimentelle Fähigkeiten

Um Ihre Forschungsaufgaben voranzubringen, ist es wichtig, schnell Zugang zu neuen innovativen Funktionen zu erhalten. Mit Braket Direct können Sie den Zugriff auf verfügbare experimentelle Funktionen, wie z. B. neue Quantengeräte mit begrenzter Verfügbarkeit, direkt in der Braket-Konsole anfordern.

Einige experimentelle Funktionen funktionieren außerhalb der Standardgerätespezifikationen und erfordern eine praktische Anleitung, die auf Ihren Anwendungsfall zugeschnitten ist. Um sicherzustellen, dass Ihre Workloads auf Erfolgskurs sind, ist der Zugriff auf Anfrage über Braket Direct möglich.

Zugang zu ionQ Forte nur mit Reservierung

Mit Braket Direct erhalten Sie nur Reservierungszugang zur IonQ Forte QPU. Aufgrund seiner begrenzten Verfügbarkeit ist dieses Gerät nur über Braket Direct erhältlich.

Gehen Sie wie folgt vor, um mehr zu erfahren und Zugriff auf IonQ Forte zu beantragen:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Menü Braket Direct aus und navigieren Sie dann unter Experimentelle Funktionen zu ionQ Forte. Wählen Sie „Gerät anzeigen“.
3. Wähle auf der Forte-Gerätedetailseite unter Zusammenfassung die Option Gerät reservieren aus.
4. Gib deine Kontaktinformationen an, einschließlich Name und E-Mail. Geben Sie eine gültige E-Mail-Adresse an, die Sie regelmäßig überprüfen.
5. Geben Sie unter Informieren Sie uns über Ihren Arbeitsaufwand Einzelheiten zum Workload an, der mit Ihrer Reservierung ausgeführt werden soll, z. B. die gewünschte Reservierungsdauer, relevante Einschränkungen oder den gewünschten Zeitplan.
6. (Optional) Wenn Sie nach Bestätigung Ihrer Reservierung an einem Braket-Experten für eine Sitzung zur Reservierungsvorbereitung teilnehmen möchten, wählen Sie Ich bin an einer Vorbereitungssitzung interessiert aus.

Sobald das Formular eingereicht wurde, wird sich das Braket-Team mit Ihnen in Verbindung setzen und Ihnen die nächsten Schritte mitteilen.

> **Note**
>
> Aufgrund der begrenzten Verfügbarkeit von Geräten ist der Zugriff auf Forte begrenzt. Kontaktiere uns, um mehr zu erfahren.

Zugang zur lokalen Abstimmung auf Aquila QuEra

Mit Braket Direct können Sie bei der Programmierung auf der QPU Zugriff auf die Steuerung der lokalen Verstimmung anfordern. QuEra Aquila Mit dieser Funktion können Sie einstellen, wie stark sich das Antriebsfeld auf jedes einzelne Qubit auswirkt.

Gehen Sie wie folgt vor, um mehr zu erfahren und Zugriff auf diese Funktion zu beantragen:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Menü Braket Direct aus und navigieren Sie dann unter Experimentelle Funktionen zu QuEra Aquila — lokale Feinabstimmung. Wählen Sie Get Access aus.
3. Geben Sie Ihre Kontaktinformationen an, einschließlich Name und E-Mail. Geben Sie eine gültige E-Mail-Adresse an, die Sie regelmäßig überprüfen.
4. Geben Sie unter Informieren Sie uns über Ihren Workload Einzelheiten zum Workload an und geben Sie an, wo Sie diese Funktion nutzen möchten.

Zugriff auf große Geometrien auf Aquila QuEra

Mit Braket Direct können Sie bei der Programmierung auf der QPU Zugriff auf erweiterte Geometrien anfordern. QuEra Aquila Mit dieser Funktion können Sie über die Funktionen der Standardgeräte hinaus experimentieren und Geometrien mit erhöhter Gitthöhe spezifizieren.

Gehen Sie wie folgt vor, um mehr zu erfahren und Zugriff auf diese Funktion zu beantragen:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Menü Braket Direct aus und navigieren Sie dann unter Experimentelle Funktionen zu QuEra Aquila — große Geometrien. Wählen Sie Zugriff abrufen aus.
3. Geben Sie Ihre Kontaktinformationen an, einschließlich Name und E-Mail. Geben Sie eine gültige E-Mail-Adresse an, die Sie regelmäßig überprüfen.
4. Geben Sie unter Informieren Sie uns über Ihren Workload Einzelheiten zum Workload an und geben Sie an, wo Sie diese Funktion nutzen möchten.

Zugang zu engen Geometrien auf Aquila QuEra

Mit Braket Direct können Sie bei der Programmierung auf der QPU Zugriff auf erweiterte Geometrien anfordern. QuEra Aquila Mit dieser Funktion können Sie über die Funktionen der Standardgeräte hinaus experimentieren und Gitterreihen mit engeren vertikalen Abständen anordnen.

Gehen Sie wie folgt vor, um mehr zu erfahren und Zugriff auf diese Funktion zu beantragen:

1. Öffnen Sie die Amazon Braket-Konsole.
2. Wählen Sie im linken Menü Braket Direct aus und navigieren Sie dann unter Experimentelle Funktionen zu QuEra Aquila — große Geometrien. Wählen Sie Zugriff abrufen aus.
3. Geben Sie Ihre Kontaktinformationen an, einschließlich Name und E-Mail. Geben Sie eine gültige E-Mail-Adresse an, die Sie regelmäßig überprüfen.
4. Geben Sie unter Informieren Sie uns über Ihren Workload Einzelheiten zum Workload an und geben Sie an, wo Sie diese Funktion nutzen möchten.

Protokollieren und Überwachen

Nachdem Sie eine Quantenaufgabe eingereicht haben, können Sie ihren Status über das Amazon Braket-SDK und die Konsole verfolgen. Wenn die Quantenaufgabe abgeschlossen ist, speichert Braket die Ergebnisse an Ihrem angegebenen Amazon S3 S3-Standort. Die Fertigstellung kann je nach Länge der Warteschlange einige Zeit in Anspruch nehmen, insbesondere bei QPU-Geräten. Zu den Statustypen gehören:

- `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

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 ein asynchroner Vorgang, 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
```

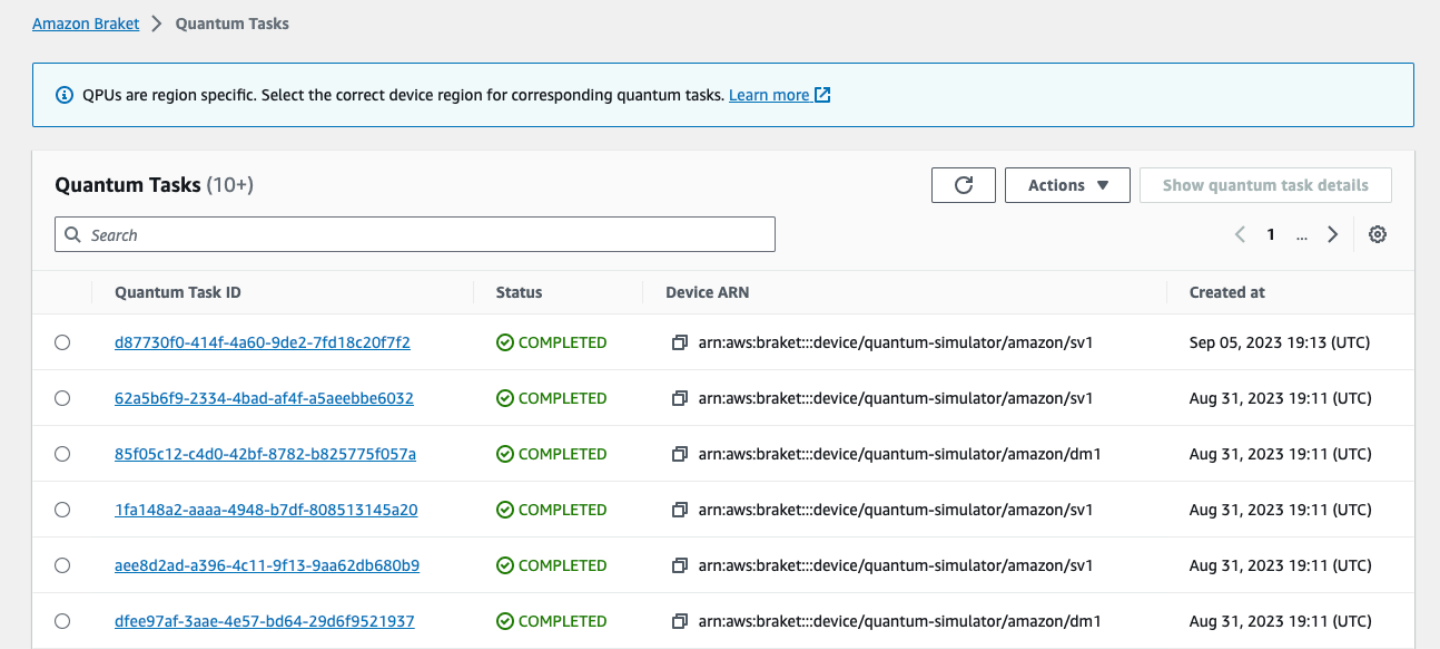
Rufen Sie eine Quantenaufgabe oder ein Ergebnis ab

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.

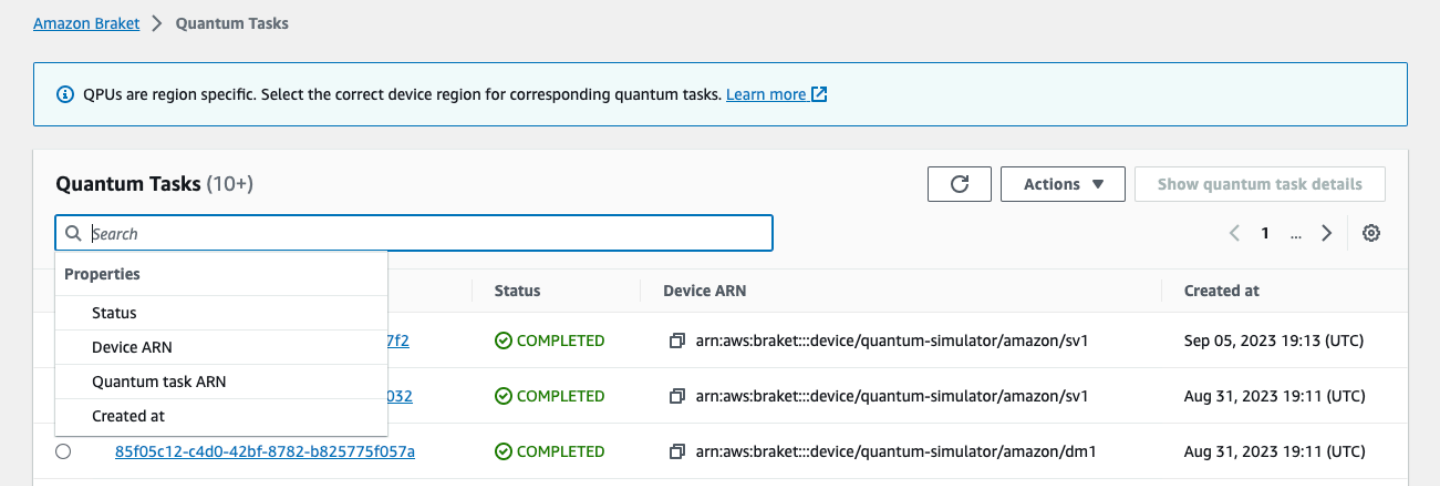
```
from braket.aws import AwsSession, AwsQuantumTask

# 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

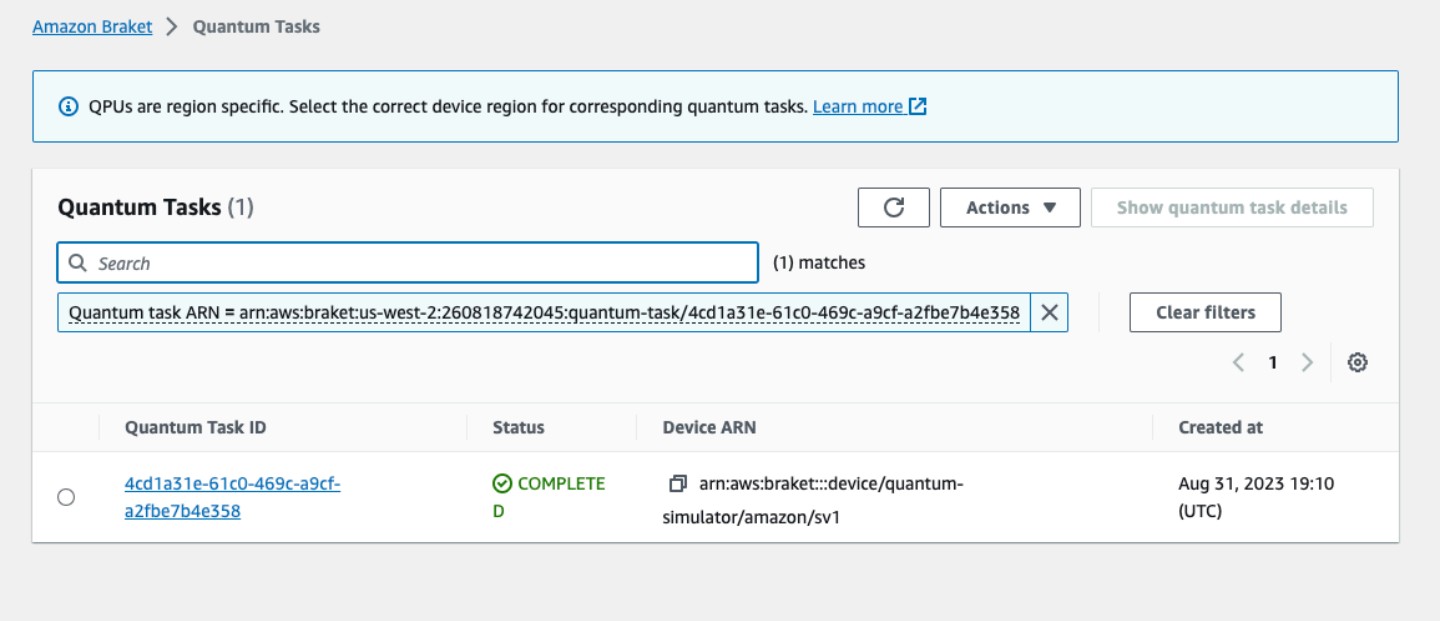
AmazonBraket bietet eine bequeme Möglichkeit, die Quantenaufgabe über die Amazon Braket-Konsole zu überwachen. Alle eingereichten Quantenaufgaben werden im Feld Quantenaufgaben aufgeführt, wie in der folgenden Abbildung dargestellt. Dieser Dienst ist regionsspezifisch, was bedeutet, dass Sie sich nur die Quantenaufgaben ansehen können, die in der jeweiligen Region erstellt wurden. AWS-Region



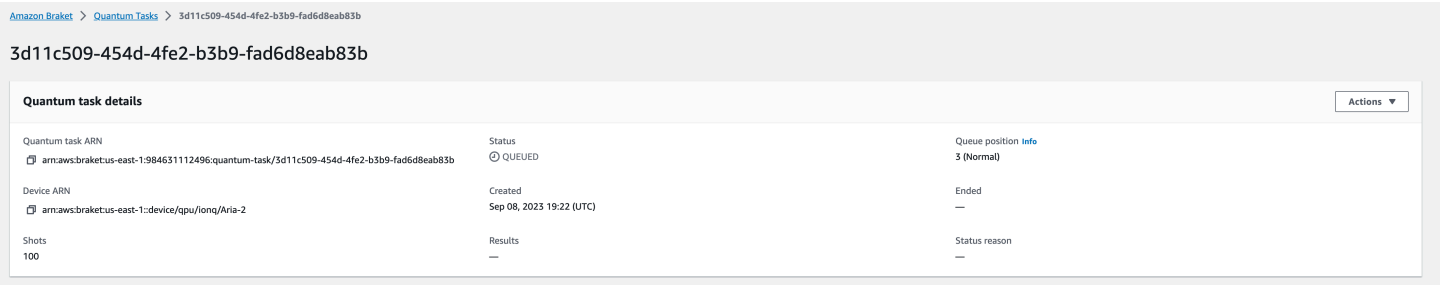
Sie können über die Navigationsleiste nach bestimmten Quantenaufgaben suchen. Die Suche kann auf Quantum Task ARN (ID), Status, Gerät und Erstellungszeit basieren. Die Optionen werden automatisch angezeigt, wenn Sie die Navigationsleiste auswählen, wie im folgenden Beispiel gezeigt.



Die folgende Abbildung zeigt ein Beispiel für die Suche nach einer Quantenaufgabe anhand ihrer eindeutigen Quantenaufgaben-ID, die durch Aufrufen abgerufen werden kann`task.id`.



Darüber hinaus kann, wie in der Abbildung unten zu sehen ist, der Status einer Quantenaufgabe überwacht werden, während sie sich in einem 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.



Quantum-Aufgaben, 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 oder AWS einer AWS-Ressource zuweisen. 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.

Verwenden von Markierungen

Mithilfe von Stichwörtern 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 das Markieren mit Tags (kurz: Tagging). So können Ressourcen aus verschiedenen Services das gleiche Tag zuweisen, um anzugeben, dass die Ressourcen zusammengehören.
- Verfolgen Sie Ihre AWS Kosten. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. Weitere Informationen finden Sie unter Use cost allocation tags (Verwendung von Kostenzuordnungs-Tags) im AWS Billing and Cost Management-Benutzerhandbuch.
- Kontrollieren Sie den Zugriff auf Ihre AWS Ressourcen. Weitere Informationen finden Sie unter Steuern des Zugriffs mithilfe von Tags.

Weitere Informationen zu AWS und Tags

- Allgemeine Informationen zum Tagging, einschließlich Benennungs- und Verwendungskonventionen, finden Sie unter AWSRessourcen mit Tags versehen in der AWSallgemeinen Referenz.
- Informationen zu Einschränkungen beim Tagging finden Sie unter Beschränkungen und Anforderungen für die Benennung von Tags in der AWS Allgemeinen Referenz.
- Bewährte Methoden und Tagging-Strategien finden Sie unter Bewährte Methoden für Tagging und AWS Tagging-Strategien.
- 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 Braket. Amazon

Unterstützte Ressourcen 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 SageMaker Amazon-Ressourcen handelt. Weitere Informationen finden Sie in der Dokumentation unter Notebook-Instance-Metadaten. SageMaker

Tag (Markierung)-Einschränkungen

Die folgenden grundlegenden Einschränkungen gelten für Tags auf 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 Braket-Konsole, die Amazon Braket-Konsole oder die anzeigen, hinzufügen, ändern, Amazon auflisten und löschen. API AWS CLI Weitere Informationen finden Sie in der Amazon Braket-API-Referenz.

Tags hinzufügen

Sie können zu folgenden Zeiten Tags zu Ressourcen hinzufügen, die mit Tags versehen werden können:

- 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 AWSAPI 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 für alle Ressourcen in Amazon Braket anzeigen, die mit Tags versehen werden können, indem Sie die Konsole verwenden, um zu der Task- oder Notizbuchressource zu navigieren, oder indem Sie den Vorgang aufrufen. AWS `ListTagsForResource` API

Sie können den folgenden AWS API Befehl verwenden, um die Tags einer Ressource anzuzeigen:

- AWS API: `ListTagsForResource`

Bearbeiten von Tags

Sie können Tags bearbeiten, indem Sie die Konsole verwenden, um zur Quantum-Task- oder Notebook-Ressource 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`

Tags entfernen

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 CLI-Tagging in Amazon Braket

Wenn Sie mit der AWS CLI arbeiten, finden Sie hier einen Beispielbefehl, der zeigt, wie Sie ein Tag erstellen, das für eine Quantenaufgabe gilt, für die Sie SV1 mit den Parametereinstellungen der Rigetti QPU erstellen. Beachten Sie, dass das Tag am Ende des Beispielbefehls angegeben ist. In diesem Fall erhält Key den Wert `state` und Value den Wert`Washington`.

```
aws braket create-quantum-task --action /
"{\"braketSchemaHeader\": {\"name\": \"braket.ir.jaqcd.program\", /
    \"version\": \"1\"}, /
    \"instructions\": [{\"angle\": 0.15, \"target\": 0, \"type\": \"rz\"}], /
    \"results\": null, /
    \"basis_rotation_instructions\": null}" /
  --device-arn "arn:aws:braket:::device/quantum-simulator/amazon/sv1" /
  --output-s3-bucket  "my-example-braket-bucket-name" /
  --output-s3-key-prefix "my-example-username"  /
  --shots 100   /
  --device-parameters /
  "{\"braketSchemaHeader\": /
     {\"name\": \"braket.device_schema.rigetti.rigetti_device_parameters\", /
      \"version\": \"1\"}, \"paradigmParameters\": /
       {\"braketSchemaHeader\": /
         {\"name\": \"braket.device_schema.gate_model_parameters\", /
```

```
        \"version\": \"1\"}, /
        \"qubitCount\": 2}}" /
        --tags {\"state\":\"Washington\"}
```

Markierung mit der Amazon Braket API

- Wenn Sie Amazon Braket verwenden, API um Tags für eine Ressource einzurichten, rufen Sie die TagResourceAPIan.

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

- Um Tags aus einer Ressource zu entfernen, rufen Sie den auf UntagResourceAPI.

```
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 ListTagsForResourceAPI.

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

Ereignisse und automatisierte Aktionen für Amazon Braket mit Amazon EventBridge

Amazon EventBridge überwacht Statusänderungsereignisse in Amazon Braket-Quantenaufgaben. Ereignisse von Amazon Braket werden fast in Echtzeit EventBridge zugestellt. Sie können einfache Regeln schreiben, die angeben, welche Ereignisse für Sie interessant sind, einschließlich automatisierter Aktionen, die durchgeführt werden sollen, wenn ein Ereignis mit einer Regel übereinstimmt. Zu den automatischen Aktionen, die ausgelöst werden können, gehören:

- Aufrufen einer AWS Lambda-Funktion
- Aktivieren eines AWS Step Functions-Zustandsautomaten
- Benachrichtigen eines Amazon SNS-Themas

EventBridge überwacht diese Ereignisse zur Änderung des Amazon Braket-Status:

- Der Status der Quantenaufgabe ändert sich

AmazonBraket garantiert die Übertragung von Ereignissen zur Änderung des Status von Quantenaufgaben. Diese Ereignisse werden mindestens einmal übermittelt, aber möglicherweise nicht in der richtigen Reihenfolge.

Weitere Informationen finden Sie unter Ereignisse und Ereignismuster unter EventBridge.

In diesem Abschnitt:

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 unter https://console.aws.amazon.com/events/.
3. Erstellen Sie mit den folgenden Werten eine EventBridge Regel:
   - Bei Rule type (Regeltyp) wählen Sie Rule with an event pattern (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 Ereignis zur Änderung des Status von Amazon Braket Quantum Task finden Sie unter Ereignisse und Ereignismuster in. 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"
      }
}
```

Überwachung von Amazon Braket mit Amazon 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.

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

Kennzahl	Beschreibung
Anzahl	Anzahl der Quantenaufgaben.
Latency	Diese Metrik wird ausgegeben, wenn eine Quantenaufgabe abgeschlossen ist. Sie stellt die Gesamtzeit von der Initialisierung bis zur Fertigstellung einer Quantenaufgabe dar.

Dimensionen für Quantum Task Metrics

Die Quanten-Task-Metriken werden mit einer Dimension veröffentlicht, die auf dem `deviceArn` Parameter basiert und die Form arn:aws:braket: ::device/xxx hat.

Unterstützte Geräte

Eine Liste der unterstützten Geräte und Geräte-ARNs finden Sie unter Braket-Geräte.

> **Note**
>
> Sie können die CloudWatch Log-Streams für Amazon Braket-Notebooks anzeigen, indem Sie auf der SageMaker Amazon-Konsole zur Notebook-Detailseite navigieren. Zusätzliche Amazon Braket-Notebook-Einstellungen sind über die SageMaker Konsole verfügbar.

Amazon Braket-API-Protokollierung mit CloudTrail

AmazonBraket ist in einen Dienst integriertAWS CloudTrail, der eine Aufzeichnung der von einem Benutzer, einer Rolle oder einem AWS-Service in Amazon Braket ausgeführten Aktionen bereitstellt. CloudTrail erfasst alle API Aufrufe von Amazon Braket als Ereignisse. Zu den erfassten Anrufen gehören Anrufe von der Amazon Braket-Konsole und Code-Aufrufe der Amazon Braket-Braket-Operationen. 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 CloudTrail gesammelten Informationen können Sie die Anfrage an Amazon Braket, 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 CloudTrailBenutzerhandbuch.

Informationen zu Amazon Braket in CloudTrail

CloudTrail ist auf Ihrem aktiviertAWS-Konto, wenn Sie das Konto erstellen. Wenn in Amazon Braket eine Aktivität stattfindet, wird diese Aktivität zusammen mit anderen AWS-Service Ereignissen in der CloudTrail Ereignishistorie in einem Ereignis aufgezeichnet. Sie können die neusten 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 SystemAWS-Konto, einschließlich der Ereignisse 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 in der AWS-Partition und stellt die Protokolldateien in dem von Ihnen angegebenen Amazon-S3-Bucket bereit. 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"
}
```


Erstellen Sie eine Amazon Braket-Notebook-Instance mit AWS CloudFormation

Sie können AWS CloudFormation damit Ihre Amazon Braket-Notebook-Instances verwalten. Braket-Notebook-Instances werden auf Amazon SageMaker erstellt. 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 Ihnen verwalten. AWS-Konto

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

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

1. Erstellen Sie ein SageMaker Amazon-Lifecycle-Konfigurationsskript.
2. Erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle, von der Sie übernommen SageMaker werden sollen.
3. Erstellen Sie eine SageMaker 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.

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

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

```
BraketNotebookInstanceLifecycleConfig:
    Type: "AWS::SageMaker::NotebookInstanceLifecycleConfig"
    Properties:
      NotebookInstanceLifecycleConfigName: BraketLifecycleConfig-${AWS::StackName}
      OnStart:
      - Content:
          Fn::Base64: |
            #!/usr/bin/env bash

            sudo -u ec2-user -i #EOS
```

```
            aws s3 cp s3://braketnotebookcdk-prod-i-
notebooklccs3bucketb3089-1cysh30vzj2ju/notebook/braket-notebook-lcc.zip braket-
notebook-lcc.zip
            unzip braket-notebook-lcc.zip
            ./install.sh
            EOS

            exit 0
```

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

Wenn Sie eine Braket-Notebook-Instance verwenden, SageMaker führt sie Operationen in Ihrem Namen aus. Nehmen wir zum Beispiel an, Sie führen ein Braket-Notebook mithilfe eines Circuits auf einem unterstützten Gerät aus. SageMaker Führt innerhalb der Notebook-Instanz den Vorgang auf Braket für Sie aus. Die Notebook-Ausführungsrolle definiert genau die Operationen, SageMaker die in Ihrem Namen ausgeführt werden dürfen. Weitere Informationen finden Sie unter SageMaker Rollen im SageMaker Amazon-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 Amazon-Notebook-Instance mit dem Präfix **`amazon-braket-`**

Verwenden Sie das SageMaker Lifecycle-Skript und die in Schritt 1 und Schritt 2 erstellte IAM-Rolle, um eine SageMaker 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 AWS CloudFormationBenutzerhandbuch.

```
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
```

Erweiterte Protokollierung

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()
```

Sicherheit in Amazon Braket

In diesem Kapitel erfahren Sie, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von Amazon Braket anwenden können. Es zeigt Ihnen, wie Sie Amazon Braket konfigurieren, um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere verwenden könnenAWS-Services, die Ihnen helfen, Ihre Amazon Braket-Ressourcen zu überwachen und zu sichern.

Die Sicherheit in der Cloud hat für AWS höchste Priorität. Als AWS-Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sie sind für andere Faktoren verantwortlich, darunter die Sensibilität Ihrer Daten, die Anforderungen Ihres Unternehmens und die geltenden Gesetze und Vorschriften.

Gemeinsame Verantwortung für die Sicherheit

Sicherheit gilt zwischen AWS und Ihnen eine geteilte Verantwortung. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud selbst – AWS ist dafür verantwortlich, die Infrastruktur zu schützen, mit der AWS-Services in der AWS Cloud ausgeführt werden. AWS stellt Ihnen außerdem Services bereit, die Sie sicher nutzen können. Auditoren von Drittanbietern testen und überprüfen die Effektivität unserer Sicherheitsmaßnahmen im Rahmen der AWS-Compliance-Programme regelmäßig. Weitere Informationen zu den Compliance-Programmen, die für Amazon Braket gelten, finden Sie unter AWSServices 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 enthält die Sicherheitskonfigurations- und Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services.

Datenschutz

Das AWS Modell der gilt für den Datenschutz in Amazon Braket. Wie in diesem Modell beschrieben, ist AWS für den Schutz der globalen Infrastruktur verantwortlich, in der die gesamte AWS Cloud ausgeführt wird. 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 Blog-Beitrag AWS-Modell der geteilten Verantwortung und in der DSGVO im AWS-Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, AWS-Konto-Anmeldeinformationen zu schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einzurichten. 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).
- Verwenden Sie SSL/TLS für die Kommunikation mit AWS-Ressourcen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit AWS CloudTrail ein.
- Verwenden Sie AWS-Verschlüsselungslösungen zusammen mit allen Standardsicherheitskontrollen in 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 auf AWS über eine Befehlszeilenschnittstelle oder über eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

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 KonsoleAWS 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

Braket erstellt einen Ressourcentyp: die Quantum-Task-Ressource. Der Amazon-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 SageMaker Amazon-Ressource, die Braket gemeinsam nutzen kann. Um ein Notizbuch 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.

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

```
                        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
                        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
                    ]
                }
            }
        }
    ]
}
```

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.

Über die AmazonBraketFullAccess Richtlinie

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 anzuhängen, 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 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 Notizbücher und Amazon Braket Hybrid-Jobs zu erstellen, darf Ihre Ressourcenanzahl die Kontingente für Ihr Konto nicht überschreiten.

Inhalt der Richtlinie

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

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

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

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

```
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:CreateLogStream",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
        },
        {
            "Effect": "Allow",
            "Action": "cloudwatch:PutMetricData",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cloudwatch:namespace": "/aws/braket"
                }
            }
        }
    ]
}
```

Über die AmazonBraketJobsExecutionPolicy Richtlinie

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

```
"Version": "2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "s3:GetObject",
   "s3:PutObject",
   "s3:ListBucket",
   "s3:CreateBucket",
   "s3:PutBucketPublicAccessBlock",
   "s3:PutBucketPolicy"
  ],
  "Resource": "arn:aws:s3:::amazon-braket-*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "ecr:GetDownloadUrlForLayer",
   "ecr:BatchGetImage",
   "ecr:BatchCheckLayerAvailability"
  ],
  "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "ecr:GetAuthorizationToken"
  ],
  "Resource": "*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "braket:CancelJob",
   "braket:CancelQuantumTask",
   "braket:CreateJob",
   "braket:CreateQuantumTask",
   "braket:GetDevice",
   "braket:GetJob",
```

```
    "braket:GetQuantumTask",
    "braket:SearchDevices",
    "braket:SearchJobs",
    "braket:SearchQuantumTasks",
    "braket:ListTagsForResource",
    "braket:TagResource",
    "braket:UntagResource"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
   "Resource": "arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*",
   "Condition": {
    "StringLike": {
     "iam:PassedToService": [
      "braket.amazonaws.com"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:ListRoles"
   ],
   "Resource": "arn:aws:iam::*:role/*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "logs:GetQueryResults"
   ],
   "Resource": [
    "arn:aws:logs:*:*:log-group:*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "logs:PutLogEvents",
```

```
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:GetLogEvents",
    "logs:DescribeLogStreams",
    "logs:StartQuery",
    "logs:StopQuery"
   ],
   "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
  },
  {
   "Effect": "Allow",
   "Action": "cloudwatch:PutMetricData",
   "Resource": "*",
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": "/aws/braket"
    }
   }
  }
 ]
}
```

Beschränken Sie den Benutzerzugriff auf bestimmte Geräte

Um den Zugriff bestimmter Benutzer auf bestimmte Braket-Geräte zu beschränken, können Sie einer bestimmten IAM Rolle eine Richtlinie zum Verweigern von Berechtigungen hinzufügen.

Die folgenden Aktionen können mit solchen Berechtigungen 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 ein. AWS-Konto `123456789012`

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

```
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
  ]
}
```

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>:<account id>:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:<account id>:device/quantum-simulator/<provider>/<device_id>`

Im Folgenden finden Sie Beispiele für verschiedene Arten des Gerätezugriffs

- So wählen Sie alle QPUs in allen Regionen aus: `arn:aws:braket:*:*:device/qpu/*`
- Um NUR alle QPUs in der Region US-West-2 auszuwählen: `arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- Um alle QPUs NUR 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:* :123456789012:device/quantum-simulator/*`
- So beschränken Sie den Zugriff auf das IonQ Harmony Gerät in der Region US-East-1: `arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- So beschränken Sie den Zugriff auf Geräte eines bestimmten Anbieters (z. B. auf Rigetti QPU Geräte): `arn:aws:braket:* :123456789012:device/qpu/rigetti/*`

- Um den Zugriff auf das TN1 Gerät einzuschränken:
  `arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

Amazon Braket-Updates für AWS verwaltete Richtlinien

Die folgende Tabelle enthält Einzelheiten zu den Aktualisierungen der AWS verwalteten Richtlinien für Braket, seit dieser Service begonnen hat, diese Änderungen zu verfolgen.

Änderung	Beschreibung	Date (Datum)
AmazonBraketFullAccess- Richtlinie für vollen Zugriff für Braket	Es wurden die GetMetric Data Aktionen servicequotas: GetServiceQuota und cloudwatc h: hinzugefügt, die in die Richtlini e aufgenommen werden sollen. AmazonBraketFullAccess .	24. März 2023
AmazonBraketFullAccess- Richtlinie für vollen Zugriff für Braket	Braket hat iam angepasst : PassRole Berechtigungen AmazonBraketFullAccess zum Einschließen des Pfads. `service-role/`	29. November 2021
AmazonBraketJobsExecutionPolicy- Richtlinie zur Ausführung von Hybrid-Jo bs 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 Richtlini en nachzuverfolgen.	29. November 2021

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 Name`amazon-braket-Alice`).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
```

```
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
${aws:username}*"
      ]
    }
  ]
}
```

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.

```
{
  "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.

Servicegebundene Rolle mit Amazon Bracket

Wenn Sie Amazon Braket aktivieren, wird in Ihrem Konto eine mit Diensten verknüpfte 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 Rolle von Amazon Braket ist so vordefiniert, dass sie alle Berechtigungen enthält, die Braket benötigt, um andere in AWS-Services Ihrem Namen anzurufen.

Eine serviceverknüpfte Rolle erleichtert die Einrichtung von Amazon Braket, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Amazon Braket definiert die Berechtigungen seiner dienstverknüpften 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 Funktion AWS Identity and Access Management (IAM) für serviceverknüpfte Rollen. Informationen zu anderen DienstenAWS-Services, die dienstverknüpfte Rollen unterstützen, finden Sie unter AWSServices, die mit IAM funktionieren. Suchen Sie in der Spalte Dienstverknüpfte Rolle nach den Diensten, für die Ja steht. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer servicegebundenen Rolle für diesen Service anzuzeigen.

Serviceverknüpfte Rollenberechtigungen für Amazon Bracket

Amazon Braket verwendet die `AWSServiceRoleForAmazonBraket` serviceverknüpfte Rolle, die darauf vertraut, dass die Entität braket.amazonaws.com die Rolle übernimmt.

Sie müssen Berechtigungen konfigurieren, damit eine IAM-Entität (z. B. eine Gruppe oder Rolle) eine dienstverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter Berechtigungen für serviceverknüpfte Rollen.

Der serviceverknüpften Rolle in Amazon Braket werden standardmäßig die folgenden Berechtigungen gewährt:

- Amazon S3 — Berechtigungen zum Auflisten der Buckets in Ihrem Konto und zum Abrufen von Objekten in 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 Log-Streams und zum Einfügen von Ereignissen in die für Amazon Braket erstellte Protokollgruppe.

Die folgende Richtlinie ist der **`AWSServiceRoleForAmazonBraket`**dienstverknüpften Rolle zugeordnet:

```
{"Version": "2012-10-17",
    "Statement": [
        {"Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:ListBucket"
            ],
            "Resource": "arn:aws:s3:::amazon-braket*"
        },
        {"Effect": "Allow",
            "Action": [
                "logs:Describe*",
                "logs:Get*",
                "logs:List*",
                "logs:StartQuery",
                "logs:StopQuery",
                "logs:TestMetricFilter",
                "logs:FilterLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/braket/*"
        },
        {"Effect": "Allow",
            "Action": "braket:*",
            "Resource": "*"
        },
        {"Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/braket.amazonaws.com/
AWSServiceRoleForAmazonBraket*",
            "Condition": {"StringEquals": {"iam:AWSServiceName": "braket.amazonaws.com"
                }
            }
```

```
        }
    ]
}
```

Resilienz in Amazon Braket

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones.

Jede Region bietet mehrere Verfügbarkeitszonen, die physisch getrennt und isoliert sind. Diese Availability Zones (AZs) sind über Netzwerke mit niedriger Latenz, hohem Durchsatz und hochredundanten Netzwerken miteinander verbunden. Daher sind Availability Zones hochverfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Sie können Anwendungen und Datenbanken entwerfen und betreiben, die automatisch und ohne Unterbrechung ein Failover zwischen AZs durchführen.

Weitere Informationen zu AWS-Regionen und Verfügbarkeitszonen finden Sie unter AWSGlobal Infrastructure.

Konformitätsvalidierung für Amazon Braket

Externe Prüfer überprüfen regelmäßig die Sicherheit und Konformität von Amazon Braket und unsere Integration mit externen Hardwareanbietern. Eine up-to-date Liste der Compliance-Informationen für Braket finden Sie unter AWS-ServicesGeltungsbereich nach Compliance-Programm. Allgemeine Informationen finden Sie unter AWSCompliance.

Die Auditberichte von Drittanbietern lassen sich mit herunterlade AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen in AWS Artifact.

> **Note**
>
> AWSDie Compliance-Berichte decken keine QPUs von Hardwareanbietern von Drittanbietern ab, die sich dafür entscheiden können, ihre eigenen unabhängigen Prüfungen durchzuführen.

Ihre Compliance-Verantwortung bei der Nutzung von Amazon Braket hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und

Vorschriften ab. AWSstellt die folgenden Ressourcen bereit, um Sie bei der Einhaltung von Vorschriften zu unterstützen:

- Kurzanleitungen für Sicherheit und Compliance – In diesen Bereitstellungsleitfäden finden Sie wichtige Überlegungen zur Architektur sowie die einzelnen Schritte zur Bereitstellung von sicherheits- und Compliance-orientierten Basisumgebungen in AWS.
- AWS-Compliance-Ressourcen – Diese Arbeitsbücher und Leitfäden könnten für Ihre Branche und Ihren Standort interessant sein.

Infrastruktursicherheit in Amazon Bracket

Als verwalteter Service ist Amazon Braket durch die AWS globalen Netzwerksicherheitsverfahren geschützt, die im Whitepaper AWS: Überblick über Sicherheitsprozesse beschrieben werden.

Um über das Netzwerk auf Amazon Braket zuzugreifen, rufen Sie veröffentlichte AWS APIs auf. Clients müssen Transport Layer Security (TLS) 1.2 oder höher unterstützen. Kunden müssen außerdem Verschlüsselungssuiten mit Perfect Forward Secrecy (PFS) wie Ephemeral Diffie-Hellman (DHE) oder Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) unterstützen. Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

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 von Einrichtungen verarbeitet, die vonAWS. Informationen über den physischen Standort undAWSDie Region, in der jede QPU verfügbar ist, finden Sie inAngaben zum GerätAbschnitt der Amazon Braket-Konsole.

Ihr Inhalt ist anonymisiert. Nur der Inhalt, der für die Bearbeitung der Schaltung erforderlich ist, wird an Dritte gesendet.AWS-KontoInformationen werden nicht an Dritte weitergegeben.

Alle Daten werden im Ruhezustand und während der Übertragung verschlüsselt. Daten werden nur zur Verarbeitung entschlüsselt. Drittanbietern von Amazon Braket ist es nicht gestattet, Ihre Inhalte für andere Zwecke als die Verarbeitung Ihrer Verbindung zu speichern oder zu 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. AWS 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 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 Interface VPC Endpoints (AWS PrivateLink) im Amazon VPC-Benutzerhandbuch.

Überlegungen zu Amazon Braket VPC-Endpunkten

Bevor Sie einen Schnittstellen-VPC-Endpunkt für Braket einrichten, stellen Sie sicher, dass Sie die Eigenschaften und Einschränkungen der 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 Steuerung des Zugriffs auf Services mit VPC-Endpunkten 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 Erstellung eines Schnittstellenendpunkts im Benutzerhandbuch für Amazon VPC.

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`

Hinweis: Wenn Sie privates DNS für den Endpunkt aktivieren, können Sie API Anfragen an Braket stellen, indem Sie beispielsweise den Standard-DNS-Namen für die Region verwenden. `braket.us-east-1.amazonaws.com`

Weitere Informationen finden Sie unter Zugriff auf einen Service über einen Schnittstellenendpunkt im Benutzerhandbuch für Amazon VPC.

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 SageMaker Amazon-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 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.

Weitere 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 über die Amazon-VPC-Konsole oder die AWS CLI finden Sie unter Creating an Interface Endpoint (Erstellen eines Schnittstellenendpunkts) im Amazon-VPC-Benutzerhandbuch.
- Informationen zum Erstellen und Konfigurieren eines Endpunkts mithilfe von finden Sie in der AWS CloudFormation Ressource AWS: :EC2: :VpcEndpoint im Benutzerhandbuch. AWS 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 Steuerung des Zugriffs auf Services mit VPC-Endpunkten 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 ohne Administratorrechte
- Kontrollieren des Zugriffs auf Services mit VPC-Endpunkten

Fehlerbehebung

Lösen Sie häufig auftretende Probleme, auf die Sie bei der Arbeit mit Amazon Braket stoßen könnten.

Themen

In diesem Abschnitt:

Amazon Braket-Kontingente

In der folgenden Tabelle sind die Servicekontingenten 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.

Ressource	Beschreibung	Limit	Einstellbar
Rate der Anfragen API	Die maximale Anzahl von Anforderungen	140	Ja

Ressource	Beschreibung	Limit	Einstellbar
	pro Sekunde, die Sie in diesem Konto in der aktuellen Region senden können.		
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
Rate der `CreateQuantumTask` Anfragen	Die maximale Anzahl von `CreateQuantumTask` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	20	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

Ressource	Beschreibung	Limit	Einstellbar
Rate der Anfragen `SearchQuantumTasks`	Die maximale Anzahl von `SearchQuantumTasks` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5	Ja
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	Ja

Ressource	Beschreibung	Limit	Einstellbar
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
Rate der Anfragen `CancelQuantumTask`	Die maximale Anzahl von `CancelQuantumTask` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	2	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	Ja

Ressource	Beschreibung	Limit	Einstellbar
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
Rate der Anfragen `SearchDevices`	Die maximale Anzahl von `SearchDevices` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5	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	Ja

Ressource	Beschreibung	Limit	Einstellbar
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
Rate der Anfragen `SearchJob`	Die maximale Anzahl von `SearchJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	5	Ja
Burst-Rate der `SearchJob` Anfragen	Die maximale Anzahl zusätzlicher `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	Ja

Ressource	Beschreibung	Limit	Einstellbar
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
Rate der Anfragen `CancelJob`	Die maximale Anzahl von `CancelJob` Anfragen, die Sie pro Sekunde in diesem Konto pro Region senden können.	2	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 (50 in US-West-1 und EU-West-2)	Nein

Ressource	Beschreibung	Limit	Einstellbar
Anzahl gleichzeitiger Quantenaufgaben DM1	Die maximale Anzahl gleichzeitiger Quantenaufgaben, die auf dem Dichtematrixsimulator (DM1) in der aktuellen Region ausgeführt werden.	100 (50 in US-West-1 und EU-West-2)	Nein
Anzahl gleichzeitiger Quantenaufgaben TN1	Die maximale Anzahl gleichzeitiger Quantenaufgaben, die auf dem Tensor-Netzwerksimulator (TN1) in der aktuellen Region ausgeführt werden.	10 (5 in EU-West-2)	Ja
Anzahl gleichzeitiger hybrider Jobs	Die maximale Anzahl gleichzeitiger Hybridjobs in der aktuellen Region.	5	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. Wenn Sie diese Kontingente erhöhen möchten, wenden Sie sich bitte an AWS Support.

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von	Die maximal zulässige Anzahl von Instances	5	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
ml.c4.xlarge für Hybrid-Jobs	des Typs ml.c4.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.		
Maximale Anzahl von Instanzen von ml.c4.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c4.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja
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
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

Ressource	Beschreibung	Einschränkungen	Einstellbar
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
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
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
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

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von ml.c5.18xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.c5.18x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.c5n.xlarge für Hybridaufträge	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
Maximale Anzahl von Instanzen von ml.c5n.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.2x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.c5n.4xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.4x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von ml.c5n.9xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.c5n.9x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.c5n.18xlarge für Hybridaufträge	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
Maximale Anzahl von Instanzen von ml.g4dn.xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.g4dn.2xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.2 xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von ml.g4dn.4xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.4 xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.g4dn.8xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.8 xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.g4dn.12xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.1 2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.g4dn.16xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.g4dn.1 6xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
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
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
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
Maximale Anzahl von Instanzen von ml.m4.10xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.10xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von ml.m4.16xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m4.16xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
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
Maximale Anzahl von Instanzen von ml.m5.xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.m5.xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	5	Ja
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

Ressource	Beschreibung	Einschränkungen	Einstellbar
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
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
Maximale Anzahl von Instanzen von ml.m5.24xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.m5.24xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
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

Ressource	Beschreibung	Einschränkungen	Einstellbar
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
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
Maximale Anzahl von Instanzen von ml.p3.2xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.p3.2xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximale Anzahl von Instanzen von ml.p3.8xlarge für Hybridaufträge	Die maximal zulässige Anzahl von Instances des Typs ml.p3.8xlarge für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja

Ressource	Beschreibung	Einschränkungen	Einstellbar
Maximale Anzahl von Instanzen von ml.p3.16xlarge für Hybrid-Jobs	Die maximal zulässige Anzahl von Instances des Typs ml.p3.16x large für alle Amazon Braket Hybrid Jobs in diesem Konto und dieser Region.	0	Ja
Maximal zulässige Recheninstanzen für einen Hybrid-Job	Die maximal zulässige Anzahl von Rechenins tanzen für einen Hybrid-Job.	5	Ja

Limit-Updates 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**
>
> P3-Instances sind in us-west-1 nicht verfügbar. Wenn Ihr Hybrid-Job die angeforderte ML-Rechenkapazität nicht bereitstellen kann, verwenden Sie eine andere Region. Wenn Sie in der Tabelle keine Instanz sehen, ist sie außerdem nicht für Hybrid-Jobs verfügbar.

Zusätzliche Kontingente und Limits

- Die Amazon Braket-Quanten-Task-Aktion ist auf eine Größe von 3 MB begrenzt.
- Die maximale Anzahl von Schüssen pro Aufgabe fürSV1,DM1, und Rigetti Geräte beträgt 100.000.
- Die maximal zulässige Anzahl von Schüssen pro Aufgabe TN1 beträgt 1000.
- Für IonQ die Geräte Aria-1 und Aria-2 sind maximal 5.000 Schüsse pro Aufgabe zulässig. Für Harmony- und Forte-Geräte und -Geräte IonQ liegt das OQC Maximum bei 10.000.
- Für beträgt QuEra die maximal zulässige Anzahl an Schüssen pro Aufgabe 1000.

- Für TN1 und die QPU Geräte müssen die Schüsse pro Aufgabe > 0 sein.

Ausnahme „Zugriff verweigert“

Wenn Sie AccessDeniedException, wie in der folgenden Abbildung gezeigt, bei der Aktivierung oder Verwendung von Braket eine Meldung erhalten, versuchen Sie höchstwahrscheinlich, Braket in einer Region zu aktivieren oder zu verwenden, auf die Ihre eingeschränkte Rolle keinen Zugriff hat.

In solchen Fällen sollten Sie sich an Ihren internen AWS Administrator wenden, um zu erfahren, welche der folgenden Bedingungen zutreffen:

- wenn es Rolleneinschränkungen gibt, die den Zugriff auf eine Region verhindern
- ob 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.

Eine Quantenaufgabe scheitert an der Schöpfung

Wenn Sie eine Fehlermeldung wie „Beim Aufrufen der CreateQuantumTask Operation ist ein Fehler aufgetreten (ValidationException) aufgetreten: Der Anrufer hat keinen Zugriff auf Amazon-Braket-...“ erhalten, stellen Sie sicher, dass Sie sich auf einen vorhandenen s3_folder beziehen, da wir nicht auto neue Amazon S3 S3-Buckets und -Präfixe für Sie erstellen.

Wenn Sie API direkt darauf zugreifen und eine Fehlermeldung wie „Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET" erhalten, stellen Sie sicher, dass Sie s3://nicht in den Amazon S3 S3-Bucket-Pfad aufnehmen.

Eine SDK-Funktion funktioniert nicht

Ihre Python-Version muss 3.9 oder höher sein. Für Amazon Braket Hybrid Jobs empfehlen wir Python 3.10.

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.

Ein Hybrid-Job schlägt aufgrund einer Überschreitung des Kontingents fehl

Mein Hybridjob schlägt fehl mit **`ServiceQuotaExceededException`**

Ein Hybridjob, 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. Die Dienstlimits werden im Thema Kontingente erklärt. Wenn Sie von Ihrem Konto aus mehrere Hybridaufträge ausführen, bei denen gleichzeitig Aufgaben auf einem Simulatorgerät ausgeführt werden, 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, können Sie entweder:

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.

In Ihrer Notebook-Instanz funktioniert etwas 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 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: Die physikalischen qubits Verbindungen in zwei qubit Gates sind nicht verbunden
- GetDevice gibt keinen OpenQASM-Ergebnisfehler zurück
- 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 Gates und physische Gates 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`

GetDevice gibt keinen OpenQASM-Ergebnisfehler zurück

Wenn Sie bei der Verwendung eines Braket-SDK keine OpenQASM-Ergebnisse in der GetDevice Antwort sehen, müssen Sie möglicherweise die Umgebungsvariable AWS_EXECUTION_ENV setzen, um den Benutzeragenten zu konfigurieren. In den folgenden Codebeispielen erfahren Sie, wie Sie dies für die Go- und Java-SDKs tun können.

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten zu konfigurieren, wenn Sie Folgendes verwenden: AWS CLI

```
% export AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0"
# Or for single execution
% AWS_EXECUTION_ENV="aws-cli BraketSchemas/1.8.0" aws braket <cmd> [options]
```

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten bei Verwendung von Boto3 zu konfigurieren:

```
import boto3
import botocore

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

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten zu konfigurieren, wenn Sie/(SDK v2) verwenden: JavaScript TypeScript

```
import Braket from 'aws-sdk/clients/braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0'  });
```

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten zu konfigurieren, wenn Sie/(SDK v3) verwenden: JavaScript TypeScript

```
import { Braket } from '@aws-sdk/client-braket';
```

```
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0' });
```

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten zu konfigurieren, wenn Sie das Go SDK verwenden:

```
os.Setenv("AWS_EXECUTION_ENV", "BraketGo BraketSchemas/1.8.0")
mySession := session.Must(session.NewSession())
svc := braket.New(mySession)
```

So legen Sie die Umgebungsvariable AWS_EXECUTION_ENV fest, um den Benutzeragenten bei Verwendung des Java-SDK zu konfigurieren:

```
ClientConfiguration config = new ClientConfiguration();
config.setUserAgentSuffix("BraketSchemas/1.8.0");
BraketClient braketClient =
 BraketClientBuilder.standard().withClientConfiguration(config).build();
```

Warnung zur Unterstützung des lokalen Simulators

Das `LocalSimulator` unterstützt erweiterte Funktionen in OpenQASM, die auf QPUs oder On-Demand-Simulatoren möglicherweise nicht 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 QPUs oder On-Demand-Simulatoren unterstützt.

Weitere Informationen zu den unterstützten OpenQASM-Funktionen finden Sie hier.

API- und SDK-Referenzleitfaden für Amazon Braket

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

- Amazon Bracket Python SDK (empfohlen)
- Amazon Bracket API-Referenz
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto)
- AWS SDK for Ruby

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

- Halterung-Tutorials GitHub

Dokumentverlauf

In der folgenden Tabelle wird die Dokumentation für diese Version von Amazon Braket beschrieben.

- APIVersion: 28. April 2022
- Letzte Aktualisierung der API Referenz: 25. September 2023
- Letzte Aktualisierung der Dokumentation: 11. April 2024

Änderung	Beschreibung	Date (Datum)
Lokale Feinabstimmung veröffentlicht	Zu den experimentellen Funktionen gehört jetzt die Funktion zur lokalen Feinabstimmung der Aquila QuEra QPU.	11. April 2024
Der Notebook-Inaktivitätsmanager 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 freigegeben	Unterstützung für Funktionen von Braket Direct wurde hinzugefügt, darunter:	8. November 2023

	• Reservierungen • Fachkundige Beratung • Experimentelle Fähigkeiten	
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.	8. November 2023
Bringen Sie Ihren eigenen Container mit (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 zur 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-I nstance-Kontingente für Hybrid-Jobs wurden aktualisi ert	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 Kostenver folgung	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 einzusehen 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 Geräuschs imulator braket_dm hinzugefü gt	25. Mai 2021
PennyLane Unterstützung	Unterstützung für PennyLane auf Amazon Braket hinzugefü gt	08. Dezember 2020
Neuer Simulator	Unterstützung für einen Tensor-Netzwerksim ulator (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 Aufgabenr essourcen	30. Oktober 2020
Support für PrivateLink	Sie können private VPC-Endpunkte für Ihre Braket-Jo bs 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

AWS-Glossar

Die neueste AWS Terminologie finden Sie im AWSGlossar in der AWSGlossarreferenz.