aws

開発者ガイド

Amazon Braket

Amazon Braket: 開発者ガイド

Amazon の商標とトレードドレスは、Amazon 以外の製品またはサービスとの関連において、顧客に混乱を招いたり、Amazon の名誉または信用を毀損するような方法で使用することはできません。Amazon が所有しない他の商標はすべてそれぞれの所有者に帰属します。所有者は必ずしも Amazon との提携や関連があるわけではありません。また、Amazon の支援を受けているとはかぎりません。

Table of Contents

Amazon Braket とは

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket は、研究者、サイエンティスト、デベロッパー AWS のサービス が量子コンピューティングを始めるのに役立つフルマネージド型です。量子コンピューティングは、量子力学の法則を利用して新しい方法で情報を処理するため、古典的なコンピュータの手の届かない計算問題を解決できる可能性があります。

量子コンピューティングハードウェアへのアクセスを得ることは、費用がかかり不便になる場合があります。アクセスが制限されているため、アルゴリズムの実行、設計の最適化、テクノロジーの現在の状態の評価、リソースをいつ投資すれば最大のメリットが得られるかの計画が困難になります。Braket は、これらの課題を克服するのに役立ちます。

Braket は、さまざまな量子コンピューティングテクノロジーへの単一のアクセスポイントを提供します。Braket を使用すると、次のことができます。

- 量子アルゴリズムとハイブリッドアルゴリズムを調べて設計します。
- さまざまな量子回路シミュレーターでアルゴリズムをテストします。
- さまざまなタイプの量子コンピュータでアルゴリズムを実行します。
- 概念実証アプリケーションを作成します。

量子問題を定義し、それを解決するための量子コンピュータのプログラミングには、新しいスキルのセットが必要です。これらのスキルを習得するために、Braket は量子アルゴリズムをシミュレートして実行するためのさまざまな環境を提供しています。要件に最適なアプローチを見つけて、ノートブック と呼ばれる一連のサンプル環境をすばやく使い始めることができます。

Braket 開発には、構築、テスト、実行の 3 つの段階があります。

構築 - Braket は、フルマネージド型の Jupyter Notebook 環境を提供し、簡単に開始できるようにします。Braket ノートブックには、Braket Amazon SDK などのサンプルアルゴリズム、リソース、開発者ツールがプリインストールされています。Amazon Braket SDK を使用すると、1 行のコードを

変更することで、量子アルゴリズムを構築し、さまざまな量子コンピュータやシミュレーターでテストして実行できます。

テスト - Braket は、フルマネージド型の高性能量子回路シミュレーターへのアクセスを提供します。回路をテストして検証できます。Braket は、基盤となるすべてのソフトウェアコンポーネントと Amazon Elastic Compute Cloud (Amazon EC2) クラスターを処理し、従来のハイパフォーマンスコンピューティング (HPC) インフラストラクチャで量子回路をシミュレートする負担を取り除きます。

Run - Braket は、さまざまなタイプの量子コンピュータへの安全なオンデマンドアクセスを提供します。IonQ、、OQCおよび からゲートベースの量子コンピュータにアクセスできRigetti、また からアナログハミルトニアンシミュレーターにアクセスできます QuEra。また、事前のコミットメントもないため、個々のプロバイダーを通じてアクセスを調達する必要はありません。

量子コンピューティングと Braket について

量子コンピューティングは開発の初期段階にあります。現在、普遍的でフォールトトレラントな量子コンピュータは存在しないことを理解することが重要です。したがって、特定のタイプの量子ハードウェアは各ユースケースに適しているため、さまざまなコンピューティングハードウェアにアクセスできることが重要です。Braket は、サードパーティープロバイダーを通じてさまざまなハードウェアを提供しています。

既存の量子ハードウェアはノイズによって制限され、エラーが生じます。業界はノイズの多い中間規模量子 (NISQ) の時代です。NISQ 時代には、量子コンピューティングデバイスがノイズが多すぎてショアのアルゴリズムやグローバーのアルゴリズムなどの純粋な量子アルゴリズムを維持できません。より良い量子誤差補正が利用可能になるまで、最も実用的な量子コンピューティングでは、ハイブリッドアルゴリズムを作成するために、従来の (従来の) コンピューティングリソースと量子コンピュータを組み合わせる必要があります。Braket は、ハイブリッド量子アルゴリズムの操作に役立ちます。

ハイブリッド量子アルゴリズムでは、量子処理装置 (QPU) が CPU のコプロセッサとして使用されるため、古典的なアルゴリズムにおける特定の計算が高速化されます。これらのアルゴリズムは、計算が古典コンピュータと量子コンピュータ間で移動する反復処理を利用します。例えば、化学、最適化、機械学習における量子コンピューティングの現在の応用は、ハイブリッド量子アルゴリズムの一種である変分量子アルゴリズムに基づいています。バリエーション量子アルゴリズムでは、古典最適化ルーチンは、機械学習トレーニングセットのエラーに基づいてニューラルネットワークの重みが繰り返し調整されるのとほぼ同じ方法で、パラメータ化された量子回路のパラメータを繰り返し調整します。Braket は、 PennyLane オープンソースソフトウェアライブラリへのアクセスを提供し、さまざまな量子アルゴリズムを支援します。

量子コンピューティングは、次の 4 つの主要な領域での計算で牽引しています。

- 数値理論 — 因数計算や暗号を含む (例えば、ショアのアルゴリズムは数値理論計算の主要な量子メソッドである)
- 最適化 — 制約の満足度、線形システムの解決、機械学習など
- 眼コンピューティング — 検索、非表示のサブグループ、順序検出結果を含む (例えば、Grover のアルゴリズムは眼コンピューティングの主要な量子メソッドです)
- シミュレーション — 直接シミュレーション、ノット不変、量子近似最適化アルゴリズム (QAOA) アプリケーションを含む

これらの計算カテゴリの用途は、金融サービス、バイオテクノロジー、製造、医薬品などが挙げられます。Braket は、特定の実用的な問題に加えて、多くの概念実証問題にすでに適用できる機能とサンプルノートブックを提供します。

Amazon Braket の用語と概念

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Braket では、次の用語と概念が使用されます。

アナログハミルトニアシミュレーション

アナログハミルトンシミュレーション (AHS) は、多体システムの時間依存量子力学を直接シミュレーションするための明確な量子コンピューティングパラダイムです。AHS では、ユーザーは時間依存のハミルトニアンを直接指定し、量子コンピュータは、このハミルトニアンの下での継続的な時間進化を直接エミュレートするように調整されます。AHS デバイスは、通常、専用デバイスであり、ゲートベースのデバイスのような汎用量子コンピュータではありません。これらは、シミュレートできるハミルトニアンのクラスに限定されます。ただし、これらのハミルトニアンはデバイスに自然に実装されるため、AHS はアルゴリズムを回路として策定し、ゲート操作を実装するために必要なオーバーヘッドに悩まされません。

Braket

Braket サービスは、量子力学の標準表記である bra-ket 表記にちなんで命名されました。量子系の状態を記述するために 1939 年に Paul Dirac によって導入され、ディラック記法とも呼ばれます。

Braket ハイブリッドジョブ

Amazon Braket には、ハイブリッドアルゴリズムのフルマネージド実行を提供する Amazon Braket Hybrid Jobs と呼ばれる機能があります。Braket ハイブリッドジョブは、次の 3 つのコンポーネントで構成されます。

1. アルゴリズムの定義。スクリプト、Python モジュール、または Docker コンテナとして提供できます。
2. アルゴリズムを実行する Amazon EC2 に基づくハイブリッドジョブインスタンス 。デフォルトは ml.m5.xlarge インスタンスです。
3. アルゴリズムの一部である量子タスクを実行する量子デバイス。 1 つのハイブリッドジョブには通常、多数の量子タスクのコレクションが含まれます。

デバイス

Amazon Braket では、デバイスは量子タスク を実行できるバックエンドです。デバイスは QPU または量子回路シミュレーターである可能性があります。詳細については、Amazon Braket がサポートするデバイス」を参照してください。

ゲートベースの量子コンピューティング

ゲートベースの量子コンピューティング (QC) では、回路ベースのQC とも呼ばれ、計算は基本オペレーション (ゲート) に分割されます。特定のゲートセットはユニバーサルです。つまり、すべての計算をそれらのゲートの有限シーケンスとして表現できます。ゲートは量子回路の構成要素であり、古典的なデジタル回路の論理ゲートに似ています。

ハミルトニア語

物理システムの量子力学は、システムの構成要素と外因性駆動力の影響の間の相互作用に関するすべての情報をエンコードするハミルトニアンによって決定されます。N 量子ビットシステムのハミルトニアンは、通常、古典的なマシン上の複雑な数値の 2^N x 2^N 行列として表されます。量子デバイスでアナログハミルトンシミュレーションを実行することで、このような指数関数的なリソース要件を回避できます。

Pulse

脈は、量子ビットに送信される一時的な物理信号です。これは、キャリア信号のサポートとして機能し、ハードウェアチャネルまたはポートにバインドされるフレームで再生される波形によっ

て記述されます。顧客は、高周波の直交キャリア信号を調整するアナログエンベロープを提供することで、独自の脈を設計できます。フレームは、周波数と、量子ビットの |0# と |1# のエネルギーレベル間のエネルギー分離で、しばしば勾配上に選択されるフェーズによって一意に記述されます。したがって、ゲートは、あらかじめ設定された形状と、その振幅、頻度、期間などのキャリブレーションされたパラメータを持つ脈動として設定されます。テンプレート波形でカバーされないユースケースは、カスタム波形を介して有効になります。カスタム波形は、固定された物理サイクル時間で区切られた値のリストを提供することで、単一のサンプル解決で指定されます。

量子回路

量子回路は、ゲートベースの量子コンピュータ上の計算を定義する命令セットです。量子回路は、一連の量子ゲートであり、測定命令とともに、qubitレジスタ上で可逆的変換です。

量子回路シミュレーター

量子回路シミュレーターは、古典的なコンピュータ上で動作し、量子サーキットの測定結果を計算するコンピュータプログラムです。一般的な回路では、量子シミュレーションのリソース要件は、シミュレートqubitsする の数とともに指数関数的に増加します。Braket は、マネージド (Braket を介してアクセスAPI) 量子回路シミュレーターとローカル (AmazonBraket SDK の一部) 量子回路シミュレーターの両方へのアクセスを提供します。

量子コンピュータ

量子コンピュータは、重ね合わせやもつれなどの量子力学現象を使用して計算を実行する物理デバイスです。量子コンピューティング (QC) には、ゲートベースのQC など、さまざまなパラダイムがあります。

量子処理ユニット (QPU)

QPU は、量子タスクで実行できる物理量子コンピューティングデバイスです。QPUsは、ゲートベースのQC など、さまざまなQC パラダイムに基づくことができます。詳細については、Amazon Braket でサポートされているデバイス」を参照してください。

QPU ネイティブゲート

QPU ネイティブゲートは、QPU 管理システムによってコントロールの脈動に直接マッピングできます。ネイティブゲートは、さらにコンパイルしなくても QPU デバイス上で実行できます。QPU がサポートするゲートのサブセット。デバイスのネイティブゲートは、Braket コンソールのデバイスページと Amazon Braket SDK から確認できます。

QPU がサポートするゲート

QPU がサポートするゲートは、QPU デバイスで受け入れられるゲートです。これらのゲートは QPU で直接実行できない場合があります。つまり、ネイティブゲートに分解する必要がある可能性があります。デバイスのサポートされているゲートは、Braket コンソールのデバイスページと Amazon Braket SDK Amazon から確認できます。

量子タスク

Braket では、量子タスクはデバイス へのアトミックリクエストです。ゲートベースのQCデバイスの場合、これには量子回路 (測定手順と の数を含むshots) やその他のリクエストメタデータが含まれます。Amazon Braket SDK または CreateQuantumTaskAPIオペレーションを直接使用して量子タスクを作成できます。量子タスクを作成すると、リクエストされたデバイスが使用可能になるまでキューに入れられます。量子タスクは、AmazonBraket コンソールの「量子タスク」ページ、または GetQuantumTask または SearchQuantumTasksAPIオペレーションを使用して表示できます。

Qubit

量子コンピュータの基本的な情報単位は、古典コンピューティングのビットと同様に qubit (量子ビット) と呼ばれます。qubit は、超伝導回路や個々のイオンや原子など、さまざまな物理実装によって実現できる 2 レベルの量子システムです。他のqubitタイプは、光子、電子スピン、スピン、またはより緻密な量子システムに基づいています。

Queue depth

Queue depth は、特定のデバイスに対してキューに入れられた量子タスクとハイブリッドジョブの数を指します。デバイスの量子タスクとハイブリッドジョブキューの数は、 Braket Software Development Kit (SDK)または からアクセスできますAmazon Braket Management Console。

1. タスクキューの深さとは、現在通常の優先度で実行を待っている量子タスクの合計数を指します。
2. 優先度タスクキューの深さとは、 を通じて実行を待機している送信された量子タスクの合計数を指しますAmazon Braket Hybrid Jobs。これらのタスクは、ハイブリッドジョブが開始されると、スタンドアロンタスクよりも優先されます。
3. ハイブリッドジョブキューの深さとは、デバイスで現在キューに入っているハイブリッドジョブの総数を指します。ハイブリッドジョブの一部としてQuantum tasks送信された は優先され、 に集約されますPriority Task Queue。

Queue position

Queue position は、それぞれのデバイスキュー内の量子タスクまたはハイブリッドジョブの現在の位置を指します。量子タスクまたはハイブリッドジョブの場合は、 Braket Software Development Kit (SDK)または を使用して取得できますAmazon Braket Management Console。

Shots

量子コンピューティングは本質的に確率的であるため、正確な結果を得るには、回路を複数回評価する必要があります。単一の回路の実行と測定は、ショットと呼ばれます。回路のショット数 (繰り返し実行) は、結果の望ましい精度に基づいて選択されます。

AWS Amazon Braket の用語とヒント

IAM ポリシー

IAM ポリシーは、 AWS のサービス および リソースへのアクセス許可を許可または拒否するドキュメントです。IAM ポリシーを使用すると、 リソースへのユーザーのアクセスレベルをカスタマイズできます。例えば、 内のすべての Amazon S3 バケットへのアクセスをユーザーに許可したり AWS アカウント、特定のバケットのみにアクセスを許可したりできます。

- ベストプラクティス:セキュリティ原則に従う最小特権アクセス許可を付与する場合。この原則に従うことで、ユーザーまたはロールが量子タスクの実行に必要な数を超えるアクセス許可を持つことを防ぐことができます。例えば、従業員が特定のバケットのみにアクセスする必要がある場合は、 内のすべてのバケットへのアクセスを従業員に許可するのではなく、IAM ポリシーでバケットを指定します AWS アカウント。

IAM ロール

IAM ロールは、アクセス許可に一時的にアクセスするために引き受けることができるアイデンティティです。ユーザー、アプリケーション、またはサービスが IAM ロールを引き受ける前に、ロールに切り替えるためのアクセス許可を付与する必要があります。IAM ロールを引き受けると、以前のロールで持っていた以前のすべてのアクセス許可を放棄し、新しいロールのアクセス許可を引き受けます。

- ベストプラクティス:IAM ロールは、長期的にではなく、サービスまたはリソースへのアクセスを一時的に付与する必要がある状況に最適です。

Amazon S3 バケット

Amazon Simple Storage Service (Amazon S3) は、データをオブジェクトとしてバケット に保存 AWS のサービス できる です。 Amazon S3 バケットは、無制限のストレージスペースを提供し

ます。Amazon S3 バケット内のオブジェクトの最大サイズは 5 TB です。イメージ、動画、テキストファイル、バックアップファイル、ウェブサイトのメディアファイル、アーカイブされたドキュメント、Braket 量子タスク結果など、任意のタイプのファイルデータを Amazon S3 バケットにアップロードできます。

- ベストプラクティス:S3 バケットへのアクセスを制御するためのアクセス許可を設定できます。詳細については、「バケットポリシーとユーザーポリシー」を参照してください。Amazon S3 ドキュメントのを参照してください。

Amazon Braket の料金

> **Tip**
>
> で量子コンピューティングの基礎を学びます AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket を使用すると、事前のコミットメントなしに、オンデマンドで量子コンピューティングリソースにアクセスできます。お支払いいただくのは、使用分の料金だけです。料金の詳細については、料金ページをご覧ください。。

ほぼリアルタイムのコスト追跡

Braket SDK には、量子ワークロードにほぼリアルタイムのコスト追跡を追加するオプションがあります。各サンプルノートブックには、Braket の量子処理ユニット (QPUs) とオンデマンドシミュレーターの最大コスト見積もりを提供するコスト追跡コードが含まれています。最大コスト見積もりは USD で表示され、クレジットや割引は含まれません。

> **Note**
>
> 表示される料金は、Amazon Braket シミュレーターと量子処理ユニット (QPU) タスクの使用状況に基づく見積もりです。表示される推定料金は、実際の料金とは異なる場合があります。推定料金は割引やクレジットを一切考慮せず、Amazon Elastic Compute Cloud (Amazon EC2) などの他の のサービスの使用に基づいて追加料金が発生する場合があります。

SV1 のコスト追跡

コスト追跡機能の使用方法を示すために、ベルステート回路を構築し、SV1 シミュレーターで実行します。まず、Braket SDK モジュールをインポートし、ベルステートを定義して、回路に Tracker()関数を追加します。

```
#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)
```

ノートブックを実行すると、ベルステートシミュレーションに対して次の出力が期待できます。トラッカー関数には、送信されたショット数、完了した量子タスク、実行期間、請求された実行期間、最大コストが USD で表示されます。実行時間はシミュレーションごとに異なる場合があります。

```
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
```

コストトラッカーを使用して最大コストを設定する

コストトラッカーを使用して、プログラムの最大コストを設定できます。特定のプログラムに費やす金額には、最大しきい値がある場合があります。このようにして、コストトラッカーを使用して、実行コードにコスト制御ロジックを構築できます。次の例では、RigettiQPU で同じ回路を使用し、コストを 1 USD に制限しています。コード内の回路の反復を 1 回実行するコストは 0.37 USD です。合計コストが 1 USD を超えるまで反復を繰り返すようにロジックを設定しました。したがって、コードスニペットは次の反復が 1 USD を超えるまで 3 回実行されます。通常、プログラムは希望する最大コストに達するまで反復し続けます。この場合、3 回の反復です。

```
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**
>
> コストトラッカーは、失敗したTN1量子タスクの継続時間を追跡しません。TN1 シミュレーション中にリハーサルが完了し、収縮ステップが失敗した場合、リハーサル料金はコストトラッカーに表示されません。

コスト削減のベストプラクティス

Amazon Braket を使用する際に次のベストプラクティスを考慮してください。時間を節約し、コストを最小限に抑え、一般的なエラーを回避します。

シミュレーターで検証する

- QPU で実行する前に、シミュレーターを使用して回路を検証します。これにより、QPU の使用料金が発生することなく回路を微調整できます。
- シミュレーターで回路を実行した結果は、QPU で回路を実行した結果と同じではない可能性がありますが、シミュレーターを使用してコーディングエラーや構成の問題を特定できます。

特定のデバイスへのユーザーアクセスを制限する

- 権限のないユーザーが特定のデバイスで量子タスクを送信できないように制限を設定できます。アクセスを制限するには、 AWS IAM を使用することをお勧めします。これを行う方法についての詳細は、アクセスの制限を参照してください。
- Amazon Braket デバイスへのユーザーアクセスを許可または制限する方法として、管理者アカウントを使用しないことをお勧めします。

請求アラームの設定

- 請求アラームを設定して、請求が事前設定された限度に達したときに通知を受けることもできます。アラームを設定する推奨方法は、 を通じてです AWS Budgets。カスタム予算を設定し、コストまたは使用量が予算額を超える可能性がある場合にアラートを受け取ることができます。情報はで入手できますAWS Budgets。

ショット数が少ないTN1量子タスクをテストする

- シミュレーターのコストは QHPsよりも低くなりますが、量子タスクをショット数の多い状態で実行すると、特定のシミュレーターのコストが高くなる可能性があります。shot タスクを少数TN1でテストすることをお勧めします。 Shot カウントは、 SV1およびローカルシミュレータータスクのコストには影響しません。

すべてのリージョンで量子タスクを確認する

- コンソールには、現在の の量子タスクのみが表示されます AWS リージョン。送信された請求可能な量子タスクを検索する場合は、必ずすべてのリージョンを確認してください。
- サポートされるデバイスドキュメントページで、デバイスおよび関連するリージョンの一覧を表示できます。

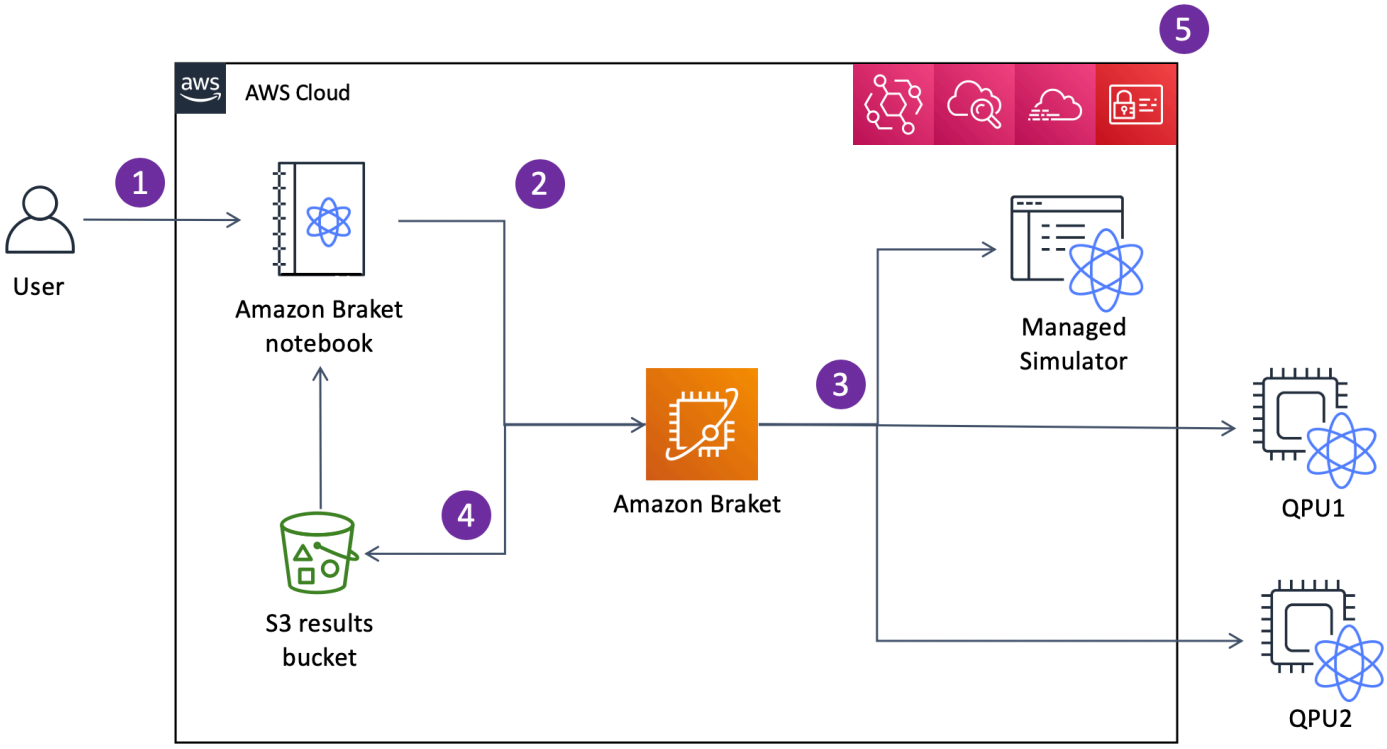
Amazon Braket の仕組み

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket は、オンデマンド回路シミュレーターやさまざまなタイプの QPUs など、量子コンピューティングデバイスへのオンデマンドアクセスを提供します。Amazon Braket では、デバイスへのアトミックリクエストは量子タスクです。ゲートベースのQCデバイスの場合、このリクエストには量子回路 (測定手順とショット数を含む) およびその他のリクエストメタデータが含まれます。アナログハミルトニアンシミュレーターの場合、量子タスクには量子レジスタの物理レイアウトと、操作フィールドの時間およびスペース依存が含まれます。

このセクションでは、AmazonBraket で量子タスクを実行する高レベルのフローについて学習します。

Amazon Braket 量子タスクフロー



Jupyter ノートブックを使用すると、Amazon Braket コンソールまたは Amazon Braket Amazon Braket SDK を使用して、量子タスクを簡単に定義、送信、モニタリングできます。量子回路は SDK で直接構築できます。ただし、アナログハミルトニアンシミュレーターでは、登録レイアウトと制御フィールドを定義します。量子タスクが定義されたら、実行するデバイスを選択して Amazon Braket API (2) に送信できます。選択したデバイスに応じて、デバイスが使用可能になり、実装のためにタスクが QPU またはシミュレーター (3) に送信されるまで、量子タスクはキューに入れられます。Amazon Braket では、さまざまなタイプの QPUs (IonQ、QuEra、、Rigetti) Oxford Quantum Circuits (OQC)、3 つのオンデマンドシミュレーター (SV1、DM1、TN1) 、2 つのローカルシミュレーター、1 つの埋め込みシミュレーターにアクセスできます。詳細については、「Amazon Braket がサポートされるデバイス」を参照してください。

量子タスクを処理すると、AmazonBraket は結果を Amazon S3 バケットに返し、そこでデータは AWS アカウント (4) に保存されます。同時に、SDK は結果をバックグラウンドでポーリングし、量子タスクの完了時に Jupyter Notebook にロードします。Braket コンソールの量子タスクページで、または Amazon AmazonBraket の `GetQuantumTask`オペレーションを使用して、量子タスクを表示および管理することもできますAPI。

Amazon Braket は、 EventBridge ユーザーアクセス管理 CloudWatch、モニタリング、ログ記録、 AWS CloudTrail イベントベースの処理 AWS Identity and Access Management (5) のために、(IAM)、Amazon 、および Amazon と統合されています。

サードパーティーのデータ処理

QPU デバイスに送信された量子タスクは、サードパーティープロバイダーが運営する施設にある量子コンピュータで処理されます。Amazon Braket のセキュリティとサードパーティー処理の詳細については、Amazon Braket ハードウェアプロバイダーのセキュリティ」を参照してください。

Braket のコアリポジトリとプラグイン

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

コアリポジトリ

以下に、Braket に使用されるキーパッケージを含むコアリポジトリのリストを示します。

- Braket Python SDK - Braket Python SDK を使用して、Python プログラミング言語でJupyterノートブックにコードを設定します。Jupyter ノートブックをセットアップしたら、Braket デバイスとシミュレーターでコードを実行できます。
- Braket スキーマ - Braket SDK と Braket サービス間の契約。
- Braket デフォルトシミュレーター - Braket 用のすべてのローカル量子シミュレーター (ステートベクトルと密度行列) 。

プラグイン

次に、さまざまなデバイスやプログラミングツールとともに使用されるさまざまなプラグインがあります。これには、Braket がサポートするプラグインと、以下に示すようにサードパーティーがサポートするプラグインが含まれます。

Amazon Braket でサポートされる ：

- Amazon Braket アルゴリズムライブラリ - Python で記述された構築済みの量子アルゴリズムのカタログ。それらをそのまま実行するか、開始点として使用してより複雑なアルゴリズムを構築します。
- Braket-PennyLane plugin - Braket の QML フレームワークPennyLaneとして使用します。

サードパーティー (Braket チームによるモニタリングと寄稿 ）：

- Qiskit-Braket プロバイダー - Qiskit SDK を使用して Braket リソースにアクセスします。
- Braket-Julia SDK - (実験的) Braket SDK の Julia ネイティブバージョン

Amazon Braket がサポートされるデバイス

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket では、デバイスは量子タスクを実行するために呼び出すことができる QPU またはシミュレーターを表します。Amazon Braket は、IonQ、、IQM、、QuEraおよび から QPU デバイスへのアクセスRigetti、3 Oxford Quantum Circuitsつのオンデマンドシミュレーター、3 つのローカルシミュレーター、および 1 つの埋め込みシミュレーターを提供します。すべてのデバイスについて、デバイストポロジ、キャリブレーションデータ、ネイティブゲートセットなどのデバイスプロパティは、Amazon Braket コンソールのデバイスタブまたは `GetDevice` API を使用して確認できます。シミュレーターを使用して回路を構築する場合、Amazon Braket では現在、連続した量子ビットまたはインデックスを使用する必要があります。Amazon Braket SDK を使用している場合は、次のコード例に示すように、デバイスプロパティにアクセスできます。

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

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

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

# get device properties
device.properties
```

サポートされている量子ハードウェアプロバイダー

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

サポートされているシミュレーター

- ローカル状態ベクトルシミュレーター (braket_sv) ('Default Simulator')
- 局所密度行列シミュレーター (`braket_dm`)
- ローカル AHS シミュレーター
- ステートベクトルシミュレーター (SV1)
- 密度行列シミュレーター (DM1)
- テンソルネットワークシミュレーター (TN1)
- PennyLaneの Lightning Simulators

量子タスクに最適なシミュレーターを選択する

- シミュレーターを比較する

> **Note**
>
> AWS リージョン 各デバイスで使用可能な を表示するには、次の表を右にスクロールします。

Amazon Braket デバイス

プロバイダー	デバイス名	パラダイム	タイプ	デバイス ARN	リージョン
IonQ	Aria 1	ゲートベース	QPU	arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1	us-east-1
IonQ	Aria 2	ゲートベース	QPU	arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2	us-east-1
IonQ	Forte 1	ゲートベース	QPU (予約のみ)	arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1	us-east-1

プロバイダー	デバイス名	パラダイム	タイプ	デバイス ARN	リージョン
IonQ	Harmony	ゲートベース	QPU	arn:aws:braket:us-east-1::device/qpu/ionq/Harmony	us-east-1
IQM	Garnet	ゲートベース	QPU	arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet	eu-north-1
Oxford Quantum Circuits	Lucy	ゲートベース	QPU	arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy	eu-west-2
QuEra	Aquila	アナログハミルトニアシミュレーション	QPU	arn:aws:braket:us-east-1::device/qpu/quera/Aquila	us-east-1
Rigetti	Aspen M-3	ゲートベース	QPU	arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3	us-west-1
AWS	braket_sv	ゲートベース	ローカルシミュレーター	該当なし (Braket SDK のローカルシミュレーター)	該当なし
AWS	braket_dm	ゲートベース	ローカルシミュレーター	該当なし (Braket SDK のローカルシミュレーター)	該当なし

プロバイダー	デバイス名	パラダイム	タイプ	デバイス ARN	リージョン
AWS	SV1	ゲートベース	オンデマンドシミュレーター	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Amazon Braket が利用可能なすべての リージョン。
AWS	DM1	ゲートベース	オンデマンドシミュレーター	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Amazon Braket が利用可能なすべての リージョン。
AWS	TN1	ゲートベース	オンデマンドシミュレーター	arn:aws:braket:::device/quantum-simulator/amazon/tn1	us-west-2、us-east-1、および eu-west-2

> **Note**
>
> 特定の QPUs「予約」を参照してください。

Amazon Braket で使用できる QPUsAmazon Braket ハードウェアプロバイダー」を参照してください。

IonQ

IonQ は、イオントラップテクノロジーに基づくゲートベースの QPUs を提供します。 IonQ's トラップされた ion QPUs は、バキュームチェンバー内のマイクロファブリケーションされた表面表面

表面表面分解トラップによって空間的に閉じ込められた 171Yb + イオンのチェーン上に構築されています。

IonQ デバイスは、次の量子ゲートをサポートします。

```
'x', 'y', 'z', 'rx', 'ry', 'rz', 'h', 'cnot', 's', 'si', 't', 'ti', 'v', 'vi', 'xx',
 'yy', 'zz', 'swap'
```

逐語的なコンパイルでは、IonQQPUsは次のネイティブゲートをサポートします。

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

ネイティブ MS ゲートを使用するときに 2 つのフェーズパラメータのみを指定すると、完全にエンタングルする MS ゲートが実行されます。完全エンタングル MS ゲートは常に π/2 ローテーションを実行します。別の角度を指定し、部分的にエンタングルする MS ゲートを実行するには、3 番目のパラメータを追加して目的の角度を指定します。詳細については、 braket.circuits.gate モジュール を参照してください。

これらのネイティブゲートは逐語的なコンパイルでのみ使用できます。逐語的なコンパイルの詳細については、「逐語的なコンパイル」を参照してください。

IQM

IQM 量子プロセッサは、超伝導トランスモン量子ビットに基づくユニバーサルおよびゲートモデルデバイスです。IQM Garnet デバイスは、正方形の格子トポロジを備えた 20 量子ビットデバイスです。

IQM デバイスは、次の量子ゲートをサポートします。

```
"ccnot", "cnot", "cphaseshift", "cphaseshift00", "cphaseshift01", "cphaseshift10",
 "cswap", "swap", "iswap", "pswap", "ecr", "cy", "cz", "xy", "xx", "yy", "zz", "h",
 "i", "phaseshift", "rx", "ry", "rz", "s", "si", "t", "ti", "v", "vi", "x", "y", "z"
```

逐語的なコンパイルでは、IQMデバイスは次のネイティブゲートをサポートします。

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

Rigetti

Rigetti 量子プロセッサは、オールチューニング可能なスーパーコンダクティング に基づくユニバーサルゲートモデルマシンですqubits。79 量子ビットAspen-M-3デバイスは、独自のマルチチップテクノロジーを活用し、2 つの 40 量子ビットプロセッサから組み立てられます。

Rigetti デバイスは、次の量子ゲートをサポートします。

```
'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'
```

逐語的なコンパイルでは、Rigetti デバイスは次のネイティブゲートをサポートします。

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

Rigetti 超伝導量子プロセッサは、「rx」ゲートを「π/2」または「π」の角度のみで実行できます。

Pulse レベルの制御は、以下のタイプの事前定義されたフレームのセットをサポートするRigettiデバイスで使用できます。

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

これらのフレームの詳細については、「フレームとポートのロール」を参照してください。

Oxford Quantum Circuits (OQC)

OQC 量子プロセッサは、スケーラブルな Coaxmon テクノロジーを使用して構築されたユニバーサルゲートモデルマシンです。OQC Lucy システムは、リングのトポロジーを持つ8-qubitデバイスであり、各リングqubitは最も近い 2 つのネイバーに接続されています。

Lucy デバイスは、次の量子ゲートをサポートします。

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

逐語的なコンパイルでは、OQC デバイスは次のネイティブゲートをサポートします。

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

Pulse レベルの制御はOQCデバイスで使用できます。OQC デバイスは、以下のタイプの事前定義されたフレームのセットをサポートします。

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

OQC デバイスは、有効なポート識別子を指定した場合、フレームの動的宣言をサポートします。これらのフレームとポートの詳細については、「フレームとポートのロール」を参照してください。

> **Note**
>
> で脈動制御を使用する場合OQC、プログラムの長さは最大 90 マイクロ秒を超えることはできません。最大継続時間は、単一量子ビットゲートでは約 50 ナノ秒、2 量子ビットゲートでは 1 マイクロ秒です。これらの数値は、使用する量子ビット、デバイスの現在のキャリブレーション、回路コンパイルによって異なります。

QuEra

QuEra は、Achemal Hamiltonian Simulation (AHS) 量子タスクを実行できる中性原子ベースのデバイスを提供します。これらの専用デバイスは、数百の同時相互作用量子ビットの時間依存量子力学を忠実に再現します。

量子ビットレジスタのレイアウトと、操作フィールドの時間的および空間的依存を再現することで、これらのデバイスをアナログハミルトンシミュレーションのパラダイムにプログラムできます。Amazon Braket は、Python SDK、 の AHS モジュールを介してそのようなプログラムを構築するためのユーティリティを提供しますbraket.ahs。

詳細については、「Analog Hamiltonian Simulation example notebooks」または「Submit an analog program using QuEra's Aquila」ページを参照してください。

ローカル状態ベクトルシミュレーター (**braket_sv**)

ローカル状態ベクトルシミュレーター (braket_sv) は、環境内でローカルで実行される Amazon Braket SDK の一部です。Braket ノートブックインスタンスまたはローカル環境のハードウェア仕様に応じて、小さな回路 (最大 25 qubits) でのラピッドプロトタイピングに適しています。

ローカルシミュレーターは Amazon Braket SDK のすべてのゲートをサポートしますが、QPU デバイスはより小さなサブセットをサポートします。デバイスのサポートされているゲートは、デバイスのプロパティで確認できます。

> **Note**
>
> ローカルシミュレーターは、QPU デバイスやその他のシミュレーターではサポートされていない可能性がある高度な OpenQASM 機能をサポートしています。サポートされている機能の詳細については、OpenQASM Local Simulator Notebook に記載されている例を参照してください。

シミュレーターを使用する方法については、「Amazon Braket の例」を参照してください。

局所密度行列シミュレーター (`braket_dm`)

ローカル密度行列シミュレーター (`braket_dm`) は、環境内でローカルで実行される Amazon Braket SDK の一部です。Braket ノートブックインスタンスまたはローカル環境のハードウェア仕様に応じて、ノイズのある小さな回路 (最大 12 個qubits) でのラピッドプロトタイピングに適しています。

ビットフリップや脱分極誤差などのゲートノイズ演算を使用して、一般的なノイズの多い回路をゼロから構築できます。また、ノイズの有無にかかわらず、既存の回路の特定の qubitsおよびゲートにノイズオペレーションを適用することもできます。

`braket_dm` ローカルシミュレーターは、指定された数の がある場合、次の結果を提供できます shots。

- 低密度行列: Shots = 0

> **Note**
>
> ローカルシミュレーターは高度な OpenQASM 機能をサポートしていますが、QPU デバイスやその他のシミュレーターではサポートされていない場合があります。サポートされている機能の詳細については、OpenQASM Local Simulator Notebook に記載されている例を参照してください。

局所密度行列シミュレーターの詳細については、「Braket 入門ノイズシミュレーターの例」を参照してください。

ローカル AHS シミュレーター (**braket_ahs**)

ローカル AHS (Analog Hamiltonian Simulation) シミュレーター (braket_ahs) は、お客様の環境でローカルで実行される Amazon Braket SDK の一部です。AHS プログラムの結果をシミュレートするために使用できます。Braket ノートブックインスタンスまたはローカル環境のハードウェア仕様に応じて、スモールレジスタ (最大 10～12 原子) でのプロトタイプ作成に適しています。

ローカルシミュレーターは、1 つのユニフォーム駆動フィールド、1 つの (非ユニフォーム) シフトフィールド、および任意の原子配置を持つ AHS プログラムをサポートしています。詳細については、Braket AHS クラスと Braket AHS プログラムスキーマを参照してください。

ローカル AHS シミュレーターの詳細については、「Hello AHS: Run your first analog Hamiltonian Simulation」ページと「Analog Hamiltonian Simulation example notebooks」を参照してください。

ステートベクトルシミュレーター (SV1)

SV1 は、オンデマンドで高性能なユニバーサルステートベクトルシミュレーターです。最大 34 の回路をシミュレートできますqubits。34-qubit、高密度、および正方形の回路 (回路深度 = 34) は、使用するゲートのタイプやその他の要因に応じて、完了までに約 1～2 時間かかることが予想されます。all-to-all ゲート付きの回路は、 に適していますSV1。完全な状態ベクトルや振幅の配列などの形式で結果を返します。

SV1 の最大ランタイムは 6 時間です。デフォルトは 35 個の同時量子タスクで、最大 100 (us-west-1 および eu-west-2 では 50) 個の同時量子タスクがあります。

SV1 結果

SV1 は、指定された数の がある場合、次の結果を提供できますshots。

- サンプル: Shots > 0
- 期待値: Shots >= 0
- 分散: Shots >= 0
- 確率: Shots > 0
- 出力: Shots = 0
- 結合グラデーション: Shots = 0

結果の詳細については、「結果タイプ」を参照してください。

SV1 は常に利用可能で、オンデマンドで回路を実行し、複数の回路を並行して実行できます。ランタイムは、オペレーションの数に応じて直線的にスケールされ、 の数に応じて指数関数的にスケールされますqubits。の数shotsはランタイムにわずかな影響を与えます。詳細については、「シミュレーターを比較する」を参照してください。

シミュレーターは Braket SDK のすべてのゲートをサポートしますが、QPU デバイスは小さなサブセットをサポートします。デバイスのサポートされているゲートは、デバイスのプロパティで確認できます。

密度行列シミュレーター (DM1)

DM1 は、オンデマンドで高性能な密度行列シミュレーターです。最大 17 の回路をシミュレートできますqubits。

DM1 の最大ランタイムは 6 時間、デフォルトは 35 個の同時量子タスク、最大 50 個の同時量子タスクです。

DM1 結果

DM1 は、指定された数の がある場合、次の結果を提供できますshots。

- サンプル: Shots > 0
- 期待値: Shots >= 0
- 分散: Shots >= 0
- 確率: Shots > 0
- 低密度行列: Shots = 0、最大 8 qubits

結果の詳細については、「結果タイプ」を参照してください。

DM1 は常に利用可能で、オンデマンドで回路を実行し、複数の回路を並行して実行できます。ランタイムは、オペレーションの数に応じて直線的にスケールされ、 の数に応じて指数関数的にスケールされますqubits。の数shotsはランタイムにわずかな影響を与えます。詳細については、「シミュレーターの比較」を参照してください。

ノイズゲートと制限

```
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]
```

テンソルネットワークシミュレーター (TN1)

TN1 は、オンデマンド、高性能、テンソルネットワークシミュレーターです。 は、最大 50 でqubits 回路深度が 1,000 以下の特定の回路タイプをシミュレートTN1できます。 TN1は、スパース回路、ローカルゲートを持つ回路、および量子フーリエ変換 (QFT) 回路などの特殊な構造を持つ他の回路に特に強力です。 は 2 つのフェーズでTN1動作します。まず、リハーサルフェーズは回路の効率的な計算パスを識別しようとします。そのため、 は収縮フェーズ と呼ばれる次のステージのランタイムを推定TN1できます。推定収縮時間がTN1シミュレーションランタイム制限を超えると、 TN1は収縮を試みません。

TN1 のランタイム制限は 6 時間です。最大 10 (eu-west-2 では 5) 個の同時量子タスクに制限されています。

TN1 結果

収縮段階は、一連の行列乗算で構成されます。一連の乗算は、結果に達するか、結果に到達できないと判断されるまで継続します。

注: は > 0 Shotsである必要があります。

結果タイプは次のとおりです。

- サンプル
- 期待
- 分散

結果の詳細については、「結果タイプ」を参照してください。

TN1 は常に利用可能で、オンデマンドで回路を実行し、複数の回路を並行して実行できます。詳細については、「シミュレーターの比較」を参照してください。

シミュレーターは Braket SDK のすべてのゲートをサポートしますが、QPU デバイスは小さなサブセットをサポートします。デバイスのサポートされているゲートは、デバイスのプロパティで確認できます。

の使用を開始するには、TN1 サンプルノートブックの Amazon Braket GitHub リポジトリにアクセスしてくださいTN1。

の使用に関するベストプラクティス TN1

- all-to-all 回路は避けてください。
- 新しい回路または少数の の回路クラスをテストしてshots、 の回路の「剛性」を学習しますTN1。
- 大規模なshotシミュレーションを複数の量子タスクに分割します。

埋め込みシミュレーター

埋め込みシミュレーターは、シミュレーションを同じコンテナ内のアルゴリズムコードに埋め込み、ハイブリッドジョブインスタンスでシミュレーションを直接実行することで機能します。これは、シミュレーションがリモートデバイスと通信することに関連するボトルネックを取り除くのに役立ちます。これにより、メモリ使用量が大幅に減少し、必要な結果を得るために回路実行の数が減り、パフォーマンスが 10 倍以上向上する可能性があります。埋め込みシミュレーターの詳細については、Amazon Braket Hybrid Jobs でハイブリッドジョブを実行する」ページを参照してください。

PennyLaneの稲妻シミュレーター

PennyLaneの稲妻シミュレーターを Braket の埋め込みシミュレーターとして使用できます。PennyLaneの稲妻シミュレーターを使用すると、結合区別 などの高度な勾配計算方法を活用して、勾配をより速く評価できます。Lightning.qubit シミュレーターは Braket NBIs経由でデバイスとして、また埋め込みシミュレーターとして使用できますが、Lightning.gpu シミュレーターは GPU イン

スタンスを備えた埋め込みシミュレーターとして実行する必要があります。Lightning.gpu の使用例については、Braket Hybrid Jobs ノートブックの「埋め込みシミュレーター」を参照してください。

シミュレーターを比較する

このセクションでは、いくつかの概念、制限、ユースケースを説明することで、量子タスクに最適な Amazon Braket シミュレーターを選択するのに役立ちます。

ローカルシミュレーターとオンデマンドシミュレーターの選択 (SV1、TN1、DM1)

ローカルシミュレーターのパフォーマンスは、シミュレーターの実行に使用される Braket ノートブックインスタンスなど、ローカル環境をホストするハードウェアによって異なります。オンデマンドシミュレーターは AWS クラウドで実行され、一般的なローカル環境を超えてスケールするように設計されています。オンデマンドシミュレーターは、より大きな回路用に最適化されていますが、量子タスクまたは量子タスクのバッチごとにレイテンシーオーバーヘッドを追加します。これは、多くの量子タスクが関与する場合、トレードオフを意味する可能性があります。これらの一般的なパフォーマンス特性を考慮すると、以下のガイダンスは、ノイズのあるシミュレーションを含むシミュレーションの実行方法を選択するのに役立ちます。

シミュレーションの場合：

- 18 未満の を使用する場合はqubits、ローカルシミュレーターを使用します。
- 18～24 個の を使用する場合はqubits、ワークロードに基づいてシミュレーターを選択します。
- 24 個を超える を使用する場合はqubits、オンデマンドシミュレーターを使用します。

ノイズシミュレーションの場合：

- 9 個未満の を使用する場合はqubits、ローカルシミュレーターを使用します。
- 9～12 個の を使用する場合はqubits、ワークロードに基づいてシミュレーターを選択します。
- 12 個を超える を使用する場合はqubits、 を使用しますDM1。

状態ベクトルシミュレーターとは何ですか?

SV1 はユニバーサルステートベクトルシミュレーターです。量子状態の全波動関数を格納し、ゲート演算を状態に順次適用します。それは、非常にありそうもないものであっても、すべての可能性を格納します。量子タスクのSV1シミュレーターの実行時間は、回路内のゲート数に応じて直線的に増加します。

密度行列シミュレーターとは何ですか?

DM1 はノイズのある量子回路をシミュレートします。システムの全密度行列を保存し、回路のゲートとノイズ操作を順番に適用します。最終的な密度行列には、回路実行後の量子状態に関する完全な情報が含まれています。ランタイムは、通常、オペレーションの数に応じて直線的にスケーリングされ、 の数に応じて指数関数的にスケーリングされますqubits。

テンソルネットワークシミュレーターとは何ですか?

TN1 は、量子回路を構造化グラフにエンコードします。

- グラフのノードは量子ゲート、または で構成されますqubits。
- グラフのエッジは、ゲート間の接続を表します。

この構造の結果として、 TN1は比較的大規模で複雑な量子回路のシミュレートされたソリューションを見つけることができます。

TN1 には 2 つのフェーズが必要です

通常、 は量子計算をシミュレートするための 2 段階のアプローチでTN1動作します。

- リハーサルフェーズ: このフェーズでは、 はグラフを効率的にトラバースする方法TN1を考え出します。これには、必要な測定を取得できるようにすべてのノードを訪問することが含まれます。は両方のフェーズを一緒にTN1実行するため、顧客にはこのフェーズは表示されません。第 1 フェーズを完了し、実用的な制約に基づいて第 2 フェーズを単独で実行するかどうかを決定します。シミュレーションの開始後、その決定への入力はありません。
- 収縮フェーズ:このフェーズは、古典的なコンピュータにおける計算の実行フェーズに似ています。フェーズは、一連の行列乗算で構成されます。これらの乗算の順序は、計算の難しさに大きな影響を与えます。したがって、グラフ全体で最も効果的な計算パスを見つけるために、リハーサルフェーズが最初に行われます。リハーサルフェーズ中に収縮パスが見つかったら、 は回路のゲートをTN1まとめてシミュレーションの結果を生成します。

TN1 グラフはマップに似ています

比喩的に、基礎となるTN1グラフを都市の道路と比較できます。計画されたグリッドがある都市では、地図を使用して目的地までのルートを簡単に見つけることができます。計画外の道路や道路名が重複している都市では、地図を見て目的地までのルートを見つけるのが難しい場合があります。

TN1 がリハーサルフェーズを実行しなかった場合は、最初に地図を見るのではなく、都市の通りを散歩して目的地を見つけるようなものです。地図を見るのにより多くの時間を費やすことは、歩く時間という点で本当に報われるでしょう。同様に、リハーサルフェーズは貴重な情報を提供します。

TN1 は、トラバースする基盤となる回路の構造に特定の「認識」があると言うかもしれません。この意識はリハーサルフェーズ中に高まります。

これらのタイプのシミュレーターに最も適した問題の種類

SV1 は、主に特定の数の qubitsとゲートを持つことに依存する問題のクラスに適しています。一般的に、必要な時間はゲートの数に応じて直線的に増加しますが、 の数には依存しませんshots。 SV1 は一般的に 28 未満の回路TN1の場合よりも高速ですqubits。

SV1 は、非常に可能性が低い可能性であっても、実際にはすべての可能性をシミュレートするため、qubit数値が高いほど遅くなる可能性があります。どの結果が出そうなのかを判断する方法はありません。したがって、30-qubit評価では、 は 2^30 設定を計算するSV1必要があります。Amazon Braket SV1シミュレータqubits一の 34 の制限は、メモリとストレージの制限による実用的な制約です。次のように考えることができます。 qubitを に追加するたびにSV1、問題は 2 倍難しくなります。

多くのクラスの問題では、 TN1はグラフの構造を利用するSV1ため、 よりもはるかに大きな回路を現実的な時間で評価TN1できます。基本的に、ソリューションの進化を最初から追跡し、効率的なトラバーサルに寄与する設定のみを保持します。別の言い方をすると、行列乗算の順序を作成するための設定が保存され、評価プロセスがよりシンプルになります。

の場合TN1、 qubitsおよび ゲートの数は重要ですが、グラフの構造はさらに重要です。例えば、TN1は、ゲートが短距離 (つまり、それぞれqubitがゲートによって最も近い隣接する にのみ接続される) の回路 (グラフqubits) と、接続 (またはゲート) が同様の範囲を持つ回路 (グラフ) を評価するのに非常に優れています。の一般的な範囲TN1は、各 が 5 qubits離れた他の とのみqubit通信qubitsすることです。構造の大部分を、より単純な関係に分解できる場合、例えば、より小さい、またはより均一な行列で表すことができる場合、 は評価を簡単にTN1実行します。

の制限事項 TN1

TN1 グラフの構造的複雑さSV1によっては、 よりも遅くなる場合があります。特定のグラフでは、 はリハーサルステージの後にシミュレーションTN1を終了し、次の 2 つの理由のいずれかで FAILEDのステータスを表示します。

- パスが見つかりません — グラフが複雑すぎると、正常なトラバーサルパスを見つけるのが難しすぎ、シミュレーターが計算をあきらめます。 は収縮を実行TN1できません。以下のような エラーメッセージが表示されることがあります。 `No viable contraction path found.`
- 収縮段階が難しすぎる — 一部のグラフでは、 はトラバーサルパスを見つけるTN1ことができますが、評価に非常に長く、非常に時間がかかります。この場合、収縮は非常に高価であるため、コ

ストは禁止され、代わりにリハーサルフェーズの後にTN1終了します。以下のようなエラーメッセージが表示されることがあります。 `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> 収縮が実行されず、 FAILEDステータスが表示されるTN1場合でも、 のリハーサルステージに対して課金されます。

予測ランタイムはshotカウントにも依存します。最悪のシナリオでは、TN1収縮時間はshotカウントに直線的に依存します。回路は、より少ない で収縮できる場合がありますshots。例えば、100 個ので量子タスクを送信しshots、 は契約不可TN1と判断しますが、10 個の のみで再送信すると、収縮が続行されます。このような状況では、100 個のサンプルを得るために、同じ回路shotsに対して 10 個の量子タスクを 10 個送信し、結果を最後に組み合わせることができます。

ベストプラクティスとして、 の数を増やす前に、回路または回路クラスを数個 shots (10 など) でテストしてTN1、回路が に対してどの程度ハードかを調べることをお勧めしますshots。

> **Note**
>
> 収縮フェーズを形成する一連の乗算は、小さな NxN 行列から始まります。例えば、2-qubit ゲートには 4x4 行列が必要です。難しすぎると判断される収縮中に必要な中間行列は巨大です。このような計算には、完了までに数日かかるでしょう。そのため、AmazonBraket は非常に複雑な収縮を試みません。

同時実行

すべての Braket シミュレーターを使用すると、複数の回路を同時に実行できます。同時実行数の制限は、シミュレーターとリージョンによって異なります。同時実行数の制限の詳細については、「クォータ」ページを参照してください。

Amazon Braket のリージョンとエンドポイント

Amazon Braket は、次の で使用できます AWS リージョン。

Amazon Braket のリージョンの可用性

リージョン名	リージョン	Braket エンドポイント	QPU
米国東部 (バージニア北部)	us-east-1	braket.us-east-1.amazonaws.com	ionQ
米国東部 (バージニア北部)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
米国西部 (北カリフォルニア)	us-west-1	braket.us-west-1.amazonaws.com	Rigetti
欧州北部 1 (ストックホルム)	eu-north-1	braket.eu-north-1.amazonaws.com	I"
欧州西部 2 (ロンドン)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Amazon Braket は利用可能なリージョンから実行できますが、各 QPU は単一のリージョンでのみ使用できます。QPU デバイスで実行される量子タスクは、そのデバイスの リージョンの Amazon Braket コンソールで表示できます。Amazon Braket SDK を使用している場合は、作業しているリージョンに関係なく、量子タスクを任意の QPU デバイスに送信できます。SDK は、指定された QPU のリージョンへのセッションを自動的に作成します。

が リージョンとエンドポイントと AWS 連携する方法の一般的な情報については、AWS 「 全般のリファレンス」の「 AWS のサービス エンドポイント」を参照してください。

量子タスクはいつ実行されますか ?

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

回路を送信すると、Amazon Braket は指定したデバイスに送信します。量子処理ユニット (QPU) とオンデマンドシミュレーターの量子タスクは、キューに入れられ、受信した順序で処理されます。量子タスクを送信した後の処理に必要な時間は、他の Amazon Braket のお客様が送信したタスクの数と複雑さ、および選択した QPU の可用性によって異なります。

E メールまたは SMS でのステータス変更通知

Amazon Braket は、QPU の可用性が変化し EventBridge たとき、または量子タスクの状態が変化したときに、Amazon にイベントを送信します。デバイスおよび量子タスクのステータス変更通知を E メールまたは SMS メッセージで受信するには、次の手順に従います。

1. Amazon SNS トピックとサブスクリプションを E メールまたは SMS に作成します。メールまたは SMS の可用性は、リージョンによって異なります。詳細については、Amazon SNSの開始方法」および「SMS メッセージの送信」を参照してください。
2. SNS トピックへの通知をトリガー EventBridge するルールを に作成します。詳細については、「Amazon による Amazon Braket のモニタリング EventBridge」を参照してください。

量子タスク完了アラート

Amazon Simple Notification Service (SNS) を介して通知を設定して、Amazon Braket 量子タスクが完了したときにアラートを受け取ることができます。アクティブな通知は、大きなタスクを送信する場合や、デバイスの可用性ウィンドウ外でタスクを送信する場合など、長い待機時間が予想される場合に便利です。タスクが完了するのを待ちたくない場合は、SNS 通知を設定してください。

Amazon Braket ノートブックでは、セットアップ手順を順を追って説明します。詳細については、「通知を設定するための Amazon Braket のサンプルノートブック」を参照してください。

QPU の可用性ウィンドウとステータス

QPU の可用性はデバイスによって異なります。

Amazon Braket コンソールのデバイスページで、現在および今後の可用性ウィンドウとデバイスのステータスを確認できます。さらに、各デバイスページには、量子タスクとハイブリッドジョブの個々のキューの深さが表示されます。

可用性ウィンドウに関係なく、お客様が利用できない場合、デバイスはオフラインと見なされます。例えば、スケジュールされたメンテナンス、アップグレード、または運用上の問題により、オフラインになる可能性があります。

キューの可視性

量子タスクまたはハイブリッドジョブを送信する前に、デバイスキューの深さを確認することで、目の前の量子タスクまたはハイブリッドジョブの数を確認できます。

キューの深さ

Queue depth は、特定のデバイスに対してキューに入れられた量子タスクとハイブリッドジョブの数を指します。デバイスの量子タスクとハイブリッドジョブキューの数は、 Braket Software Development Kit (SDK)または からアクセスできますAmazon Braket Management Console。

1. タスクキューの深さとは、現在通常の優先度で実行を待っている量子タスクの合計数を指します。
2. 優先度タスクキューの深さとは、 を通じて実行を待機している送信された量子タスクの総数を指しますAmazon Braket Hybrid Jobs。これらのタスクは、スタンドアロンタスクの前に実行されます。
3. ハイブリッドジョブキューの深さとは、デバイスで現在キューに入っているハイブリッドジョブの総数を指します。ハイブリッドジョブの一部としてQuantum tasks送信された は優先され、 に集約されますPriority Task Queue。

を介してキューの深さを表示したいお客様は、次のコードス二ペットを変更して、量子タスクまたはハイブリッドジョブのキュー位置を取得Braket SDKできます。

```
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'
```

量子タスクまたはハイブリッドジョブを QPU に送信すると、ワークロードが QUEUED状態になる可能性があります。Amazon Braket は、量子タスクとハイブリッドジョブキューの位置を可視化します。

キューの位置

Queue position は、それぞれのデバイスキュー内の量子タスクまたはハイブリッドジョブの現在の位置を指します。量子タスクまたはハイブリッドジョブの場合は、 Braket Software Development Kit (SDK)または を使用して取得できますAmazon Braket Management Console。

を介してキューの位置を表示したいお客様は、次のコードスニペットを変更して、量子タスクまたはハイブリッドジョブのキューの位置を取得Braket SDKできます。

```
# 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
```

Amazon Braket の使用を開始する

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket を有効にする」の指示に従ったら、AmazonBraket の使用を開始できます。

開始する手順は次のとおりです。

- Amazon Braket を有効にする
- Amazon Braket ノートブックインスタンスを作成する
- Amazon Braket Python SDK を使用して最初のサーキットを実行します。
- 最初の量子アルゴリズムを実行する

Amazon Braket を有効にする

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

アカウントで Amazon Braket を有効にするには、AWS コンソール を使用します。

前提条件

Amazon Braket を有効にして実行するには、Amazon Braket アクションを開始する権限を持つユーザーまたはロールが必要です。 Amazon Braket これらのアクセス許可は IAM AmazonBraketFullAccess ポリシー (arn:aws:iam::aws:policy/AmazonBraketFullAccess) に含まれています。

> **Note**
>
> 管理者の場合:
> 他のユーザーに Amazon Braket へのアクセスを許可するには、AmazonBraketFullAccessポリシーをアタッチするか、作成したカスタムポリシーをアタッチして、ユーザーに許可を付与します。Amazon Braket の使用に必要なアクセス許可の詳細については、「Amazon Braket へのアクセスを管理する」を参照してください。

Amazon Braket を有効にする手順

1. を使用して Amazon Braket コンソールにサインインします AWS アカウント。
2. Amazon Braket コンソールを開きます。
3. Braket ランディングページから、開始方法をクリックしてサービスダッシュボードページに移動します。サービスダッシュボードの上部にあるアラートは、次の 3 つのステップを順を追って説明します。
   a. サービスにリンクされたロール (SLR) の作成
   b. サードパーティーの量子コンピュータへのアクセスの有効化
   c. 新しい Jupyter Notebook インスタンスの作成

サードパーティーの量子デバイスを使用するには、自分自身 AWSとそれらのデバイス間のデータ転送に関する特定の条件に同意する必要があります。本ライセンス条項の条項は、Amazon Braket コンソールの「アクセス許可と設定」ページの「一般」タブに記載されています。

> **Note**
>
> Braket ローカルシミュレーターやオンデマンドシミュレーターなど、サードパーティーが関係しない量子デバイスは、「サードパーティーデバイスの有効化」の契約に同意せずに使用できます。
> サードパーティーのハードウェアにアクセスする場合は、これらの条件に同意してサードパーティーのデバイスを使用できるようにするだけで、アカウントごとに 1 回だけ実行できます。

Amazon Braket ノートブックインスタンスを作成する

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket は、フルマネージドの Jupyter ノートブックを提供し、作業を始めることができます。Amazon Braket ノートブックインスタンスは、Amazon SageMaker ノートブックインスタンスに基づいています。次の手順では、新規および既存のお客様向けに新しいノートブックインスタンスを作成するために必要な手順の概要を説明します。

Amazon Braket の新規顧客

1. Amazon Braket コンソールを開き、左側のペインのダッシュボードページに移動します。
2. ダッシュボードページの中央にある「Amazon Braket へようこそ」モーダルで「開始方法」をクリックして、ノートブック名を指定します。 Amazon Braket これにより、デフォルトの Jupyter Notebook が作成されます。
3. ノートブックの作成には数分かかる場合があります。ノートブックはノートブックページに一覧表示され、ステータスは保留中です。ノートブックインスタンスを使用する準備ができると、ステータスは に変わりますInService。ノートブックの更新ステータスを表示するには、ページを更新する必要がある場合があります。

既存の Amazon Braket のお客様

1. Amazon Braket コンソールを開き、左側のペインでノートブックを選択し、ノートブックインスタンスの作成を選択します。ノートブックがゼロの場合は、標準セットアップを選択してデフォルトの Jupyter Notebook を作成し、英数字とハイフンのみを使用してノートブックインスタンス名を入力し、任意のビジュアルモードを選択します。次に、ノートブックの非アクティブマネージャーを有効または無効にします。
   a. 有効にした場合、ノートブックがリセットされるまでに必要なアイドル時間を選択します。ノートブックをリセットすると、コンピューティング料金は発生しなくなりますが、ストレージ料金は続行されます。
   b. ノートブックインスタンスの残りのアイドル時間を表示するには、コマンドバーに移動し、Braket タブを選択し、次に Inactivity Manager タブを選択します。

> **Note**
>
> 作業が失われないようにするには、SageMaker ノートブックインスタンスを git リポジトリ と統合することを検討してください。別の方法として、作業を フォルダと /Braket Examplesフォルダの外に移動する/Braket Algorithmsと、ノートブックインスタンスの再起動によってファイルが上書きされなくなります。

2. (オプション) 詳細設定では、アクセス許可、追加設定、ネットワークアクセス設定を含むノートブックを作成できます。
   a. ノートブック設定で、インスタンスタイプを選択します。デフォルトでは、標準で費用対効果の高いインスタンスタイプ ml.t3.medium が選択されています。インスタンスの料金の詳細については、「Amazon の SageMaker 料金」を参照してください。パブリック Github リポジトリをノートブックインスタンスに関連付ける場合は、Git リポジトリドロップダウンをクリックし、リポジトリドロップダウンメニューから URL からパブリック git リポジトリのクローンを選択します。 Git リポジトリ URL テキストバーにリポジトリの URL を入力します。
   b. アクセス許可 で、オプションの IAM ロール、ルートアクセス、および暗号化キーを設定します。
   c. ネットワーク で、Jupyter Notebookインスタンスのカスタムネットワークとアクセス設定を設定します。
3. 設定を確認し、タグを設定してノートブックインスタンスを識別し、 の起動をクリックします。

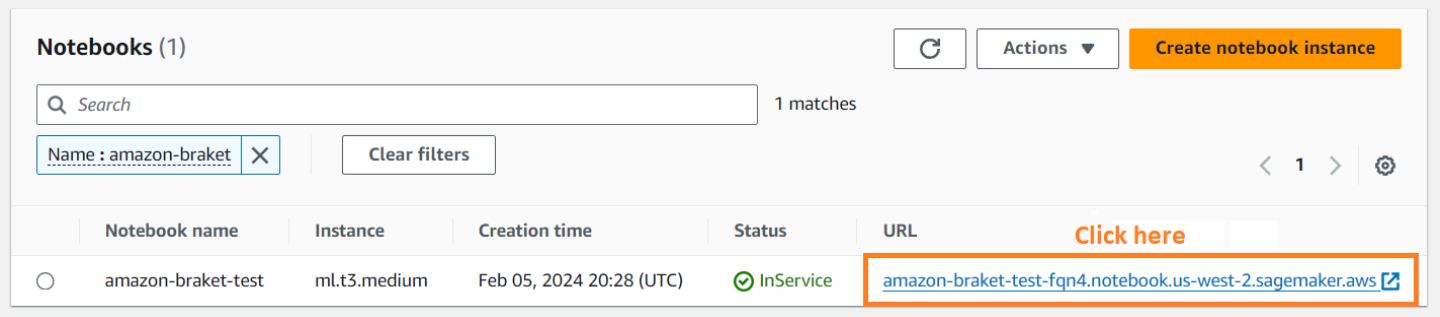
> **Note**
>
> Amazon Braket ノートブックインスタンスは、Amazon Braket および Amazon Braket SageMaker コンソールで表示および管理できます。追加の Amazon Braket ノートブック設定は、SageMaker コンソール から利用できます。

Amazon Braket SDK 内の AWS Amazon Braket コンソールで作業している場合、プラグインは作成したノートブックにプリロードされます。独自のマシンで を実行する場合は、 コマンドを実行するとき、pip install amazon-braket-sdkまたはプラグインpip install amazon-braket-pennylane-pluginで使用するコマンドを実行するときに SDK と PennyLane プラグインをインストールできます。

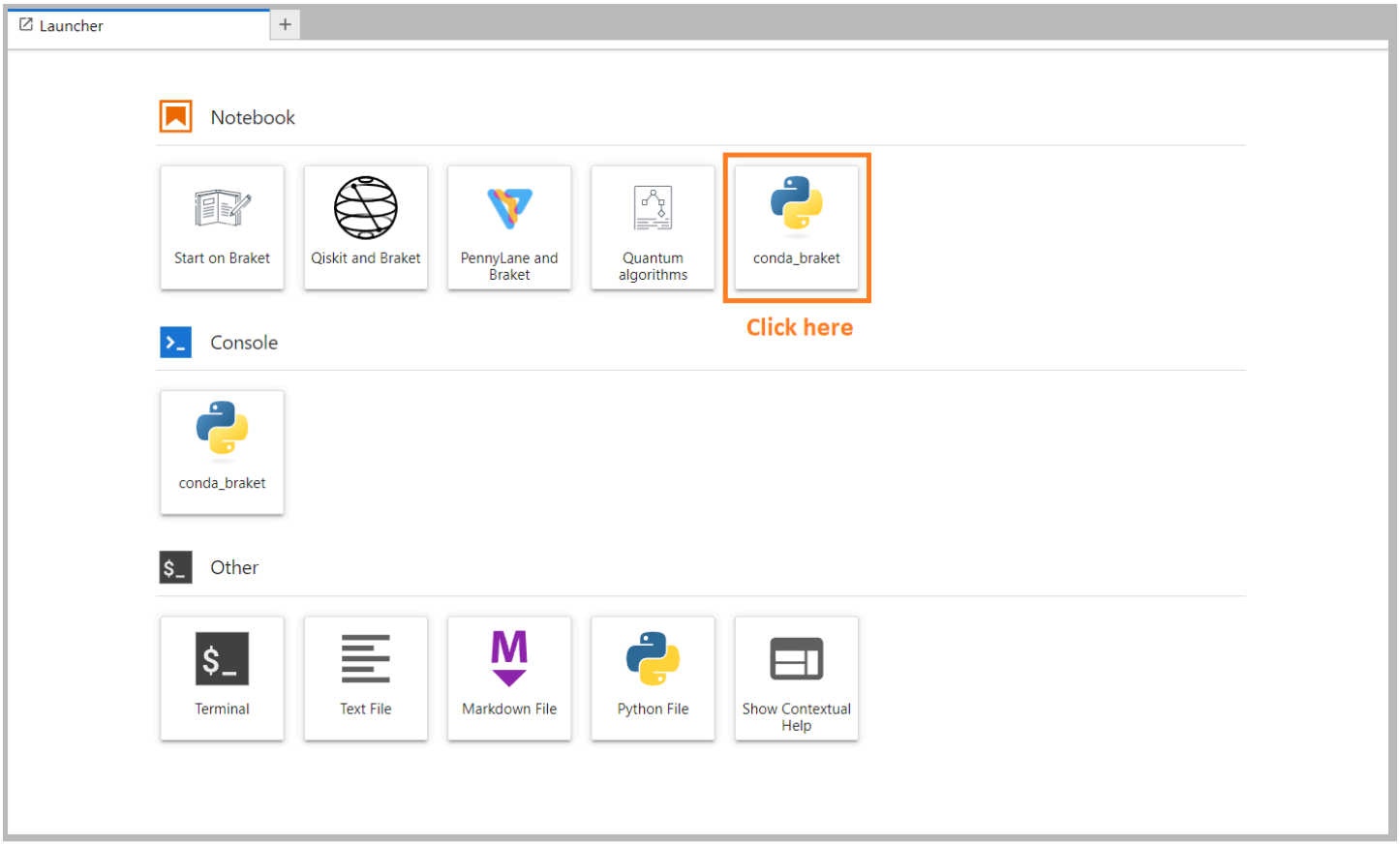
Amazon Braket Python SDK を使用して最初のサーキットを実行します。

> **Tip**
>
> による量子コンピューティングの基礎について説明します AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

ノートブックインスタンスが起動したら、作成したノートブックを選択して、標準の Jupyter インターフェイスでインスタンスを開きます。



Amazon Braket ノートブックインスタンスには、Braket SDK Amazon とそのすべての依存関係がプリインストールされています。`conda_braket` カーネルを使用して新しいノートブックを作成することから始めます。



シンプルな「こんにちは、世界！」という 例から始めることができます。まず、ベル状態を準備する回路を構築し、その回路を異なるデバイスで実行して結果を取得します。

まず、AmazonBraket SDK モジュールをインポートし、単純なベルステート回路を定義します。

```
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)
```

次のコマンドで回路を視覚化できます。

```
print(bell)
```

[Run your circuit on the local simulator] (ローカルシミュレーターで回路を実行する)

次に、回路を実行する量子デバイスを選択します。Amazon Braket SDK には、ラピッドプロトタイピングとテスト用のローカルシミュレーターが付属しています。小規模な回路にはローカルシミュレーターを使用することをお勧めします。これは最大 25 です qubits (ローカルハードウェアによって異なります) 。

ローカルシミュレーターをインスタンス化する方法は次のとおりです。

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

そして、回路を実行します。

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

次のような結果が表示されます。

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

準備した特定のベル状態は |00> と |11> の等号重ね合わせであり、期待どおりに測定結果として 00 と 11 のほぼ等しい (shotノイズまで) 分布がわかります。

オンデマンドシミュレーターで回路を実行する

Amazon Braket は、より大きな回路を実行するためのオンデマンドの高性能シミュレーター へのアクセスも提供します。 SV1はSV1、最大 34 個の量子回路のシミュレーションを可能にするオンデマンドのステートベクトルシミュレーターですqubits。詳細については、SV1「サポートされているデバイス」セクションと AWS コンソールを参照してください。量子タスクを SV1 (およびTN1任意の QPU で) 実行すると、量子タスクの結果はアカウントの S3 バケットに保存されます。バケットを指定しない場合、Braket SDK によってデフォルトのバケットが作成されますamazon-braket-{region}-{accountID}。詳細については、「Amazon Braket へのアクセスの管理」を参照してください。

> **Note**
>
> 実際の既存のバケット名を入力します。次の例では、バケット名として example-bucket が表示されます。Amazon Braket のバケット名は、常に で始まり、その後に追加した他の

識別文字amazon-braket-が続きます。S3 バケットの設定方法に関する情報が必要な場合は、Amazon S3 の開始方法」を参照してください。

```
# 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)
```

で回路を実行するにはSV1、.run( )コールで位置引数として以前に選択した S3 バケットの場所を指定する必要があります。

```
# 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)
```

Amazon Braket コンソールには、量子タスクに関する詳細情報が表示されます。コンソールの量子タスクタブに移動すると、量子タスクがリストの上部にあるはずです。または、一意の量子タスク ID またはその他の基準を使用して量子タスクを検索することもできます。

> **Note**
>
> 90 日後、AmazonBraket は量子タスクに関連するすべての量子タスク IDs とその他のメタデータを自動的に削除します。詳細については、データ保持期間を参照してください。

QPU で実行する

Amazon Braket では、1 行のコードを変更するだけで、物理量子コンピュータで前の量子回路の例を実行できます。Amazon Braket は、IonQ、、Oxford Quantum Circuits、QuEraおよび からQPU デバイスへのアクセスを提供しますRigetti。さまざまなデバイスと可用性ウィンドウに関する情報は、サポートされているデバイスセクションと、 AWS コンソールのデバイスタブにあります。次の例は、Rigettiデバイスをインスタンス化する方法を示しています。

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

このコードのIonQデバイスを選択します。

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

デバイスを選択し、ワークロードを実行する前に、次のコードでデバイスキューの深さをクエリして、量子タスクまたはハイブリッドジョブの数を判断できます。さらに、お客様は のデバイスページでデバイス固有のキューの深さを表示できますAmazon Braket Management Console。

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

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

タスクを実行すると、Amazon Braket SDK は結果をポーリングします (デフォルトのタイムアウトは 5 日) 。次の例に示すように、 .run( ) コマンドで poll_timeout_secondsパラメータを変更することで、このデフォルトを変更できます。ポーリングタイムアウトが短すぎると、QPU が使用できず、ローカルタイムアウトエラーが返されるなど、ポーリング時間内に結果が返されない可能性があることに注意してください。task.result( ) 関数を呼び出して、ポーリングを再開できます。

```
# 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)
```

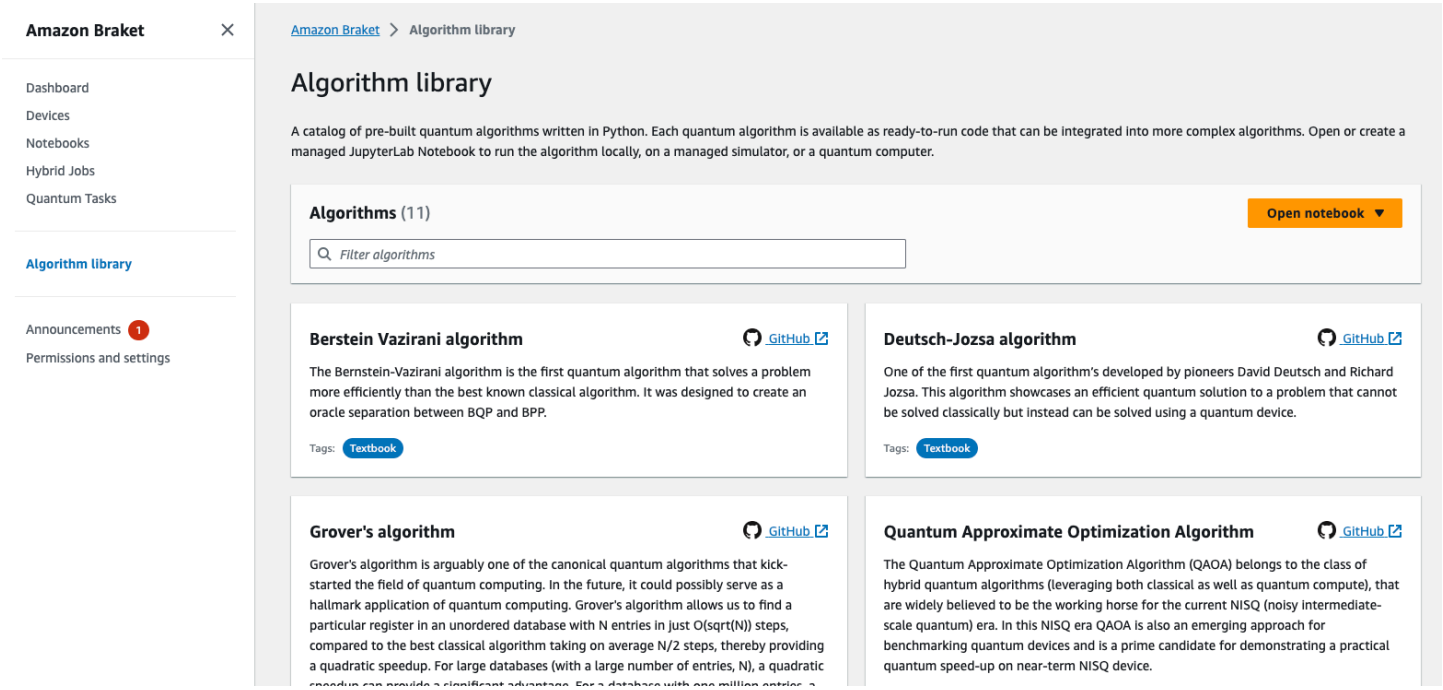
さらに、量子タスクまたはハイブリッドジョブを送信した後、 queue_position( )関数を呼び出してキューの位置を確認できます。

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

最初の量子アルゴリズムを実行する

> **Tip**
>
> で量子コンピューティングの基礎を学びます AWS。Amazon Braket Digital Learning Plan に登録し、一連の学習コースとデジタル評価を完了したら、独自のデジタルバッジを獲得します。

Amazon Braket アルゴリズムライブラリは、Python で記述された構築済みの量子アルゴリズムのカタログです。これらのアルゴリズムをそのまま実行することも、開始点として使用してより複雑なアルゴリズムを構築することもできます。Braket コンソールからアルゴリズムライブラリにアクセスできます。Github の Braket アルゴリズムライブラリにアクセスすることもできます: https://github.com/aws-samples/amazon-braket-algorithm-library。



Braket コンソールには、アルゴリズムライブラリで使用可能な各アルゴリズムの説明が表示されます。 GitHub リンクを選択して各アルゴリズムの詳細を表示するか、ノートブックを開くを選択して、使用可能なすべてのアルゴリズムを含むノートブックを開くか作成します。ノートブックオプションを選択すると、ノートブックのルートフォルダに Braket アルゴリズムライブラリが表示されます。

Amazon Braket を使用する

このセクションでは、量子回路を設計し、これらの問題を量子タスクとしてデバイスに送信し、Amazon Braket SDK で量子タスクをモニタリングする方法について説明します。

Amazon Braket のリソースを操作する主な方法は、次のとおりです。

- Amazon Braket コンソールには、リソースと量子タスクの作成、管理、モニタリングに役立つデバイス情報とステータスが用意されています。
- Amazon Braket Python SDK および コンソールから量子タスクを送信して実行します。SDK には、事前設定された Amazon Braket ノートブックからアクセスできます。
- Amazon Braket API は、AmazonBraket Python SDK およびノートブックからアクセスできます。量子コンピューティングをプログラムで操作するアプリケーションを構築するAPI場合は、 に直接呼び出しを行うことができます。

このセクションの例では、AmazonBraket Python SDK と Amazon AWS Python SDK for Braket (Boto3) APIを直接使用する方法を示します。

Amazon Braket Python SDK の詳細

Amazon Braket Python SDK を使用するには、まず AWS Python SDK for Braket (Boto3) をインストールして、 と通信できるようにします AWS API。Amazon Braket Python SDK は、量子カスタマーにとって Boto3 の便利なラッパーと考えることができます。

- Boto3 には、 を利用するために必要なインターフェイスが含まれています AWS API。(Boto3 は、と通信する大規模な Python SDK であることに注意してください AWS API。 ほとんどの は Boto3 インターフェイス AWS のサービス をサポートしています) 。
- Amazon Braket Python SDK には、回路、ゲート、デバイス、結果タイプ、および量子タスクのその他の部分用のソフトウェアモジュールが含まれています。プログラムを作成するたびに、その量子タスクに必要なモジュールをインポートします。
- Amazon Braket Python SDK はノートブックからアクセスできます。ノートブックには、量子タスクの実行に必要なすべてのモジュールと依存関係がプリロードされています。
- ノートブックを操作したくない場合は、AmazonBraket Python SDK から任意の Python スクリプトにモジュールをインポートできます。

Boto3 をインストールすると、AmazonBraket Python SDK を使用して量子タスクを作成する手順の概要は次のようになります。

1. (オプション) ノートブックを開きます。
2. 回路に必要な SDK モジュールをインポートします。
3. QPU またはシミュレーターを指定します。
4. 回路をインスタンス化します。
5. 回路を実行します。
6. 結果を収集します。

このセクションの例では、各ステップについて詳しく説明します。

その他の例については、 のAmazon Braket Examples」リポジトリを参照してください GitHub。

このセクションの内容:

- Hello AHS: 最初のアナログハミルトニアシミュレーションを実行する
- SDK で回路を構築する
- QPUs とシミュレーターへの量子タスクの送信
- OpenQASM 3.0 で回路を実行する
- QuEraの Aquila を使用してアナログプログラムを送信する
- Boto3 を使用する

Hello AHS: 最初のアナログハミルトニアシミュレーションを実行する

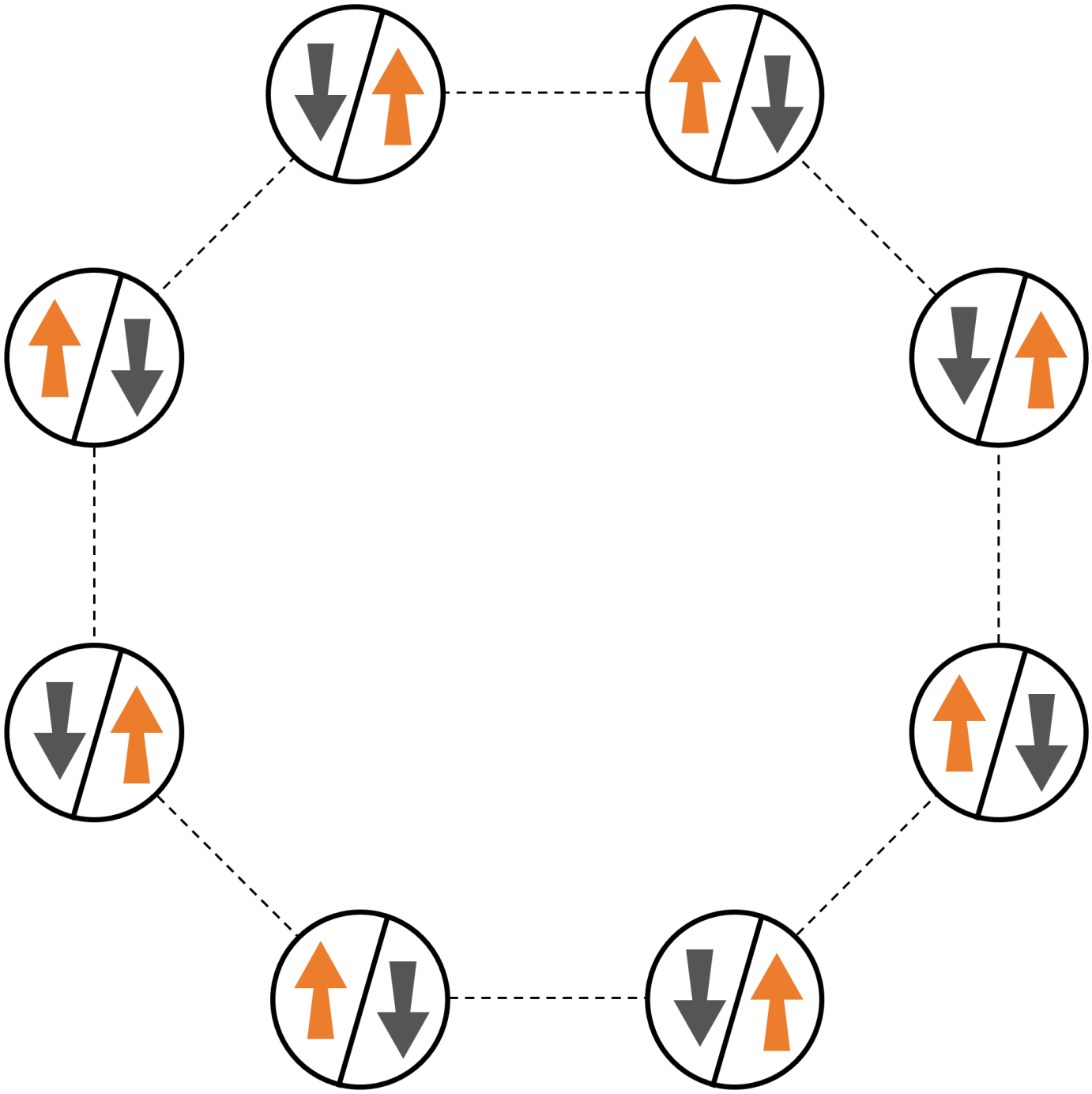
AHS

アナログハミルトニアシミュレーション (AHS) は、量子回路とは異なる量子コンピューティングのパラダイムです。各ゲートは一度に数量子ビットにのみ作用する一連のゲートではなく、AHS プログラムは問題のハミルトニアンの時間依存および空間依存パラメータによって定義されます。システムのハミルトニアンは、そのエネルギーレベルと外部力の影響をエンコードし、それらが組み合わさってその状態の時間の進化を管理します。N 量子ビットシステムの場合、ハミルトニアンは複雑な数値の 2^N X2^N 二乗行列で表すことができます。

AHS を実行できる量子デバイスは、パラメータ (例えば、コヒーレント駆動場の振幅やデチューニング) を調整して、カスタムハミルトニアンにおける量子システムの時間進化に密接に近似します。AHS パラダイムは、多くの相互作用するパーティクルの量子系の静的および動的特性をシミュレートするのに適しています。の Aquila デバイスなど、専用の QPUs QuEraを使用すると、従来のハードウェアでは不可能なサイズを持つシステムの時間進化をシミュレートできます。

インタラクションスピンチェーン

多数の相互作用するパーティクルのシステムの正規例については、8 つのスピンのリングを考えてみましょう (それぞれが「アップ」|↑# および「ダウン」|↓# 状態になる可能性があります) 。小規模ではありますが、このモデルシステムはすでに自然に発生する磁気マテリアルのいくつかの興味深い現象を示しています。この例では、連続するスピンが反対方向に向く、いわゆる強直性防止順序を準備する方法を示します。



配置

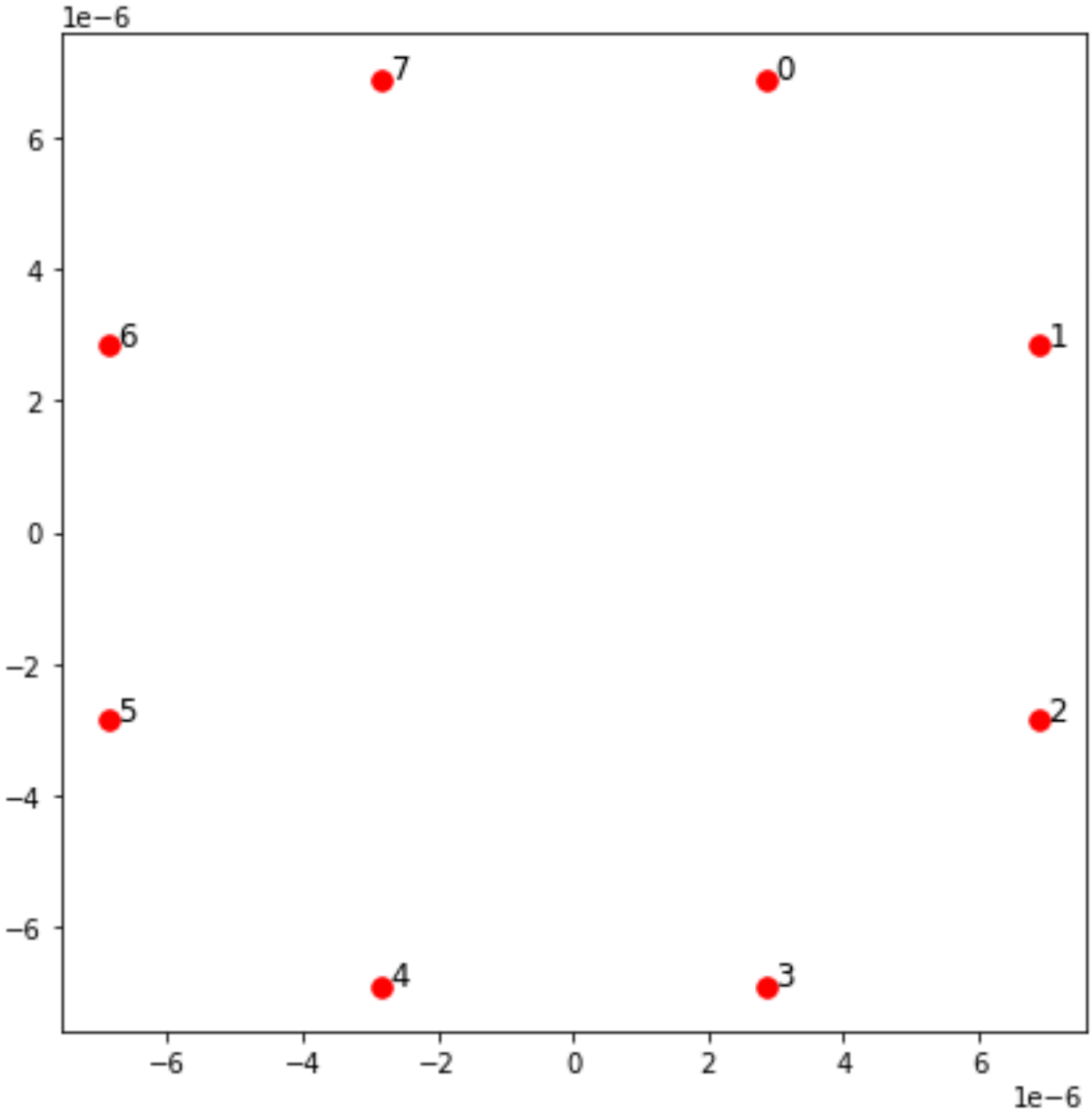
スピンごとに 1 つの中立アトムを使用し、「上」スピン状態と「下」スピン状態は、それぞれアトムの勾配 Rydberg 状態とグラウンド状態でエンコードされます。まず、2 D 配置を作成します。上記のスピンリングは、次のコードでプログラムできます。

前提条件： Braket SDK をインストールする必要があります。(Braket がホストするノートブックインスタンスを使用している場合、この SDK にはノートブックがプリインストールされています)。プロットを再現するには、シェルコマンド を使用して matplotlib を個別にインストールする必要もあります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)
```

また、 を使用してプロットすることもできます。

```
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
```



インタラクション

強固化防止フェーズを準備するには、隣接するスピン間の相互作用を誘発する必要があります。このために van der Waals インタラクションを使用します。これは、中立な原子デバイス (のAquilaデ

バイスなど) によってネイティブに実装されますQuEra。スピン表現を使用すると、このインタラクションのハミルトニアン用語を、すべてのスピンペア (j,k) の合計として表現できます。

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

ここで、nj =$_{jj}$|↑##↑| はスピン j が「up」状態にある場合にのみ 1 の値を取り、それ以外の場合は 0 の値を取る演算子です。強度は V $_{j,k}$=C $_6$/(d $_{j,k}$) ^{6} で、C $_6$は固定係数、d $_{j,k}$はスピン j と k の間のユークリッド距離です。このインタラクション期間の即時効果は、スピン j とスピン k の両方が「アップ」している状態が、(V の量だけ) エネルギーが増加していることです$_{j,k}$。AHS プログラムの残りの部分を慎重に設計することで、この相互作用により、隣接するスピンが両方とも「アップ」状態になるのを防ぐことができます。これは、一般的に「Rydberg ブロック」と呼ばれる効果です。

運転フィールド

AHS プログラムの開始時、すべてのスピン (デフォルトでは) は「ダウン」状態で始まり、いわゆる強分解フェーズにあります。強固化防止フェーズを準備するという目標を念頭に置いて、スピンをこの状態から「アップ」状態が優先される多体状態にスムーズに移行する時間依存コヒーレント駆動フィールドを指定します。対応するハミルトニアンは、次のように記述できます。

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

ここで、「(t)」、「(t)」、「」、「(t)」は、すべてのスピンに均一に影響する駆動フィールドの時間依存的なグローバル振幅 (Rabi 周波数)、フェーズ、およびデチューニングです。ここで、S $_{-,k}$=k $_k$|↓##↑| と S $_{+,k}$=(S $_{-,k}$) †=$_{kk}$|↑##↓| はスピン k の降格演算子と降格演算子であり、n $_k$=$_{kk}$|↑##↑| は以前と同じ演算子です。走行フィールドの 部分は、すべてのスピンの「ダウン」状態と「アップ」状態を同時に一貫して結合し、一方、「アップ」状態のエネルギー報酬を制御します。

強熱フェーズから強熱フェーズへのスムーズな移行をプログラムするには、次のコードで駆動フィールドを指定します。

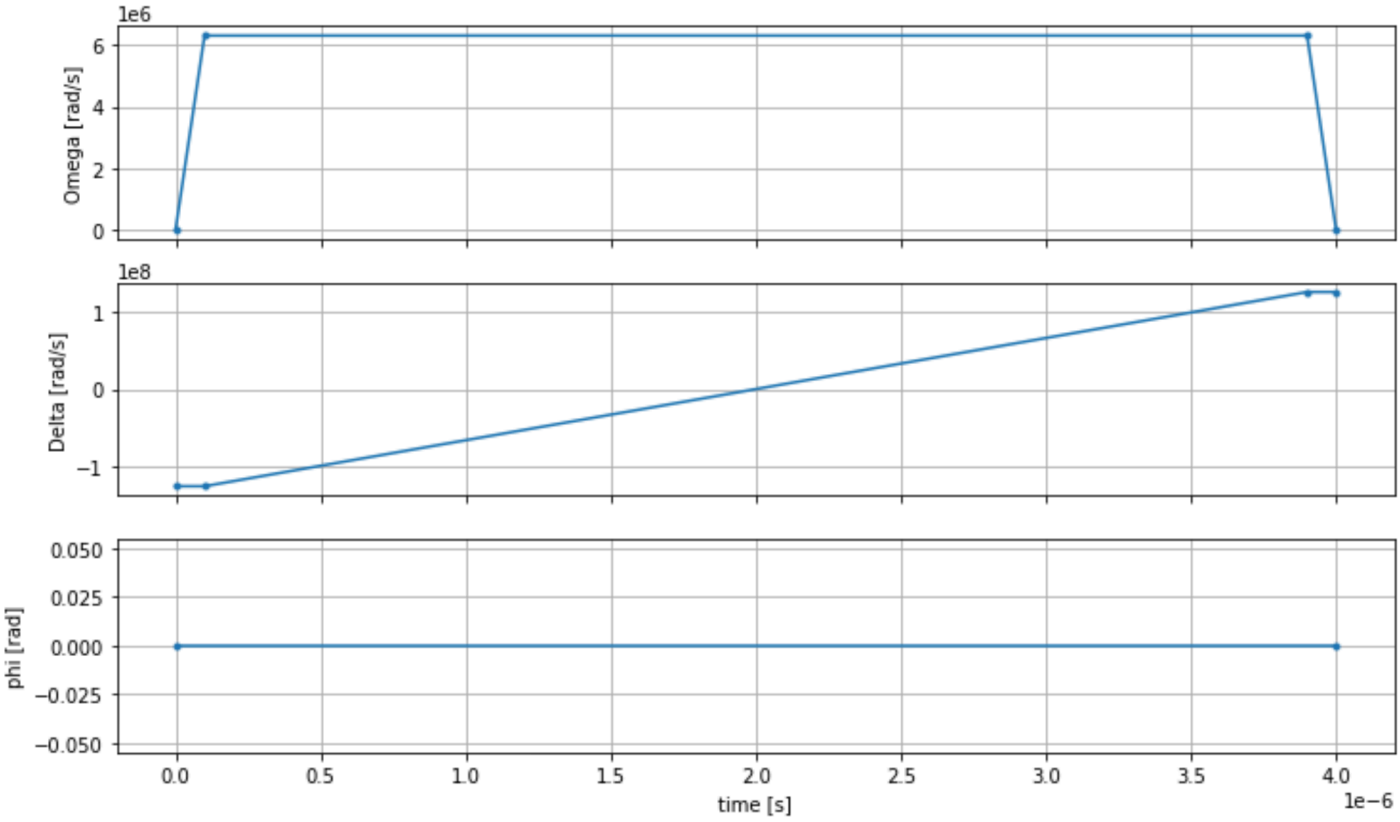
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField

# 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
)
```

次のスクリプトを使用して、運転フィールドの時系列を視覚化できます。

```
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 プログラム

登録、運転フィールド (および暗黙的な van der Waals インタラクション) は、Analog Hamiltonian Simulation プログラム を構成しますahs_program。

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

ローカルシミュレーターでの実行

この例は小さい (スピンが 15 回未満) ため、AHS 互換 QPU で実行する前に、Braket SDK に付属するローカル AHS シミュレーターで実行できます。ローカルシミュレーターは Braket SDK で無料で利用できるため、コードが正しく実行されるようにするのがベストプラクティスです。

ここでは、ショット数を高い値 (つまり、100 万) に設定することができます。ローカルシミュレーターは量子状態の時間進化を追跡し、最終状態からサンプルを抽出するため、ショット数を増やしながら、合計ランタイムをわずかにだけ増やすためです。

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

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

シミュレーターの結果の分析

各スピンの状態 (「ダウン」の場合は「d」、「アップ」の場合は「u」、または空のサイトの場合は「e」) を推測する次の関数を使用してショット結果を集計し、各設定がショット全体で発生した回数をカウントできます。

```
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
```

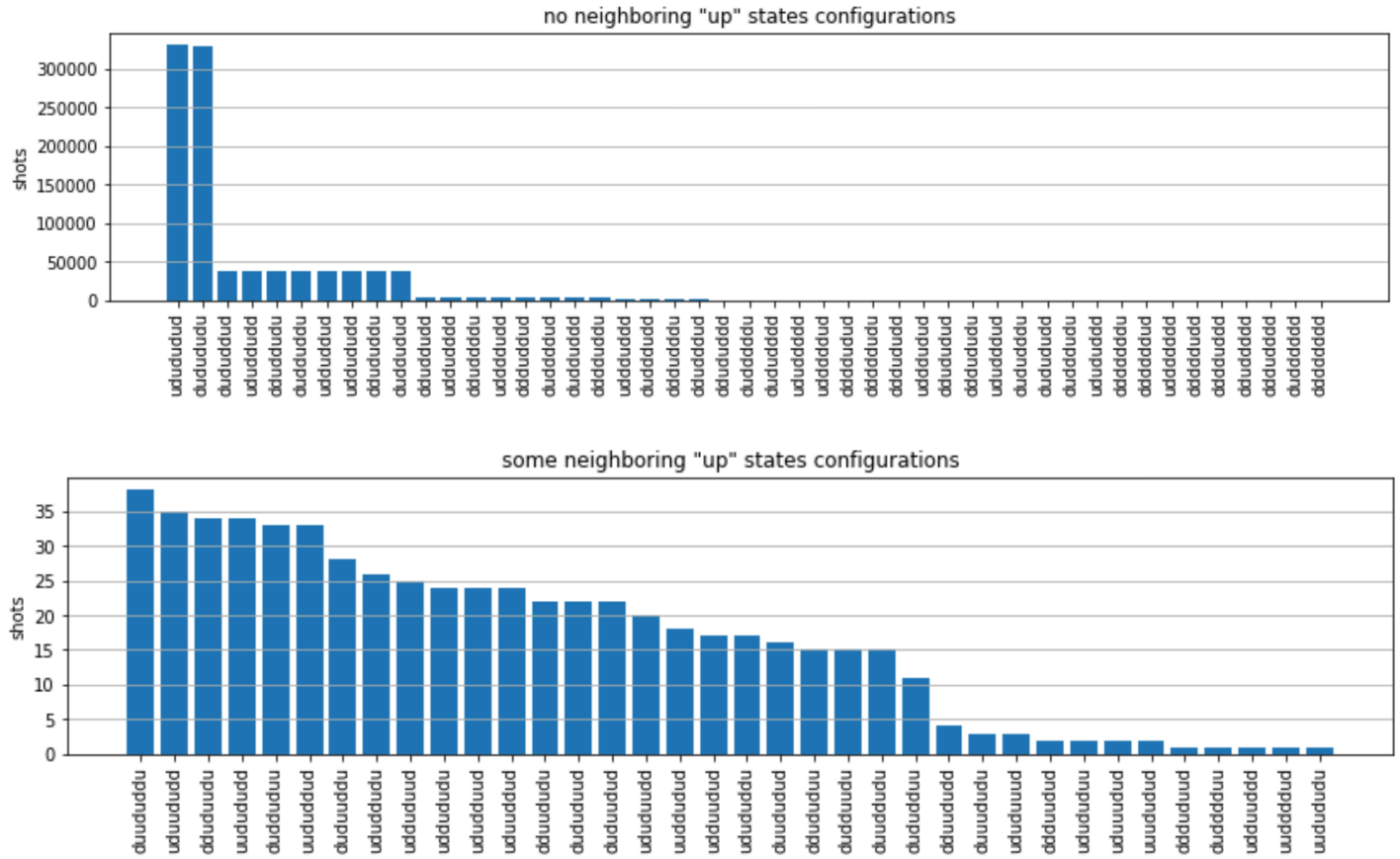
```
    """
    state_counts = Counter()
    states = ['e', 'u', 'd']
    for shot in result.measurements:
        pre = shot.pre_sequence
        post = shot.post_sequence
        state_idx = np.array(pre) * (1 + np.array(post))
        state = "".join(map(lambda s_idx: states[s_idx], state_idx))
        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, 'dudуddud': 38033, ...}
```

以下はcounts、ショット全体で各状態設定が観測された回数をカウントするディクショナリです。また、次のコードで視覚化することもできます。

```
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)
```



プロットから、次の観測値を読み、強分解防止フェーズを正常に準備したことを確認できます。

1. 一般に、ブロックされていない状態 (隣接するスピンが 2 つも「アップ」状態ではない状態) は、隣接するスピンの少なくとも 1 つのペアが両方とも「アップ」状態である状態よりも一般的です。
2. 一般的に、設定がブロックされていない限り、「アップ」の興奮が多い状態が優先されます。
3. 最も一般的な状態は、実際には完全な強磁気状態 "dudududu"および です"udududud"。
4. 2 番目に一般的な状態は、連続する分離が 1、2、2 の「アップ」エキサイテーションが 3 つしかない状態です。これは、van der Waals のインタラクションが、最も近い近傍にも影響する (ただし、はるかに小さい) ことを示しています。

QuEraの Aquila QPU での実行

前提条件 : Braket SDK をインストールする pip とは別に、Amazon Braket を初めて使用する場合は、必要な「使用開始」ステップを完了していることを確認してください。

> **Note**
>
> Braket がホストするノートブックインスタンスを使用している場合、Braket SDK にはインスタンスがプリインストールされています。

すべての依存関係をインストールすると、AquilaQPU に接続できます。

```
from braket.aws import AwsDevice

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

AHS プログラムをQuEraマシンに適したものにするには、AquilaQPU で許容される精度レベルに準拠するようにすべての値を四捨五入する必要があります。 (これらの要件は、名前に「解決」が含まれるデバイスパラメータによって管理されます。 ノートブックaquila_qpu.properties.dict()で を実行することで確認できます。 Aquila の機能と要件の詳細については、「Aquila の概要ノートブック」を参照してください) 。これを行うには、discretizeメソッドを呼び出します。

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

これで、プログラム (現在は 100 ショットのみを実行) を Aquila QPU で実行できるようになりました。

> **Note**
>
> このプログラムをAquilaプロセッサで実行すると、コストが発生します。Amazon Braket SDK には、お客様がコスト制限を設定し、コストをほぼリアルタイムで追跡できる Cost Tracker が含まれています。

```
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
```

量子タスクの実行にかかる時間のばらつきが大きいため (可用性ウィンドウと QPU 使用率によって異なります) 、量子タスク ARN を書き留めておくことをお勧めします。したがって、後で次のコードスニペットでそのステータスを確認できます。

```
# 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
```

ステータスが完了すると (Amazon Braket コンソール の量子タスクページからも確認できます)、次の方法で結果をクエリできます。

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

QPU 結果の分析

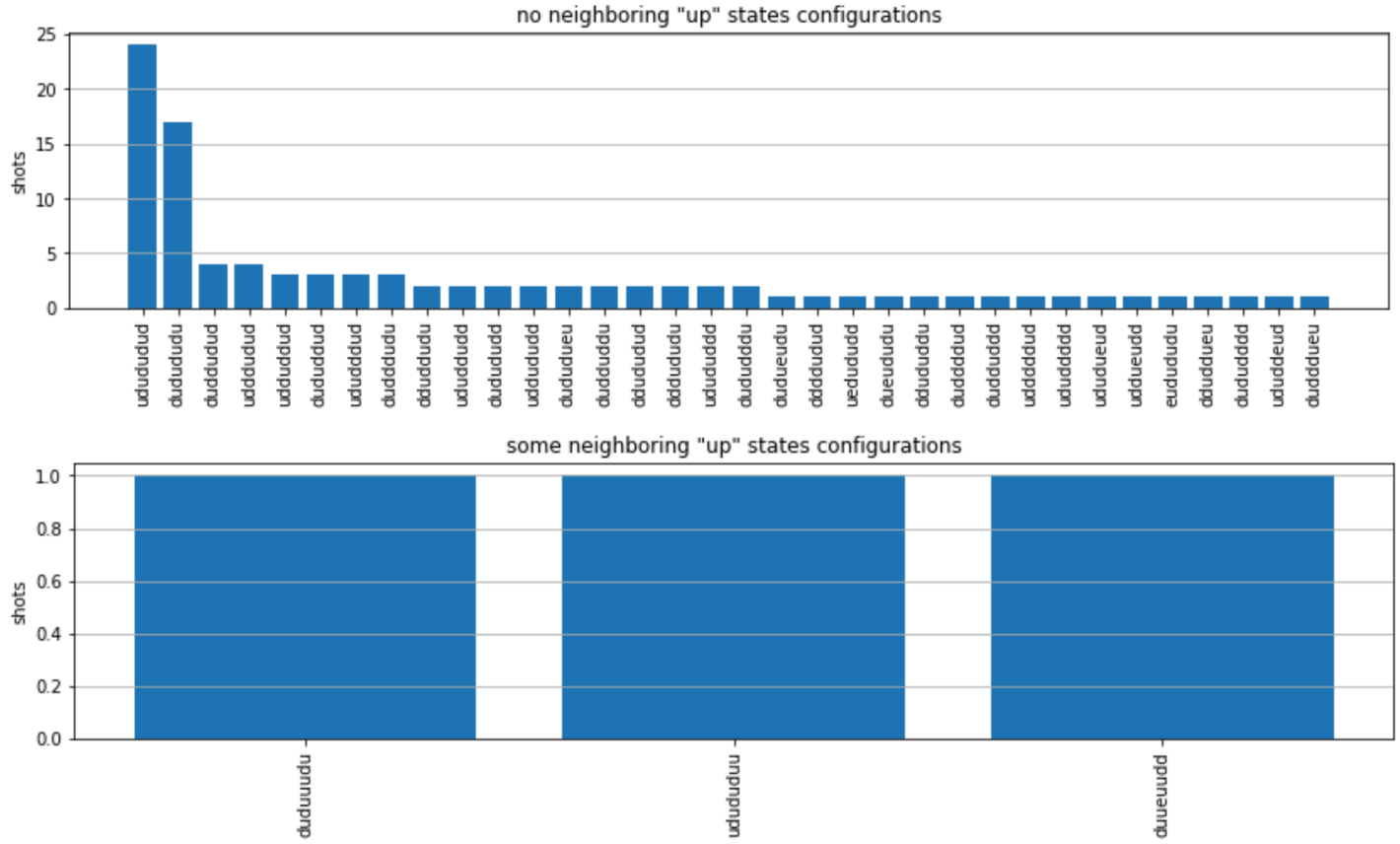
以前と同じget_counts関数を使用すると、カウントを計算できます。

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

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

と でプロットしますplot_counts。

```
plot_counts(counts_aquila)
```



ごく一部のショットに空のサイト (「e」でマーク) があることに注意してください。これは、AquilaQPU の原子ごとの準備の不完全性が 1～2% であることが原因です。これとは別に、ショット数が少ないため、結果は予想される統計変動内でシミュレーションと一致します。

次へ

これで、ローカル AHS シミュレーターと Aquila QPU を使用して Amazon Braket で最初の AHS ワークロードを実行できました。

Rydberg 物理学、アナログハミルトニアンシミュレーション、Aquilaデバイスの詳細については、サンプルノートブック を参照してください。

SDK で回路を構築する

このセクションでは、回路の定義、使用可能なゲートの表示、回路の拡張、および各デバイスがサポートするゲートの表示の例を示します。また、手動で を割り当てqubits、定義されたとおりに回路を実行するようにコンパイラに指示し、ノイズシミュレーターを使用してノイズの多い回路を構築する方法についても説明します。

また、特定の QPUs。詳細については、Amazon Braket での Pulse Control」を参照してください。

このセクションの内容:

- ゲートと回路
- 部分的な測定
- 手動qubit割り当て
- 逐語的なコンパイル
- ノイズシミュレーション
- 回路の検査
- 結果タイプ

ゲートと回路

量子ゲートと回路は Braket Amazon Python SDK の `braket.circuits` クラスで定義されます。SDK から、`Circuit()` を呼び出して、新しい回路オブジェクトをインスタンス化できます。

例: 回路を定義する

この例では、標準単一量子ビットのハダマードゲート`q0`q1q2と 2 量子ビットの CNOT ゲートで構成される 4 つの qubits (ラベル付き`q3`) サンプル回路を定義します。次の例に示すように、 `print`関数を呼び出すことで、この回路を視覚化できます。

```
# 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 |
```

例: パラメータ化された回路を定義する

この例では、空きパラメータに依存するゲートを持つ回路を定義します。これらのパラメータの値を指定して新しい回路を作成するか、回路を送信するときに特定のデバイスで量子タスクとして実行できます。

```
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)
```

パラメータ化された回路からパラメータ化されていない回路を新しく作成するには、次のように単一の `float` (すべての空きパラメータが取る値) または各パラメータの値を指定するキーワード引数のいずれかを指定します。

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

`my_circuit` は変更されていないため、これを使用して固定パラメータ値で多くの新しい回路をインスタンス化できます。

例: 回路のゲートを変更する

次の例では、制御修飾子と電力修飾子を使用するゲートを持つ回路を定義します。これらの変更を使用して、制御されたゲートなどの新しいRyゲートを作成できます。

```
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)
```

ゲート修飾子はローカルシミュレーターでのみサポートされています。

例: 使用可能なすべてのゲートを見る

次の例は、AmazonBraket で使用可能なすべてのゲートを確認する方法を示しています。

```
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)
```

このコードからの出力には、すべてのゲートが一覧表示されます。

```
['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']
```

これらのゲートは、そのタイプの回路のメソッドを呼び出して、回路に追加できます。例えば、 を呼び出して`circ.h(0)`、最初の にハダマードゲートを追加しますqubit。

> **Note**
>
> ゲートが所定の位置に追加され、以下の例では、前の例に挙げたすべてのゲートを同じ回路に追加しています。

```
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)
```

定義済みのゲートセットとは別に、自己定義のユニタリゲートを回路に適用することもできます。これらは、単一量子ビットゲート (次のソースコードを参照) でも、 targetsパラメータでqubits定義された に適用されるマルチ量子ビットゲートでもかまいません。

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

例: 既存の回路を拡張する

命令を追加することで、既存の回路を拡張できます。Instruction は、量子デバイスで実行する量子タスクを記述する量子ディレクティブです。 Instruction演算子には、 タイプのオブジェクトGateのみが含まれます。

```
# 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})
```

例: 各デバイスがサポートするゲートの表示

シミュレーターは Braket SDK のすべてのゲートをサポートしますが、QPU デバイスは小さなサブセットをサポートします。デバイスのサポートされているゲートは、デバイスのプロパティで確認できます。IonQ デバイスの例を次に示します。

```
# 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']
```

サポートされているゲートは、量子ハードウェアで実行する前に、ネイティブゲートにコンパイルする必要があります。回路を送信すると、AmazonBraket はこのコンパイルを自動的に実行します。

例: デバイスでサポートされているネイティブゲートの忠実度をプログラムで取得する

Braket コンソールのデバイスページで忠実度情報を表示できます。プログラムで同じ情報にアクセスすると役立つ場合があります。次のコードは、QPU の 2 つのqubitゲート間で 2 つのゲート忠実度を抽出する方法を示しています。

```
# 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"])
```

部分的な測定

前の例に従って、量子回路内のすべての量子ビットを測定しました。ただし、個々の量子ビットまたは量子ビットのサブセットを測定することは可能です。

例: 量子ビットのサブセットを測定する

この例では、回路の末尾にターゲット量子ビットを持つmeasure命令を追加して、部分的な測定を示します。

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

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

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

```
# Get the results
result = task.result()

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

手動qubit割り当て

から量子コンピュータで量子回路を実行する場合Rigetti、オプションで手動qubit割り当てを使用して、アルゴリズムに使用される qubitsを制御できます。Amazon Braket コンソールと Amazon Braket SDK は、選択した量子処理ユニット (QPU) デバイスの最新のキャリブレーションデータを検査するのに役立つため、実験qubitsに最適なものを選択できます。

手動qubit割り当てにより、回路をより正確に実行し、個々のqubitプロパティを調査できます。研究者や上級ユーザーは、最新のデバイスキャリブレーションデータに基づいて回路設計を最適化し、より正確な結果を得ることができます。

次の例は、qubits明示的に を割り当てる方法を示しています。

```
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)
```

詳細については、「」の Amazon Braket の例 GitHub、またはより具体的には、このノートブック: QPU デバイスでの Qubit の割り当て」を参照してください。

> **Note**
>
> OQC コンパイラは の設定をサポートしていませんdisable_qubit_rewiring=True。このフラグを に設定すると、 というエラーTrueが発生します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。

逐語的なコンパイル

Rigetti、、または Oxford Quantum Circuits (OQC) から量子コンピュータで量子回路を実行する場合IonQ、コンパイラは回路を変更せずに定義されたとおりに実行するように指示できます。逐語的な

コンパイルを使用して、回路全体を指定されたとおりに正確に保持するか (Rigetti、、および でサポートOQC) IonQ、その特定の部分のみを保持するか (Rigettiでのみサポート) を指定できます。ハードウェアベンチマークまたはエラー軽減プロトコルのアルゴリズムを開発する場合、ハードウェアで実行しているゲートと回路レイアウトを正確に指定するオプションが必要です。逐語的なコンパイルでは、特定の最適化ステップをオフにすることでコンパイルプロセスを直接制御できるため、回路が設計どおりに動作します。

逐語的なコンパイルは現在Rigetti、、、および Oxford Quantum Circuits (OQC) デバイスでサポートされておりIonQ、ネイティブゲートを使用する必要があります。逐語的なコンパイルを使用する場合は、デバイスのトポロジをチェックして、ゲートが接続時に呼び出されqubits、回路がハードウェアでサポートされているネイティブゲートを使用することを確認することをお勧めします。次の例は、デバイスでサポートされているネイティブゲートのリストにプログラムでアクセスする方法を示しています。

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

の場合Rigetti、逐語的なコンパイルで使用するdisableQubitRewiring=Trueには、 を設定してqubit再ワイヤリングをオフにする必要があります。コンパイルで逐語的なボックスを使用するときに disableQubitRewiring=Falseが設定されている場合、量子回路は検証に失敗し、実行されません。

回路に対して逐語的なコンパイルが有効で、それをサポートしていない QPU で実行すると、サポートされていないオペレーションによってタスクが失敗したことを示すエラーが生成されます。コンパイラー関数をネイティブにサポートする量子ハードウェアが増えるにつれて、この機能は拡張され、これらのデバイスを含めるようになります。逐語的なコンパイルをサポートするデバイスは、次のコードで照会されたときに、サポートされているオペレーションとしてそれを含めます。

```
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
```

逐語的なコンパイルを使用しても追加コストは発生しません。Braket QPU デバイス、ノートブックインスタンス、オンデマンドシミュレーターで実行された量子タスクについては、Amazon Braketの料金ページで指定されている現在の料金に基づいて引き続き課金されます。詳細については、逐語的なコンパイルサンプルノートブックを参照してください。

> **Note**
>
> OpenQASM を使用して OQCおよび IonQデバイスの回路を書き込み、回路を物理量子ビットに直接マッピングする場合は、 `#pragma braket verbatimdisableQubitRewiring`フラグが OpenQASM によって完全に無視されるため、 を使用する必要があります。

ノイズシミュレーション

ローカルノイズシミュレーターをインスタンス化するには、次のようにバックエンドを変更できます。

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

ノイズの多い回路は、次の 2 つの方法で構築できます。

1. ノイズの多い回路を下部から構築します。
2. 既存のノイズのない回路を使用し、全体にノイズを挿入します。

次の例は、脱分極ノイズとカスタム Kraus チャンネルを備えた単純な回路を使用するアプローチを示しています。

```
# 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)
```

回路の実行は、次の 2 つの例に示すように、以前と同じユーザーエクスペリエンスです。

例 1

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

または

例 2

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

その他の例については、「Braket 入門ノイズシミュレーターの例」を参照してください。

回路の検査

Amazon Braket の量子回路には、 という擬似時間の概念がありますMoments。各 は、 ごとに 1 つのゲートを経験qubitできますMoment。の目的は、回路とそのゲートに対処しやすくし、時間的構造を提供することMomentsです。

> **Note**
>
> モーメントは通常、QPU でゲートが実行されるリアルタイムに対応していません。

回路の深さは、その回路内のモーメントの総数によって与えられます。次の例circuit.depthに示すように、 メソッドを呼び出す回路深度を表示できます。

```
# 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
```

上記の回路の合計回路深度は 3 です (モーメント 0、1、および として表示されます2) 。各モーメントのゲート動作を確認することができます。

Moments は、キーと値のペアのディクショナリとして機能します。

- キーは でMomentsKey( )、擬似時間とqubit情報が含まれています。
- この値は、Instructions( ) のタイプで割り当てられます。

```
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)]))
```

また、Moments を介して回路にゲートを追加することもできます。

```
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|
```

結果タイプ

Amazon Braket は、 を使用して回路を測定すると、さまざまなタイプの結果を返すことができますResultType。回路は次のタイプの結果を返すことができます。

- AdjointGradient は、指定されたオブザーバブルの期待値の勾配 (ベクトル派生) を返します。このオブザーバビリティは、結合区別方法を使用して、指定されたパラメータに関して指定されたターゲットで動作します。このメソッドは、shots=0 の場合にのみ使用できます。
- Amplitude は、出力ウェーブ関数で指定された量子状態の振幅を返します。これは、 SV1およびローカルシミュレーターでのみ使用できます。
- Expectation は、特定のオブザーバビリティの期待値を返します。この値は、この章で後ほど紹介した Observable クラスで指定できます。オブザーバブルの測定qubitsに使用されるターゲットを指定する必要があります。また、指定されたターゲットの数は、オブザーバブルが動作するの数qubitsと等しくなければなりません。ターゲットを指定しない場合、オブザーバブルは 1 でのみ動作qubitし、すべての に並行qubitsして適用されます。
- Probability は、計算基準の状態を測定する確率を返します。ターゲットが指定されていない場合、Probability はすべての基底状態を測定する確率を返します。ターゲットが指定されている場合、指定されたベースベクトルのわずかな確率のみqubitsが返されます。

- `Reduced density matrix` は、指定されたターゲットのサブシステムの密度行列qubitsを のシステムから返しますqubits。この結果タイプのサイズを制限するため、Braket はターゲットの数qubitsを最大 8 に制限します。
- `StateVector` はフルステートベクトルを返します。ローカルシミュレーターで利用できます。
- `Sample` は、指定されたターゲットqubitセットとオブザーバブルの測定数を返します。ターゲットを指定しない場合、オブザーバブルは 1 でのみ動作qubitし、すべての に並行qubitsして適用されます。ターゲットを指定する場合、指定されたターゲットの数は、オブザーバブルが動作するの数qubitsと等しくなければなりません。
- `Variance` は、指定されたターゲットqubitセットの分散 (`mean([x-mean(x)]`^{2}`)`) を返し、要求された結果タイプとして観測可能を返します。ターゲットを指定しない場合、オブザーバブルは 1 でのみ動作qubitし、すべての に並行qubitsして適用されます。それ以外の場合、指定するターゲットの数は、オブザーバブルを適用できる の数qubitsと等しくなければなりません。

さまざまなデバイスでサポートされる結果タイプ:

	ローカルシム	SV1	DM1	TN1	Rigetti	IonQ	OQC
結合グラデーション	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
期待	Y	Y	Y	Y	Y	Y	Y
見込み	Y	Y	Y	N	Y*	Y	Y
低密度マトリックス	Y	N	Y	N	N	N	N
状態ベクトル	Y	N	N	N	N	N	N
サンプル	Y	Y	Y	Y	Y	Y	Y

分散	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> * は、最大 40 の確率結果タイプRigettiのみをサポートしますqubits。

次の例に示すように、サポートされている結果タイプを確認するには、デバイスのプロパティを調べます。

```
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
```

を呼び出すにはResultType、次の例に示すように、回路に追加します。

```
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**
>
> デバイスによっては、結果として測定値 (などRigetti) を提供し、結果として確率 (IonQ や など) を提供するデバイスもありますOQC。SDK は結果に測定プロパティを提供しますが、確率を返すデバイスについては、計算後に行われます。したがって、 IonQや によって提供されるようなデバイスはOQC、ショットごとの測定値が返されないため、確率によって決定された測定値を持ちます。このファイル に示すように、結果オブジェクトmeasurements_copied_from_deviceで を表示することで、結果がポストコンピューティングされているかどうかを確認できます。

オブザーバブル

Amazon Braket には、測定するオブザーバブルを指定するために使用できる Observable クラスが含まれています。

各 には、最大 1 つの一意の非 ID オブザーバブルを適用できますqubit。同じ に 2 つ以上の異なる非 ID オブザーバビリティを指定するとqubit、エラーが表示されます。この目的のために、テンソル製品の各要素は個々のオブザーバビリティとしてカウントされるため、同じ で動作する複数のテンソル製品を持つことは許可されます。ただしqubit、その で動作する要素qubitが同じである場合に限ります。

オブザーバブルをスケーリングし、オブザーバブル (スケーリングの有無にかかわらず) を追加することもできます。これにより、AdjointGradient結果タイプSumで使用できる が作成されます。

Observable クラスには、次のオブザーバブルが含まれます。

```
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)))
```

パラメータ

回路には、「1 回構築 - 複数回実行」方式で使用したり、勾配を計算するために使用できる空きパラメータが含まれている場合があります。フリーパラメータには文字列エンコードされた名前があり、値を指定したり、値に関して区別するかどうかを決定したりできます。

```
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]
```

区別するパラメータには、名前 (文字列) または直接参照を使用して指定します。AdjointGradient 結果タイプを使用した勾配の計算は、観測可能な の期待値に関して行われることに注意してください。

注： パラメータ化された回路に引数として渡して空きパラメータの値を修正した場合、結果タイプAdjointGradientとして とパラメータを指定して回路を実行すると エラーが発生します。これは、 との区別に使用しているパラメータが存在しないためです。次の例を参照してください。

```
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
```

QPUs とシミュレーターへの量子タスクの送信

Amazon Braket は、量子タスクを実行できる複数のデバイスへのアクセスを提供します。量子タスクは個別に送信することも、量子タスクのバッチ処理を設定することもできます。

QPU

量子タスクはいつでも QPUs に送信できますが、タスクは Amazon Braket コンソールのデバイスページに表示される特定の可用性ウィンドウ内で実行されます。量子タスクの結果は、次のセクションで紹介する量子タスク ID を使用して取得できます。

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

シミュレーター

- 密度行列シミュレーター、DM1: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`

- ステートベクトルシミュレーター、SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- テンソルネットワークシミュレーター、TN1: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- ローカルシミュレーター : `LocalSimulator()`

> **Note**
>
> QPUsおよびオンデマンドシミュレーターの CREATED状態の量子タスクをキャンセルできます。オンデマンドシミュレーターと QPUs では、ベストエフォートベースで QUEUED状態の量子タスクをキャンセルできます。QPU 量子タスクは、QPU QUEUED の可用性ウィンドウ中に正常にキャンセルされる可能性は低いことに注意してください。

このセクションの内容:

Amazon Braket の量子タスクの例

このセクションでは、デバイスの選択から結果の表示まで、量子タスクの例を実行する段階について説明します。Amazon Braket のベストプラクティスとして、まず などのシミュレーターで回路を実行することをお勧めしますSV1。

このセクションの内容:

- 結果のポーリング
- サンプル結果の表示

デバイスを指定する

まず、量子タスクのデバイスを選択して指定します。この例では、シミュレーター を選択する方法を示します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")
```

このデバイスのプロパティの一部は、次のように表示できます。

```
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)])
```

量子タスクの例を送信する

オンデマンドシミュレーターで実行する量子タスクの例を送信します。

```
# 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()
```

device.run() コマンドは API を使用して量子タスクを作成しますCreateQuantumTaskhttps://docs.aws.amazon.com/braket。初期化時間が短いと、量子タスクは、デバイス上で量子タスクを実行する容量が存在するまでキューに入れられます。この場合、デバイスは ですSV1。デバイスが計算を完了すると、AmazonBraket は呼び出しで指定された Amazon S3 の場所に結果を書き込みます。位置引数 s3_location はローカルシミュレーターを除くすべてのデバイスで必要です。

> **Note**
>
> Braket 量子タスクアクションのサイズは 3MB に制限されています。

パラメータ化されたタスクを送信する

Amazon Braket オンデマンドおよびローカルシミュレーターと QPUsは、タスク送信時の空きパラメータの値の指定もサポートしています。これを行うには、次の例に示すようにdevice.run()、の inputs引数を使用します。は文字列と浮動小数点のペアのディクショナリinputsである必要があります。ここで、キーはパラメータ名です。

パラメトリックコンパイルは、特定の QPUs。サポートされている QPU に量子タスクとしてパラメトリック回路を送信すると、Braket は回路を 1 回コンパイルし、結果をキャッシュします。同じ回路への後続のパラメータ更新の再コンパイルは行われないため、同じ回路を使用するタスクの実行時間が短縮されます。Braket は、回路をコンパイルするときに、ハードウェアプロバイダーから更新されたキャリブレーションデータを自動的に使用して、最高品質の結果を実現します。

> **Note**
>
> パラメトリックコンパイルは、 Rigetti Computingおよび からのすべての超電導ゲートベースの QPUs でサポートされています。ただし、 の脈動レベルプログラムを除きOxford Quantum Circuitsます。

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

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

# create a circuit with a result type
circ = Circuit().rx(0, alpha).ry(1, alpha).cnot(0,2).xx(0, 2, beta)
circ.variance(observable=Observable.Z(), target=0)
# add another result type
circ.probability(target=[0, 2])
# submit the quantum task to run
my_task = device.run(circ, inputs={'alpha': 0.1, 'beta':0.2})
```

shots を指定する

shots 引数は、必要な測定 の数を参照しますshots。などのシミュレーターは、2 つのシミュレーションモードSV1をサポートします。

- shots = 0 の場合、シミュレーターは正確なシミュレーションを実行し、すべての結果タイプの true 値を返します。(では使用できません) TN1。
- のゼロ以外の値の場合shots、シミュレータは出力ディストリビューションからサンプリングして、実際の QPU のshotノイズをエミュレートします。 QPUs QPU デバイスは shots > 0 のみを許可します。

量子タスクあたりの最大ショット数については、「Braket Quotas」を参照してください。

結果のポーリング

を実行すると`my_task.result()`、SDK は量子タスクの作成時に定義したパラメータを使用して結果のポーリングを開始します。

- poll_timeout_seconds は、オンデマンドシミュレーターや QPU デバイスで量子タスクを実行するときにタイムアウトする前に量子タスクをポーリングする秒数です。デフォルト値は 432,000 秒 (5 日) です。
- 注： Rigettiや などの QPU デバイスの場合はIonQ、数日かかることをお勧めします。ポーリングタイムアウトが短すぎると、ポーリング時間内に結果が返されないことがあります。例えば、QPU が利用できない場合、ローカルタイムアウトエラーが返されます。
- poll_interval_seconds は、量子タスクがポーリングされる頻度です。オンデマンドシミュレーターと QPU デバイスで量子タスクが実行されたときAPIに、Braket を呼び出してステータスを取得する頻度を指定します。デフォルト値は 1 秒です。

この非同期実行により、常に使用できるとは限らない QPU デバイスの操作が容易になります。例えば、通常のメンテナンス期間中はデバイスを使用できない場合があります。

返される結果には、量子タスクに関連付けられたメタデータの範囲が含まれます。測定結果は、次のコマンドで確認できます。

```
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}
```

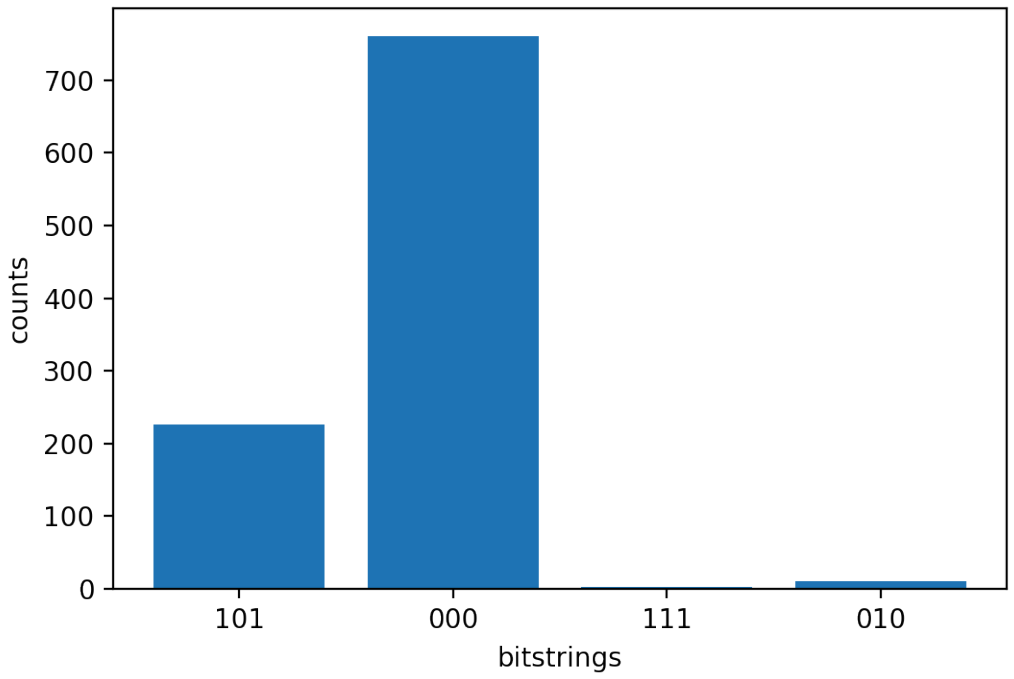
サンプル結果の表示

ResultType も指定したので、返される結果を表示できます。結果タイプは、回路に追加された順に表示されます。

```
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]
```



QPU への量子タスクの送信

Amazon Braket を使用すると、QPU デバイスで量子回路を実行できます。次の例は、量子タスクを Rigettiまたは IonQ デバイスに送信する方法を示しています。

Rigetti Aspen-M-3 デバイスを選択し、関連する接続グラフを確認します。

```
# 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']}}
```

前述のディクショナリconnectivityGraphには、現在のRigettiデバイスの接続に関する情報が含まれています。

IonQ Harmony デバイスを選択する

IonQ Harmony デバイスの場合、次の例に示すように、 connectivityGraphは空です。これは、デバイスがオールツーオール接続を提供しているためです。したがって、詳しい connectivityGraph は必要ありません。

```
# 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': {}}
```

次の例に示すように、デフォルトバケット以外の場所を指定した場合、結果を保存する S3 バケットshotsの場所 poll_timeout_seconds ()、 (デフォルト = 432000 = 5 日) 、poll_interval_seconds (デフォルト = 1s3_location) を調整するオプションがあります。

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

IonQ および Rigettiデバイスは、提供された回路をそれぞれのネイティブゲートセットに自動的にコンパイルし、抽象qubitインデックスをそれぞれの QPU qubitsの物理にマッピングします。

> **Note**
>
> QPU デバイスの容量は限られています。容量に達すると、待機時間が長くなることが予想されます。

Amazon Braket は特定の可用性ウィンドウ内で QPU 量子タスクを実行できますが、対応するすべてのデータとメタデータが適切な S3 バケットに確実に保存されるため、いつでも量子タスクを送信できます (24/7)。次のセクションに示すように、 AwsQuantumTaskと一意の量子タスク ID を使用して量子タスクを復元できます。

ローカルシミュレーターで量子タスクを実行する

量子タスクをローカルシミュレーターに直接送信して、ラピッドプロトタイピングとテストを行うことができます。このシミュレーターはローカル環境で実行されるため、Amazon S3 の場所を指定する必要はありません。結果はセッションで直接計算されます。ローカルシミュレーターで量子タスクを実行するには、 shotsパラメータのみを指定する必要があります。

> **Note**
>
> ローカルシミュレータqubitsーが処理できる実行速度と最大数は、AmazonBraket ノートブックインスタンスタイプ、またはローカルハードウェア仕様によって異なります。

次のコマンドはすべて同一で、ステートベクトル (ノイズフリー) ローカルシミュレーターをインスタンス化します。

```
# 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")
```

次に、以下を使用して量子タスクを実行します。

```
my_task = device.run(circ, shots=1000)
```

ローカル密度行列 (ノイズ) シミュレーターをインスタンス化するために、お客様はバックエンドを次のように変更します。

```
# import the LocalSimulator module
from braket.devices import LocalSimulator
device = LocalSimulator(backend="braket_dm")
```

ローカルシミュレーターでの特定の量子ビットの測定

ローカルステートベクトルシミュレーターとローカル密度行列シミュレーターは、回路の量子ビットのサブセットを測定できる回路の実行をサポートします。これは、部分測定 と呼ばれることがよくあります。

例えば、次のコードでは、2 量子ビット回路を作成し、ターゲット量子ビットを持つmeasure命令を回路の末尾に追加することによってのみ、最初の量子ビットを測定できます。

```
# Import the LocalSimulator module
from braket.devices import LocalSimulator

# Use the local simulator device
device = LocalSimulator()

# Define a bell circuit and only measure
circuit = Circuit().h(0).cnot(0, 1).measure(0)

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

# Get the results
result = task.result()

# Print the measurement counts for qubit 0
print(result.measurement_counts)
```

量子タスクバッチ処理

量子タスクバッチ処理は、ローカルシミュレーターを除くすべての Amazon Braket デバイスで使用できます。バッチ処理は、複数の量子タスクを並行して処理できるため、オンデマンドシミュレーター (TN1 または SV1) で実行する量子タスクに特に役立ちます。さまざまな量子タスクのセットアップに役立つように、Amazon Braket はサンプルノートブック を提供しています。

バッチ処理を使用すると、量子タスクを並行して起動できます。例えば、10 個の量子タスクを必要とする計算を行い、それらの量子タスクの回路が互いに独立している場合は、バッチ処理を使用することをお勧めします。そうすれば、ある量子タスクが完了するのを待ってから別のタスクが開始される必要はありません。

次の例は、量子タスクのバッチを実行する方法を示しています。

```
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
```

詳細については、「」の Amazon Braket の例 GitHubまたはバッチ処理に関するより具体的な情報を含む量子タスクバッチ処理」を参照してください。 https://github.com/aws/amazon-braket-sdk-python/blob/main/src/braket/aws/aws_quantum_task_batch.py

量子タスクのバッチ処理とコストについて

量子タスクのバッチ処理と請求コストについて、いくつかの注意点があります。

- デフォルトでは、量子タスクバッチ処理はすべてのタイムアウトを再試行するか、量子タスクを 3 回失敗させます。
- qubits の 34 など、長時間実行される量子タスクのバッチにはSV1、大きなコストが発生する可能性があります。量子タスクのバッチを開始する前に、`run_batch`割り当て値を慎重に再確認してください。TN1 で を使用することはお勧めしませんrun_batch。
- TN1 では、失敗したリハーサルフェーズタスクのコストが発生する可能性があります (詳細については、TN1 の説明を参照してください)。自動再試行によってコストが増加する可能性があるため、 を使用する場合は、バッチ処理の「max_retries」の数を 0 に設定することをお勧めします TN1 (「量子タスクバッチ処理」、186 行目を参照)。

量子タスクのバッチ処理と PennyLane

次の例に示すように、AmazonBraket PennyLane で を使用している場合は、Braket デバイスをインスタンス化`parallel = True`するときに Amazon を設定してバッチ処理を活用します。

```
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,)
```

を使用したバッチ処理の詳細については PennyLane、「量子回路の並列最適化」を参照してください。

タスクバッチ処理とパラメータ化された回路

パラメータ化された回路を含む量子タスクバッチを送信するときは、バッチ内のすべての量子タスクに使用される`inputs`ディクショナリ、または入力ディクショナリ`list`の を提供することができます。この場合、次の例に示すように、i-th ディクショナリは i-th タスクとペアになります。

```
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)
```

また、単一のパラメトリック回路の入力ディクショナリのリストを準備し、量子タスクバッチとして送信することもできます。リストに N 個の入力ディクショナリがある場合、バッチには N 個の量子タスクが含まれます。i-th 量子タスクは、i-th 入力ディクショナリで実行される回路に対応します。

```
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)
```

SNS 通知を設定する (オプション)

Amazon Simple Notification Service (SNS) を介して通知を設定して、AmazonBraket 量子タスクが完了したときにアラートを受け取ることができます。アクティブな通知は、大きな量子タスクを送信する場合や、デバイスの可用性ウィンドウ外で量子タスクを送信する場合など、長い待機時間が予想される場合に便利です。量子タスクが完了するまで待機しない場合は、SNS 通知を設定できます。

Amazon Braket ノートブックでは、セットアップ手順について説明します。詳細については、「」の Amazon Braket の例 GitHub、特に通知を設定するためのサンプルノートブックを参照してください。

コンパイルされた回路の検査

回路がハードウェアデバイスで実行されるときは、QPU でサポートされているネイティブゲートに回路を変換するなど、許容可能な形式でコンパイルする必要があります。実際のコンパイル済み出力を検査することは、デバッグの目的で非常に役立ちます。以下のコードを使用して、 Rigettiと OQC デバイスの両方でこの回路を表示できます。

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

> **Note**
>
> 現在、IonQデバイスのコンパイル済み回路を表示することはできません。

OpenQASM 3.0 で回路を実行する

Amazon Braket は、ゲートベースの量子デバイスとシミュレーターで OpenQASM 3.0 をサポートするようになりました。このユーザーガイドでは、Braket でサポートされている OpenQASM 3.0 のサブセットについて説明します。Braket のお客様は、 SDK を使用して Braket 回路を送信するか、Amazon Braket API と Amazon Braket Python SDK を使用してすべてのゲートベースのデバイスに直接 OpenQASM 3.0 文字列を提供するかを選択できるようになりました。 Amazon Braket

このガイドのトピックでは、以下の量子タスクを完了する方法のさまざまな例について説明します。

- 異なる Braket デバイスで OpenQASM 量子タスクを作成して送信する
- サポートされているオペレーションと結果タイプにアクセスする
- OpenQASM でノイズをシミュレートする
- OpenQASM で逐語的なコンパイルを使用する
- OpenQASM の問題のトラブルシューティング

このガイドでは、Braket の OpenQASM 3.0 で実装できる特定のハードウェア固有の機能の概要と、その他のリソースへのリンクも提供します。

このセクションの内容:

OpenQASM 3.0 とは

Open Quantum Assembly Language (OpenQASM) は、量子命令の中間表現です。OpenQASM はオープンソースフレームワークであり、ゲートベースのデバイス用の量子プログラムの仕様に広く使用されています。OpenQASM を使用すると、ユーザーは量子計算の構成要素を形成する量子ゲートと測定操作をプログラムできます。OpenQASM の以前のバージョン (2.0) は、単純なプログラムを記述するために多数の量子プログラミングライブラリで使用されました。

OpenQASM の新しいバージョン (3.0) では、以前のバージョンを拡張して、エンドユーザーインターフェイスとハードウェア記述言語間のギャップを埋めるために、脈動レベルの制御、ゲートタイミング、クラシック制御フローなどの機能を追加しました。現在のバージョン 3.0 の詳細と仕様は、GitHub OpenQASM 3.x Live Specification で入手できます。OpenQASM の将来の開発はOpenQASM 3.0 技術諜報委員会 によって管理され、そのうちの AWS は IBM、Microsoft、および""sbruck 大学のメンバーです。

OpenQASM 3.0 を使用するタイミング

OpenQASM は、アーキテクチャ固有ではない低レベル制御を通じて量子プログラムを指定する表現的フレームワークを提供するため、複数のゲートベースのデバイスにわたる表現として最適です。OpenQASM の Braket サポートは、ゲートベースの量子アルゴリズムを開発するための一貫したアプローチとして採用をさらに進め、ユーザーが複数のフレームワークでライブラリを学習して維持する必要性を減らします。

OpenQASM 3.0 に既存のプログラムのライブラリがある場合は、これらの回路を完全に書き換えるのではなく、Braket での使用に適応させることができます。研究者やデベロッパーは、OpenQASM でのアルゴリズム開発をサポートしている利用可能なサードパーティーライブラリの数が増えていることからも恩恵を受けるはずです。

OpenQASM 3.0 の仕組み

Braket の OpenQASM 3.0 のサポートにより、現在の中間表現と同等の機能が提供されます。つまり、Braket を使用してハードウェアデバイスやオンデマンドシミュレーターで現在できることはすべて、Braket を使用して OpenQASM で実行できるということですAPI。OpenQASM 3.0 プログラムを実行するには、OpenQASM 文字列をすべてのゲートベースのデバイスに直接供給します。これは、回路が Braket 上のデバイスに現在供給されている方法と似ています。Braket ユーザーは、OpenQASM 3.0 をサポートするサードパーティーライブラリを統合することもできます。このガイドの残りの部分では、Braket で使用する OpenQASM 表現を開発する方法について説明します。

前提条件

Amazon Braket で OpenQASM 3.0 を使用するには、Amazon Braket Python スキーマのバージョン v1.8.0 および Amazon Braket Python SDK のバージョン v1.17.0 以降が必要です。

Amazon Braket を初めて使用する場合は、Braket Amazon を有効にする必要があります。手順については、Amazon Braket を有効にする」を参照してください。

Braket はどのような OpenQASM 機能をサポートしていますか？

次のセクションでは、Braket でサポートされている OpenQASM 3.0 データ型、ステートメント、およびプラグマ命令を一覧表示します。

このセクションの内容:

- サポートされている OpenQASM データ型

- サポートされている OpenQASM ステートメント
- Braket OpenQASM プラグマ
- Local Simulator での OpenQASM の高度な機能のサポート
- でサポートされているオペレーションと文法 OpenPulse

サポートされている OpenQASM データ型

Amazon Braket では、次の OpenQASM データ型がサポートされています。

- 負以外の整数は (仮想および物理) 量子ビットインデックスに使用されます。
  - `cnot q[0], q[1];`
  - `h $0;`
- 浮動小数点数または定数は、ゲートローテーション角度に使用できます。
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi は OpenQASM に組み込まれた定数であり、パラメータ名として使用することはできません。

- 複雑な数値の配列 (架空の部分については OpenQASM `im`表記) は、一般的なヘルミット観測物を定義するための結果型プラグマと単一プラグマで使用できます。
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

サポートされている OpenQASM ステートメント

Braket では、次の OpenQASM Amazon ステートメントがサポートされています。

- `Header: OPENQASM 3;`
- クラシックビット宣言：
  - `bit b1;` (同等、`creg b1;`)

  - `bit[10] b2;` (同等、`creg b2[10];`)
- 量子ビット宣言 :
  - `qubit b1;` (同等、`qreg b1;`)
  - `qubit[10] b2;` (同等、`qreg b2[10];`)
- 配列内のインデックス作成: `q[0]`
- 入力: `input float alpha;`
- 物理 の仕様qubits: `$0`
- デバイスでサポートされているゲートとオペレーション :
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> デバイスのサポートされているゲートは、OpenQASM アクションのデバイスプロパティにあります。これらのゲートを使用するにはゲート定義は必要ありません。

- 逐語的なボックスステートメント。現在、ボックス期間表記はサポートされていません。逐語的なボックスにはネイティブゲートと物理ゲートqubitsが必要です。

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

- qubit またはレジスタqubits全体に対する測定と測定の割り当て。
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`

> **Note**
>
> pi は OpenQASM に組み込まれた定数であり、パラメータ名として使用することはできません。

Braket OpenQASM プラグマ

以下の OpenQASM プラグマ手順は Braket Amazon でサポートされています。

- ノイズプラグマ
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- 逐語的なプラグマ
  - `#pragma braket verbatim`
- 結果タイプのプラグマ
  - 基本不変の結果タイプ :
    - ステートベクトル: `#pragma braket result state_vector`
    - 密度マトリックス: `#pragma braket result density_matrix`
  - グラデーション計算プラグマ :
    - 結合勾配: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Z ベースの結果タイプ :
    - 出力: `#pragma braket result amplitude "01"`
    - 確率: `#pragma braket result probability q[0], q[1]`
  - ローテーションされた基本結果タイプ
    - 期待値: `#pragma braket result expectation x(q[0]) @ y([q1])`
    - 分散: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
    - サンプル: `#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0 は OpenQASM 2.0 と下位互換性があるため、2.0 を使用して記述されたプログラムは Braket で実行できます。ただし、Braket がサポートする OpenQASM 3.0 の機能には、 `qreg` と `creg`、 など、構文のわずかな違い`qubit`があります`bit`。測定構文にも違いがあり、これらは正しい構文でサポートされる必要があります。

Local Simulator での OpenQASM の高度な機能のサポート

は、Braket の QPU またはオンデマンドシミュレーターの一部として提供されていない高度な OpenQASM 機能`LocalSimulator`をサポートしています。以下の機能のリストは、 でのみサポートされています`LocalSimulator`。

- ゲート修飾子
- OpenQASM 組み込みゲート
- クラシック変数
- クラシックオペレーション
- カスタムゲート
- クラシックコントロール
- QASM ファイル
- サブルーチン

各高度な機能の例については、このサンプルノートブック を参照してください。OpenQASM の完全な仕様については、OpenQASM ウェブサイト を参照してください。

でサポートされているオペレーションと文法 OpenPulse

サポートされている OpenPulse データ型

Cal ブロック：

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

デフォルトブロック：

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

フレーム：

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

ウェーブフォーム：

```
// 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};
```

カスタムゲートキャリブレーションの例：

```
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;
```

任意的脈の例：

```
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);
}
```

OpenQASM 3.0 量子タスクの例を作成して送信する

Amazon Braket Python SDK、Boto3、または を使用して、OpenQASM 3.0 量子タスクを Amazon Braket デバイス AWS CLI に送信できます。

このセクションの内容:

- OpenQASM 3.0 プログラムの例
- Python SDK を使用して OpenQASM 3.0 量子タスクを作成する

- Boto3 を使用して OpenQASM 3.0 量子タスクを作成する
- を使用して OpenQASM 3.0 タスク AWS CLI を作成する

OpenQASM 3.0 プログラムの例

OpenQASM 3.0 タスクを作成するには、次の例に示すように、GHZ 状態を準備するシンプルな OpenQASM 3.0 プログラム (ghz.qasm) から開始できます。

```
// 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;
```

Python SDK を使用して OpenQASM 3.0 量子タスクを作成する

Amazon Braket Python SDK を使用して、次のコードでこのプログラムを Amazon Braket デバイスに送信できます。

```
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,
)
```

Boto3 を使用して OpenQASM 3.0 量子タスクを作成する

次の例に示すように、AWS Python SDK for Braket (Boto3) を使用して OpenQASM 3.0 文字列を使用して量子タスクを作成することもできます。次のコードスニペットは、上記のように GHZ 状態を準備する ghz.qasm を参照しています。

```
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,
)
```

を使用して OpenQASM 3.0 タスク AWS CLI を作成する

次の例に示すように、 AWS Command Line Interface (CLI) を使用して OpenQASM 3.0 プログラムを送信することもできます。

```
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)
    }'
```

異なる Braket デバイスでの OpenQASM のサポート

OpenQASM 3.0 をサポートしているデバイスの場合、 `action`フィールドは、 Rigettiおよび IonQ デバイスの次の例に示すように、 `GetDevice`レスポンスによる新しいアクションをサポートします。

```
//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"
            ],
            ….
        }
    }
}
```

脈動制御をサポートするデバイスの場合、 pulseフィールドがGetDeviceレスポンスに表示されます。次の例は、 Rigettiおよび OQC デバイスのpulseこのフィールドを示しています。

```
// 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
    }
  }
}
```

前述のフィールドでは、以下について詳しく説明しています。

ポート：

特定のポートの関連プロパティに加えて、QPU で宣言された事前に作成された外部 (extern) デバイスポートについて説明します。この構造にリストされているすべてのポートは、ユーザーが送信したOpenQASM 3.0プログラム内で有効な識別子として事前に宣言されています。ポートの追加プロパティは次のとおりです。

- ポート ID (portId)
  - OpenQASM 3.0 で識別子として宣言されたポート名。
- 方向 (方向)

    - ポートの方向。ドライブポートは、 の脈 (方向「tx」) を送信し、測定ポートは の脈 (方向「rx」) を受信します。
- ポートタイプ (portType)
    - このポートが担当するアクションのタイプ (ドライブ、キャプチャ、ff - fast-flux など) 。
- dt (dt)
    - 指定されたポートの 1 つのサンプル時間ステップを表す秒単位の時間。
- 量子ビットマッピング (qubitMappings)
    - 指定されたポートに関連付けられた量子ビット。
- 中心周波数 (centerFrequencies)
    - ポート上のすべての事前宣言済みフレームまたはユーザー定義フレームに関連する中心周波数のリスト。詳細については、「フレーム」を参照してください。
- QHP 固有のプロパティ (qhp SpecificProperties)
    - QHP に固有のポートに関する既存のプロパティを詳述するオプションのマップ。

フレーム：

QPU で宣言された事前に作成された外部フレームと、そのフレームに関連するプロパティについて説明します。この構造にリストされているすべてのフレームは、ユーザーが送信したOpenQASM 3.0プログラム内で有効な識別子として事前に宣言されています。フレームの追加プロパティは次のとおりです。

- フレーム ID (frameId)
    - OpenQASM 3.0 で識別子として宣言されたフレーム名。
- ポート ID (portId)
    - フレームに関連付けられたハードウェアポート。
- 頻度 (頻度)
    - フレームのデフォルトの初期頻度。
- 中心周波数 (centerFrequency)
    - フレームの周波数帯域幅の中心。通常、フレームは中心周波数の周りの特定の帯域幅にのみ調整できます。その結果、頻度調整は中心周波数の所定の差分内にとどまる必要があります。帯域幅の値は、検証パラメータで確認できます。
- フェーズ (フェーズ)
    - フレームのデフォルトの初期フェーズ。

- 関連ゲート (associatedGate)
  - 指定されたフレームに関連付けられたゲート。
- 量子ビットマッピング (qubitMappings)
  - 指定されたフレームに関連付けられた量子ビット。
- QHP 固有のプロパティ (qhp SpecificProperties)
  - QHP に固有のフレームに関する既存のプロパティの詳細を示すオプションのマップ。

SupportsDynamicフレーム：

OpenPulse `newframe` 関数を介してフレームを `cal`または `defcal`ブロックで宣言できるかどうかを記述します。これが false の場合、フレーム構造にリストされているフレームのみをプログラム内で使用できます。

SupportedFunctions:

特定のOpenPulse関数に関連する引数、引数タイプ、および戻り値タイプに加えて、デバイスでサポートされている関数について説明します。OpenPulse 関数の使用例については、仕様 OpenPulse を参照してください。現時点では、Braket は以下をサポートしています。

- shift_phase
  - フレームのフェーズを指定された値だけシフトします。
- set_phase
  - フレームのフェーズを指定された値に設定します。
- shift_frequency
  - フレームの頻度を指定された値でシフトします。
- set_frequency
  - フレームの頻度を指定された値に設定します。
- play
  - 波形をスケジュールする
- capture_v0
  - キャプチャフレームの値をビットレジスタに返す

SupportedQhpTemplateWaveforms:

デバイスで使用可能な事前構築済みの波形関数、および関連する引数とタイプについて説明します。デフォルトでは、Braket Pulse はすべてのデバイスで事前構築済みの波形ルーチンを提供します。

定数

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

τ は波形の長さで、 iqは複雑な数値です。

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

ガウス語

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

τ は波形の長さ、 σはガウス語の幅、 Aは振幅です。ZaE を に設定するとTrue、ガウシアンはオフセットされ、波形の開始時と終了時にゼロに等しくなるように再スケーリングされ、A最大値に達します。

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

ガウス語をドラッグ

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

τ は波形の長さ、 σはガウシアンの幅、 βは自由パラメータ、 Aは振幅です。ZaE を に設定するとTrue、Adiabatic Gate (DRAG) Gaussian による微分除去がオフセットされ、波形の開始と終了時にゼロに等しくなり、実際の部分がA最大に達するように再スケーリングされます。DRAG 波形の詳細については、ホワイトペーパー「Unakly Nonlinear Qubits」の「Simple Pulses for Elimination of Packagingage」を参照してください。

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

SupportsLocalPulseElements:

ポート、フレーム、波形などの脈動要素をdefcalブロックでローカルに定義できるかどうかを記述します。値が の場合false、要素は calブロックで定義する必要があります。

SupportsNonNativeGatesWithPulses:

非ネイティブゲートを脈動プログラムと組み合わせて使用できるかどうかについて説明します。例えば、最初に使用済み量子ビットのゲートスルーを定義しないと、プログラム内のHゲートのような非ネイティブゲートdefcalを使用することはできません。ネイティブゲートnativeGateSetキーのリストは、デバイスの機能にあります。

ValidationParameters:

以下を含む、脈要素の検証境界について説明します。

- 波形の最大スケール/最大振幅値 (任意および事前構築)
- 指定された中心周波数からの最大周波数帯域幅を Hz 単位で表示
- 秒単位の最小脈長/持続時間
- 秒単位の最大脈長/時間

OpenQASM でサポートされているオペレーション、結果、結果タイプ

各デバイスがサポートする OpenQASM 3.0 の機能を確認するには、デバイス機能出力の actionフィールドの braket.ir.openqasm.program キーを参照してください。例えば、Braket State Vector Simulator でサポートされているオペレーションと結果タイプを次に示します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
        }
      ]
    }
  },
...
```

OpenQASM 3.0 でノイズをシミュレートする

OpenQASM3 でノイズをシミュレートするには、プラグマ命令を使用してノイズ演算子を追加します。例えば、前述の GHZ プログラムのノイズの多いバージョンをシミュレートするには、次の OpenQASM プログラムを送信できます。

```
// 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;
```

サポートされているすべてのプラグマノイズ演算子の仕様を次のリストに示します。

```
#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
```

クラウス演算子

Kraus 演算子を生成するには、行列のリストを繰り返し処理し、行列の各要素を複雑な式として出力します。

Kraus 演算子を使用する場合は、次の点に注意してください。

- の数qubitsは 2 を超えることはできません。スキーマの現在の定義によって、この制限が設定されます。
- 引数リストの長さは 8 の倍数である必要があります。つまり、2x2 行列のみで構成される必要があります。
- 合計長は 2^{2*num_qubits} 行列を超えないようにします。つまり、1 には 4 つの行列qubit、2 には 16 の行列ですqubits。
- 提供されるすべてのマトリックスは、完全陽性トレース保存 (CPTP) です。
- Kraus 演算子とトランスポジット結合の積は、ID 行列に合計する必要があります。

Qubit OpenQASM 3.0 を使用した再ワイヤリング

Amazon Braket は、Rigettiデバイスの OpenQASM 内の物理qubit表記をサポートしています (詳細については、このページを参照してください)。ナイーブ再ワイヤリング戦略 qubitsで物理 を使用する場合は、 qubitsが選択したデバイスに接続されていることを確認します。または、代わりにqubitレジスタを使用する場合、PARTIAL 再ワイヤリング戦略はRigettiデバイスでデフォルトで有効になります。

```
// ghz.qasm
```

```
// Prepare a GHZ state
OPENQASM 3;

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

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

OpenQASM 3.0 を使用した逐語的なコンパイル

Rigetti、、OQCおよび から量子コンピュータで量子回路を実行する場合IonQ、コンパイラは回路を定義されたとおりに、変更を加えることなく実行するように指示できます。この機能は逐語的なコンパイル と呼ばれます。Rigetti デバイスを使用すると、回路全体または特定の部分のみなど、保持されるものを正確に指定できます。回路の特定の部分のみを保持するには、保存されたリージョン内でネイティブゲートを使用する必要があります。現在、 IonQと は回路全体の逐語的なコンパイルOQCのみをサポートしているため、回路内のすべての命令を逐語的なボックスで囲む必要があります。

OpenQASM を使用すると、ハードウェアの低レベルコンパイルルーチンによって変更されず、最適化されていないコードボックスの周りに逐語的なプラグマを指定できます。次のコード例は、 の使用方法を示しています`#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;
```

逐語的なコンパイルの詳細については、逐語的なコンパイルのサンプルノートブックを参照してください。

Braket コンソール

OpenQASM 3.0 タスクは使用でき、AmazonBraket コンソール内で管理できます。コンソールでは、OpenQASM 3.0 で量子タスクを送信した経験は、既存の量子タスクを送信した経験と同じです。

その他のリソース

OpenQASM は、すべての Amazon Braket リージョンで使用できます。

Braket での OpenQASM の使用を開始するためのノートブックの例については、Amazon「Braket チュートリアル GitHub」を参照してください。

OpenQASM 3.0 を使用した勾配の計算

Amazon Braket は、結合差別化方法を使用して、オンデマンドシミュレーターとローカルシミュレーターの両方で `shots=0` (正確な) モードで勾配の計算をサポートします。次の例に示すように、適切なプラグマを指定して、計算する勾配を指定できます。

```
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
```

すべてのパラメータを個別に一覧表示する代わりに、 プラグマ`all`で を指定することもできます。これにより、リストされているすべての`input`パラメータに関する勾配が計算されます。これは、パラメータの数が非常に多い場合に便利です。この場合、プラグマは次の例のようになります。

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

個々の演算子、テンソル製品、ヘルミット観測物、 など、すべての観測可能なタイプがサポートされていますSum。勾配の計算に使用する演算子はexpectation()、カプセル化器でラップし、各用語が作用する量子ビットを指定する必要があります。

OpenQASM 3.0 を使用した特定の量子ビットの測定

ローカルステートベクトルシミュレーターとローカル密度行列シミュレーターは、回路の量子ビットのサブセットを測定できるOpenQASMプログラムの送信をサポートします。これは多くの場合、部分測定と呼ばれます。例えば、次のコードでは、2 量子ビット回路を作成し、最初の量子ビットのみを測定できます。

```
partial_measure_qasm = """
OPENQASM 3.0;
bit[1] b;
qubit[2] q;
h q[0];
cnot q[0], q[1];
b[0] = measure q[0];
"""
```

量子ビットは 2 つq[0]ありますq[1]が、ここで測定するのは量子ビット 0 のみですb[0] = measure q[0]。次に、ローカル状態ベクトルシミュレーターで以下を実行します。

```
from braket.devices import LocalSimulator

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

デバイスが部分測定をサポートしているかどうかは、アクションプロパティの requiresAllQubitsMeasurementフィールドを調べることで確認できます。 の場合False、部分測定がサポートされます。

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

ここで、 requiresAllQubitsMeasurementは です。これはFalse、すべての量子ビットを測定する必要がないことを示します。

QuEraの Aquila を使用してアナログプログラムを送信する

このページでは、 のAquilaマシンの機能に関する包括的なドキュメントを提供しますQuEra。ここで説明する詳細は次のとおりです。1) でシミュレートされたパラメータ化されたハミルトニアンAquila、2) AHS プログラムパラメータ、3) AHS 結果コンテンツ、4) Aquila機能パラメータ。Ctrl+Fテキスト検索を使用して、質問に関連するパラメータを検索することをお勧めします。

ハミルトニア語

のAquilaマシンは、次の (時間依存) ハミルトニアンをネイティブにQuEraシミュレートします。

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

> **Note**
>
> ローカルチューニングへのアクセスは実験的な機能であり、Braket Direct を通じてリクエストによって利用できます。

この場合、次のようになります。

- $H_{\text{drive},k}(t) = (\frac{1}{2}\Omega(t)e^{i\phi(t)}S_{-,k} + \frac{1}{2}\Omega(t)e^{-i\phi(t)}S_{+,k}) + (-\Delta_{\text{global}}(t)n_k)$ 、
  - 「Ω(t)」は、時間依存的でグローバルな運転振幅 (Rabi 周波数) を (rad / s) 単位で表したものです。
  - 「ϕ(t)」は時間依存のグローバルフェーズで、ラジアンで測定されます。
  - $S_{-,k}$ と $S_{+,k}$は、原子 k のスピン下降および上昇演算子です (ベース =|↓#|g#, |↑#=|r#, S_{-}=|g##r|, $S_{+}=(S_{-})^{\dagger}$=|r##g|)。
  - $\Delta_{\text{global}}(t)$ は時間依存的でグローバルな調整です
  - n_k は、原子 k の Rydberg 状態の射影演算子です (例: n=|r##r|)。
- $H_{\text{local detuning},k}(t) = -\Delta_{\text{local}}(t)h_k n_k$
  - Δ_{local}「Ω(t)」は、ローカル周波数シフトの時間依存係数で、 (rad / s) の単位です。
  - h_k はサイト依存係数で、ディメンションレスの数値は 0.0 ~ 1.0 です。
- $V_{\text{vdw},k,l} = C_6/(d_{k,l})^6 n_k n_l$、
  - C_6は、(rad / s) * (m)^6 の単位の van der Waals 係数です。
  - $d_{k,l}$ は、原子 k と l の間のユークリッド距離で、メートル単位で測定されます。

ユーザーは、Braket AHS プログラムスキーマを介して以下のパラメータを制御できます。

- 2-d 原子配置 (各原子 k の x 座標$_k$と y $_k$座標、um 単位) 。これは、k、l=1、2、...N で原子距離 $d_{k,l}$ のペアを制御します。
- 「(t)」、時間依存のグローバル Rabi 周波数、 (rad / s) 単位
- 「(t)」、時間依存のグローバルフェーズ、 (rad) 単位
- $_{global}$「」(t)、時間依存、グローバル調整、 (rad / s) 単位
- $_{local}$「」(t)、ローカルチューニングの大きさの時間依存 (グローバル) 係数、(rad / s) の単位
- h $_k$: ローカル調整の大きさの (静的) サイト依存係数、0.0 ~ 1.0 のディメンションレス数

> **Note**
>
> ユーザーは、関係するレベル (S 、S $_-$、n 演算子が固定されている) $_+$や Rydberg-Rydberg 相互作用係数の強度 (C) を制御することはできません$_6$。

Braket AHS プログラムスキーマ

braket.ir.ahs.program_v1.Program オブジェクト (例)

```
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 (例)

```
{
```

```
    "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]
                }
            }
        ]
    }
}
```

メインフィールド

プログラムフィールド	type	説明
setup.ahs_register.sites	List[List[10 進数]]	ピンセットが原子をトラップする 2 次元座標のリスト
setup.ahs_register.filling	List[int]	トラップサイトを占有するアトムを 1、空のサイトを 0 でマークします
hamiltonian.drivingFields []amplitude.time_series.times	List[10 進数〕	走行振幅の時間ポイント、Omega(t)
hamiltonian.drivingFields []amplitude.time_series.values	List[10 進数〕	走行振幅の値、Omega(t)
hamiltonian.drivingFields []amplitude.pattern	str	走行振幅の空間パターン、Omega(t)。「均一」である必要があります
hamiltonian.drivingFields []phase.time_series.times	List[10 進数〕	走行フェーズの時点、phi(t)
hamiltonian.drivingFields []phase.time_series.values	List[10 進数〕	走行フェーズの値、phi(t)

プログラムフィールド	type	説明
hamiltonian.drivingFields [].phase.pattern	str	走行フェーズの空間パターン、phi(t)。「均一」である必要があります
hamiltonian.drivingFields [].detuning.time_series.times	List[10 進数〕	運転調整の時点、Delta_global(t)
hamiltonian.drivingFields [].detuning.time_series.values	List[10 進数〕	運転調整の値、Delta_global(t)
hamiltonian.drivingFields [].detuning.pattern	str	運転チューニングの空間パターン、Delta_global(t)。「均一」である必要があります
hamiltonian.localDetuning [].magnitude.time_series.times	List[10 進数〕	ローカル調整マグニチュードの時間依存係数の時間ポイント、Delta_local(t)
hamiltonian.localDetuning [].magnitude.time_series.values	List[10 進数〕	Delta_local(t) のローカル調整マグニチュードの時間依存係数の値
hamiltonian.localDetuning [].magnitude.pattern	List[10 進数〕	ローカル調整の大きさのサイト依存係数 h_k (値は setup.ahs_register.sites のサイトに対応します)

メタデータフィールド

プログラムフィールド	type	説明
braket.SchemaHeadername	str	スキーマの名前。braket.ir.ahs.program」である必要があります
braket SchemaHeader.version	str	スキーマのバージョン

Braket AHS タスク結果スキーマ

braket.tasks.analog_hamiltonian_simulation_quantum_task_result .AnalogHamiltonianSimulationQuantumTas
(例)

```
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 (例)

```
{
    "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
        }
    }
}
```

メインフィールド

タスク結果フィールド	type	説明
measurements[].shotResult.preSequence	List[int]	各ショットのシーケンス前測定ビット (アトミックサイトごとに 1 つ): サイトが空の場合は 0、サイトがいっぱいの場合は 1。量子進化を実行する一連の脈の前に測定されます。
measurements[].shotResult.postSequence	List[int]	各ショットのシーケンス後測定ビット: 原子が Rydberg 状態またはサイトが空の場合は 0、原子が地面状態の場合は 1。量子進化を実行する一連の脈の最後に測定されます。

メタデータフィールド

タスク結果フィールド	type	説明
braket.SchemaHeadername	str	スキーマの名前。'braket.task_result.analog_hamiltonia

タスク結果フィールド	type	説明
		n_simulation_task_result'
braket SchemaHeader.version	str	スキーマのバージョン
taskMetadata.braketSchemaHeadername	str	スキーマの名前。「braket.task_result.task_metadata」である必要があります。
taskMetadata.braketSchemaHeaderversion	str	スキーマのバージョン
taskMetadata.id	str	量子タスクの ID。 AWS 量子タスクの場合、これは量子タスク ARN です。
taskMetadata.shots	整数	量子タスクのショット数
taskMetadata.shots.deviceId	str	量子タスクが実行されたデバイスの ID。 AWS デバイスの場合、これはデバイス ARN です。

タスク結果フィールド	type	説明
taskMetadata.shots.createdAt	str	作成のタイムスタンプ。形式は ISO-8601/RFC3339 文字列形式 YYYY-MM-DDTHH:mm:ss.sssZである必要があります。デフォルトは None です。
taskMetadata.shots.endedAt	str	量子タスクが終了した時刻のタイムスタンプ。形式は ISO-8601/RFC3339 文字列形式 YYYY-MM-DDTHH:mm:ss.sssZである必要があります。デフォルトは None です。

タスク結果フィールド	type	説明
taskMetadata.shots.status	str	量子タスクのステータス (CREATED、QUEUED、RUNNING、COMPLETED、FAILED)。デフォルトは None です。
taskMetadata.shots.failureReason	str	量子タスクの失敗の理由。デフォルトは None です。
additionalMetadata.action	braket.ir.ahs.program_v1.Program	(Braket AHS プログラムスキーマセクションを参照)
additionalMetadata.action.braketSchemaHeaderqueraMetadata.name	str	スキーマの名前。'braket.task_result.quera_metadata' である必要があります
additionalMetadata.action.braketSchemaHeaderqueraMetadata.version	str	スキーマのバージョン

タスク結果フィールド	type	説明
additionalMetadata.action.numSuccessfulShots	整数	完全に成功したショットの数。リクエストされたショットの数と同じである必要があります
measurements[].shotMetadata.shotStatus	整数	ショットのステータス (成功、部分成功、失敗) 。「成功」である必要があります

QuEra デバイスプロパティスキーマ

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

```
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 (例)

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

サービスプロパティフィールド

サービスプロパティフィールド	type	説明
service.executionWindows [].executionDay	ExecutionDay	実行ウィンドウの日数。「毎日」、「平日」、「週末」、「月曜日」、「火曜日」、「水曜日」、「木曜日」、「金曜日」、「土曜日」、または「日曜日」である必要があります
service.executionWindows [].windowStartHour	datetime.time	実行ウィンドウが開始される時刻の UTC 24 時間形式
service.executionWindows [].windowEndHour	datetime.time	実行ウィンドウが終了する時刻の UTC 24 時間形式
service.qpu_capabilities.service.shotsRange	Tuple[int, int]	デバイスの最小ショット数と最大ショット数
service.qpu_capabilities.service.deviceCost.price	フロート	米ドル単位のデバイスの価格
service.qpu_capabilities.service.deviceCost.unit	str	料金の請求単位。例：「分」、「時間」、「ショット」、「タスク」

メタデータフィールド

メタデータフィールド	type	説明
action[].version	str	AHS プログラムスキーマのバージョン
action[].actionType	ActionType	AHS プログラムスキーマ名。braket.ir.ahs.program」である必要があります
service.braket.SchemaHeadername	str	スキーマの名前。'braket.device_schema.device_service_properties' である必要があります
service.braket.SchemaHeaderversion	str	スキーマのバージョン
service.deviceDocumentation.imageUrl	str	デバイスのイメージの URL
service.deviceDocumentation.summary	str	デバイスの簡単な説明
service.deviceDocumentation.externalDocumentationUrl	str	外部ドキュメント URL
service.deviceLocation	str	デバイスの地理的位置
service.updatedAt	datetime	デバイスプロパティが最後に更新された時刻

Boto3 を使用する

Boto3 は AWS SDK for Python です。Boto3 を使用すると、Python デベロッパーは Amazon Braket などの を作成、設定 AWS のサービス、管理できます。Boto3 は、オブジェクト指向の と APIAmazonBraket への低レベルアクセスを提供します。

「Boto3 クイックスタートガイド」の指示に従って、Boto3 をインストールして設定する方法を確認してください。

Boto3 は、AmazonBraket Python SDK と連携するコア機能を提供し、量子タスクの設定と実行を支援します。Python のお客様は常に Boto3 をインストールする必要があります。これがコア実装だからです。追加のヘルパーメソッドを使用する場合は、Braket SDK Amazon もインストールする必要があります。

例えば、 を呼び出すと`CreateQuantumTask`、AmazonBraket SDK は Boto3 にリクエストを送信し、Braket SDK は を呼び出します AWS API。

このセクションの内容:

- Amazon Braket Boto3 クライアントを有効にする
- Boto3 と Amazon Braket SDK の AWS CLI プロファイルを設定する

Amazon Braket Boto3 クライアントを有効にする

Amazon Braket で Boto3 を使用するには、Boto3 をインポートし、Braket Amazon への接続に使用するクライアントを定義する必要がありますAPI。次の例では、Boto3 クライアントの名前は ですbraket。

> **Note**
>
> の古いバージョンとの下位互換性のために BraketSchemas、OpenQASM 情報はGetDeviceAPI呼び出しから省略されます。この情報を取得するには、ユーザーエージェントは最新バージョンの BraketSchemas (1.8.0 以降) を提示する必要があります。Braket SDK はこれを自動的に報告します。Braket SDK の使用時に OpenQASM の結果がGetDeviceレスポンスに表示されない場合は、AWS_EXECUTION_ENV環境変数を設定してユーザーエージェントを設定する必要がある場合があります。、Boto3、Go、Java AWS CLI、 JavaScript/SDK に対してこれを行う方法については、GetDevice 「 が

OpenQASM の結果エラーを返さない」トピックに記載されているコード例を参照してください。TypeScript SDKs

```
import boto3
import botocore

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

braket クライアントが確立されたら、AmazonBraket サービスからリクエストを行い、レスポンスを処理できます。リクエストとレスポンスのデータの詳細については、API リファレンスをご覧ください。

次の例は、デバイスと量子タスクの操作方法を示しています。

- デバイスを検索します。
- デバイスを取得する
- 量子タスクを作成する
- 量子タスクを取得する
- 量子タスクの検索
- 量子タスクのキャンセル

デバイスを検索します。

- `search_devices(**kwargs)`

指定されたフィルターを使用してデバイスを検索します。

```
# 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'])
```

デバイスを取得する

- `get_device(deviceArn)`

Amazon Braket で使用可能なデバイスを取得します。

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

量子タスクを作成する

- `create_quantum_task(**kwargs)`

量子タスクを作成します。

```
# 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")
```

量子タスクを取得する

- get_quantum_task(quantumTaskArn)

指定された量子タスクを取得します。

```
# 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'])
```

量子タスクの検索

- search_quantum_tasks(**kwargs)

指定されたフィルター値に一致する量子タスクを検索します。

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

量子タスクのキャンセル

- `cancel_quantum_task(quantumTaskArn)`

指定された量子タスクをキャンセルします。

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

Boto3 と Amazon Braket SDK の AWS CLI プロファイルを設定する

Amazon Braket SDK は、明示的に指定しない限り、デフォルトの AWS CLI 認証情報に依存します。マネージド Amazon Braket ノートブックで を実行するときは、デフォルトのままにすることをお勧めします。ノートブックインスタンスを起動する権限を持つ IAM ロールを指定する必要があるためです。

オプションで、コードをローカル (Amazon EC2 インスタンスなど) で実行する場合、名前付き AWS CLI プロファイルを確立できます。デフォルトのプロファイルを定期的に上書きするのではなく、各プロファイルに異なる権限セットを与えることができます。

このセクションでは、このような CLI を設定する方法`profile`と、そのプロファイルを Amazon Braket に組み込む方法を簡単に説明し、そのプロファイルからのアクセス許可でAPI呼び出しを行います。

このセクションの内容:

- ステップ 1: ローカル を設定する AWS CLI`profile`
- ステップ 2: Boto3 セッションオブジェクトを確立する
- ステップ 3: Boto3 セッションを Braket に組み込む AwsSession

ステップ 1: ローカル を設定する AWS CLI**`profile`**

ユーザーの作成方法とデフォルト以外のプロファイルの設定方法については、このドキュメントの範囲外です。これらのトピックの詳細については、以下を参照してください。

- IAM の使用開始

- を使用する AWS CLI ための の設定 AWS IAM Identity Center

Amazon Braket を使用するには、このユーザー、および関連する CLI `profile`に必要な Braket アクセス許可を付与する必要があります。例えば、AmazonBraketFullAccessポリシーをアタッチできます。

ステップ 2: Boto3 セッションオブジェクトを確立する

Boto3 セッションオブジェクトを確立するには、次のコード例を使用します。

```
from boto3 import Session

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

> **Note**
>
> 予想されるAPI呼び出しに`profile`デフォルトのリージョンと一致しないリージョンベースの制限がある場合は、次の例に示すように Boto3 セッションのリージョンを指定できます。

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

として指定された引数で`region`、、 など`us-east-1`、AmazonBraket AWS リージョン が利用可能な のいずれかに対応する値に置き換え`us-west-1`ます。

ステップ 3: Boto3 セッションを Braket に組み込む AwsSession

次の例は、Boto3 Braket セッションを初期化し、そのセッションでデバイスをインスタンス化する方法を示しています。

```
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)
```

この設定が完了したら、インスタンス化されたAwsDeviceオブジェクトに量子タスクを送信できます (例えば、 device.run(…) コマンドを呼び出す) 。そのデバイスによって行われたすべてのAPI呼び出しは、以前に として指定した CLI プロファイルに関連付けられた IAM 認証情報を活用できますprofile。

Amazon Braket でのPulse 制御

このセクションでは、Amazon Braket のさまざまな QPUs で の脈動制御を使用する方法について説明します。

このセクションの内容:

Braket Pulse

Pulse は、量子コンピュータ内の量子ビットを制御するアナログ信号です。Amazon Braket の特定のデバイスでは、 の 脈動制御機能にアクセスして、 脈動を使用して回路を送信できます。Braket SDK、OpenQASM 3.0 を使用するか、Braket APIs を使用して直接、 から の脈動制御にアクセスできます。まず、Braket での脈動制御の主要な概念をいくつか紹介します。

[フレーム]

フレームは、量子プログラム内のクロックとフェーズの両方として機能するソフトウェア抽象化です。クロック時間は、使用量ごと、および周波数で定義されるステートフルキャリア信号ごとに増分されます。量子ビットに信号を送信するとき、フレームは量子ビットのキャリア周波数、フェーズオフセット、および波形エンベロープが出力される時間を決定します。Braket Pulse では、フレームの構築はデバイス、周波数、フェーズによって異なります。デバイスに応じて、事前定義されたフレームを選択するか、ポートを指定して新しいフレームをインスタンス化できます。

```
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)
```

ポート

ポートは、量子ビットを制御する入出力ハードウェアコンポーネントを表すソフトウェア抽象化です。ハードウェアベンダーが、ユーザーが操作して量子ビットを観察できるインターフェイスを提供するのに役立ちます。ポートは、コネクタの名前を表す単一の文字列によって特徴付けられます。この文字列は、波形をどれだけ細かく定義できるかを指定する最小時間増分も公開します。

```
from braket.pulse import Port
Port0 = Port("channel_0", dt=1e-9)
```

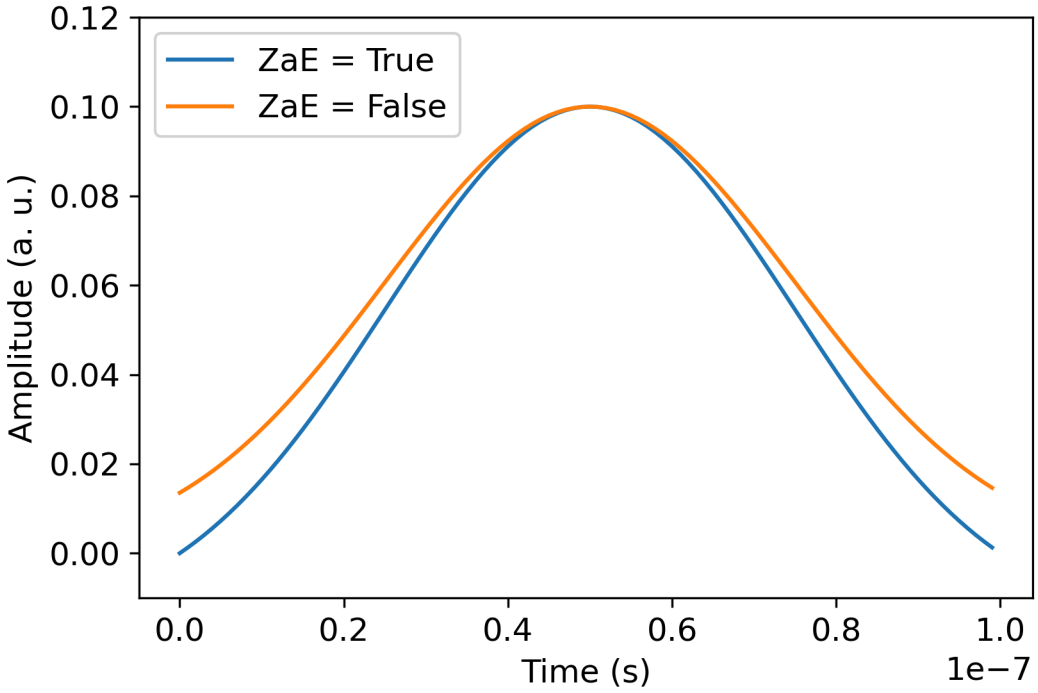
ウェーブフォーム

波形は、出力ポートで信号を発したり、入力ポートを介して信号をキャプチャしたりするために使用できる時間依存エンベロープです。複雑な数値のリストを通じて、または波形テンプレートを使用してハードウェアプロバイダーからリストを生成することで、波形を直接指定できます。

```
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 は、一定の波形、ガウス波形、およびダイアバティックゲートによる微分除去 (DRAG) 波形を含む波形の標準ライブラリを提供します。次の例に示すように、 `sample`関数を使用して波形データを取得して、波形のシェープを描画できます。

```
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))
```



上の画像は、 から作成されたガウス波形を示していますGaussianWaveform。100 ns、幅 25 ns、振幅 0.1 (任意の単位) の 1 つの 1 つの 100 ns の 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの 1 つの つの 1 つの 1 つの つの 1 つの つの 波形は、脈波ウィンドウの中央に配置されます。 はブール引数 zero_at_edges (凡例の ZaE) GaussianWaveformを受け入れます。に設定するとTrue、この引数は t=0 および t=length のポイントがゼロになるようにガウス波形をオフセットし、最大値がamplitude引数に対応するようにその振幅を再スケーリングします。

ここまでで、「脈線レベルアクセスの基本概念」を学習しました。次に、ゲートと脈線を使用して回路を構築する方法を説明します。

フレームとポートのロール

このセクションでは、各デバイスで使用できる事前定義されたフレームとポートについて説明します。また、特定のフレームで脈が再生される際に関係するメカニズムについても簡単に説明します。

Rigetti

[Frames] (フレーム)

Rigetti デバイスは、関連する量子ビットで勾配に達するように周波数とフェーズが
キャリブレートされた事前定義されたフレームをサポートします。命名規則は、 q{i}

[_q{j}]_{role}_frameが{i}最初の量子ビット数を{j}指し、 がフレームが 2 量子ビットインタラクションをアクティブ化する場合の 2 番目の量子ビット数を指し、 がフレームのロール{role}を指します。ロールは次のとおりです。

- rf は、量子ビットの 0~1 遷移を駆動するフレームです。Pulse は、以前に setおよび shift関数を通じて提供された周波数とフェーズの一時的な信号として送信されます。信号の時間依存的な振幅は、フレームで再生される波形によって決まります。フレームは、単一量子ビットの対角線外インタラクションをプラグします。詳細については、「Krantz et al.」および「Rahamim et al.」を参照してください。
- rf_f12 は に似rfており、そのパラメータは 1~2 の移行をターゲットにしています。
- ro_rx は、結合された共平面導波路を介して量子ビットの分散読み取りを実現するために使用されます。読み取り波形の頻度、フェーズ、およびパラメータの完全なセットは事前にキャリブレーションされています。現在、フレーム識別子以外の引数capture_v0を必要としない を介して使用されます。
- ro_tx は、リソネーターから信号を送信するためのものです。現在使用されていません。
- cz は、2 量子ビットczゲートを有効にするようにキャリブレートされたフレームです。ff ポートに関連付けられているすべてのフレームと同様に、ネイバーと のペアのチューニング可能な量子ビットを調整することで、フラックスラインを介したエンタングルインタラクションを有効にします。エンタングルメカニズムの詳細については、「Reagor et al.」、「Caldwell et al.」、および「Didier et al.」を参照してください。
- cphase は、2 量子ビットcphaseshiftゲートを有効にするようにキャリブレートされたフレームで、ffポートにリンクされています。エンタングルメカニズムの詳細については、czフレームの説明を参照してください。
- xy は、2 量子ビット XY (θ) ゲートを有効にするようにキャリブレートされたフレームで、ffポートにリンクされています。エンタングルメカニズムと XY ゲートの達成方法の詳細については、czフレームの説明と Abrams et al. を参照してください。

ff ポートに基づくフレームが調整可能な量子ビットの頻度をシフトすると、量子ビットに関連する他のすべての駆動フレームは、振幅と頻度シフトの期間に関連する量だけ遅延されます。したがって、対応するフェーズシフトを隣接する量子ビットのフレームに追加して、この効果を補正する必要があります。

ポート

Rigetti デバイスは、デバイス機能を通じて検査できるポートのリストを提供します。ポート名は、q{i}_{type} が量子ビット番号{i}を参照し、 がポートのタイプ{type}を参照する規則に従いま

す。すべての量子ビットにポートの完全なセットがあるわけではないことに注意してください。ポートのタイプは次のとおりです。

- `rf` は、単一量子ビット移行を駆動するメインインターフェイスを表します。これは `rf`および `rf_f12`フレームに関連付けられています。量子ビットに容量的に結合されているため、ギガヘルツ範囲で走行できます。
- `ro_tx` は、量子ビットに容量的に結合されたリードアウトレゾネーターに信号を送信する役割を果たします。読み取り信号配信は、8 倍に 8 倍に 8 八角で多重化されます。
- `ro_rx` は、量子ビットに結合された読み取りリゾネーターから信号を受信する役割を果たします。
- `ff` は、量子ビットに帰納的に結合された高速フラックス線を表します。これを使用して、トランスモンの頻度を調整できます。高度に調整可能なように設計された量子ビットのみが`ff`ポートを持ちます。このポートは、隣接するトランスモンの各ペア間に静的容量結合があるため、量子ビットと量子ビットの相互作用をアクティブにします。

アーキテクチャの詳細については、「Valery et al」を参照してください。

OQC

[Frames] (フレーム)

OQC デバイスは、関連する量子ビットで跳ね返るように周波数とフェーズがキャリブレーションされた事前定義されたフレームをサポートします。これらのフレームの命名規則は次のとおりです。

- 駆動フレーム: `q{i}[_q{j}]_{role}` ここで、 `{i}` は最初の量子ビット数を指し、 `{j}`はフレームが 2 量子ビットの相互作用をアクティブ化する場合の 2 番目の量子ビット数`{role}`を指し、 は以下で説明するようにフレームのロールを指します。
- 量子ビット読み取りフレーム: `r{i}_{role}` ここで、 `{i}`は量子ビット数を`{role}`指し、 は以下で説明するようにフレームのロールを指します。

設計上のロールには、次のように各フレームを使用することをお勧めします。

- `drive` は、量子ビットの 0~1 遷移を駆動するメインフレームとして使用されます。Pulse は、以前に `set`および `shift`関数を通じて提供された周波数とフェーズの一時的な信号として送信されます。信号の時間依存的な振幅は、フレームで再生される波形によって決まります。フレームは、単一量子ビットの対角線外インタラクションをプラグします。詳細については、「Krantz et al.」および「Rahamim et al.」を参照してください。

- second_state はdriveフレームと同等ですが、その頻度は 1~2 の移行で跳ね返って調整されます。
- measure は読み取り用です。読み取り波形の頻度、フェーズ、およびパラメータの完全なセットは事前にキャリブレーションされています。現在、フレーム識別子以外の引数capture_v0を必要としない を通じて使用されます。
- acquire は、リソネーターからシグナルをキャプチャするためのものです。現在使用されていません。
- cross_resonance は、ターゲット量子ビット の移行頻度iで制御量子ビット を駆動jすることで、量子ビット i と の間の交差結合相互作用をアクティブ化しますj。したがって、フレーム周波数はターゲット量子ビットの頻度を使用して設定されます。相互作用は、このクロスレゾナンシードライブの振幅に比例するレートで発生します。クロスタルクのタイプが異なると、修正が必要な望ましくない影響が発生します。軸形状のトランスモン量子ビット (「coaxmons」) との交差共鳴相互作用の詳細については、「Patterson et al.」を参照してください。
- cross_resonance_cancellation は、交差共鳴相互作用がアクティブ化されたときにクロスタルクによって誘発される有害な効果を抑制するための修正を追加するのに役立ちます。初期フレーム周波数は、制御量子ビット の移行周波数に設定されますi。キャンセル方法の詳細については、「Patterson et al.」を参照してください。

ポート

OQC デバイスは、デバイス機能を通じて検査できるポートのリストを提供します。前述のフレームは、ID によって識別されるポートに関連付けられます。channel_{N}ここで、 {N}は整数です。ポートは、Coaxmons に接続されたライン (方向 tx) と読み取りレゾネーター (方向 rx) を制御するためのインターフェイスです。各量子ビットは、1 つのコントロールラインと 1 つの読み取りレゾネーターに関連付けられます。送信ポートは、単一量子ビットおよび 2 量子ビット操作のインターフェイスです。受信ポートは、量子ビットの読み取りを行います。

Hello Pulse

ここでは、シンプルなベルペアを の脈で直接構築し、この脈動プログラムをRigettiデバイスで実行する方法について説明します。ベルペアは、最初の量子ビットのハダマードゲートと、それに続く最初の量子ビットと 2 番目の量子ビットの間のcnotゲートで構成される 2 量子ビット回路です。脈でエンタングル状態を作成するには、ハードウェアタイプとデバイスアーキテクチャに依存する特定のメカニズムが必要です。ネイティブメカニズムを使用してcnotゲートを作成することはありません。代わりに、czゲートをネイティブに有効にする特定の波形とフレームを使用します。この例で

は、単一量子ビットのネイティブゲートを使用してハダマードゲートを作成しrxrz、 および は の脈を使用してczゲートを表現します。

まず、必要なライブラリをインポートします。Circuit クラスに加えて、 PulseSequence クラスをインポートする必要もあります。

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

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

次に、デバイスの Amazon リソースネーム (ARN) を使用して、新しい Braket Rigetti Aspen-M-3 デバイスをインスタンス化します。デバイスのレイアウトを表示するには、Amazon Braket コンソールのRigetti Aspen-M-3「デバイス」ページを参照してください。

```
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")
```

ハダマードゲートはRigettiデバイスのネイティブゲートではないため、 の脈と組み合わせて使用することはできません。したがって、 rxとrzネイティブゲートのシーケンスに分解する必要があります。

```
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)
    )
```

cz ゲートでは、キャリブレーション段階でハードウェアプロバイダーによって事前に定義されたパラメータ (振幅、立上り/立下り時間、継続時間) を含む任意の波形を使用します。この波形は に適用されますq10_q113_cz_frame。ここで使用されている任意の波形の最新バージョンについては、

Rigettiウェブサイトの「QCS」を参照してください。場合によっては、QCS アカウントを作成する必要があります。

```
a_b_cz_wfm = ArbitraryWaveform([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00017888439538396808, 0.00046751103636033026, 0.0011372942989106456,
 0.002577059611929697, 0.005443941944632366, 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')
```

これは以下を返します。

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

これで、先ほど定義した波形を使用してczゲートを構築できます。制御量子ビットが |1>状態にある場合、czゲートはターゲット量子ビットのフェーズフリップで構成されていることを思い出してください。

```
phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

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

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

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

a_b_cz_wfm 波形は、高速フラックスポートに関連付けられたフレームで再生されます。その役割は、量子ビット周波数をシフトして量子ビットと量子ビットの相互作用をアクティブ化することです。詳細については、「 フレームとポートのロール」を参照してください。周波数が変化するにつれて、量子ビットフレームは、そのまま維持される単一量子ビットrfフレームとは異なるレートでローテーションします。後者のフレームはデフェーズされます。これらのフェーズシフトは事前にRamseyシーケンスで調整されており、ここでは phase_shift_aおよび phase_shift_b (全期間)を通じてハードコードされた情報として提供されます。rf フレームshift_phaseの指示を使用して、このデフェージングを修正しました。このシーケンスは、量子ビットに関連するXYフレームがなくab、これらのフレームで発生するフェーズシフトを補正しないため、使用されるプログラムでのみ機能することに注意してください。これは、 フレームrfと czフレームのみを使用するこの単一のベルペアプログラムの場合です。詳細については、「Caldwell et al.」を参照してください。

ここまでで、ベルペアと の脈線を作成する準備が整いました。

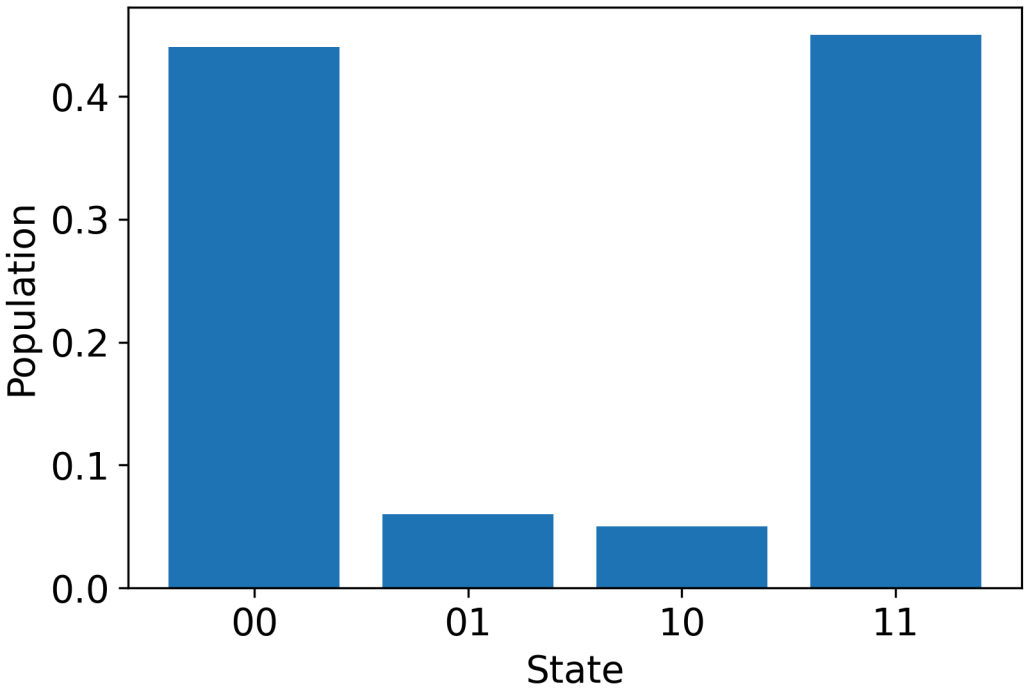
```
bell_circuit_pulse = (
    Circuit()
    .rigetti_native_h(a)
    .rigetti_native_h(b)
```

```
    .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    |
```

このベルペアをRigettiデバイスで実行しましょう。このコードブロックを実行すると料金が発生することに注意してください。これらのコストの詳細については、Amazon Braket の料金」ページを参照してください。少量のショットを使用して回路をテストし、ショット数を増やす前にデバイスで実行できることを確認することをお勧めします。

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

counts = task.result().measurement_counts

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



を使用した Hello Pulse OpenPulse

OpenPulse は、一般的な量子デバイスの脈波レベルの制御を指定するための言語であり、OpenQASM 3.0 仕様の一部です。Amazon Braket はOpenPulse、OpenQASM 3.0 表現を使用した直接プログラミングの をサポートしています。

Braket は、ネイティブ命令で脈を表現するための基盤となる中間表現OpenPulseとして を使用します。 は、 `defcal` (「キャリブレーションの定義」の略) 宣言の形式で命令キャリブレーションの追加OpenPulseをサポートしています。これらの宣言を使用すると、下位レベルの制御文法内でゲート命令の実装を指定できます。

この例では、OpenQASM 3.0 を使用してベル回路を構築し、周波数調整可能なトランスモンを使用してデバイスOpenPulseを構築します。ベル回路は、最初の量子ビットのハダマードゲートと、それに続く 2 つの量子ビット間のcnotゲートで構成される 2 量子ビット回路であることを思い出してください。cnot ゲートは基本変換によってのみczゲートと異なるため、ここでは代わりにハダマードとczゲートを使用してベルペアを定義します。デバイスがこのデモンストレーションのczゲートを作成するより簡単な方法を提供するためです。

まず、デバイスのネイティブゲートを使用してハダマードゲートを定義します。

```
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;
}
```

CZ ゲートでは、事前に事前定義されたパラメータ (振幅、立上り/立下り時間、継続時間) を持つ任意の波形を使用します。この波形は に適用されます`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, 0.48443981200754405, 0.484439812007528 4,
0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.484432196472809 6,
0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.481411707540660 3,
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;
}
```

q10_q113_cz_wfm 波形の期間は 124 サンプルで、最小時間増分dtは 1 ns であるため、124 ns に対応します。

q10_q113_cz_wfm 波形は、高速フラックスポートにバインドされたフレームで再生されます。その役割は、量子ビット周波数をシフトして量子ビットと量子ビットの相互作用をアクティブ化することです。詳細については、「フレームとポートのロール」を参照してください。頻度が変化するにつれて、量子ビットフレームは、変更されないままの単一量子ビットrfフレームと比較して異なるレートでローテーションします。後者のフレームはデフェーズされます。このデフェージングは、キャリブレーション段階でRamseyシーケンスで測定し、 rfおよび xyフレームshift_phaseの指示で補正できます。詳細については、「Caldwell et al.」を参照してください。

これで、いくつかのハダマードとcnotゲートを使用してczゲートを分解したベルペア回路を実行できるようになりました。

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

ネイティブのゲートと脈の組み合わせを使用して構築されたベル回路の完全な OpenQASM 3.0 表現は次のとおりです。

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
```

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

```
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
 0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
 0.48443979687791755, 0.4844396697373718, 0.484438680618381 7, 0.484432196472809 6,
 0.484396376639855 8, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
 0.475381140971073 4, 0.4612131109596855 3, 0.433155425132028 5, 0.3863175050956295 7,
 0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
 0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
 0.00086447604313 74626, 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;
```

Braket SDK を使用して、次のコードを使用してRigettiデバイスでこの OpenQASM 3.0 プログラムを実行できるようになりました。

```
# 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,
)
```

パルスを使用したネイティブゲートへのアクセス

多くの場合、研究者は、特定の QPU でサポートされているネイティブゲートが、どのように実装されるかを正確に把握する必要があります。Pulse シーケンスはハードウェアプロバイダーによって慎重に調整されていますが、それらにアクセスすると、研究者はより良いゲートを設計したり、特定のゲートの脈を伸張してノイズゼロ外挿などのエラー緩和のためのプロトコルを探索したりできます。

Amazon Braket は、Rigetti からのネイティブゲートへのプログラムによるアクセスをサポートしています。

```
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**
>
> ハードウェアプロバイダーは定期的に QPU をキャリブレートし、多くの場合 1 日に 2 回以上キャリブレートします。Braket SDK を使用すると、最新のゲートキャリブレーションを取得できます。

```
device.refresh_gate_calibrations()
```

RX や XY ゲートなど、特定のネイティブゲートを取得するには、`Gate`オブジェクトと目的の量子ビットを渡す必要があります。例えば、0 に適用された RX(π/2) qubit の脈波実装を検査できます。

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

`filter` 関数を使用して、フィルタリングされたキャリブレーションのセットを作成できます。ゲートのリストまたは のリストを渡しますQubitSet。次のコードは、RX(π/2) と 0 のすべてのキャリブレーションを含む 2 qubit つのセットを作成します。

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

これで、カスタムキャリブレーションセットをアタッチして、ネイティブゲートのアクションを提供または変更できます。例えば、次の回路を考えてみましょう。

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

PulseSequence オブジェクトのディクショナリをgate_definitionsキーワード引数に渡すqubit 0ことで、 のrxゲートのカスタムゲートキャリブレーションで実行できます。GateCalibrations オブジェクトpulse_sequencesの 属性からディクショナリを作成できます。指定されていないすべてのゲートは、量子ハードウェアプロバイダーの脈波キャリブレーションに置き換えられます。

```
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 ユーザーガイド

このセクションでは、AmazonBraket でハイブリッドジョブをセットアップして管理する方法について説明します。

Braket のハイブリッドジョブには、以下を使用してアクセスできます。

- Amazon Braket Python SDK。
- Amazon Braket コンソール 。
- Amazon Braket API。

このセクションの内容:

- を使用してハイブリッドジョブを直接操作する API

ハイブリッドジョブとは

Amazon Braket Hybrid Jobs は、古典的な AWS リソースと量子処理単位 (QPUs) の両方を必要とするハイブリッド量子古典アルゴリズムを実行する方法を提供します。Hybrid Jobs は、リクエストされたクラシックリソースをスピンアップし、アルゴリズムを実行し、完了後にインスタンスを解放するように設計されているため、 の使用分に対してのみ料金が発生します。

Hybrid Jobs は、古典リソースと量子リソースの両方を含む長時間実行される反復アルゴリズムに最適です。アルゴリズムを送信して実行すると、Braket はスケーラブルなコンテナ化された環境で実行し、アルゴリズムが完了すると結果を取得します。

さらに、ハイブリッドジョブから作成された量子タスクは、ターゲット QPU へのキューイングの優先度が高いという利点があります。これにより、量子タスクがキュー内の他のタスクよりも先に処理および実行されます。これは、後続のタスクが以前の量子タスクの結果に依存する反復ハイブリッドアルゴリズムに特に有益です。このようなアルゴリズムの例としては、量子近似最適化アルゴリズム (QAOA)、バリエーション量子固有ソルバー、量子機械学習 などがあります。また、アルゴリズムの進行状況をほぼリアルタイムでモニタリングできるため、コスト、予算、トレーニング損失や期待値などのカスタムメトリクスを追跡できます。

Amazon Braket Hybrid Jobsを使用する時期

Amazon Braket Hybrid Jobs を使用すると、従来のコンピューティングリソースと量子コンピューティングデバイスを組み合わせて今日の量子システムのパフォーマンスを最適化する、Variational Quantum Eigensolver (VQE) や Quantum Approximate Optimization Algorithm (QAOA) などのハイブリッド量子古典アルゴリズムを実行できます。Amazon Braket Hybrid Jobs には、主に次の 3 つの利点があります。

1. パフォーマンス: Amazon Braket Hybrid Jobs は、お客様の環境からハイブリッドアルゴリズムを実行するよりも優れたパフォーマンスを提供します。ジョブの実行中に、選択したターゲット QPU に優先的にアクセスできます。ジョブのタスクは、デバイスでキューに入れられた他のタスクよりも先に実行されます。これにより、ハイブリッドアルゴリズムのランタイムが短くなり、予測しやすくなります。Amazon Braket Hybrid Jobs は、パラメトリックコンパイルもサポートしています。フリーパラメータを使用して回路を送信でき、Braket は回路を 1 回コンパイルします。同じ回路にパラメータを後で更新するために再コンパイルする必要がないため、ランタイムがさらに短縮されます。

2. 利便性: Amazon Braket Hybrid Jobs は、コンピューティング環境のセットアップと管理を簡素化し、ハイブリッドアルゴリズムの実行中も実行し続けることができます。アルゴリズムスクリプトを指定し、実行する量子デバイス (量子処理ユニットまたはシミュレーター) を選択するだけです。Amazon Braket は、ターゲットデバイスが使用可能になるまで待機し、クラシックリソースをスピンアップし、構築済みのコンテナ環境でワークロードを実行し、結果を Amazon Simple Storage Service (Amazon S3) に返し、コンピューティングリソースを解放します。
3. メトリクス: Amazon Braket Hybrid Jobs は、実行中のアルゴリズムに関する on-the-fly インサイトを提供し、カスタマイズ可能なアルゴリズムメトリクスをほぼリアルタイムで Amazon CloudWatch および Amazon Braket コンソールに配信して、アルゴリズムの進行状況を追跡できるようにします。

ローカルコードをハイブリッドジョブとして実行する

Amazon Braket Hybrid Jobs は、Amazon EC2 コンピューティングリソースと Amazon Braket Quantum Processing Unit (QPU) アクセスを組み合わせた、ハイブリッド量子クラシックアルゴリズムのフルマネージドオーケストレーションを提供します。ハイブリッドジョブで作成された量子タスクは、個々の量子タスクよりも優先キューイングされるため、量子タスクキューの変動によってアルゴリズムが中断されることはありません。各 QPU は個別のハイブリッドジョブキューを維持し、一度に実行できるハイブリッドジョブは 1 つだけです。

このセクションの内容:

- ローカル Python コードからハイブリッドジョブを作成する
- 追加の Python パッケージとソースコードをインストールする
- ハイブリッドジョブインスタンスにデータを保存してロードする
- ハイブリッドジョブデコレータのベストプラクティス

ローカル Python コードからハイブリッドジョブを作成する

ローカル Python コードを Amazon Braket Hybrid Job として実行できます。これを行うには、次のコード例に示すように、コードにデ`@hybrid_job`コレータで注釈を付けます。カスタム環境では、Amazon Elastic Container Registry (ECR) のカスタムコンテナを使用することを選択できます。

> **Note**
>
> デフォルトでは、Python 3.10 のみがサポートされています。

デ@hybrid_jobコレータを使用して関数に注釈を付けることができます。Braket は、デコレータ内のコードを Braket ハイブリッドジョブアルゴリズムスクリプト に変換します。次に、ハイブリッドジョブは Amazon EC2 インスタンスのデコレータ内で 関数を呼び出します。ジョブの進行状況は、`job.state()`または Braket コンソールでモニタリングできます。次のコード例は、 で 5 つの状態のシーケンスを実行する方法を示していますState Vector Simulator (SV1) device。

```
from braket.aws import AwsDevice
from braket.circuits import Circuit, FreeParameter, Observable
from braket.devices import Devices
from braket.jobs.hybrid_job import hybrid_job
from braket.jobs.metrics import log_metric

device_arn = Devices.Amazon.SV1


@hybrid_job(device=device_arn)  # choose priority device
def run_hybrid_job(num_tasks=1):
    device = AwsDevice(device_arn)  # declare AwsDevice within the hybrid job

    # create a parametric circuit
    circ = Circuit()
    circ.rx(0, FreeParameter("theta"))
    circ.cnot(0, 1)
    circ.expectation(observable=Observable.X(), target=0)

    theta = 0.0  # initial parameter

    for i in range(num_tasks):
        task = device.run(circ, shots=100, inputs={"theta": theta})  # input parameters
        exp_val = task.result().values[0]

        theta += exp_val  # modify the parameter (possibly gradient descent)

        log_metric(metric_name="exp_val", value=exp_val, iteration_number=i)

    return {"final_theta": theta, "final_exp_val": exp_val}
```

ハイブリッドジョブを作成するには、通常の Python 関数と同じように 関数を呼び出します。ただし、デコレータ関数は、関数の結果ではなくハイブリッドジョブハンドルを返します。完了後に結果を取得するには、 を使用しますjob.result()。

```
job = run_hybrid_job(num_tasks=1)
```

```
result = job.result()
```

デ@hybrid_jobコレータのデバイス引数は、ハイブリッドジョブが優先的にアクセスできるデバイスを指定します。この場合はシミュレータSV1ーです。QPU の優先度を取得するには、関数内で使用されるデバイス ARN がデコレータで指定されたものと一致することを確認する必要があります。便宜上、ヘルパー関数を使用してget_job_device_arn()、 で宣言されたデバイス ARN をキャプチャできます@hybrid_job。

> **Note**
>
> 各ハイブリッドジョブは、Amazon EC2 でコンテナ化された環境を作成しているため、起動時間が少なくとも 1 分あります。したがって、1 つの回路や回路のバッチなど、非常に短いワークロードでは、量子タスクを使用するだけで十分です。

ハイパーパラメータ

run_hybrid_job() 関数は 引数num_tasksを使用して、作成された量子タスクの数を制御します。ハイブリッドジョブは、これをハイパーパラメータ として自動的にキャプチャします。

> **Note**
>
> ハイパーパラメータは、2500 文字に制限された文字列として Braket コンソールに表示されます。

メトリクスとログ記録

run_hybrid_job() 関数内では、反復アルゴリズムからのメトリクスは で記録されますlog_metrics。メトリクスは、ハイブリッドジョブタブの Braket コンソールページに自動的にプロットされます。Braket コストトラッカー を使用して、ハイブリッドジョブ実行中の量子タスクコストをほぼリアルタイムで追跡できます。上記の例では、結果タイプ の最初の確率を記録するメトリクス名「確率」を使用しています。

結果の取得

ハイブリッドジョブが完了したら、 job.result()を使用してハイブリッドジョブの結果を取得します。return ステートメント内のすべてのオブジェクトは Braket によって自動的にキャプチャされます。関数によって返されるオブジェクトは、各要素がシリアル化可能なタプルである必要があることに注意してください。例えば、次のコードは、動作中の と失敗の例を示しています。

```
@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
```

ジョブ名

デフォルトでは、このハイブリッドジョブの名前は関数名から推測されます。最大 50 文字のカスタム名を指定することもできます。例えば、次のコードでは、ジョブ名は「my-job-name」です。

```
@hybrid_job(device=Devices.Amazon.SV1, job_name="my-job-name")
def function():
    pass
```

ローカルモード

ローカルジョブは、 引数をデlocal=Trueコレータに追加することによって作成されます。これにより、ラップトップなどのローカルコンピューティング環境のコンテナ化された環境でハイブリッドジョブが実行されます。ローカルジョブには、量子タスクの優先キューイングはありません。マルチノードや MPI などの高度なケースでは、ローカルジョブが必要な Braket 環境変数にアクセスできる場合があります。次のコードは、 デバイスを SV1 シミュレーターとして使用してローカルハイブリッドジョブを作成します。

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

その他のハイブリッドジョブオプションはすべてサポートされています。オプションのリストについては、braket.jobs.quantum_job_creation module を参照してください。

追加の Python パッケージとソースコードをインストールする

任意の Python パッケージを使用するようにランタイム環境をカスタマイズできます。requirements.txt ファイル、パッケージ名のリスト、または独自のコンテナの持ち込み (BYOC) のいずれかを使用できます。requirements.txt ファイルを使用してランタイム環境をカスタマイズするには、次のコード例を参照してください。

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

例えば、 requirements.txt ファイルにはインストールする他のパッケージが含まれている場合があります。

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

または、次のようにパッケージ名を Python リストとして指定することもできます。

```
@hybrid_job(device=Devices.Amazon.SV1, dependencies=["qiskit", "pennylane>=0.31",
 "mitiq==0.29"])
def run_hybrid_job(num_tasks = 1):
    return ...
```

追加のソースコードは、モジュールのリストとして指定することも、次のコード例のように単一のモジュールとして指定することもできます。

```
@hybrid_job(device=Devices.Amazon.SV1, include_modules=["my_module1", "my_module2"])
def run_hybrid_job(num_tasks = 1):
    return ...
```

ハイブリッドジョブインスタンスにデータを保存してロードする

入力トレーニングデータの指定

ハイブリッドジョブを作成するときは、Amazon Simple Storage Service (Amazon S3) バケットを指定して、入力トレーニングデータセットを指定できます。ローカルパスを指定して、Braket が s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name> で Amazon S3 にデータを自動的にアップロードすることもできます。ローカルパスを指定すると、チャンネル名はデフォルトで「入力」になります。次のコードは、ローカルパス からの numpy ファイルを示しています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 ...
```

S3 の場合は、`get_input_data_dir()`ヘルパー関数を使用する必要があります。

```
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")
```

チャンネル値と S3 URIs またはローカルパスのディクショナリを提供することで、複数の入力データソースを指定できます。

```
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**
>
> 入力データが大きい (>1GB) 場合、ジョブが作成されるまでの待機時間が長くなります。これは、ローカル入力データが最初に S3 バケットにアップロードされたときに、S3 パスがジョブリクエストに追加されるためです。最後に、ジョブリクエストは Braket サービスに送信されます。

S3 への結果の保存

修飾された関数の return ステートメントに含まれていない結果を保存するには、すべてのファイル書き込みオペレーションに正しいディレクトリを追加する必要があります。次の例は、numpy 配列と matplotlib 図の保存を示しています。

```
@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 ...
```

すべての結果は、 という名前のファイルに圧縮されますmodel.tar.gz。結果は、Python 関数 を使用して、または Braket job.result() マネジメントコンソールのハイブリッドジョブページから結果フォルダに移動してダウンロードできます。

チェックポイントの保存と再開

長時間実行されるハイブリッドジョブの場合、アルゴリズムの中間状態を定期的に保存することをお勧めします。組み込みsave_job_checkpoint()ヘルパー関数を使用するか、AMZN_BRAKET_JOB_RESULTS_DIRパスにファイルを保存できます。後者は、ヘルパー関数 で使用できますget_job_results_dir()。

以下は、ハイブリッドジョブデコレータを使用してチェックポイントを保存およびロードするための最小限の作業例です。

```
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()
```

最初のハイブリッドジョブでは、 save_job_checkpoint()は保存するデータを含むディクショナリで呼び出されます。デフォルトでは、すべての値をテキストとしてシリアル化する必要があります。numpy 配列など、より複雑な Python オブジェクトをチェックポイントするには、 を設定できますdata_format = PersistedJobDataFormat.PICKLED_V4。このコードは、「checkpoints」というサブフォルダのハイブリッドジョブアーティファクト<jobname>.jsonにデフォルト名を持つチェックポイントファイルを作成して上書きします。

チェックポイントから続行する新しいハイブリッドジョブを作成するには、 が前のジョブのハイブリッドジョブ ARN copy_checkpoints_from_job=job_arnjob_arnである を渡す必要があります。次にload_job_checkpoint(job_name)、 を使用してチェックポイントからロードします。

ハイブリッドジョブデコレータのベストプラクティス

非同期性を採用する

デコレータアノテーションで作成されたハイブリッドジョブは非同期であり、古典リソースと量子リソースが利用可能になると実行されます。アルゴリズムの進行状況をモニタリングするには、Braket Management Consoleまたは Amazon を使用します CloudWatch。アルゴリズムを実行して送信すると、Braket はスケーラブルなコンテナ化された環境でアルゴリズムを実行し、アルゴリズムが完了すると結果を取得します。

反復的なバリエーションアルゴリズムを実行する

ハイブリッドジョブは、反復的な量子古典アルゴリズムを実行するためのツールを提供します。純粋な量子問題の場合は、量子タスク または量子タスク のバッチ を使用します。特定の QPUs への優先アクセスは、QPU への反復呼び出しを複数回必要とする実行時間の長いバリエーションアルゴリズム QPUs に最も有益です。

ローカルモードを使用したデバッグ

QPU でハイブリッドジョブを実行する前に、まずシミュレーター SV1 で を実行して、期待どおりに実行されていることを確認することをお勧めします。小規模なテストでは、ローカルモードで を実行して、迅速な反復とデバッグを行うことができます。

Bring your own container (BYOC) による再現性の向上

コンテナ化された環境内でソフトウェアとその依存関係をカプセル化して、再現可能な実験を作成します。すべてのコード、依存関係、および設定をコンテナにパッケージ化することで、潜在的な競合やバージョニングの問題を防ぐことができます。

マルチインスタンス分散シミュレーター

多数の回路を実行するには、組み込みの MPI サポートを使用して、単一のハイブリッドジョブ内の複数のインスタンスでローカルシミュレーターを実行することを検討してください。詳細については、「埋め込みシミュレーター」を参照してください。

パラメトリック回路を使用する

ハイブリッドジョブから送信するパラメトリック回路は、パラメトリックコンパイルを使用して特定の QPUs で自動的にコンパイルされ、アルゴリズムのランタイムが向上します。 ???

定期的にチェックポイントする

長時間実行されるハイブリッドジョブの場合、アルゴリズムの中間状態を定期的に保存することをお勧めします。

その他の例、ユースケース、ベストプラクティスについては、Amazon Braket の例 GitHub」を参照してください。

Amazon Braket Hybrid Jobs でハイブリッドジョブを実行する

Amazon Braket Hybrid Jobs でハイブリッドジョブを実行するには、まずアルゴリズムを定義する必要があります。Amazon Braket Python SDK または を使用して、アルゴリズムスクリプトと、オプションで他の依存関係ファイルを記述することで定義できますPennyLane。他の (オープンソースまたは独自の) ライブラリを使用する場合は、これらのライブラリを含む Docker を使用して独自のカスタムコンテナイメージを定義できます。詳細については、「自分のコンテナを持参 (BYOC)」を参照してください。

いずれの場合も、次に Amazon Braket を使用してハイブリッドジョブを作成します。ここでAPIは、アルゴリズムスクリプトまたはコンテナを指定し、ハイブリッドジョブが使用するターゲット量子デバイスを選択し、さまざまなオプション設定から選択します。これらのオプション設定で提供されるデフォルト値は、ほとんどのユースケースで機能します。ターゲットデバイスがハイブリッドジョブを実行するには、QPU、オンデマンドシミュレーター (、DM1、 などTN1) SV1、または従来のハイブリッドジョブインスタンス自体のいずれかを選択できます。オンデマンドシミュレーターまたは QPU を使用すると、ハイブリッドジョブコンテナはリモートデバイスに API コールを行います。組み込みシミュレーターを使用すると、シミュレーターはアルゴリズムスクリプトと同じコンテナに埋め込まれます。の稲妻シミュレーター PennyLane には、デフォルトの構築済みハイブリッドジョブコンテナが埋め込まれているため、使用できます。埋め込み PennyLane シミュレーターまたはカスタムシミュレーターを使用してコードを実行する場合は、インスタンスタイプと使用するインスタン

スの数を指定できます。各選択肢に関連するコストについては、Amazon Braket の料金ページを参照してください。

ターゲットデバイスがオンデマンドシミュレーターまたは埋め込みシミュレーターの場合、Amazon Braket はハイブリッドジョブの実行をすぐに開始します。ハイブリッドジョブインスタンスを起動し (API 呼び出しでインスタンスタイプをカスタマイズできます)、アルゴリズムを実行し、結果を Amazon S3 に書き込み、リソースを解放します。このリリースのリソースを使用すると、使用した分に対してのみお支払いいただくことができます。

量子処理ユニット (QPU) あたりの同時ハイブリッドジョブの合計数は制限されています。現在、1 つの QPU で実行できるハイブリッドジョブは 1 回に 1 つだけです。キューは、許可される制限を超えないように、実行できるハイブリッドジョブの数を制御するために使用されます。ターゲットデバイスが QPU の場合、ハイブリッドジョブは最初に選択した QPU のジョブキューに入ります。Amazon Braket は、必要なハイブリッドジョブインスタンスを起動し、デバイスでハイブリッドジョブを実行します。アルゴリズムの期間中、ハイブリッドジョブには優先アクセスがあります。つまり、ジョブ量子タスクが数分に 1 回 QPU に送信されていれば、ハイブリッドジョブの量子タスクはデバイスでキューに入れられた他の Braket 量子タスクよりも先に実行されます。ハイブリッドジョブが完了すると、リソースが解放されます。つまり、使用した分に対してのみ料金が発生します。

> **Note**
>
> デバイスはリージョナルであり、ハイブリッドジョブはプライマリデバイス AWS リージョン と同じ で実行されます。

シミュレーターと QPU ターゲットシナリオの両方で、アルゴリズムの一部としてハミルトニアンのエネルギーなどのカスタムアルゴリズムメトリクスを定義するオプションがあります。これらのメトリクスは Amazon に自動的に報告 CloudWatch され、そこから Amazon Braket コンソールにほぼリアルタイムで表示されます。

> **Note**
>
> GPU ベースのインスタンスを使用する場合は、Braket の埋め込みシミュレーターで使用可能な GPU ベースのシミュレーターのいずれかを使用してください (例: `lightning.gpu`) 。CPU ベースの埋め込みシミュレーター (、、 など`braket:default-simulator`) のいずれかを選択した場合`lightning.qubit`、GPU は使用されず、不要なコストが発生する可能性があります。

最初のハイブリッドジョブを作成する

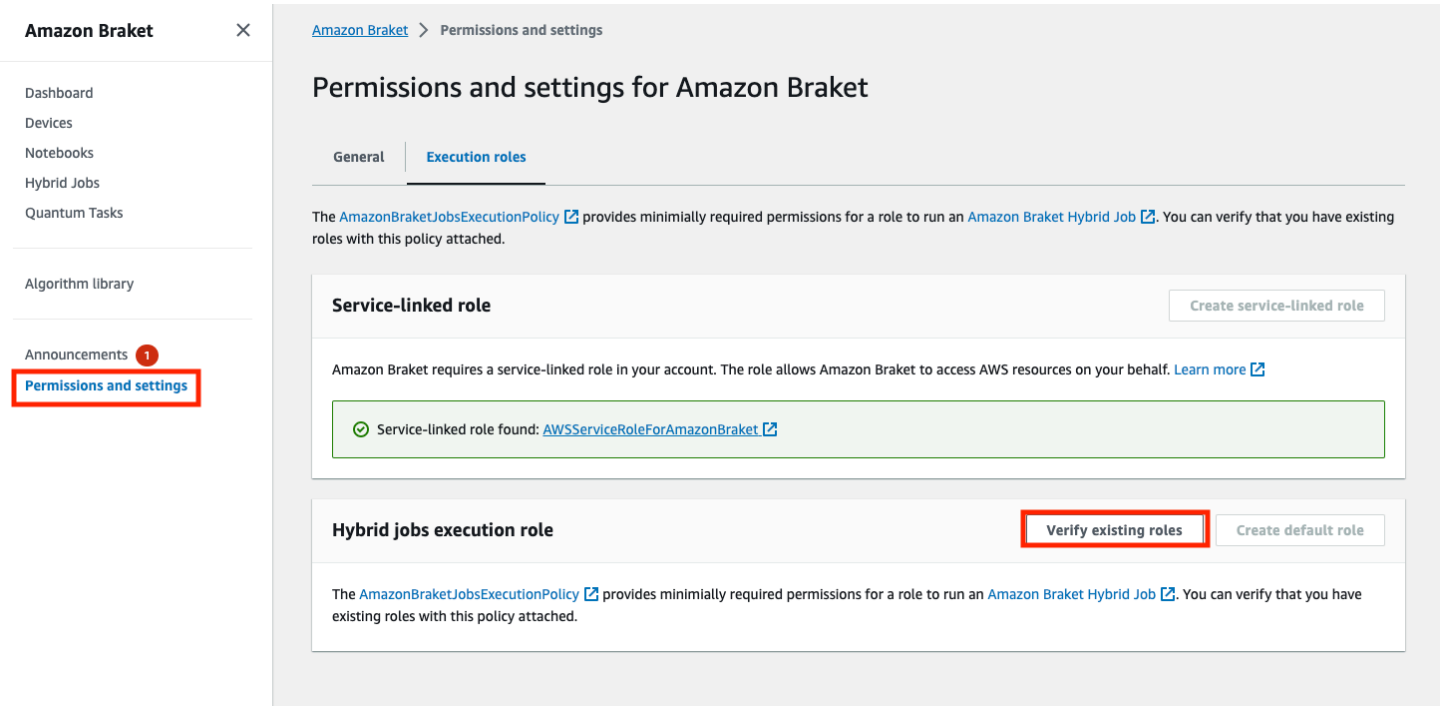
このセクションでは、Python スクリプトを使用してハイブリッドジョブを作成する方法について説明します。または、希望する統合開発環境 (IDE) や Braket ノートブックなどのローカル Python コードからハイブリッドジョブを作成するには、「」を参照してくださいローカルコードをハイブリッドジョブとして実行する。

このセクションの内容:

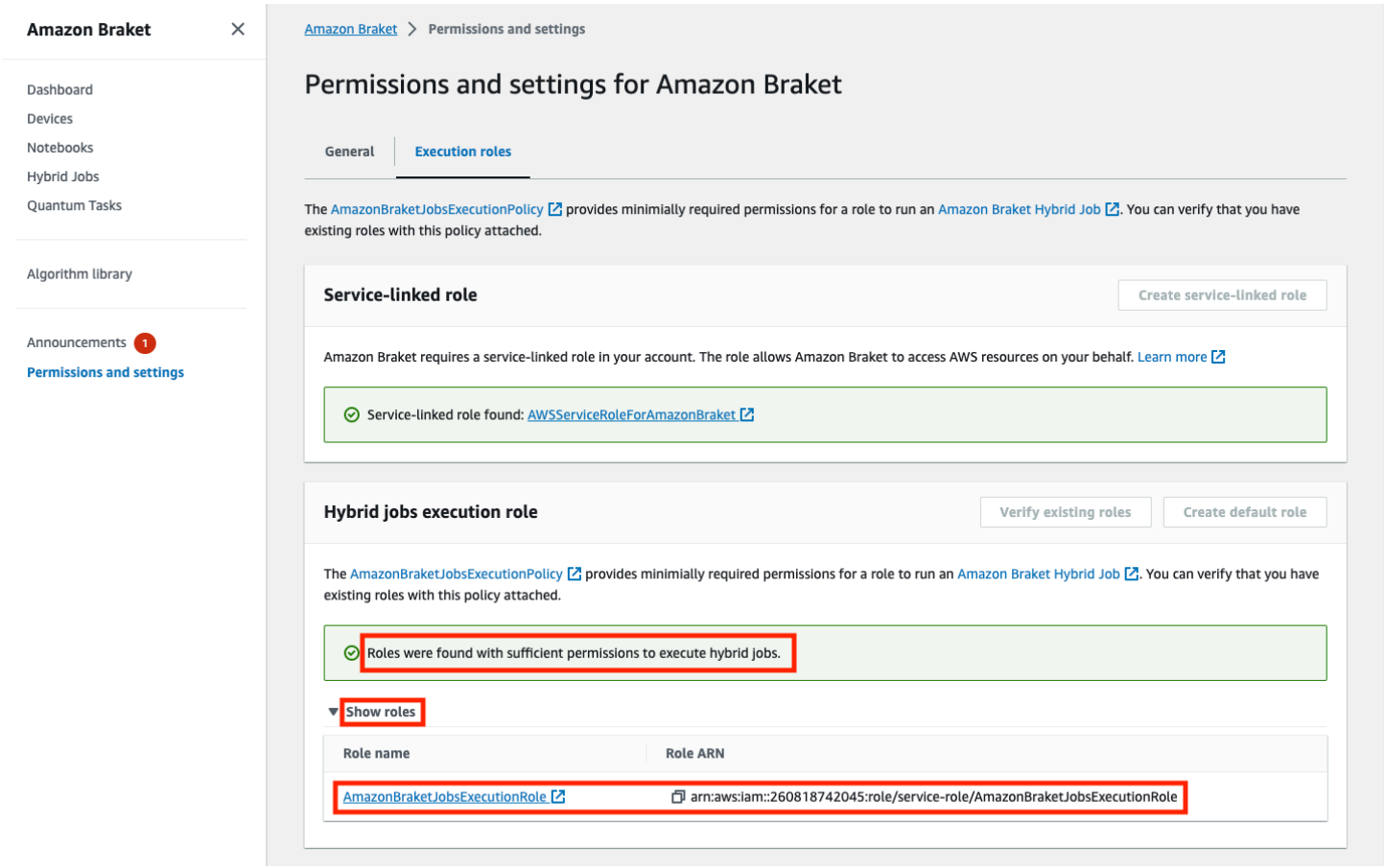
- アクセス許可を設定する
- を作成して実行する
- 結果をモニタリングする

アクセス許可を設定する

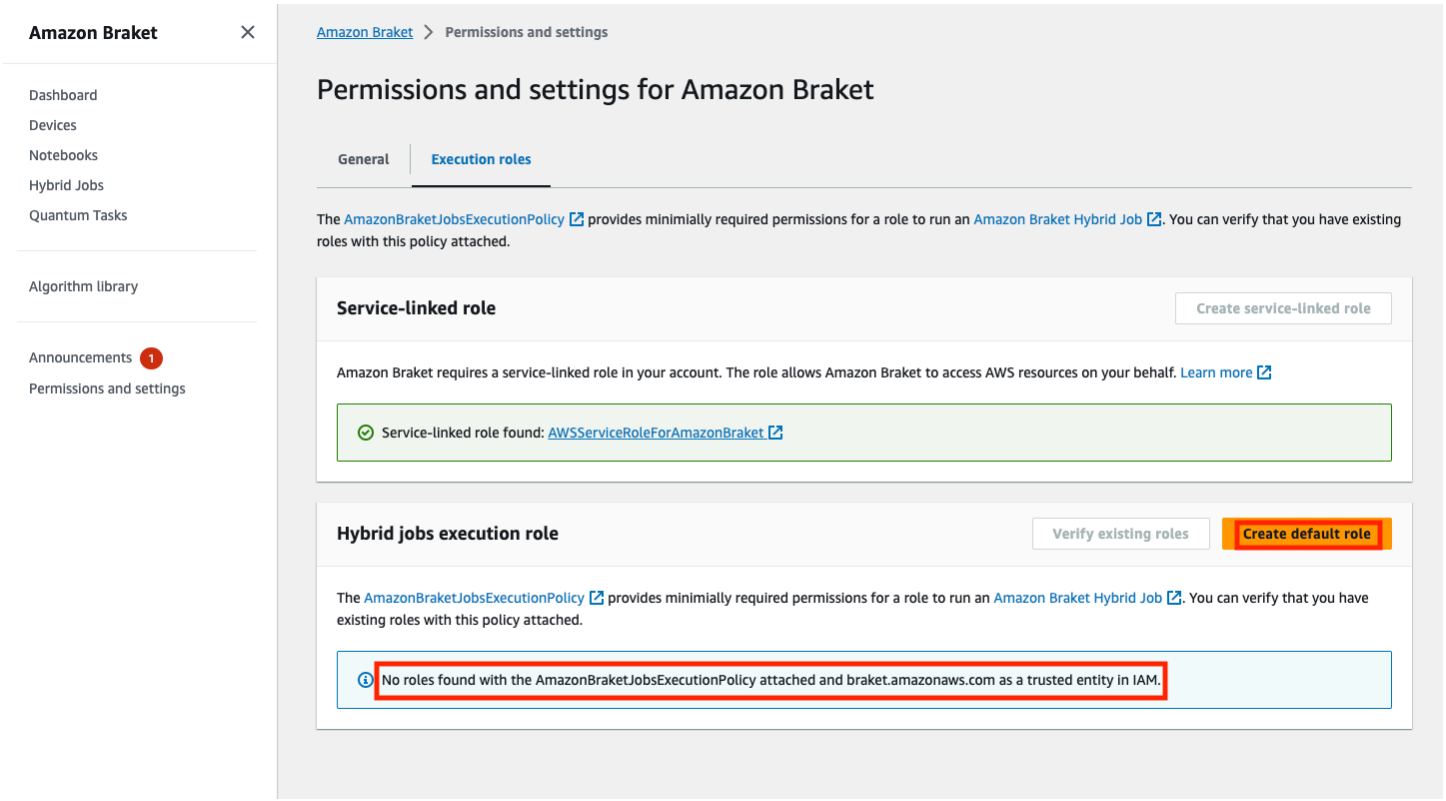
最初のハイブリッドジョブを実行する前に、このタスクを続行するのに十分なアクセス許可があることを確認する必要があります。適切なアクセス許可があることを確認するには、Braket コンソールの左側にあるメニューから[Permissions] (アクセス許可) を選択します。Amazon Braket のアクセス許可管理ページでは、既存のロールの 1 つにハイブリッドジョブを実行するのに十分なアクセス許可があるかどうかを確認するのに便利です。また、ハイブリッドジョブを実行するために使用できるデフォルトのロールがまだない場合は、そのロールの作成をガイドします。



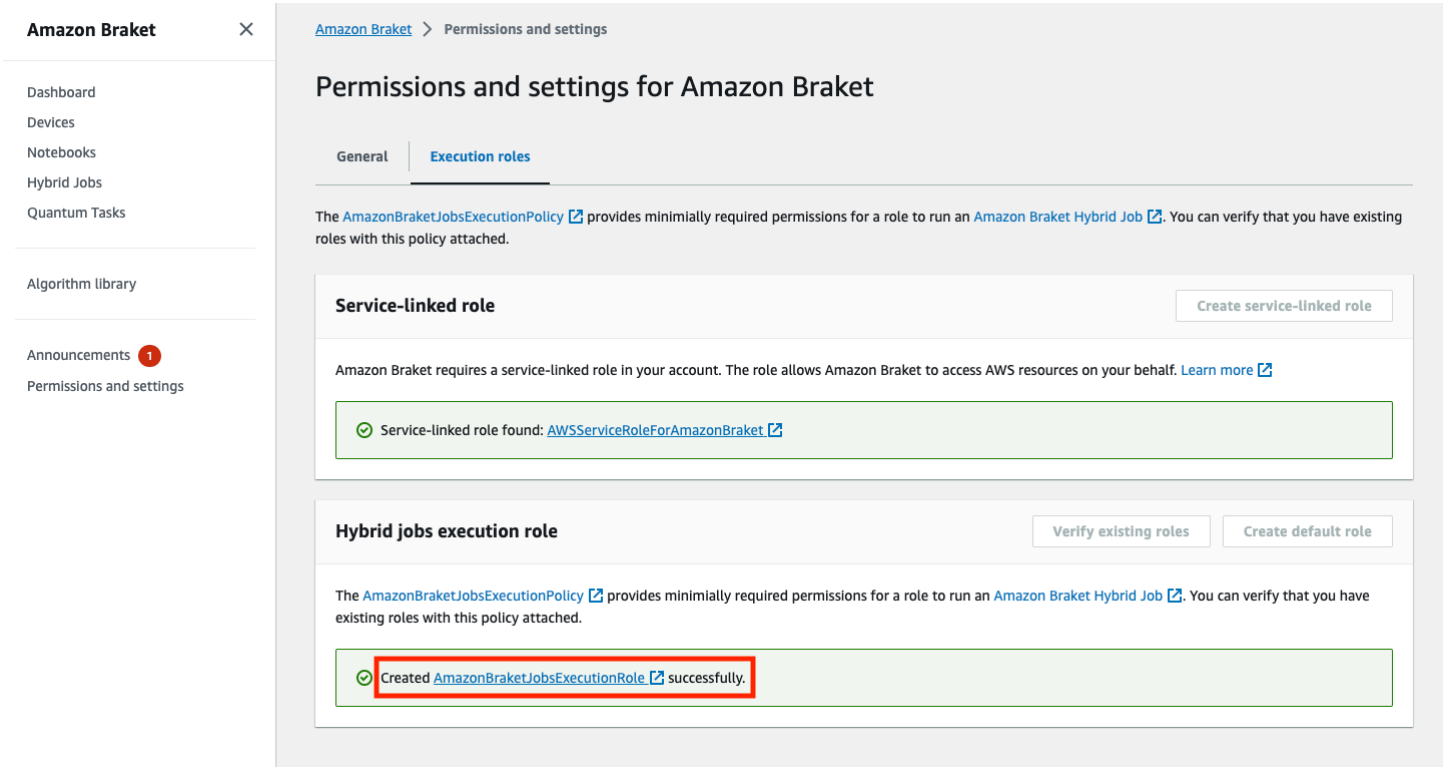
ハイブリッドジョブを実行するのに十分なアクセス許可を持つロールがあることを確認するには、既存のロールの検証ボタンを選択します。使用すると、ロールが見つかったというメッセージが表示されます。ロールの名前とそのロール ARN を表示するには、[Show roles] (ロールを表示する) ボタンを選択します。



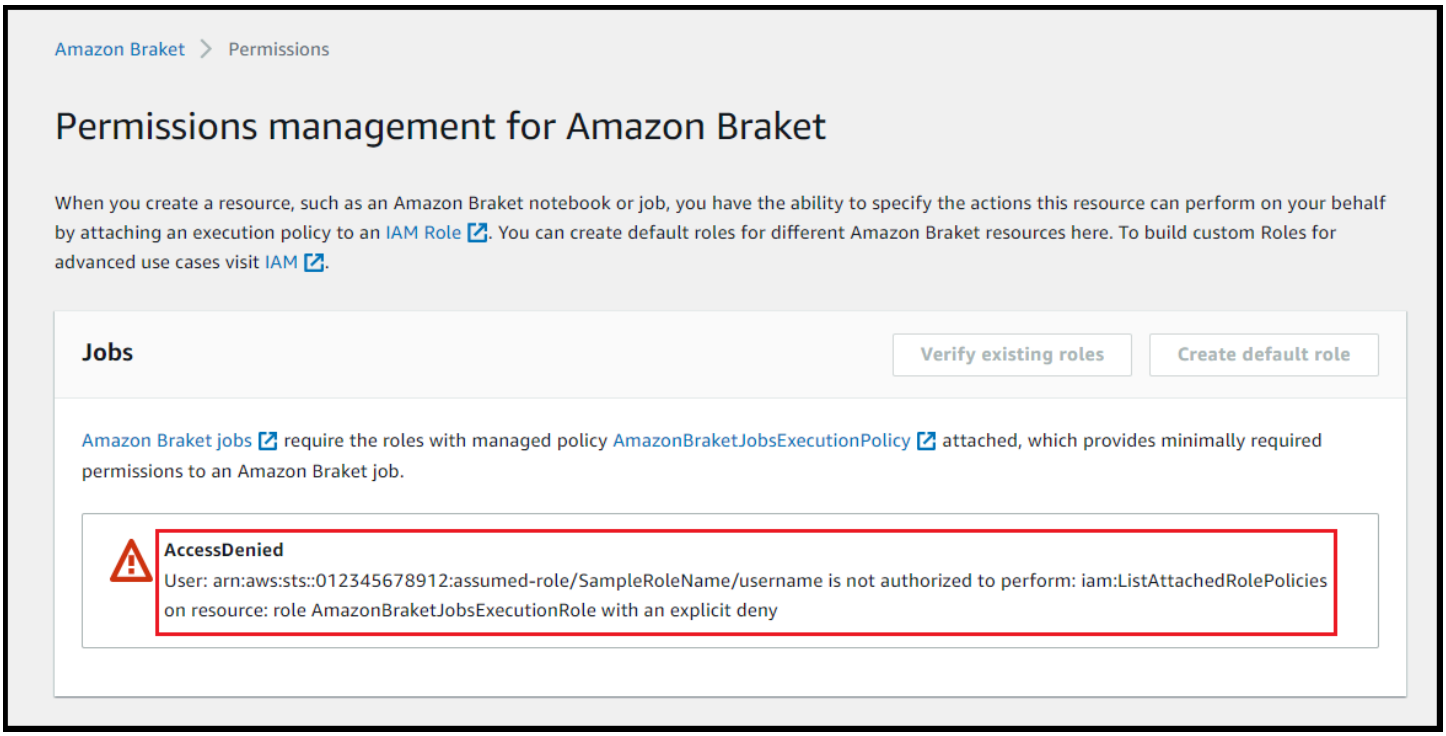
ハイブリッドジョブを実行するのに十分なアクセス許可を持つロールがない場合は、そのようなロールが見つからなかったというメッセージが表示されます。[Create default role] (デフォルトのロールの作成) ボタンを選択して、十分な権限を持つロールを取得します。



ロールが正常に作成された場合は、これを確認するメッセージが表示されます。

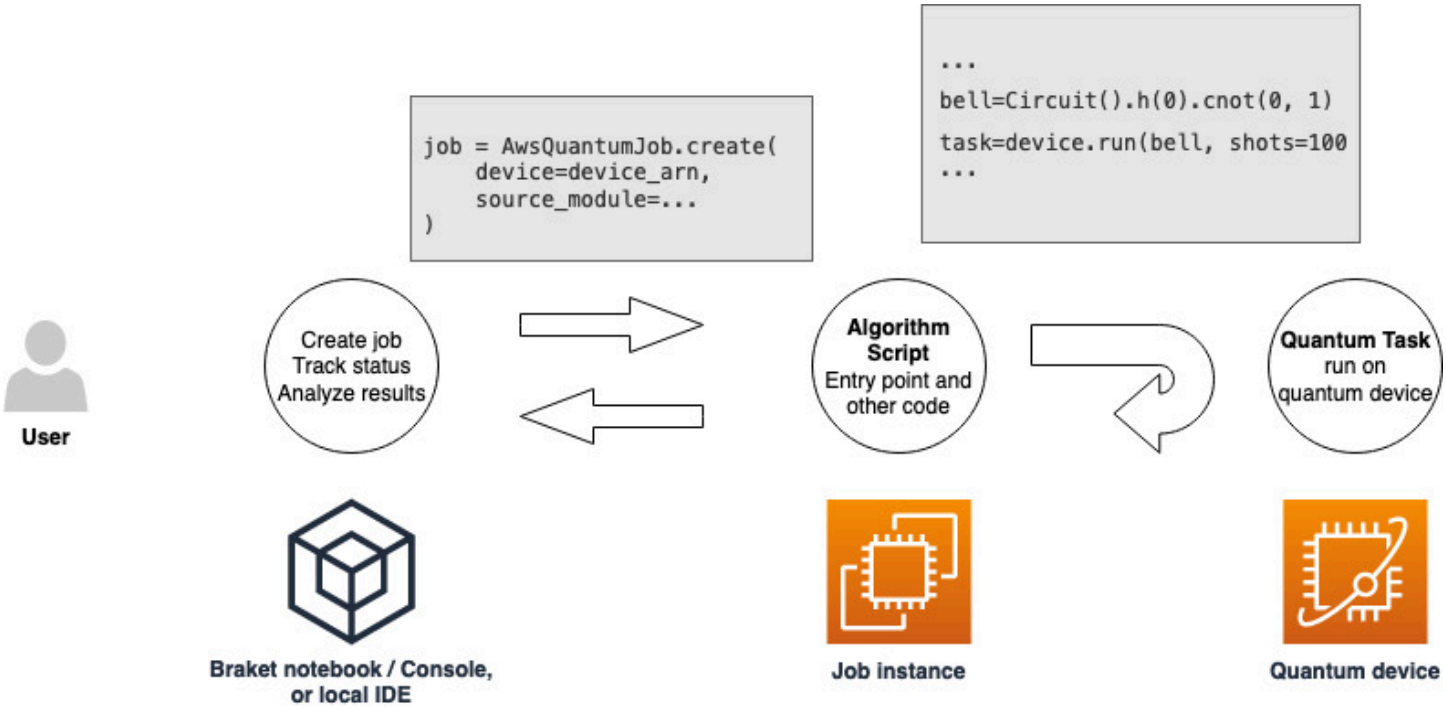


この問い合わせを行う権限がない場合、アクセスは拒否されます。この場合、内部 AWS 管理者に連絡してください。



を作成して実行する

ハイブリッドジョブを実行するアクセス許可を持つロールを取得したら、続行する準備が整います。最初の Braket ハイブリッドジョブのキー部分は、アルゴリズムスクリプト です。実行するアルゴリズムを定義し、アルゴリズムの一部である古典的な論理と量子タスクが含まれています。アルゴリズムスクリプトに加えて、他の依存関係ファイルを指定することもできます。アルゴリズムスクリプトとその依存関係は、ソースモジュールと呼ばれます。エントリポイントは、ハイブリッドジョブの開始時にソースモジュールで実行する最初のファイルまたは関数を定義します。



まず、5 つのベル状態を作成し、対応する測定結果を出力するアルゴリズムスクリプトの基本的な例を考えてみましょう。

```
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!")
```

このファイルをalgorithm_script.py という名前で、Braket ノートブックまたはローカル環境の現在の作業ディレクトリに保存します。algorithm_script.py ファイルには計画されたエントリーポイントとして `start_here()` が含まれます。

次に、algorithm_script.py ファイルと同じディレクトリに Python ファイルまたは Python ノートブックを作成します。このスクリプトはハイブリッドジョブを開始し、関心のあるステータスや主要な結果の印刷などの非同期処理を処理します。少なくとも、このスクリプトはハイブリッドジョブスクリプトとプライマリデバイスを指定する必要があります。

> **Note**
>
> Braket ノートブックを作成する方法、または algorithm_script.py ファイルなどのファイルをノートブックと同じディレクトリにアップロードする方法の詳細については、Amazon Braket Python SDK を使用して最初の回路を実行する」を参照してください。

この基本的な最初のケースでは、シミュレーターをターゲットにします。ターゲットとする量子デバイス、シミュレーター、または実際の量子処理ユニット (QPU) のタイプにかかわらず、次のスクリプト`device`で で指定したデバイスがハイブリッドジョブのスケジュールに使用され、アルゴリズムスクリプトが環境変数 として使用できます`AMZN_BRAKET_DEVICE_ARN`。

> **Note**
>
> ハイブリッドジョブ AWS リージョン の で使用できるデバイスのみを使用できます。Amazon Braket SDK はこの を自動選択します AWS リージョン。例えば、us-east-1 のハイブリッドジョブではIonQ、、SV1、DM1、および TN1 デバイスを使用できますが、Rigettiデバイスは使用できません。

シミュレーターの代わりに量子コンピュータを選択した場合、Braket はハイブリッドジョブがすべての量子タスクを優先アクセスで実行するようにスケジュールします。

```
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
)
```

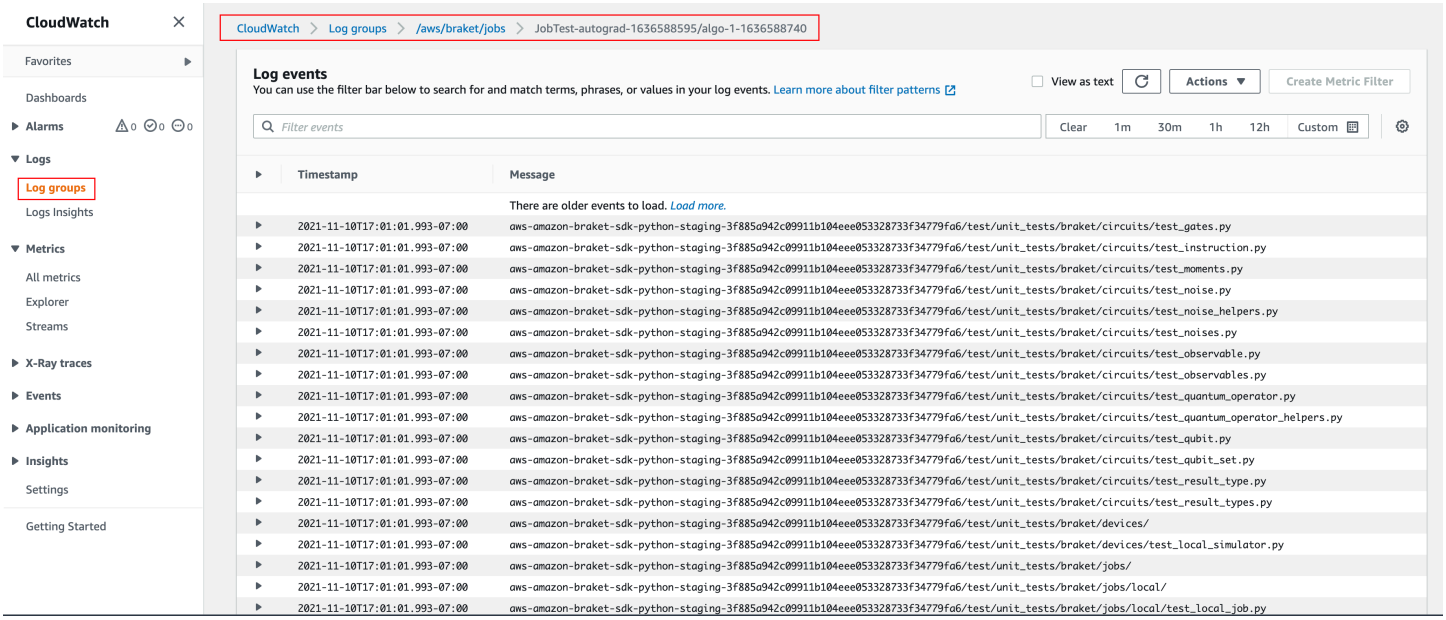
パラメータ wait_until_complete=True は、冗長モードを設定して、ジョブが実行中に実際のジョブからの出力を出力するようにします。次の例のような出力が表示されます。

```
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
```

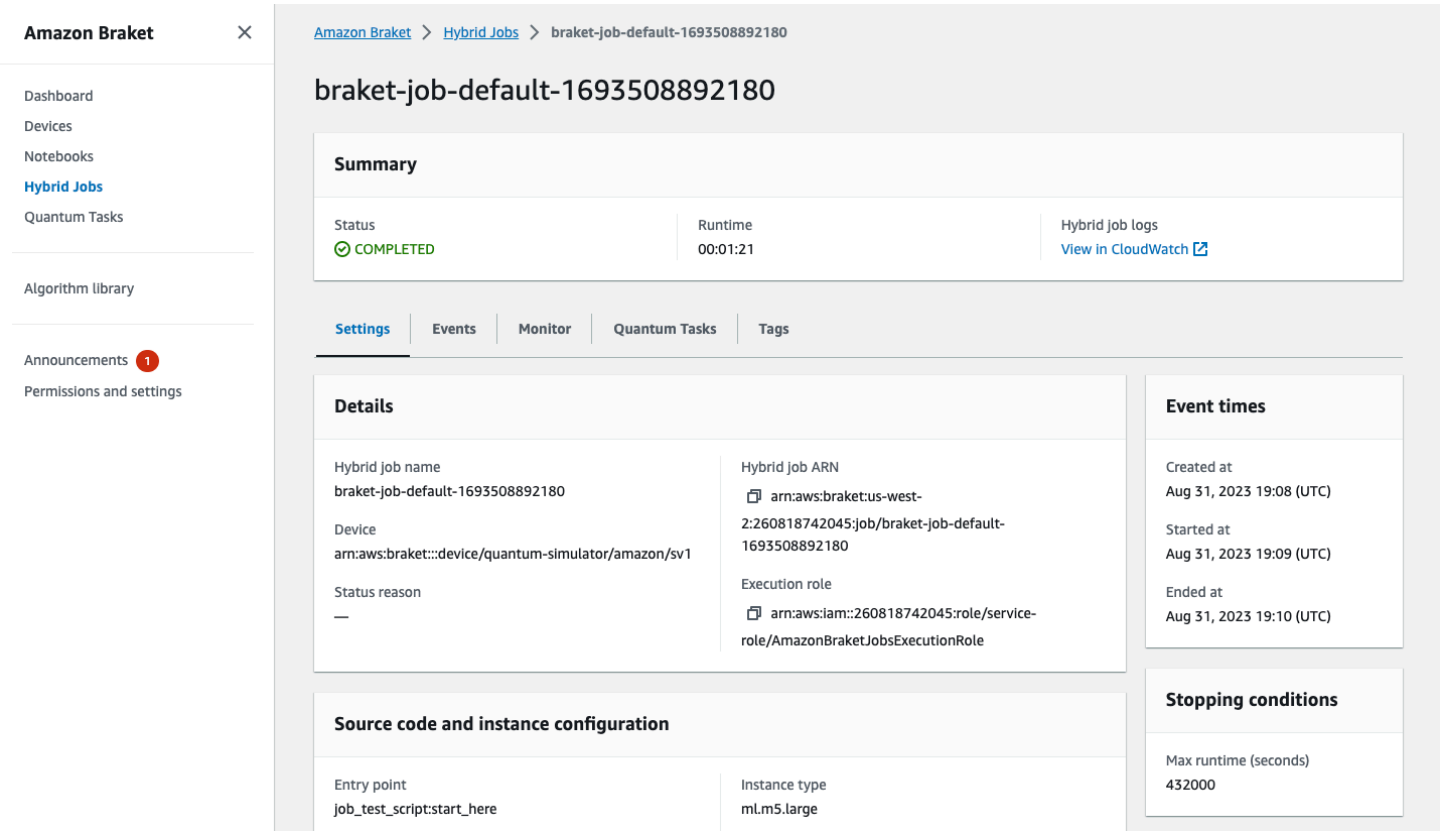
> **Note**
>
> JobAwsQuantum.create メソッドでカスタムメイドモジュールを使用するには、その場所 (ローカルディレクトリまたはファイルへのパス、または tar.gz ファイルの S3 URI) を渡すこともできます。実際の例については、「Amazon Braket サンプル Github リポジトリ」の「ハイブリッドジョブフォルダの Parallelize_training_for_"L.ipynb ファイル」を参照してください。 Amazon Braket

結果をモニタリングする

または、Amazon からのログ出力にアクセスすることもできます CloudWatch。これを行うには、ジョブの詳細ページの左側のメニューにあるロググループタブに移動し、ロググループ を選択しaws/braket/jobs、ジョブ名を含むログストリームを選択します。上記の例で、これは braket-job-default-1631915042705/algo-1-1631915190 です。



ハイブリッドジョブページを選択し、設定 を選択して、コンソールでハイブリッドジョブのステータスを表示することもできます。



ハイブリッドジョブは、実行中に Amazon S3 にアーティファクトを生成します。デフォルトの S3 バケット名は amazon-braket-<region>-<accountid> で、コンテンツはjobs/<jobname>/<timestamp> ディレクトリにあります。Braket Python SDK でハイブリッドジョブを作成するcode_locationときに別の を指定することで、これらのアーティファクトが保存される S3 の場所を設定できます。

> **Note**
>
> この S3 バケットは、ジョブスクリプト AWS リージョン と同じ に配置する必要があります。

jobs/<jobname>/<timestamp> ディレクトリには、 model.tar.gz ファイル内のエントリポイントスクリプトからの出力を含むサブフォルダが含まれています。また、 という名前のディレクトリscriptには、 source.tar.gz ファイルにアルゴリズムスクリプトアーティファクトが含まれています。実際の量子タスクの結果は、jobs/<jobname>/tasks という名前のディレクトリにあります。

入力、出力、環境変数、およびヘルパー関数

完全なアルゴリズムスクリプトを構成する ファイルに加えて、ハイブリッドジョブには追加の入力と出力を含めることができます。ハイブリッドジョブが開始されると、AmazonBraket はハイブリッドジョブの作成の一部として提供された入力を、アルゴリズムスクリプトを実行するコンテナにコピーします。ハイブリッドジョブが完了すると、アルゴリズム中に定義されたすべての出力が、指定された Amazon S3 の場所にコピーされます。

> **Note**
>
> アルゴリズムメトリクスはリアルタイムで報告されますので、この出力手順に従わないでください。

Amazon Braket には、コンテナの入出力とのやり取りを簡素化するための環境変数とヘルパー関数もいくつか用意されています。

このセクションでは、AmazonBraket Python SDK によって提供される `AwsQuantumJob.create`関数の主要な概念と、コンテナファイル構造へのマッピングについて説明します。

このセクションの内容:

入力

入力データ : 入力データは、 `input_data`引数でディクショナリとして設定された入力データファイルを指定することで、ハイブリッドアルゴリズムに提供できます。ユーザーは SDK の `AwsQuantumJob.create`関数内で引`input_data`数を定義します。これにより、環境変数 によって指定された場所にあるコンテナファイルシステムに入力データがコピーされます`"AMZN_BRAKET_INPUT_DIR"`。ハイブリッドアルゴリズムでの入力データの使用方法のいくつかの例については、Amazon Braket Hybrid Jobs を使用した QAOA」と PennyLaneAmazon Braket Hybrid Jobs Jupyter Notebooks での量子機械学習」を参照してください。

> **Note**
>
> 入力データが大きい (>1GB) 場合、ハイブリッドジョブが送信されるまでに長い待機時間がかかります。これは、ローカル入力データが最初に S3 バケットにアップロードされ、次に S3 パスがハイブリッドジョブリクエストに追加され、最後にハイブリッドジョブリクエストが Braket サービスに送信されるためです。

ハイパーパラメータ: 通り過ぎたらhyperparametersの場合、環境変数で使用できます。"AMZN_BRAKET_HP_FILE"。

> **Note**
>
> ハイパーパラメータと入力データを作成し、この情報をハイブリッドジョブスクリプトに渡す方法の詳細については、「ハイパーパラメータを使用する」セクションとこの github ページを参照してください。

チェックポイント : 新しいハイブリッドジョブで使用するjob-arnチェックポイントの を指定するには、 copy_checkpoints_from_job コマンドを使用します。このコマンドは、チェックポイントデータを新しいハイブリッドジョブcheckpoint_configs3Uriの にコピーし、ジョブの実行AMZN_BRAKET_CHECKPOINT_DIR中に環境変数によって指定されたパスで使用できるようにします。デフォルトは です。つまりNone、別のハイブリッドジョブのチェックポイントデータは新しいハイブリッドジョブでは使用されません。

出力

量子タスク : 量子タスクの結果は S3 の場所 に保存されますs3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks。

ジョブの結果: アルゴリズムスクリプトが環境変数 "AMZN_BRAKET_JOB_RESULTS_DIR" で指定されたディレクトリに保存するものはすべて、output_data_config で指定された S3 の場所にコピーされます。この値を指定していない場合は、デフォルトで s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data になります。SDK ヘルパー関数 **save_job_result** を提供しています。この関数を使用すると、アルゴリズムスクリプトから呼び出されたときに結果をディクショナリの形式で簡単に保存できます。

チェックポイント: チェックポイントを使用する場合は、環境変数 "AMZN_BRAKET_CHECKPOINT_DIR" で指定されたディレクトリに保存できます。代わりに、SDK ヘルパー関数 save_job_checkpoint を使用することもできます。

アルゴリズムメトリクス : ハイブリッドジョブの実行中に Amazon に出力 CloudWatch され、AmazonBraket コンソールにリアルタイムで表示されるアルゴリズムスクリプトの一部としてアルゴリズムメトリクスを定義できます。アルゴリズムメトリクスの使用方法の例については、Amazon Braket Hybrid Jobs を使用して QAOA アルゴリズムを実行する」を参照してください。

環境変数

Amazon Braket には、コンテナの入力と出力とのやり取りを簡素化するための環境変数がいくつか用意されています。次のコードは、Braket が使用する環境変数を一覧表示します。

```
# 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"]
```

ヘルパー関数

Amazon Braket には、コンテナの入出力とのやり取りを簡素化するためのヘルパー関数がいくつか用意されています。これらのヘルパー関数は、ハイブリッドジョブの実行に使用されるアルゴリズムスクリプト内から呼び出されます。次の例は、その使用方法を示しています。

```
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
```

ジョブ結果の保存

アルゴリズムスクリプトによって生成された結果を保存して、ハイブリッドジョブスクリプトのハイブリッドジョブオブジェクトと Amazon S3 の出力フォルダ (model.tar.gz という名前の tar 圧縮ファイル) から結果を利用できるようにします。

出力は Object Notation (JSON) JavaScript 形式を使用してファイルに保存する必要があります。numpy 配列の場合のように、データをテキストに簡単にシリアル化できない場合は、ピクルドデータ形式を使用してシリアル化するオプションを渡すことができます。詳細については、braket.jobs.data_persistence モジュールを参照してください。

ハイブリッドジョブの結果を保存するには、#ADD でコメントされた次の行をアルゴリズムスクリプトに追加します。

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

次に、#ADD でコメントされた行 **print(job.result())** を追加することで、ジョブスクリプトのジョブの結果を表示できます。

```
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
```

この例では、wait_until_complete=True を削除して冗長出力を抑制します。デバッグ用に再度追加できます。このハイブリッドジョブを実行すると、識別子 と が出力されjob-arn、ハイブリッドジョブの状態が 10 秒ごとにハイブリッドジョブの状態が になるまで続きます。その後COMPLETED、ベル回路の結果が表示されます。次の例を参照してください。

```
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}]}
```

チェックポイントを使用してハイブリッドジョブを保存および再起動する

チェックポイントを使用して、ハイブリッドジョブの中間反復を保存できます。前のセクションのアルゴリズムスクリプトの例では、#ADD でコメントされた次の行を追加して、チェックポイントファイルを作成します。

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

ハイブリッドジョブを実行すると、デフォルト/opt/jobs/checkpointsパスを持つチェックポイントディレクトリのハイブリッドジョブアーティファクトにファイル <jobname>-checkpoint-1.json が作成されます。このデフォルトパスを変更しない限り、ハイブリッドジョブスクリプトは変更されません。

以前のハイブリッドジョブによって生成されたチェックポイントからハイブリッドジョブをロードする場合、アルゴリズムスクリプトは を使用しますfrom braket.jobs import load_job_checkpoint。アルゴリズムスクリプトにロードするロジックは次のとおりです。

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

このチェックポイントをロードした後、checkpoint-1 にロードされたコンテンツに基づいてロジックを続行できます。

> **Note**
>
> checkpoint_file_suffix は、チェックポイントの作成時に以前に指定した接尾辞と一致する必要があります。

オーケストレーションスクリプトでは、前のハイブリッドジョブjob-arnの を指定し、#ADD でコメントされた行を指定する必要があります。

```
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
    )
```

アルゴリズムスクリプトの環境を定義する

Amazon Braket は、アルゴリズムスクリプトのコンテナによって定義された 3 つの環境をサポートしています。

- ベースコンテナ (が指定されていない場合`image_uri`はデフォルト)
- Tensorflow と を持つコンテナ PennyLane
- PyTorch および を含むコンテナ PennyLane

次の表は、コンテナとそれらに含まれるライブラリの詳細を示しています。

Amazon Braket コンテナ

タイプ	PennyLane で TensorFlow	PennyLane で PyTorch	ペンニラネ
基本	292282985366.dkr.ecr.us-east-1.amazonaws.com/amazon-braket-tensorflow-jobs:latest	292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:latest	292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:latest
継承ライブラリ	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
追加のライブラリ	• amazon-braket-default-simulator • amazon-braket-PennyLane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras	• amazon-braket-default-simulator • amazon-braket-PennyLane-plugin • amazon-braket-schemas • amazon-braket-sdk • ipykernel • keras	• amazon-braket-default-simulator • amazon-braket-PennyLane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • boto3

タイプ	PennyLane で TensorFlow	PennyLane で PyTorch	ペンニラネ
	• matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-Lightning-gpu • cuQuantum	• matplotlib • networkx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-Lightning-gpu • cuQuantum	• ipykernel • matplotlib • networkx • numpy • openbabel • pandas • PennyLane • protobuf • psi4 • rsa • scipy

オープンソースのコンテナ定義は、aws/amazon-braket-containers で表示およびアクセスできます。ユースケースに一致するコンテナを選択します。コンテナは、ハイブリッドジョブを呼び出す AWS リージョン 元の にある必要があります。ハイブリッドジョブの作成時にコンテナイメージを指定するには、次の 3 つの引数のいずれかをハイブリッドジョブスクリプトの`create(…)` 呼び出しに追加します。Amazon Braket コンテナにはインターネット接続があるため、ランタイムに選択したコンテナに追加の依存関係をインストールできます (起動時またはランタイムのコストがかかります)。次の例は、us-west-2 リージョン用です。

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

は、Braket SDK の Amazon `retrieve_image()`関数を使用して取得`image-uris`することもできます。次の例は、us-west-2 からそれらを取得する方法を示しています AWS リージョン。

```
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")
```

ハイパーパラメータの使用

ハイブリッドジョブを作成するときに、学習速度やステップサイズなど、アルゴリズムに必要なハイパーパラメータを定義できます。ハイパーパラメータ値は通常、アルゴリズムのさまざまな側面を制御するために使用され、多くの場合、アルゴリズムのパフォーマンスを最適化するために調整できます。Braket ハイブリッドジョブでハイパーパラメータを使用するには、それらの名前と値をディクショナリとして明示的に指定する必要があります。値は文字列データ型である必要があります。最適な値のセットを検索するときにテストするハイパーパラメーター値を指定します。ハイパーパラメータを使用する最初のステップは、ハイパーパラメータをディクショナリとして設定して定義することです。これは、次のコードで確認できます。

```
#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
}
```

次に、上記のコードスニペットで定義されたハイパーパラメータを渡して、選択したアルゴリズムで次のようになります。

```
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**
>
> 入力データの詳細については、「入力」セクションを参照してください。

ハイパーパラメータは、次のコードを使用してハイブリッドジョブスクリプトにロードされます。

```
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**
>
> 入力データやデバイス ARN などの情報をハイブリッドジョブスクリプトに渡す方法の詳細については、この github ページ を参照してください。

Amazon Braket Hybrid Jobs を使用した QAOA とAmazon Braket Hybrid Jobs PennyLane Amazon Braket のチュートリアルで量子機械学習を学ぶために非常に役立つガイドがいくつかあります。

アルゴリズムスクリプトを実行するようにハイブリッドジョブインスタンスを設定する

アルゴリズムによっては、要件が異なる場合があります。デフォルトでは、AmazonBraket はアルゴリズムスクリプトを`ml.m5.large`インスタンスで実行します。ただし、次のインポート引数と設定引数を使用してハイブリッドジョブを作成するときに、このインスタンスタイプをカスタマイズできます。

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.p3.8xlarge"), # Use NVIDIA Tesla
 V100 instance with 4 GPUs.
    ...
    ),
```

埋め込みシミュレーションを実行していて、デバイス設定でローカルデバイスを指定している場合、instanceCount を指定し、それを 1 つ以上に設定 InstanceConfig することで、 で複数のインスタンスを追加でリクエストできます。上限は 5 です。例えば、次のように 3 つのインスタンスを選択できます。

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

複数のインスタンスを使用する場合は、データ並列機能を使用してハイブリッドジョブを分散することを検討してください。この Braket サンプル を表示する方法の詳細については、次のサンプルノートブックを参照してください。

次の 3 つの表は、標準、コンピューティング最適化、高速コンピューティングインスタンスで使用可能なインスタンスタイプと仕様の一覧です。

> **Note**
>
> Hybrid Jobs のデフォルトのクラシックコンピューティングインスタンスクォータを表示するには、このページを参照してください。

スタンダードインスタンス	vCPU	「メモリ」
ml.m5.large (デフォルト)	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

コンピュート最適化インスタンス	vCPU	「メモリ」
ml.c4.xlarge	4	7.5 GIB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB

コンピュート最適化インスタンス	vCPU	「メモリ」
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.xlarge	4	10.5 GiB
ml.c5n.2xlarge	8	21 GiB
ml.c5d.4xlarge	16	42 GiB
ml.c5d.9xlarge	36	96 GiB
ml.c5d.18xlarge	72	192 GiB

高速コンピューティングインスタンス	vCPU	「メモリ」
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB
ml.g4dn.xlarge	4	16 GiB

高速コンピューティングインスタンス	vCPU	「メモリ」
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 インスタンスは us-west-1 では使用できません。ハイブリッドジョブがリクエストされた ML コンピューティング容量をプロビジョニングできない場合は、別のリージョンを使用します。

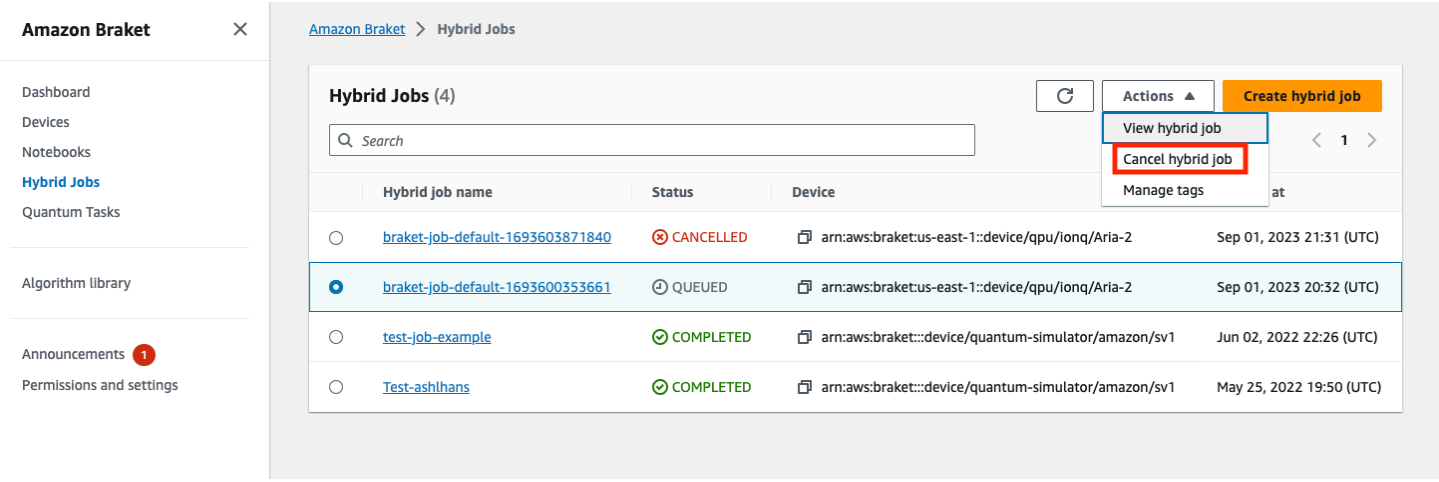
各インスタンスは、30 GB のデータストレージ (SSD) のデフォルト設定を使用します。ただし、ストレージは、`instanceType` を設定するのと同じ方法で調整できます。。次の例では、ストレージの合計を 50 GB に増やす方法を示します。

```
from braket.jobs.config import InstanceConfig

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

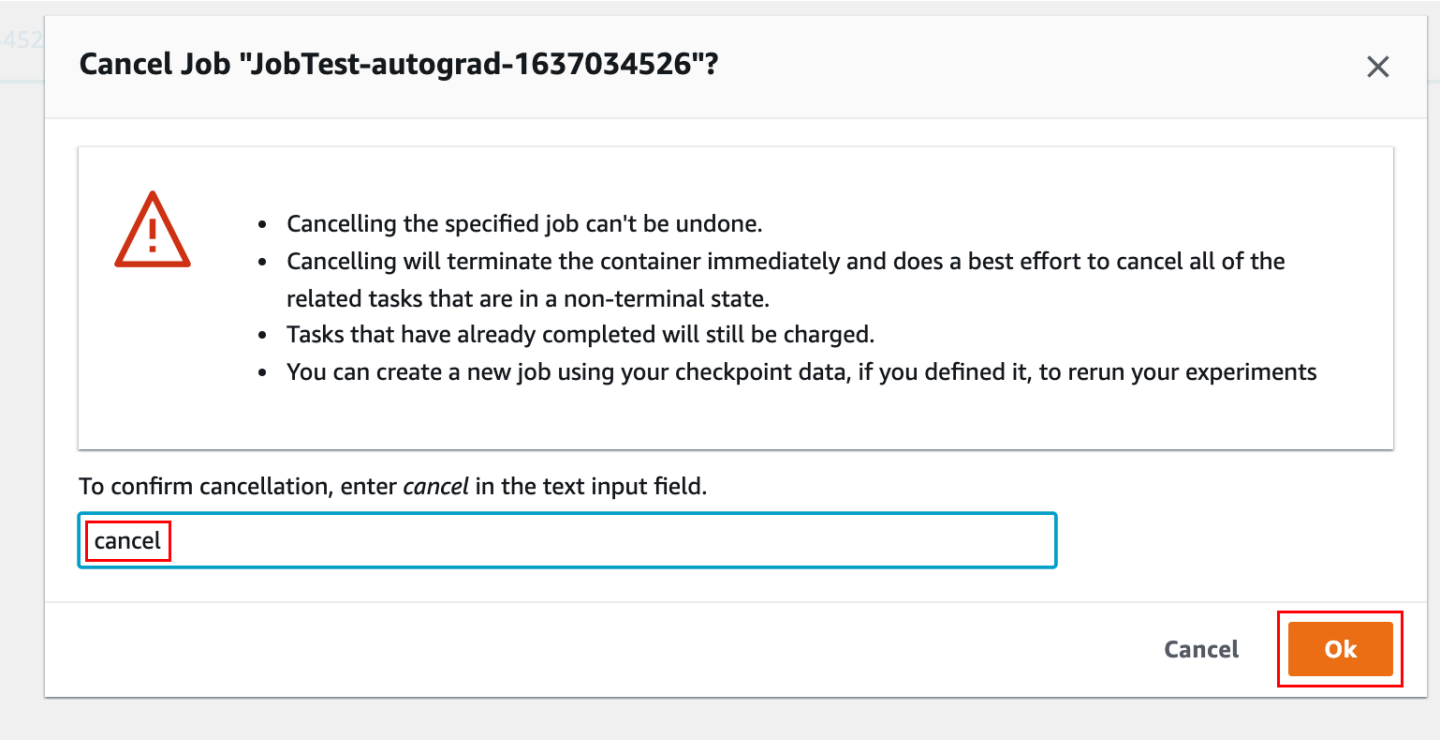
ハイブリッドジョブをキャンセルする

非ターミナル状態のハイブリッドジョブをキャンセルする必要がある場合があります。これは、コンソールまたはコードで行うことができます。

コンソールでハイブリッドジョブをキャンセルするには、ハイブリッドジョブ ページからキャンセルするハイブリッドジョブを選択し、アクション ドロップダウンメニューからハイブリッドジョブをキャンセル を選択します。



キャンセルを確認するには、プロンプトが表示されたら、と入力フィールドにキャンセルと入力し、OK を選択します。



Braket Python SDK のコードを使用してハイブリッドジョブをキャンセルするには、 を使用してハイブリッドジョブjob_arnを識別し、次のコードに示すように、そのジョブで cancel コマンドを呼び出します。

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

cancel このコマンドは、クラシックハイブリッドジョブコンテナを直ちに終了し、非ターミナル状態にある関連する量子タスクをすべてキャンセルするよう最善を尽くします。

パラメトリックコンパイルを使用してハイブリッドジョブを高速化する

Amazon Braket は、特定の QPUs でのパラメトリックコンパイルをサポートしています。これにより、ハイブリッドアルゴリズムの反復ごとに回路を 1 回だけコンパイルすることで、計算コストの高いコンパイルステップに関連するオーバーヘッドを削減できます。これにより、各ステップで回路を再コンパイルする必要がなくなるため、Hybrid Jobs のランタイムが大幅に向上します。パラメータ化された回路を、サポートされているQPUs に Braket Hybrid Job として送信するだけです。長時間実行されるハイブリッドジョブの場合、Braket は回路をコンパイルするときにハードウェアプロバイダーから更新されたキャリブレーションデータを自動的に使用して、最高品質の結果を実現します。

パラメトリック回路を作成するには、まずアルゴリズムスクリプトの入力としてパラメータを指定する必要があります。この例では、小さなパラメトリック回路を使用し、各反復間の従来の処理を無視します。一般的なワークロードでは、多数の回路をバッチで送信し、各反復のパラメータの更新などの従来の処理を実行します。

```
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.")
```

次のジョブスクリプトを使用して、ハイブリッドジョブとして実行するアルゴリズムスクリプトを送信できます。パラメトリックコンパイルをサポートする QPU で Hybrid Job を実行する場合、回路は最初の実行時にのみコンパイルされます。次の実行では、コンパイルされた回路が再利用され、コード行を追加することなく Hybrid Job のランタイムパフォーマンスが向上します。

```
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="algorithm_script.py",
)
```

> **Note**
>
> パラメトリックコンパイルは、 Rigetti Computingおよび からのすべての超電導ゲートベースの QPUs でサポートされています。ただし、 の脈動レベルプログラムを除きOxford Quantum Circuitsます。

Amazon Braket PennyLane で を使用する

ハイブリッドアルゴリズムは、古典命令と量子命令の両方を含むアルゴリズムです。クラシック命令はクラシックハードウェア (EC2 インスタンスまたはラップトップ) で実行され、量子命令はシミュレーターまたは量子コンピュータで実行されます。Hybrid Jobs 機能を使用してハイブリッドアルゴリズムを実行することをお勧めします。詳細については、Amazon Braket Jobs を使用するタイミング」を参照してください。

Amazon Braket を使用すると、Braket PennyLane プラグイン 、または Amazon Braket Python SDK Amazon とサンプルノートブックリポジトリを使用して、ハイブリッド量子アルゴリズムをセットアップして実行できます。 AmazonBraket サンプルノートブックは SDK に基づいており、PennyLane プラグインなしで特定のハイブリッドアルゴリズムをセットアップして実行できます。ただし、よりリッチなエクスペリエンス PennyLane を提供するため、 をお勧めします。

ハイブリッド量子アルゴリズムについて

ハイブリッド量子アルゴリズムは、今日の業界にとって重要です。なぜなら、近代的な量子コンピューティングデバイスは一般的にノイズを生成し、したがって エラーが発生するからです。計算に追加される量子ゲートごとにノイズが増える可能性が高くなるため、長時間実行されるアルゴリズムはノイズに圧倒され、計算が失敗する可能性があります。

Shor の （量子フェーズ推定の例) や Grover の (Grover の例) などの純粋な量子アルゴリズムには、数千または数百万のオペレーションが必要です。このため、これらは既存の量子デバイスでは実行できない可能性があります。これらの量子デバイスは、一般にノイズの多い中間スケール量子(NISQ) デバイスと呼ばれます。

ハイブリッド量子アルゴリズムでは、特に古典的なアルゴリズムにおける特定の計算を高速化するために、量子処理装置 (QPU) は古典的 CPU のコプロセッサとして機能します。今日のデバイスの機能の手の届く範囲内で、回路の実行がはるかに短くなります。

を使用した Amazon Braket PennyLane

Amazon Braket はPennyLane、量子識別可能なプログラミングの概念に基づいて構築されたオープンソースのソフトウェアフレームワークである をサポートしています。このフレームワークを使用して、量子分子、量子機械学習、最適化における計算上の問題に対する解決策を見つけるためにニューラルネットワークをトレーニングするのと同じ方法で量子回路をトレーニングできます。

この PennyLane ライブラリは、 PyTorch や などの使い慣れた機械学習ツールへのインターフェイスを提供し TensorFlow、量子回路のトレーニングを迅速かつ直感的に行います。

- PennyLane ライブラリ -- PennyLane は Amazon Braket ノートブックにプリインストールされています。から Amazon Braket デバイスにアクセスするには PennyLane、ノートブックを開き、次のコマンドを使用して PennyLane ライブラリをインポートします。

```
import pennylane as qml
```

チュートリアルノートブックで、すぐに開始できます。または、選択した IDE から PennyLane Amazon Braket で を使用することもできます。

- Amazon Braket PennyLane プラグイン — 独自の IDE を使用するには、AmazonBraket PennyLane プラグインを手動でインストールできます。プラグインは Amazon Braket Python SDK PennyLane に接続するため、AmazonBraket デバイスで PennyLane で回路を実行できます。 PennyLane プラグインをインストールするには、次のコマンドを使用します。

```
pip install amazon-braket-pennylane-plugin
```

次の例は、 で Amazon Braket デバイスへのアクセスを設定する方法を示しています 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)
```

チュートリアルの例と の詳細については PennyLane、Amazon Braket サンプルリポジトリ」を参照してください。

Amazon Braket PennyLane プラグインを使用すると、AmazonBraket QPU と埋め込みシミュレーターデバイスを 1 行のコード PennyLane で切り替えることができます。2 つの Amazon Braket 量子デバイスが で動作します PennyLane。

- `braket.aws.qubit` QPUs やシミュレーターなど、AmazonBraket サービスの量子デバイスで実行するための
- `braket.local.qubit` Amazon Braket SDK のローカルシミュレーターで実行するための

Amazon Braket PennyLane プラグインはオープンソースです。PennyLane プラグイン GitHub リポジトリ からインストールできます。

の詳細については PennyLane、 PennyLane ウェブサイトのドキュメントを参照してください。

Amazon Braket のハイブリッドアルゴリズムのサンプルノートブック

Amazon Braket には、ハイブリッドアルゴリズムの実行に PennyLane プラグインに依存しないさまざまなサンプルノートブックが用意されています。量子近似最適化アルゴリズム (QAOA) や変分量子固有値ソルバー (VQE) などの変分法を説明する Amazon Braket ハイブリッドサンプルノートブックのいずれからでも開始できます。

Amazon Braket サンプルノートブックは Amazon Braket Python SDK に依存しています。SDK は、AmazonBraket を介して量子コンピューティングハードウェアデバイスとやり取りするためのフレームワークを提供します。これは、ハイブリッドワークフローの量子部分を支援するように設計されたオープンソースライブラリです。

Amazon Braket は、サンプルノートブック でさらに詳しく調べることができます。

PennyLane シミュレーターが埋め込まれたハイブリッドアルゴリズム

Amazon Braket Hybrid Jobs には、 の高性能 CPU および GPU ベースの組み込みシミュレーターが付属するようになりましたPennyLane。この組み込みシミュレーターファミリーはハイブリッドジョブコンテナに直接埋め込むことができ、高速ステートベクトル`lightning.qubit`シミュレーター、NVIDIA の cuQuantum ライブラリ を使用して高速化された`lightning.gpu`シミュレーターなどが含まれます。これらの組み込みシミュレーターは、結合分化方法 などの高度な方法の利点を享受できる量子機械学習などのバリエーションアルゴリズムに最適です。これらの埋め込みシミュレーターは、1 つ以上の CPU または GPU インスタンスで実行できます。

Hybrid Jobs では、従来のコプロセッサと QPU、 などの Amazon Braket オンデマンドシミュレーター、または の埋め込みシミュレーターを直接使用してSV1、バリエーションアルゴリズムコードを実行できるようになりました PennyLane。

埋め込みシミュレーターは Hybrid Jobs コンテナで既に利用可能です。メインの Python 関数を デコレータで`@hybrid_job`デコレートするだけで済みます。シミュレーターを使用するには PennyLane `lightning.gpu`、次のコードスニペット`InstanceConfig`に示すように、 で GPU インスタンスも指定する必要があります。

```
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)
    ...
```

Hybrid Jobs で PennyLane 埋め込みシミュレーターの使用を開始するには、サンプルノートブックを参照してください。

Amazon Braket シミュレータ PennyLane ーによる の結合勾配

Amazon Braket のPennyLaneプラグインを使用すると、ローカル状態ベクトルシミュレーターまたは SV1 で実行するときに、結合区別方法を使用して勾配を計算できます。 Amazon Braket

注： 結合の区別方法を使用するには、 ではなくqnode、 diff_method='device'で を指定する必要がありますdiff_method='adjoint'。次の例を参照してください。

```
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**
>
> 現在、 PennyLaneは QAOA ハミルトニアンのグループ化インデックスを計算し、それらを使用してハミルトニアンを複数の期待値に分割します。から QAOA を実行するときに SV1 の結合差別化機能を使用する場合はPennyLane、次のようにグループ化インデックスを削除してコストハミルトニアンを再構築する必要があります。
> `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False) cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Amazon Braket Hybrid Jobs と PennyLane を使用して QAOA アルゴリズムを実行する

このセクションでは、学習した内容を使用して、パラメトリックコンパイル PennyLane で を使用して実際のハイブリッドプログラムを作成します。アルゴリズムスクリプトを使用して、量子近似最適化アルゴリズム (QAOA) 問題に対処します。プログラムは、従来の Max Cut 最適化問題に対応するコスト関数を作成し、パラメータ化された量子回路を指定し、単純な勾配降下法を使用してパラメータを最適化して、コスト関数を最小限に抑えるようにします。この例では、わかりやすくするためにアルゴリズムスクリプトで問題グラフを生成しますが、より一般的なユースケースでは、ベストプラクティスとして、入力データ設定の専用チャネルを介して問題仕様を提供することです。フラグのparametrize_differentiableデフォルトは Trueであるため、サポートされている QPUs でのパラメトリックコンパイルにより、ランタイムパフォーマンスが自動的に向上するメリットが得られます。

```
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**
>
> パラメトリックコンパイルは、 Rigetti Computingおよび からのすべての超電導ゲートベースの QPUs でサポートされています。ただし、 の脈動レベルプログラムを除きOxford Quantum Circuitsます。

の組み込みシミュレーターを使用してハイブリッドワークロードを高速化する PennyLane

Amazon Braket Hybrid Jobs PennyLane で の埋め込みシミュレーターを使用してハイブリッドワークロードを実行する方法を見てみましょう。Pennylane の GPU ベースの埋め込みシミュレーター はlightning.gpu、Nvidia cuQuantum ライブラリを使用して回路シミュレーションを高速化します。組み込み GPU シミュレーターは、ユーザーがすぐに使用できるすべての Braket ジョブコンテナに事前設定されています。このページでは、 を使用してハイブリッドワークロードlightning.gpuを高速化する方法を示します。

量子近似最適化アルゴリズムワークロード**lightning.gpu**での の使用

このノートブックの量子近似最適化アルゴリズム (QAOA) の例を考えてみましょう。埋め込みシミュレーターを選択するには、 引device数を 形式の文字列として指定します"local:<provider>/<simulator_name>"。例えば、 "local:pennylane/lightning.gpu"に を設定しますlightning.gpu。起動時にハイブリッドジョブに渡すデバイス文字列は、環境変数 としてジョブに渡されます"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)
```

このページでは、2 つの埋め込み PennyLane 状態ベクトルシミュレーター lightning.qubit (CPU ベース) と lightning.gpu (GPU ベース) を比較します。さまざまな勾配を計算するには、シミュレーターにいくつかのカスタムゲート分解を提供する必要があります。

これで、ハイブリッドジョブ起動スクリプトを準備する準備が整いました。QAOA アルゴリズムは、 m5.2xlargeと の 2 つのインスタンスタイプを使用して実行しますp3.2xlarge。m5.2xlarge インスタンスタイプは、標準のデベロッパーラップトップと同等です。p3.2xlarge は、16GB のメモリを備えた単一の NVIDIA Volta GPU を備えた高速コンピューティングインスタンスです。

すべてのハイブリッドジョブhyperparametersの は同じになります。さまざまなインスタンスとシミュレーターを試すために必要なのは、次のように 2 行を変更することだけです。

```
# 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')
```

または

```
# 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**
>
> GPU ベースのインスタンスを使用して instance_configとして指定し、埋め込み CPU ベースのシミュレーター (lightning.qubit) deviceとして を選択した場合、GPU は使用されません。GPU をターゲットにする場合は、必ず組み込み GPU ベースのシミュレーターを使用してください。

まず、2 つのハイブリッドジョブを作成し、18 個の頂点を持つグラフで QAOA を使用して Max-Cut を解決できます。これは 18 量子ビット回路に変換されます。比較的小さく、ラップトップやm5.2xlargeインスタンスですばやく実行できます。

```
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,
)
```

m5.2xlarge インスタンスの平均反復時間は約 25 秒で、p3.2xlargeインスタンスの平均反復時間は約 12 秒です。この 18 量子ビットのワークフローでは、GPU インスタンスによって 2 倍の高速化が得られます。Amazon Braket Hybrid Jobs の料金ページを見ると、インスタンスの 1 分あたりのコストは 0.00768 USD で、p3.2xlargeインスタンスのコストは 0.06375 USD であることがわかります。 m5.2xlargeここで行ったように、合計 5 回のイテレーションを実行するには、CPU インスタンスを使用して 0.016 USD、または GPU インスタンスを使用して 0.06375 USD の費用がかかります。どちらもかなり安価です。

次に、問題をより難しくし、24 量子ビットに変換される 24 頂点グラフで Max-Cut の問題を解決してみましょう。同じ 2 つのインスタンスでハイブリッドジョブを再度実行し、コストを比較します。

> **Note**
>
> CPU インスタンスでこのハイブリッドジョブを実行する時間は、約 5 時間になる場合があります。

```
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,
)
```

m5.2xlarge インスタンスの平均反復時間は約 1 時間で、p3.2xlargeインスタンスの平均反復時間は約 2 分です。この大きな問題では、GPU インスタンスは桁違いに高速です。この高速化の恩恵を受けるために必要なのは、2 行のコードを変更して、インスタンスタイプとローカルシミュレーターをスワップすることだけです。ここで行ったように、合計 5 回の反復を実行するには、CPU インスタンスを使用して約 2.27,072 USD、または GPU インスタンスを使用して約 0.77,5625 USD のコストがかかります。CPU 使用率はより高価であるだけでなく、実行にさらに時間がかかります。NVIDIA がサポートする PennyLaneの埋め込みシミュレーターを使用して AWS、 で利用可能な GPU インスタンスでこのワークフローを加速することで、中間量子ビット数 (20 から 30 の間) でワークフローを実行できるため CuQuantum、合計コストが低く、時間も短縮されます。つまり、ラップトップや同様のサイズのインスタンスですぐに実行できない大きな問題でも、量子コンピューティングを試すことができます。

量子機械学習とデータ並列処理

ワークロードタイプがデータセットでトレーニングする量子機械学習 ("L) である場合は、データ並列処理を使用してワークロードをさらに高速化できます。QML では、モデルには 1 つ以上の量子回路が含まれています。モデルには古典ニューラルネットが含まれている場合と含まれない場合もあります。データセットを使用してモデルをトレーニングすると、モデルのパラメータが更新され、損失関数が最小限に抑えられます。損失関数は通常、単一のデータポイントに対して定義され、データセット全体の平均損失の合計が定義されます。QML では、勾配計算の合計損失を平均化する前に、損失は通常連続して計算されます。特に数百のデータポイントがある場合、この手順には時間がかかります。

あるデータポイントからの損失は他のデータポイントに依存しないため、損失は並行して評価できます。異なるデータポイントに関連する損失と勾配は、同時に評価できます。これはデータ並列処理と呼ばれます。 SageMakerの分散データ並列ライブラリを使用すると、Amazon Braket Hybrid Jobs は、データ並列処理を活用してトレーニングを高速化することを容易にします。

二項分類の例として、よく知られている UCI リポジトリの Sonar データセットデータセットを使用するデータ並列処理には、次の QML ワークロードを検討してください。Sonar データセットには 208 個のデータポイントがあり、それぞれに 60 個の特徴があり、ソナーシグナルから収集され、マ

テリアルが跳ね返ります。各データポイントは、地雷の場合は「M」、石の場合は「R」のラベルが付けられます。当社の QML モデルは、入力レイヤー、隠しレイヤーとしての量子回路、および出力レイヤーで構成されます。入力レイヤーと出力レイヤーは、 に実装されている古典ニューラルネットです PyTorch。量子回路は、 の qml.qnn モジュールを使用して PyTorch ニューラルネットと統合 PennyLaneされます。ワークロードの詳細については、サンプルノートブックを参照してください。上記の QAOA の例と同様に、 などの組み込み GPU ベースのシミュレーターを使用して GPU の能力を活用して PennyLanelightning.gpu、組み込み CPU ベースのシミュレーターよりもパフォーマンスを向上させることができます。

ハイブリッドジョブを作成するには、 を呼び出しAwsQuantumJob.create、キーワード引数を使用してアルゴリズムスクリプト、デバイス、およびその他の設定を指定できます。

```
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,
    ...
)
```

データ並列処理を使用するには、 SageMaker 分散ライブラリのアルゴリズムスクリプト内の数行のコードを変更して、トレーニングを正しく並列化する必要があります。まず、smdistributedパッケージをインポートします。パッケージは、ワークロードを複数の GPUsと複数のインスタンスに分散するために、ほとんどの負荷の高いリフトを実行します。このパッケージは Braket PyTorch および TensorFlow コンテナで事前設定されています。dist モジュールは、トレーニング (world_size) の GPUs の合計数と GPU コアlocal_rankの rankおよび をアルゴリズムスクリプトに指示します。 rankはすべてのインスタンスにおける GPU の絶対インデックスであり、 local_rank はインスタンス内の GPU のインデックスです。例えば、トレーニングにそれぞれ 8 つの GPUs が割り当てられた 4 つのインスタンスがある場合、 rankの範囲は 0～31、local_rankの範囲は 0～7 です。

```
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)
```

次に、 world_sizeおよび DistributedSamplerに従って を定義rankし、それをデータローダーに渡します。このサンプラーは、GPUsがデータセットの同じスライスにアクセスするのを回避します。

```
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,
)
```

次に、 DistributedDataParallel クラスを使用してデータ並列処理を有効にします。

```
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"])
```

上記の変更は、データ並列処理を使用するために必要なものです。QML では、多くの場合、結果を保存してトレーニングの進行状況を出力します。各 GPU が 保存コマンドと印刷コマンドを実行すると、ログは繰り返される情報でフラッディングされ、結果は互いに上書きされます。これを回避するには、0 の GPU rank からのみ保存および印刷できます。

```
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 は、 SageMaker 分散データ並列ライブラリのml.p3.16xlargeインスタンスタイプをサポートしています。インスタンスタイプは、Hybrid Jobs のInstanceConfig引数を使用して設定します。 SageMaker 分散データ並列ライブラリがデータ並列処理が有効になっていることを知るには、 を に設定"true"し、"sagemaker_distributed_dataparallel_enabled"を使用中のインスタンスタイプ"sagemaker_instance_type"に設定する 2 つのハイパーパラメータを追加する必要があります。これら 2 つのハイパーパラメータはsmdistributedパッケージで使用されます。アルゴリズムスクリプトは明示的に使用する必要はありません。Amazon Braket SDK では、便利なキーワード引数 を提供しますdistribution。ハイブリッドジョブの作成distribution="data_parallel"では、Amazon Braket SDK によって 2 つのハイパーパラメータが自動的に挿入されます。Amazon Braket API を使用する場合は、これら 2 つのハイパーパラメータを含める必要があります。

インスタンスとデータ並列処理を設定して、ハイブリッドジョブを送信できるようになりました。ml.p3.16xlarge インスタンスには 8 GPUs があります。を設定するとinstanceCount=1、ワークロードはインスタンス内の 8 GPUs に分散されます。instanceCount 複数の を設定すると、ワークロードはすべてのインスタンスで使用できる GPUsに分散されます。複数のインスタンスを使用する場合、各インスタンスの使用時間に基づいて料金が発生します。例えば、4 つのインスタンスを使用する場合、ワークロードを同時に実行しているインスタンスが 4 つあるため、請求対象時間はインスタンスあたりの実行時間の 4 倍になります。

```
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**
>
> 上記のハイブリッドジョブの作成では、 `train_dp.py`はデータ並列処理を使用するための変更されたアルゴリズムスクリプトです。データ並列処理は、上記のセクションに従ってアルゴリズムスクリプトを変更する場合にのみ正しく機能することに注意してください。データ並列処理オプションが正しく変更されたアルゴリズムスクリプトなしで有効になっている場合、ハイブリッドジョブがエラーをスローしたり、各 GPU が同じデータスライスを繰り返し処理したりすることがありますが、これは非効率です。

上記の二項分類問題に対して 26 量子ビットの量子回路でモデルをトレーニングする場合の例で、実行時間とコストを比較してみましょう。この例で使用されている`ml.p3.16xlarge`インスタンスの料金は 1 分あたり 0.4692 USD です。データ並列処理を行わない場合、シミュレーターがモデルを 1 エポック (つまり、208 データポイント以上) でトレーニングするのに約 45 分かかり、約 20 USD かかります。1 つのインスタンスと 4 つのインスタンスのデータ並列処理では、それぞれ 6 分と 1.5 分しかかからず、どちらも約 2.8 USD になります。4 つのインスタンスでデータ並列処理を使用すると、実行時間を 30 倍向上させるだけでなく、コストを 1 桁削減できます。

ローカルモードでハイブリッドジョブを構築およびデバッグする

新しいハイブリッドアルゴリズムを構築する場合、ローカルモードはアルゴリズムスクリプトのデバッグとテストに役立ちます。ローカルモードは、AmazonBraket Hybrid Jobs で使用する予定のコードを実行できますが、Braket がハイブリッドジョブを実行するインフラストラクチャを管理する必要はありません。代わりに、Braket Notebook インスタンスまたはラップトップやデスクトップコンピュータなどの優先クライアントでハイブリッドジョブをローカルで実行します。ローカルモードでは、量子タスクを実際のデバイスに送信することはできますが、ローカルモードで実際の QPU に対して実行すると、パフォーマンス上の利点は得られません。

ローカルモードを使用するには、発生する場所で `AwsQuantumJob` を `LocalQuantumJob` に変更します。例えば、最初のハイブリッドジョブ を作成する の例を実行するには、次のようにハイブリッドジョブスクリプトを編集します。

```
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**
>
> この機能を使用するには、Amazon Braket ノートブックに既にプリインストールされている Docker をローカル環境にインストールする必要があります。Docker のインストール手順については、https://docs.docker.com/get-docker/「」を参照してください。さらに、すべてのパラメータがローカルモードでサポートされているわけではありません。

独自のコンテナ

Amazon Braket Hybrid Jobs には、さまざまな環境でコードを実行するための 3 つの構築済みコンテナが用意されています。これらのコンテナのいずれかがユースケースをサポートしている場合は、ハイブリッドジョブを作成するときにのみアルゴリズムスクリプトを提供する必要があります。欠落している軽微な依存関係は、アルゴリズムスクリプトまたは を使用するrequirements.txtファイルから追加できますpip。

これらのコンテナのいずれもユースケースをサポートしていない場合、またはそれらを拡張する場合、Braket Hybrid Jobs は独自のカスタムDockerコンテナイメージを使用したハイブリッドジョブの実行、または独自のコンテナの持ち込み (BYOC) をサポートします。ただし、詳しく知る前に、実際にユースケースに適した機能であることを確認してください。

自分のコンテナを持ち込むのはどのような場合が適切ですか？

独自のコンテナ (BYOC) を Braket Hybrid Jobs に持ち込むと、パッケージ化された環境にインストールすることで、独自のソフトウェアを柔軟に使用できます。特定のニーズに応じて、完全な BYOC Dockerビルド - Amazon ECR アップロード - カスタムイメージ URI サイクルを経ることなく、同じ柔軟性を実現する方法がある場合があります。

> **Note**
>
> 公開されている少数の Python パッケージ (通常は 10 個未満) を追加する場合は、BYOC が適していない可能性があります。例えば、 を使用している場合です PyPi。

この場合、構築済みの Braket イメージのいずれかを使用して、ジョブの送信時にソースディレクトリに`requirements.txt`ファイルを含めることができます。ファイルは自動的に読み取`pip`られ、通常どおり指定されたバージョンでパッケージがインストールされます。多数のパッケージをインストールする場合、ジョブのランタイムが大幅に増加する可能性があります。Python を確認し、該当する場合は、ソフトウェアが動作するかどうかのテストに使用する構築済みコンテナの CUDA バージョンを確認します。

BYOC は、ジョブスクリプトに Python 以外の言語 (C++ や Rust など) を使用する場合や、Braket 構築済みコンテナでは利用できない Python バージョンを使用する場合に必要です。また、次のような場合にも適しています。

- ライセンスキーを持つソフトウェアを使用しているため、ソフトウェアを実行するにはライセンスサーバーに対してそのキーを認証する必要があります。BYOC を使用すると、ライセンスキーを Dockerイメージに埋め込み、認証用のコードを含めることができます。
- 公開されていないソフトウェアを使用している。例えば、ソフトウェアは、アクセスに特定の SSH キーを必要とするプライベート GitLab または GitHub リポジトリでホストされます。
- Braket が提供するコンテナにパッケージ化されていない大規模なソフトウェアスイートをインストールする必要があります。BYOC を使用すると、ソフトウェアのインストールによるハイブリッドジョブコンテナの長い起動時間を排除できます。

また、BYOC では、ソフトウェアでDockerコンテナを構築し、ユーザーが利用できるようにすることで、カスタム SDK またはアルゴリズムを顧客が利用できるようにします。これを行うには、Amazon ECR で適切なアクセス許可を設定します。

> **Note**
>
> 該当するすべてのソフトウェアライセンスを遵守する必要があります。

独自のコンテナを持ち込むためのレシピ

このセクションでは、Braket Hybrid Jobs bring your own container (BYOC)に必要なこと、つまりカスタムDockerイメージを起動して実行するためにそれらを組み合わせるスクリプト、ファイル、ステップの step-by-step ガイドを提供します。2 つの一般的なケースのレシピを提供しています。

1. Docker イメージに追加のソフトウェアをインストールし、ジョブで Python アルゴリズムスクリプトのみを使用します。
2. Hybrid Jobs で Python 以外の言語で記述されたアルゴリズムスクリプト、または x86 以外の CPU アーキテクチャを使用します。

コンテナエントリスクリプトの定義は、ケース 2 ではより複雑です。

Braket がハイブリッドジョブを実行すると、リクエストされた数とタイプの Amazon EC2 インスタンスを起動し、Dockerイメージ URI 入力で指定されたイメージを実行してジョブを作成します。BYOC 機能を使用する場合は、読み取りアクセス権があるプライベート Amazon ECR リポジトリでホストされているイメージ URI を指定します。Braket Hybrid Jobs は、そのカスタムイメージを使用してジョブを実行します。

Hybrid Jobs で使用できるDockerイメージの構築に必要な特定のコンポーネント。の記述と構築に慣れていない場合は`Dockerfiles`、これらの手順を読みながら、必要に応じて Dockerfile ドキュメントと Amazon ECR CLIドキュメントを参照することをお勧めします。

必要なものの概要は次のとおりです。

- Dockerfile のベースイメージ
- (オプション) 変更されたコンテナエントリポイントスクリプト
- 必要なソフトウェアDockerfileをインストールし、コンテナスクリプトを含む 。

Dockerfile のベースイメージ

Python を使用していて、Braket が提供するコンテナで提供されているものの上にソフトウェアをインストールする場合、ベースイメージのオプションは、GitHub リポジトリと Amazon ECR でホストされている Braket コンテナイメージの 1 つです。イメージをプルしてその上に構築するには、Amazon ECR に対して認証する必要があります。例えば、BYOC Docker ファイルの最初の行は次のようになります。 `FROM [IMAGE_URI_HERE]`

次に、 の残りの部分に記入Dockerfileして、コンテナに追加するソフトウェアをインストールしてセットアップします。構築済みの Braket イメージには既に適切なコンテナエントリポイントスクリプトが含まれているため、これを含めることを心配する必要はありません。

C++、Rust、Julia などの Python 以外の言語を使用する場合、または ARM などの x86 以外の CPU アーキテクチャ用のイメージを構築する場合は、ベアボーンパブリックイメージの上に構築する必要がある場合があります。このようなイメージの多くは、Amazon Elastic Container Registry Public Gallery にあります。CPU アーキテクチャに適したものを選択し、必要に応じて使用する GPU を選択してください。

(オプション) 変更されたコンテナエントリポイントスクリプト

> **Note**
>
> 構築済みの Braket イメージにのみソフトウェアを追加する場合は、このセクションをスキップできます。

ハイブリッドジョブの一部として Python 以外のコードを実行するには、コンテナエントリポイントを定義する Python スクリプトを変更する必要があります。例えば、`braket_container.py`Amazon Braket Github の Python スクリプトです。これは、Braket によって事前に構築されたイメージがアルゴリズムスクリプトを起動し、適切な環境変数を設定するために使用するスクリプトです。コンテナエントリポイントスクリプト自体は Python にある必要がありますが、Python 以外のスクリプトを起動できます。構築済みの例では、Python アルゴリズムスクリプトが Python サブプロセスまたはまったく新しいプロセス として起動されていることがわかります。このロジックを変更することで、エントリポイントスクリプトを有効にして Python 以外のアルゴリズムスクリプトを起動できます。例えば、ファイル拡張子の終了に応じて Rust プロセスを起動するように `thekick_off_customer_script()`関数を変更できます。

まったく新しい を記述することもできます`braket_container.py`。入力データ、ソースアーカイブ、およびその他の必要なファイルを Amazon S3 からコンテナにコピーし、適切な環境変数を定義する必要があります。

必要なソフトウェア**Dockerfile**をインストールし、コンテナスクリプトを含む 。

> **Note**
>
> 構築済みの Braket イメージをDockerベースイメージとして使用している場合、コンテナスクリプトは既に存在します。

前のステップで変更したコンテナスクリプトを作成した場合は、コンテナにコピーし、環境変数をSAGEMAKER_PROGRAMに定義するかbraket_container.py、新しいコンテナエントリポイントスクリプトに名前を付けたものを定義する必要があります。

以下は、GPU アクセラレーションDockerfileされた Jobs インスタンスで Julia を使用できるようにする の例です。

```
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
```

この例では、 は が提供するスクリプトをダウンロードして実行 AWS し、関連するすべてのオープンソースライセンスを確実に遵守します。例えば、 によって管理されるインストール済みコードに適切に帰属させますMIT license。

プライベート GitHub または GitLab リポジトリでホストされているインスタンスコードなど、非パブリックコードを含める必要がある場合は、Dockerイメージに SSH キーを埋め込んでアクセスしな

いでください。代わりに、ビルドDocker Compose時に を使用して、ビルドされているホストマシン上の SSH Dockerへのアクセスを に許可します。詳細については、「Docker で SSH キーを安全に使用してプライベート Github リポジトリにアクセスする」ガイドを参照してください。

Dockerイメージの構築とアップロード

これで、適切に定義された を使用してDockerfile、プライベート Amazon ECR リポジトリ がまだ存在しない場合は、そのリポジトリを作成するステップを実行する準備が整いました。コンテナイメージを構築、タグ付け、リポジトリにアップロードすることもできます。

イメージを構築、タグ付け、プッシュする準備が整いました。のオプションの完全な説明docker buildといくつかの例については、Docker ビルドドキュメントを参照してください。

上記のサンプルファイルでは、以下を実行できます。

```
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
```

適切な Amazon ECR アクセス許可の割り当て

Braket Hybrid Jobs Docker イメージはプライベート Amazon ECR リポジトリでホストする必要があります。デフォルトでは、プライベート Amazon ECR リポジトリは、 への読み取りアクセスや、共同作業者Braket Hybrid Jobs IAM roleや学生など、イメージを使用したい他のユーザーへの読み取りアクセスを提供しません。適切なアクセス許可を付与するには、リポジトリポリシーを設定する必要があります。一般に、イメージにアクセスする特定のユーザーとIAMロールにのみアクセス許可を付与し、 を持つすべてのユーザーにイメージimage URIのプルを許可しないでください。

Braket ハイブリッドジョブを独自のコンテナで実行する

独自のコンテナを使用してハイブリッドジョブを作成するには、 image_uri を指定された 引数AwsQuantumJob.create()で呼び出します。QPU、オンデマンドシミュレーターを使用するか、Braket Hybrid Jobs で利用可能なクラシックプロセッサでコードをローカルで実行できます。実際の QPU で実行する前に、SV1, DM1、TN1 などのシミュレーターでコードをテストすることをお勧めします。

クラシックプロセッサでコードを実行するには、 を更新して、instanceCount使用する instanceTypeと を指定しますInstanceConfig。instance_count > 1 を指定する場合は、

コードが複数のホストで実行できることを確認する必要があります。選択できるインスタンス数の上限は 5 です。例:

```
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**
>
> デバイス ARN を使用して、ハイブリッドジョブメタデータとして使用したシミュレーターを追跡します。許容値は の形式に従う必要がありますdevice = "local:<provider>/<simulator_name>"。<provider> および は、文字、数字、、_、-および のみで構成される<simulator_name>必要があることに注意してください.。文字列は 256 文字に制限されています。
>
> BYOC を使用する予定で、Braket SDK を使用して量子タスクを作成していない場合は、環境変数の値をCreateQuantumTaskリクエストの jobTokenパラメータAMZN_BRAKET_JOB_TOKENに渡す必要があります。そうしないと、量子タスクは優先順位を付けず、通常のスタンドアロン量子タスクとして請求されます。

AwsSession でデフォルトのバケットを設定します。

独自のAwsSessionを提供すると、デフォルトバケットの場所など、柔軟性が向上します。デフォルトでは、AwsSession のデフォルトのバケットの場所は f"amazon-braket-{id}-{region}" です。ただし、AwsSession を作成するときにこのデフォルトをオーバーライドできます。次のコード例aws_sessionに示すように、ユーザーはオプションでパラメータ名AwsQuantumJob.createで AwsSession オブジェクトを に渡すことができます。

```
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
)
```

を使用してハイブリッドジョブを直接操作する API

を使用して Amazon Braket Hybrid Jobs に直接アクセスして操作できますAPI。ただし、 APIを直接使用する場合、デフォルトと便利なメソッドは使用できません。

> **Note**
>
> Amazon Braket Python SDK を使用して Amazon Braket Hybrid Jobs を操作することを強くお勧めします。ハイブリッドジョブを正常に実行するのに役立つ便利なデフォルトと保護を提供します。

このトピックでは、 の使用の基本について説明しますAPI。API を使用する場合は、このアプローチがより複雑になり、ハイブリッドジョブを実行するための反復処理が複数回行われる可能性があることに注意してください。

API を使用するには、アカウントに `AmazonBraketFullAccess`管理ポリシーを持つロールが必要です。

> **Note**
>
> `AmazonBraketFullAccess` マネージドポリシーでロールを取得する方法の詳細については、Amazon Braket を有効にする」ページを参照してください。

さらに、実行ロール が必要です。このロールはサービスに渡されます。 Amazon Braket コンソールを使用してロールを作成できます。アクセス許可と設定ページの「実行ロール」タブを使用して、ハイブリッドジョブのデフォルトロールを作成します。

`CreateJob` API では、ハイブリッドジョブに必要なすべてのパラメータを指定する必要があります。Python を使用するには、アルゴリズムスクリプトファイルを input.tar.gz ファイルなどの tar バンドルに圧縮し、次のスクリプトを実行します。コードの部分を角括弧 (<>) で更新して、ハイブリッドジョブが開始されるパス、ファイル、メソッドを指定するアカウント情報とエントリポイントと一致させます。

```
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
    },
)
```

ハイブリッドジョブを作成したら、 GetJobAPIまたはコンソールからハイブリッドジョブの詳細にアクセスできます。前の例のようにcreateJobコードを実行した Python セッションからハイブリッドジョブの詳細を取得するには、次の Python コマンドを使用します。

```
getJob = client.get_job(jobArn=job["jobArn"])
```

ハイブリッドジョブをキャンセルするには、ジョブ () Amazon Resource Nameの CancelJobAPIを使用して を呼び出します'JobArn'。

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

checkpointConfig パラメータcreateJobAPIを使用して、 の一部としてチェックポイントを指定できます。

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> のローカルパス checkpointConfig は、予約済みパスの /opt/ml、/opt/braket、/tmp、または /usr/local/nvidia のいずれかで開始することはできません。

エラーの軽減

量子エラー軽減は、量子コンピュータのエラーの影響を減らすことを目的とした一連の手法です。

量子デバイスは、実行される計算の品質を低下させる環境ノイズの影響を受けます。耐障害性量子コンピューティングはこの問題への解決策を保証しますが、現在の量子デバイスは量子ビット数と比較的高いエラー率によって制限されます。短期的にこれに対処するため、研究者はノイズの多い量子計算の精度を向上させる方法を調査しています。このアプローチは量子エラー軽減 と呼ばれ、さまざまな手法を使用してノイズの多い測定データから最適なシグナルを抽出します。

でのエラー軽減 IonQ Aria

エラーの軽減には、複数の物理回路を実行し、その測定値を組み合わせて結果を改善することが含まれます。IonQ Aria デバイスには、 の偏見というエラー軽減方法があります。

バイアス解除は、回路を、異なる量子ビット順列または異なるゲート分解で動作する複数のバリアントにマッピングします。これにより、測定結果をバイアスする可能性のある回路のさまざまな実装を使用することで、ゲートのオーバーローテーションや単一の障害のある量子ビットなどの体系的なエラーの影響が軽減されます。これは、複数の量子ビットとゲートをキャリブレーションするための余分なオーバーヘッドの犠牲になります。

バイアス解除の詳細については、「対称化による量子コンピュータのパフォーマンスの向上」を参照してください。

> **Note**
>
> デバイアスを使用するには、最低 2500 ショットが必要です。

次のコードを使用して、 IoQ Ariaデバイスでデバイアスを伴う量子タスクを実行できます。

```
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
```

量子タスクが完了すると、量子タスクから測定確率と結果タイプを確認できます。すべてのバリアントの測定確率とカウントは、1 つの分布に集約されます。期待値など、回路で指定された結果タイプは、集計測定カウントを使用して計算されます。

シャープニング

また、シャープニングと呼ばれる別の後処理戦略で計算された測定確率にアクセスすることもできます。シャープニングは、各バリアントの結果を比較し、一貫性のないショットを破棄し、バリアント間で最も可能性の高い測定結果を優先します。詳細については、「対称化による量子コンピュータのパフォーマンスの向上」を参照してください。

重要なのは、シャープニングは、出力分布の形式が、確率の高い状態が少なく、確率のない状態が多いことを前提としています。この仮定が有効でない場合、確率分布が歪む可能性があります。

`GateModelTaskResult` Braket Python SDK の の `additional_metadata`フィールドのシャープなディストリビューションから確率にアクセスできます。シャープニングは測定数を返さず、代わりに再正規化された確率分布を返すことに注意してください。次のコードスニペットは、シャープニング後にディストリビューションにアクセスする方法を示しています。

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

Braket Direct

Braket Direct を使用すると、選択したさまざまな量子デバイスへの専用アクセスを予約し、量子コンピューティングスペシャリストに接続してワークロードのガイダンスを受け取り、可用性が制限された新しい量子デバイスなどの次世代機能に早期にアクセスできます。

このセクションの内容:

予約

予約すると、選択した量子デバイスへの排他的なアクセスが可能になります。予約は都合の良いタイミングでスケジュールできるため、ワークロードの実行の開始と終了を正確に把握できます。予約は 1 時間単位で可能で、48 時間前までキャンセルできます。追加料金はかかりません。今後の予約のために量子タスクとハイブリッドジョブを事前にキューに入れるか、予約中にワークロードを送信するかを選択できます。

専用デバイスアクセスのコストは、量子処理ユニット (QPU) で実行する量子タスクとハイブリッドジョブの数に関係なく、予約の期間に基づきます。

予約には、次の量子コンピュータを使用できます。

- IonQ の Aria
- I"" のガーネット
- QuEraの Aquila
- リゲッティの Aspen-M-3

予約を使用するタイミング

予約で専用デバイスアクセスを利用すると、量子ワークロードの実行の開始と終了を正確に把握する利便性と予測可能性が得られます。タスクやハイブリッドジョブをオンデマンドで送信する場合と比較して、他のカスタマータスクでキューに待機することはありません。予約中にデバイスへの排他的なアクセス権があるため、予約中はワークロードのみがデバイス上で実行されます。

オンデマンドアクセスは、研究の設計とプロトタイプ作成の段階で使用し、アルゴリズムを迅速かつコスト効率の高い方法で反復できるようにすることをお勧めします。最終的な実験結果を作成する準備ができたら、プロジェクトまたは公開の期限を確実に守られるように、都合の良いときにデバイス予約をスケジュールすることを検討してください。また、ライブデモや量子コンピュータでのワークショップなど、特定の時間帯にタスクを実行する場合は、予約を使用することをお勧めします。

このセクションの内容:

予約を作成する

予約を作成するには、以下の手順に従って Braket チームにお問い合わせください。

1. Amazon Braket コンソールを開きます。
2. 左のペインで Braket Direct を選択し、予約セクションでデバイスの予約 を選択します。
3. 予約するデバイスを選択します。
4. 名前 や E メール などの連絡先情報を入力します。定期的にチェックする有効な E メールアドレスを必ず指定してください。
5. 「ワークロードについて教えてください」で、予約を使用して実行するワークロードに関する詳細を入力します。例えば、希望する予約期間、関連する制約、または希望するスケジュールなどです。
6. 予約の確認後に予約準備セッションのために Braket の専門家と接続することに関心がある場合は、オプションで準備セッション に関心がある を選択します。

以下の手順に従って、予約を作成するためにお問い合わせいただくこともできます。

1. Amazon Braket コンソールを開きます。
2. 左側のペインでデバイスを選択し、予約するデバイスを選択します。
3. 概要 セクションで、デバイス を予約 を選択します。
4. 前の手順のステップ 4 ~ 6 に従います。

フォームを送信すると、Braket チームから、予約を作成するための次のステップが記載された E メールが届きます。予約が確認されると、予約 ARN が E メールで届きます。

> **Note**
>
> 予約は、予約 ARN を受け取った後にのみ確認されます。

予約は最低 1 時間単位で利用可能で、特定のデバイスには追加の予約時間制限 (最小予約期間と最大予約期間を含む) がある場合があります。Braket チームは、予約を確認する前に、関連情報をすべて共有します。

予約準備セッションに関心を示した場合、Braket チームは E メールで連絡し、Braket の専門家との 30 分間のセッションを手配します。

予約でワークロードを実行する

予約中は、ワークロードのみがデバイスで実行されます。デバイス予約中に実行する量子タスクとハイブリッドジョブを指定するには、有効な予約 ARN を使用する必要があります。

> **Note**
>
> 予約は AWS アカウントとデバイスによって異なります。予約を作成した AWS アカウントのみが予約 ARN を使用できます。さらに、予約 ARN は、選択した開始時刻と終了時刻の予約済みデバイスでのみ有効です。

予約時間を最大限に活用するには、予約前にタスクとジョブをキューに入れることを選択できます。これらのワークロードは、予約が開始されるまで QUEUEDステータスのままになります。予約が開始されると、キューに入れられたワークロードは送信された順序で実行されます。ジョブタスクは、スタンドアロン量子タスクよりも優先されます。

> **Note**
>
> 予約中に実行されるのはワークロードのみであるため、予約 ARN で送信されたタスクやジョブのキューの可視性はありません。

予約の量子タスクを作成するためのコード例：

1. OpenQASM fomat で GHZ 状態を準備する回路を定義します。

```
// 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. 回路と予約 ARN を使用して量子タスクを作成します。

```
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>"
)
```

Braket Direct 予約のハイブリッドジョブを作成するためのコード例：

1. アルゴリズムスクリプトを定義します。

```
//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. アルゴリズムスクリプトと予約 ARN を使用してハイブリッドジョブを作成します。

```
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. リモートデコレータを使用してハイブリッドジョブを作成します。

```
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
```

予約の最後に何が起こるか

予約が終了すると、デバイスへの専用アクセスはできなくなります。この予約でキューに入っている残りのワークロードは自動的にキャンセルされます。

> **Note**
>
> 予約が終了した時点でRUNNINGステータスだったジョブはすべてキャンセルされます。チェックポイントを使用して、都合の良いときにジョブを保存および再起動することをお勧めします。

予約開始後や予約終了前などの継続的な予約は、各予約がスタンドアロンの専用デバイスアクセスを表すため、延長できません。例えば、2 つの back-to-back 予約は別々のものと見なされ、最初の予約から保留中のタスクは自動的にキャンセルされます。2 番目の予約では再開されません。

> **Note**
>
> 予約は AWS 、アカウント専用のデバイスアクセスを表します。デバイスがアイドル状態であっても、他のお客様はデバイスを使用できません。したがって、使用時間に関係なく、予約時間の長さに対して課金されます。

既存の予約をキャンセルまたは再スケジュールする

予約は、スケジュールされた予約開始時刻の 48 時間以上前にキャンセルできます。キャンセルするには、キャンセルリクエストで受け取った予約確認 E メールに応答します。

スケジュールを変更するには、既存の予約をキャンセルしてから、新しい予約を作成する必要があります。

専門家のアドバイス

Braket マネジメントコンソールで量子コンピューティングの専門家に直接接続して、ワークロードに関する追加のガイダンスを取得します。

Braket Direct を介してエキスパートアドバイスオプションを調べるには、Braket コンソールを開き、左側のペインで Braket Direct を選択し、エキスパートアドバイスセクションに移動します。以下のエキスパートアドバイスオプションを利用できます。

- Braket の営業時間： Braket の営業時間は 1 対 1 のセッションで、先着順で毎月行われます。オフィス時間スロットは 30 分ごとに無料です。Braket の専門家に相談すると、 use-case-to-device 適合を探索し、アルゴリズムに Braket を最適に活用するためのオプションを特定し、Amazon Braket Hybrid Jobs、Braket Pulse、アナログハミルトニアシミュレーションなどの特定の Braket 機能の使用方法に関するレコメンデーションを取得することで、アイデアから実行までの時間を短縮できます。
  - Braket の営業時間にサインアップするには、サインアップを選択し、連絡先情報、ワークロードの詳細、および希望するディスカッショントピックを入力します。
  - 利用可能な次のスロットへのカレンダーの招待がメールで届きます。

> **Note**
>
> 緊急の問題やトラブルシューティングに関する簡単な質問については、AWS サポート に問い合わせることをお勧めします。緊急でない質問については、AWS re:Post フォーラムまたは Quantum Computing Stack Exchange を使用することもできます。ここでは、以前に回答された質問を参照し、新しい質問をすることができます。

- 量子ハードウェアプロバイダーサービス： IonQ 、Oxford Quantum Circuits QuEra、および Rigetti は、それぞれ 経由でプロフェッショナルサービスを提供します AWS Marketplace。
  - サービスを調べるには、接続を選択してリストを参照します。
  - のプロフェッショナルサービスの詳細については AWS Marketplace、「プロフェッショナルサービス製品」を参照してください。
- Amazon 量子ソリューションラボ (QSL): QSL は、量子コンピューティングのエキスパートが所属する共同研究およびプロフェッショナルサービスチームであり、量子コンピューティングを効果的に探索し、このテクノロジーの現在のパフォーマンスを評価するのに役立ちます。
  - QSL に連絡するには、接続 を選択し、連絡先情報とユースケースの詳細を入力します。
  - QSL チームは、次のステップを記載した E メールで連絡します。

実験的な機能

研究ワークロードを進化させるには、新しい革新的な機能にすばやくアクセスすることが重要です。Braket Direct を使用すると、可用性が制限された新しい量子デバイスなど、利用可能な実験的な機能へのアクセスを Braket コンソールで直接リクエストできます。

一部の実験的な機能は、標準のデバイス仕様外で動作し、ユースケースに合わせた実践的なガイダンスが必要です。ワークロードが成功するように設定するために、Braket Direct を通じてリクエストによりアクセスが可能です。

IonQ Forte への予約専用アクセス

Braket Direct を使用すると、IonQ Forte QPU への予約専用アクセスが可能になります。可用性が限られているため、このデバイスは Braket Direct でのみ使用できます。

詳細を確認し、IonQ Forte へのアクセスをリクエストするには、次の手順に従います。

1. Amazon Braket コンソールを開きます。

2. 左のメニューで Braket Direct を選択し、実験機能で IonQ Forte に移動します。デバイスの表示を選択します。
3. Forte デバイスの詳細ページで、概要 でデバイスを予約 を選択します。
4. 名前 や E メール などの連絡先情報を入力します。定期的にチェックする有効な E メールアドレスを入力します。
5. 「ワークロードについて教えてください」で、予約を使用して実行するワークロードの詳細を、希望する予約期間、関連する制約、または希望するスケジュールなどで指定します。
6. (オプション) 予約の確認後に予約準備セッションのために Braket の専門家と接続することに関心がある場合は、準備セッション に関心がある を選択します。

フォームが送信されると、Braket チームが次のステップについて連絡します。

> **Note**
>
> デバイスの可用性が制限されているため、Forte へのアクセスは制限されています。詳細については、お問い合わせください。

QuEra Aquila でのローカルデチューニングへのアクセス

Braket Direct を使用すると、QuEra AquilaQPU でのプログラミング時にローカルチューニングを制御するアクセスをリクエストできます。この機能を使用すると、運転フィールドが各特定の量子ビットにどの程度影響するかを調整できます。

詳細を確認し、この機能へのアクセスをリクエストするには、次の手順に従います。

1. Amazon Braket コンソールを開きます。
2. 左のメニューで Braket Direct を選択し、実験機能で QuEra Aquila - local detuning に移動します。「アクセスの取得」を選択します。
3. 名前 や E メール などの連絡先情報を入力します。定期的にチェックする有効な E メールアドレスを入力します。
4. 「ワークロードについて教えてください」で、ワークロードの詳細と、この機能を使用する予定の場所を指定します。

QuEra Aquila の背の高いジオメトリへのアクセス

Braket Direct を使用すると、QuEra AquilaQPU でプログラミングするときに、拡張されたジオメトリへのアクセスをリクエストできます。この機能を使用すると、標準のデバイス機能を超えて実験し、格子の高さを増やしてジオメトリを指定できます。

詳細を確認し、この機能へのアクセスをリクエストするには、次の手順に従います。

1. Amazon Braket コンソールを開きます。
2. 左のメニューで Braket Direct を選択し、実験機能で QuEra Aquila - 高さジオメトリ に移動します。「アクセスの取得」を選択します。
3. 名前 や E メール などの連絡先情報を入力します。定期的にチェックする有効な E メールアドレスを入力します。
4. 「ワークロードについて教えてください」で、ワークロードの詳細と、この機能を使用する予定の場所を指定します。

QuEra Aquila での厳しいジオメトリへのアクセス

Braket Direct を使用すると、QuEra AquilaQPU でプログラミングするときに、拡張されたジオメトリへのアクセスをリクエストできます。この機能を使用すると、標準のデバイス機能を超えて実験し、垂直方向の間隔を狭めながら格子行を配置できます。

詳細を確認し、この機能へのアクセスをリクエストするには、次の手順に従います。

1. Amazon Braket コンソールを開きます。
2. 左のメニューで Braket Direct を選択し、実験機能で QuEra Aquila - 高さジオメトリ に移動します。「アクセスの取得」を選択します。
3. 名前 や E メール などの連絡先情報を入力します。定期的にチェックする有効な E メールアドレスを入力します。
4. 「ワークロードについて教えてください」で、ワークロードの詳細と、この機能を使用する予定の場所を指定します。

ログ記録とモニタリング

量子タスクを送信すると、AmazonBraket SDK とコンソールを使用してステータスを追跡できます。量子タスクが完了すると、Braket は指定された Amazon S3 の場所に結果を保存します。キューの長さによっては、特に QPU デバイスでは、完了に時間がかかる場合があります。ステータスタイプは次のとおりです。

- `CREATED` — Amazon Braket が量子タスクを受け取りました。
- `QUEUED` — Amazon Braket が量子タスクを処理し、デバイスでの実行を待っています。
- `RUNNING` – 量子タスクは QPU またはオンデマンドシミュレーターで実行されています。
- `COMPLETED` – 量子タスクが QPU またはオンデマンドシミュレーターで実行されました。
- `FAILED` – 量子タスクの実行が試行され、失敗しました。量子タスクが失敗した理由に応じて、量子タスクを再度送信してみてください。
- `CANCELLED` — 量子タスクをキャンセルしました。量子タスクが実行されませんでした。

このセクションの内容:

Amazon Braket SDK からの量子タスクの追跡

コマンドは、一意の量子タスク ID を持つ量子タスク`device.run(…)` を定義します。次の例に示すように、`task.state()` を使用してステータスを照会および追跡できます。

注: `task = device.run()`は非同期オペレーションです。つまり、システムが量子タスクをバックグラウンドで処理している間も作業を続けることができます。

結果を取得する

を呼び出すとtask.result()、SDK は Amazon Braket のポーリングを開始し、量子タスクが完了したかどうかを確認します。SDK は、.run() で定義したポーリングパラメータを使用します。量子タスクが完了すると、SDK は S3 バケットから結果を取得し、QuantumTaskResultオブジェクトとして返します。

```
# 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
```

量子タスクをキャンセルする

量子タスクをキャンセルするには、次の例に示すように、 cancel()メソッドを呼び出します。

```
# cancel quantum task
task.cancel()
```

```
status = task.state()
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

メタデータの確認

次の例に示すように、完成した量子タスクのメタデータを確認できます。

```
# 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
```

量子タスクまたは結果を取得する

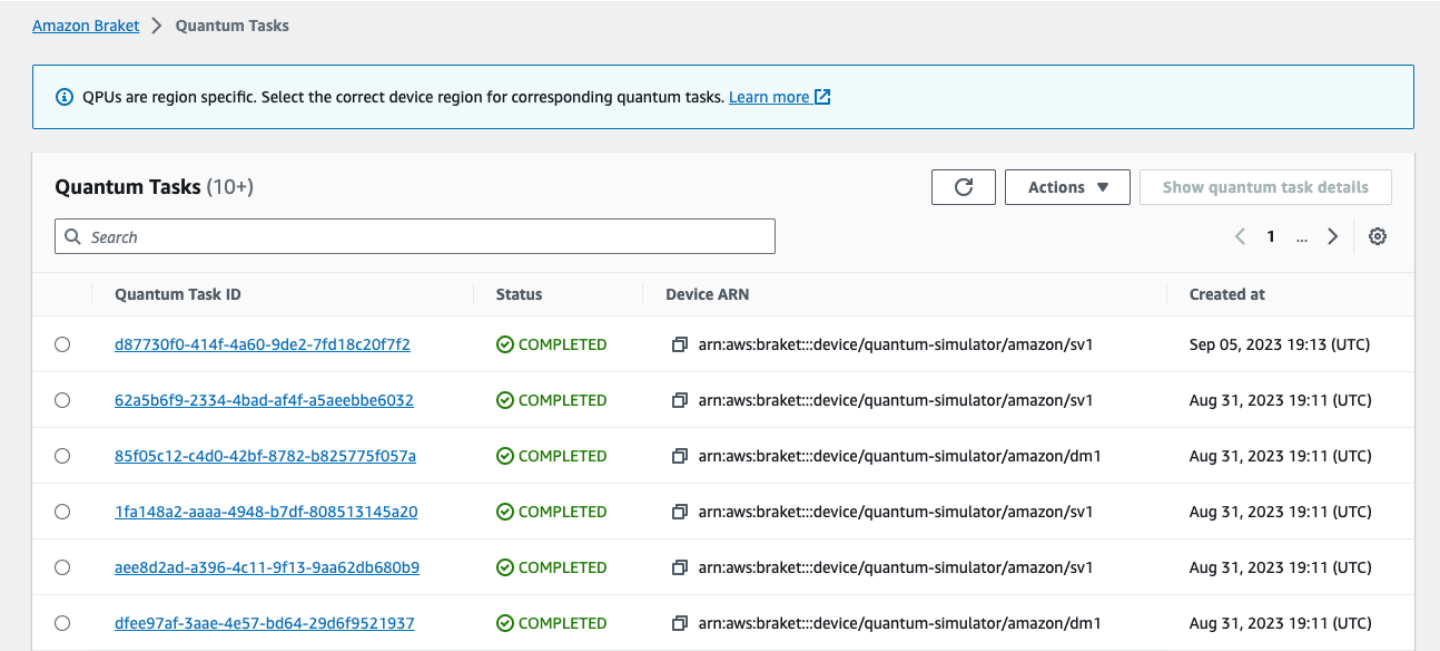
量子タスクを送信した後、またはノートブックまたはコンピュータを閉じた後にカーネルがなくなった場合は、一意の ARN (量子タスク ID) を使用してtaskオブジェクトを再構築できます。次に `task.result()` を呼び出して、保存されている S3 バケットから結果を取得します。

```
from braket.aws import AwsSession, AwsQuantumTask
```

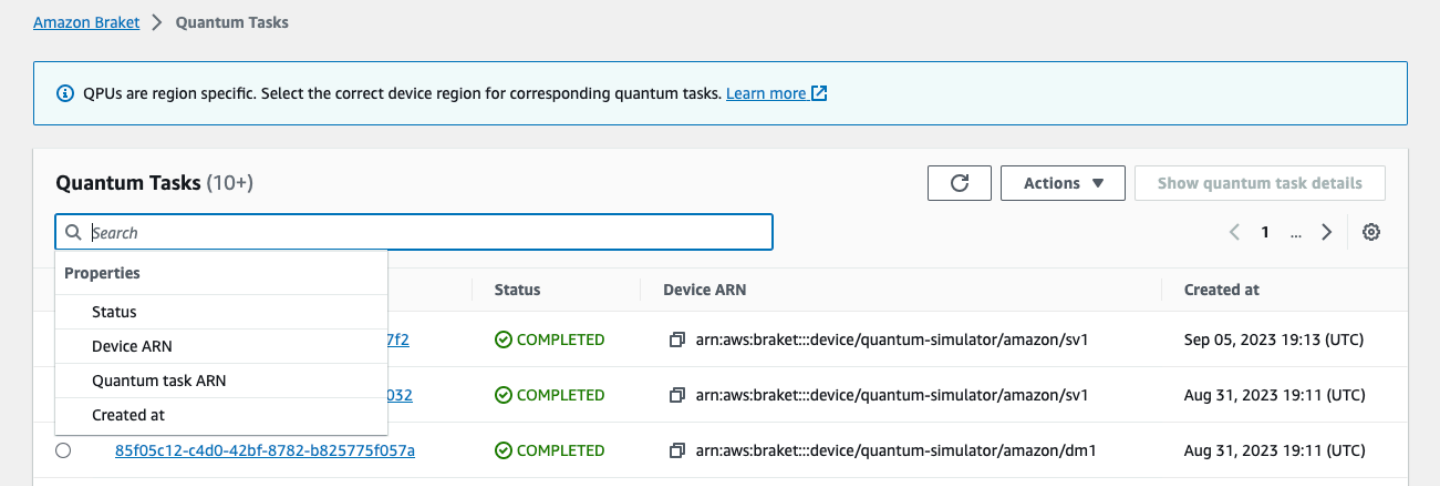
```
# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Amazon Braket コンソールによる量子タスクのモニタリング

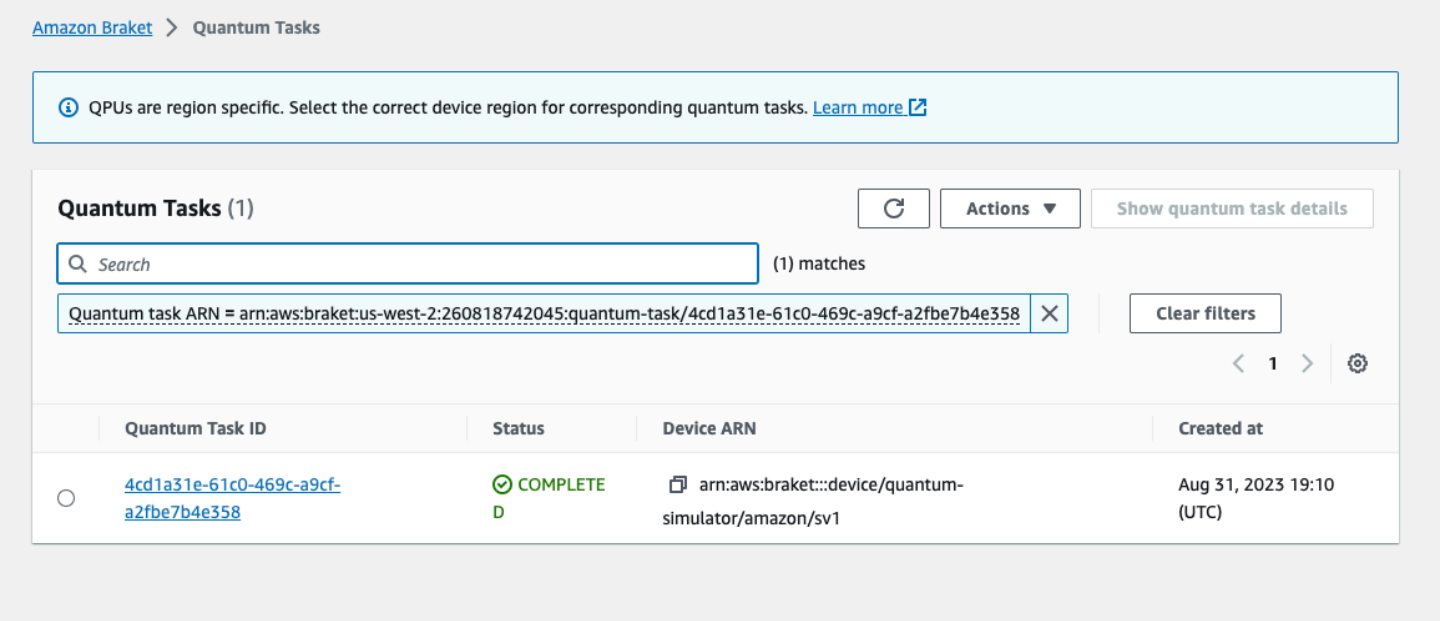
Amazon Braket は、Amazon Braket コンソール を介して量子タスクをモニタリングする便利な方法を提供します。次の図に示すように、送信されたすべての量子タスクが量子タスクフィールドに一覧表示されます。このサービスはリージョン固有の です。つまり、特定の で作成された量子タスクのみを表示できます AWS リージョン。



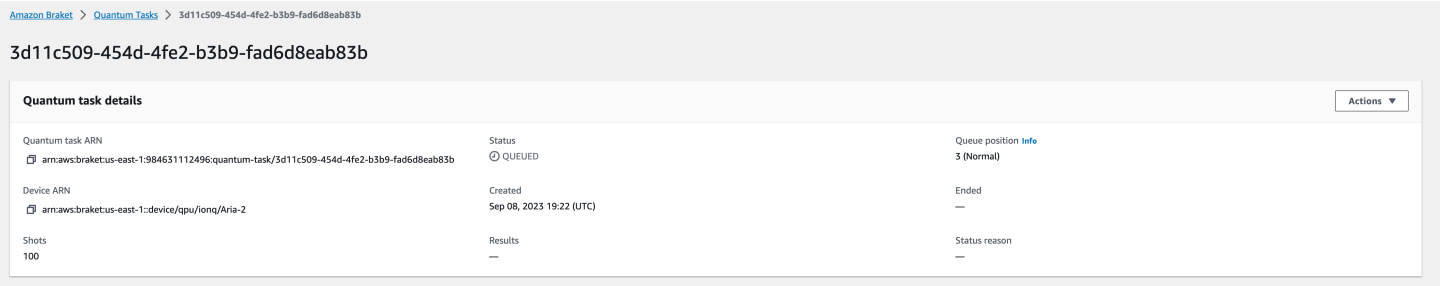
ナビゲーションバーから特定の量子タスクを検索できます。検索は、量子タスク ARN (ID)、ステータス、デバイス、作成時間に基づいて行うことができます。次の例に示すように、ナビゲーションバーを選択すると、オプションが自動的に表示されます。



次の図は、 を呼び出すことで取得できる一意の量子タスク ID に基づいて量子タスクを検索する例を示していますtask.id。



さらに、次の図に示すように、量子タスクのステータスは、 QUEUED状態のときにモニタリングできます。量子タスク ID をクリックすると、詳細ページが表示されます。このページには、処理するデバイスに関連する量子タスクの動的キュー位置が表示されます。



ハイブリッドジョブの一部として送信される量子タスクは、キューに入っているときに優先されます。ハイブリッドジョブの外部で送信される量子タスクは、通常のキューイング優先度を持ちます。

Braket SDK をクエリしたいお客様は、量子タスクとハイブリッドジョブキューの位置をプログラムで取得できます。詳細については、「タスクが実行されるタイミング」ページを参照してください。

Amazon Braket リソースのタグ付け

タグは、 AWS リソースに割り当てる、または AWS に割り当てるカスタム属性ラベルです。タグはリソースについて詳しく説明するメタデータです。各タグは、キーと値から構成されます。これらは共にキーと値のペアと呼ばれます。ユーザーが割り当てるタグでは、ユーザーがキーと値を定義します。

Amazon Braket コンソールでは、量子タスクまたはノートブックに移動し、それに関連付けられたタグのリストを表示できます。タグの追加、タグの削除、またはタグの修正を行うことができます。量子タスクまたはノートブックの作成時にタグを付け、コンソール、 AWS CLI、または を使用して関連するタグを管理できますAPI。

タグの使用

タグを使用すると、リソースを自分に役立つカテゴリに整理できます。例えば、「部門」タグを割り当てて、このリソースを所有する部門を指定できます。

各 タグは 2 つの部分で構成されます:

- タグキー (、環境CostCenter、プロジェクト など) 。タグキーでは、大文字と小文字が区別されます。
- タグ値と呼ばれるオプションのフィールド (例: 111122223333 または Production) 。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値では大文字と小文字が区別されます。

タグは、以下のことに役立ちます。

- AWS リソースを特定して整理します。多くの はタグ付け AWS のサービス をサポートしているため、異なる サービスのリソースに同じタグを割り当てて、リソースが関連していることを示すことができます。
- AWS コストを追跡します。 AWS Billing and Cost Management ダッシュボードでこれらのタグをアクティブ化します。 はタグ AWS を使用してコストを分類し、毎月のコスト配分レポートを配

信します。詳細については、「AWS Billing and Cost Management ユーザーガイド」の「コスト配分タグを使用する」を参照してください。

- AWS リソースへのアクセスを制御します。詳細については、「タグを使用したアクセスの制御」を参照してください。

AWS および タグの詳細

- 命名規則や使用規則など、タグ付けに関する一般的な情報については、 AWS 全般のリファレンスの「 AWS リソースのタグ付け」を参照してください。
- タグ付けの制限については、「 AWS 全般のリファレンス」の「タグの命名制限と要件」を参照してください。
- ベストプラクティスとタグ付け戦略については、「タグ付けのベストプラクティス」とAWS 「タグ付け戦略」を参照してください。
- タグの使用をサポートするサービスのリストについては、「リResource Groups のタグ付け API リファレンス」を参照してください。

以下のセクションでは、AmazonBraket のタグに関する詳細情報を提供します。

Amazon Braket でサポートされるリソース

Amazon Braket の次のリソースタイプはタグ付けをサポートしています。

- `quantum-task` リソース:
- リソース名： `AWS::Service::Braket`
- ARN 正規表現： `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

注： Braket コンソールで Amazon AmazonBraket ノートブックのタグを適用および管理するには、コンソールを使用してノートブックリソースに移動しますが、ノートブックは実際には Amazon SageMaker リソースです。詳細については、 SageMaker ドキュメントの「ノートブックインスタンスメタデータ」を参照してください。

タグの制限

Amazon Braket リソースのタグには、次の基本的な制限が適用されます。

- リソースに割り当てることができるタグの最大数 :50
- キーの最大長: 128 文字 (ユニコード)
- 値の最大長: 256 文字 (ユニコード)
- キーと値の有効な文字:a-z, A-Z, 0-9, spaceと、次の文字。_ . : / = + - と @
- キーと値は大文字と小文字が区別されます。
- キーのプレフィックスawsとして を使用しないでください。 AWS 用に予約されています。

Amazon Braket でタグを管理する

リソースのプロパティとしてタグを設定します。Braket コンソール、AmazonBraket 、または Amazon を使用して、タグを表示、追加、変更、一覧表示API、削除できます AWS CLI。詳細については、「Amazon Braket API リファレンス」を参照してください。

タグを追加する

タグ付きリソースには、次の場合にタグを追加できます。

- リソースを作成する場合： コンソールを使用するか、 AWS API の Createオペレーションに Tagsパラメータを含めます。
- リソースを作成した後： コンソールを使用して量子タスクまたはノートブックリソースに移動するか、 AWS API で TagResourceオペレーションを呼び出します。

リソースの作成時にタグを追加するには、指定したタイプのリソースを作成する権限も必要です。

タグの表示

Amazon Braket のタグ付け可能なリソースのタグを表示するには、コンソールを使用してタスクまたはノートブックリソースに移動するか、 ListTagsForResourceAPIオペレーションを呼び出します AWS 。

次の AWS APIコマンドを使用して、リソースのタグを表示できます。

- AWS API: ListTagsForResource

タグの編集

コンソールを使用して量子タスクまたはノートブックリソースに移動するか、次のコマンドを使用してタグ付け可能なリソースにアタッチされたタグの値を変更することで、タグを編集できます。すでに存在するタグキーを指定すると、そのキーの値が上書きされます。

- AWS API: TagResource

タグを削除する

リソースからタグを削除するには、削除するキーを指定するか、 コンソールを使用して量子タスクまたはノートブックリソースに移動するか、 UntagResourceオペレーションを呼び出す必要があります。

- AWS API: UntagResource

Amazon Braket での CLI タグ付けの例

AWS CLI を使用している場合は、RigettiQPU のパラメータ設定SV1を使用して作成した量子タスクに適用されるタグを作成する方法を示すコマンドの例を次に示します。タグは、例のコマンドの最後に指定されていることに注目してください。この場合は、キーには state の値が与えられ、値には Washington の値が与えられます。

```
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\"}
```

Amazon Braket API を使用したタグ付け

- Amazon Braket を使用してリソースにタグAPIを設定する場合は、 を呼び出しますTagResourceAPI。

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

- リソースからタグを削除するには、 を呼び出しますUntagResourceAPI。

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

- 特定のリソースにアタッチされているすべてのタグを一覧表示するには、 を呼び出しますListTagsForResourceAPI。

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

Amazon Braket with Amazon のイベントと自動アクション EventBridge

Amazon は Amazon Braket 量子タスクのステータス変更イベントを EventBridge モニタリングします。Amazon Braket からのイベントは EventBridge、ほぼリアルタイムで に配信されます。簡単なルールを記述して、注目するイベント (イベントがルールに一致した場合に自動的に実行するアクションを含む) を指定できます。トリガーできる自動アクションには、次が含まれます。

- AWS Lambda 関数の呼び出し
- AWS Step Functions ステートマシンのアクティブ化
- Amazon SNS トピックへの通知

EventBridge は、以下の Amazon Braket ステータス変更イベントをモニタリングします。

- qauntum タスクの状態の変更

Amazon Braket は、量子タスクステータス変更イベントの配信を保証します。これらのイベントは少なくとも 1 回配信されますが、順序が乱れている可能性があります。

詳細については、「」の「イベントとイベントパターン EventBridge」を参照してください。

このセクションの内容:

で量子タスクのステータスをモニタリングする EventBridge

を使用すると EventBridge、Braket が Amazon Braket 量子タスクに関するステータス変更の通知を送信するときに実行するアクションを定義するルールを作成できます。例えば、量子タスクのステータスが変わるたびに E メールメッセージを送信するルールを作成できます。

1. EventBridge と Amazon Braket を使用するアクセス許可を持つ AWS アカウントを使用して にログインします。
2. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
3. 次の値を使用して、 EventBridge ルールを作成します。
   - ルールタイプ では、イベントパターンを持つルール を選択します。
   - Event source] (イベントソース) では、Other] (その他) を選択します。
   - [Event pattern] (イベントパターン) セクションで [Custom patterns (JSON editor)] (カスタムパターン (JSONエディター)) を選択し、次のイベントパターンをテキストエリアに貼付けます。

     ```
     {
       "source": [
         "aws.braket"
       ],
       "detail-type": [
         "Braket Task State Change"
       ]
     }
     ```

     Amazon Braket からすべてのイベントをキャプチャするには、次のコードに示すように `detail-type`セクションを除外します。

     ```
     {
       "source": [
     ```

```
    "aws.braket"
  ]
}
```

- ターゲットタイプ では AWS のサービスを選択し、ターゲット の選択 では Amazon SNS トピックや AWS Lambda 関数などのターゲットを選択します。ターゲットは、AmazonBraket から量子タスク状態変更イベントを受信するとトリガーされます。

  例えば、Amazon Simple Notification Service (SNS) トピックを使用して、イベントが発生したときに E メールまたはテキストメッセージを送信できます。これを行うには、Amazon SNS コンソールを使用して Amazon SNS トピックを作成する必要があります。詳細については、「ユーザー通知に Amazon SNS を使用する」を参照してください。

ルールの作成の詳細については、「イベント に反応する Amazon EventBridge ルールの作成」を参照してください。

Amazon Braket EventBridge イベントの例

Amazon Braket Quantum タスクステータス変更イベントのフィールドについては、「」の「イベントとイベントパターン EventBridge」を参照してください。

JSON の「詳細」フィールドには、次の属性が表示されます。

- **quantumTaskArn** (str): このイベントが生成された量子タスク。
- **status** (オプション[str]): 量子タスクが移行されたステータス。
- **deviceArn** (str): この量子タスクが作成されたユーザーによって指定されたデバイス。
- shots (int): ユーザーがshotsリクエストした の数。
- **outputS3Bucket** (str): ユーザーによって指定された出力バケット。
- **outputS3Directory** (str): ユーザーが指定した出力キープレフィックス。
- **createdAt** (str): ISO-8601 文字列としての量子タスクの作成時間。
- **endedAt** (オプション[str]): 量子タスクが終了状態になった時刻。このフィールドは、量子タスクが終了状態に移行した場合にのみ存在します。

次の JSON コードは、AmazonBraket Quantum Task Status Change イベントの例を示しています。

```
{
    "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"
      }
}
```


Amazon による Amazon Braket のモニタリング CloudWatch

Amazon Braket は CloudWatch、raw データを収集し、読み取り可能なほぼリアルタイムのメトリクスに処理する Amazon を使用してモニタリングできます。Amazon CloudWatch コンソールで過去 15 か月までに生成された履歴情報を表示するか、過去 2 週間に更新されたメトリクスを検索して、Amazon Braket の動作をより的確に把握できます。詳細については、「メトリクスの使用 CloudWatch 」を参照してください。

Amazon Braket のメトリクスとディメンション

メトリクスは、 の基本的な概念です CloudWatch。メトリクスは、 に発行される時系列のデータポイントのセットを表します CloudWatch。すべてのメトリクスは、一連のディメンションによって特徴付けられます。のメトリクスディメンションの詳細については CloudWatch、「 ディメンションCloudWatch 」を参照してください。

Amazon Braket は、Amazon Braket に固有の次のメトリクスデータを Amazon CloudWatch メトリクスに送信します。

量子タスクメトリクス

量子タスクが存在する場合、メトリクスを使用できます。 CloudWatch コンソールの AWS/Braket/By Device の下に表示されます。

メトリクス	説明
カウント	量子タスクの数。
レイテンシー	このメトリクスは、量子タスクが完了したときに出力されます。量子タスクの初期化から完了までの合計時間を表します。

量子タスクメトリクスのディメンション

量子タスクメトリクスは、arn:aws:braket::device/xxx という形式を持つ `deviceArn`パラメータに基づくディメンションで発行されます。

サポートされるデバイス

サポートされるデバイスおよびデバイス ARN のリストについては、「Braket デバイス」を参照してください。。

> **Note**
>
> Amazon Braket ノートブックの CloudWatch ログストリームを表示するには、Amazon SageMaker コンソールのノートブックの詳細ページに移動します。Amazon Braket ノートブックの追加設定は、SageMaker コンソール から利用できます。

を使用した Amazon Braket API ログ記録 CloudTrail

Amazon Braket は、ユーザー AWS CloudTrail、ロール、または Braket の によって実行されたアクションを記録するサービスである と統合 AWS のサービス されています。 Amazon は、AmazonBraket のすべてのAPI呼び出しをイベントとして CloudTrail キャプチャします。キャプチャされた呼び出しには、AmazonBraket コンソールからの呼び出しと、Braket Braket Amazon オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、AmazonBraket の

CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、 CloudTrail コンソールのイベント履歴 で最新のイベントを表示できます。によって収集された情報を使用して CloudTrail、AmazonBraket に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

の詳細については CloudTrail、「AWS CloudTrail ユーザーガイド」を参照してください。

の Amazon Braket 情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 で が有効になります。Amazon Braket でアクティビティが発生すると、そのアクティビティは CloudTrail イベント履歴 の他の AWS のサービス イベントとともにイベントに記録されます。 で最近のイベントを表示、検索、ダウンロードできます AWS アカウント。詳細については、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

Amazon Braket のイベントなど AWS アカウント、 のイベントの継続的な記録については、証跡を作成します。証跡により、 はログファイル CloudTrail を Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成するときに、証跡がすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、他の を設定 AWS のサービス して、 CloudTrail ログで収集されたイベントデータをより詳細に分析し、それに基づく対応を行うことができます。詳細については、次を参照してください:

- 証跡の作成のための概要
- CloudTrail サポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

すべての Amazon Braket アクションは によってログに記録されます CloudTrail。例えば、`GetQuantumTask`または `GetDevice`アクションを呼び出すと、 CloudTrail ログファイルにエントリが生成されます。

各イベントまたはログエントリには、誰がリクエストを生成したかという情報が含まれます。アイデンティティ情報は、以下を判別するのに役立ちます:

- リクエストがロールまたはフェデレーションユーザーのテンポラリなセキュリティ認証情報を使用して行われたかどうか。

- リクエストが、別の AWS のサービスによって送信されたかどうか。

詳細については、CloudTrail userIdentity Element」を参照してください。

Amazon Braket ログファイルエントリの概要

証跡は、指定した Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリックAPIコールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例では、量子タスクの詳細を取得する GetQuantumTask アクションのログエントリを示します。


```
{
  "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"
}
```

以下に、デバイスイベントの詳細を返す GetDevice アクションのログエントリを示します。

```
{
  "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"
}
```


を使用して Amazon Braket ノートブックインスタンスを作成する AWS CloudFormation

AWS CloudFormation を使用して、Amazon Braket ノートブックインスタンスを管理できます。Braket ノートブックインスタンスは Amazon 上に構築されています SageMaker。を使用すると CloudFormation、意図した設定を記述するテンプレートファイルを使用してノートブックインスタンスをプロビジョニングできます。テンプレートファイルは JSON または YAML 形式で記述されます。インスタンスの作成、更新、削除は、順序的かつ繰り返し可能な方法で行うことができます。これは、 で複数の Braket ノートブックインスタンスを管理する場合に便利です AWS アカウント。

Braket ノートブックの CloudFormation テンプレートを作成したら、 AWS CloudFormation を使用してリソースをデプロイします。詳細については、 ユーザーガイドの AWS CloudFormation 「コンソールでのスタックの作成AWS CloudFormation 」を参照してください。

を使用して Braket ノートブックインスタンスを作成するには CloudFormation、次の 3 つのステップを実行します。

1. Amazon SageMaker ライフサイクル設定スクリプトを作成します。
2. が引き受ける AWS Identity and Access Management (IAM) ロールを作成します SageMaker。
3. プレフィックスが付いた SageMaker ノートブックインスタンスを作成する **amazon-braket-**

ライフサイクル設定は、作成したすべての Braket ノートブックで再利用できます。また、同じ実行アクセス許可を割り当てた Braket ノートブックの IAM ロールを再利用することもできます。

ステップ 1: Amazon SageMaker ライフサイクル設定スクリプトを作成する

次のテンプレートを使用して、SageMaker ライフサイクル設定スクリプト を作成します。このスクリプトは Braket の SageMaker ノートブックインスタンスをカスタマイズします。ライフサイクル CloudFormation リソースの設定オプションについては、ユーザーガイドのAWS::SageMaker::NotebookInstanceLifecycleConfigAWS CloudFormation 「」を参照してください。

```
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
```

ステップ 2: Amazon が引き受ける IAM ロールを作成する SageMaker

Braket ノートブックインスタンスを使用する場合、 はユーザーに代わってオペレーション SageMaker を実行します。例えば、サポートされているデバイスで回路を使用して Braket ノートブックを実行するとします。ノートブックインスタンス内で、 は Braket で オペレーション SageMaker を実行します。ノートブック実行ロール SageMaker は、ユーザーに代わって実行できる正確なオペレーションを定義します。詳細については、「Amazon デベロッパーガイド」の「SageMaker ロール」を参照してください。 SageMaker

次の例を使用して、必要なアクセス許可を持つ Braket ノートブック実行ロールを作成します。必要に応じてポリシーを変更できます。

> **Note**
>
> ロールに、プレフィックスが の Amazon S3 バケットに対する s3:ListBucketおよび s3:GetObjectオペレーションのアクセス許可があることを確認しますbraketnotebookcdk-"。ライフサイクル設定スクリプトには、Braket ノートブックのインストールスクリプトをコピーするためのこれらのアクセス許可が必要です。

```
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: "*"
```

ステップ 3: プレフィックスを持つ Amazon SageMaker ノートブックインスタンスを作成する **amazon-braket-**

ステップ 1 とステップ 2 で作成した SageMaker ライフサイクルスクリプトと IAM ロールを使用して、 SageMaker ノートブックインスタンスを作成します。ノートブックインスタンスは Braket 用にカスタマイズされており、Amazon Braket コンソールからアクセスできます。この CloudFormation リソースの設定オプションの詳細については、ユーザーガイドのAWS::SageMaker::NotebookInstanceAWS CloudFormation 「」を参照してください。

```
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
```

高度なロギング

ロガーを使用して、タスク処理プロセス全体を記録できます。これらの高度なロギング技術により、バックグラウンドポーリングを確認し、後でデバッグするためのレコードを作成できます。

ロガーを使用するには、 パラメータpoll_timeout_secondsと poll_interval_secondsパラメータを変更して、量子タスクを長時間実行し、量子タスクのステータスを継続的にログに記録し、結果をファイルに保存することを推奨します。このコードを Jupyter ノートブックではなく Python スクリプトに転送して、スクリプトをバックグラウンドでプロセスとして実行できるようにします。

ロガーを設定する

まず、次の例の行に示すように、すべてのログがテキストファイルに自動的に書き込まれるように、ロガーを設定します。

```
# 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
```

回路を作成して実行する

これで、回路を作成し、実行するデバイスに送信して、この例に示すように何が起こっているかを確認できます。

```
# 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
)
```

ログファイルを確認する

次のコマンドを入力して、ファイルに書き込まれる内容を確認できます。

```
# 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})
```

ログファイルから ARN を取得する

前の例に示すように、返されるログファイルの出力から、ARN 情報を取得できます。ARN ID を使用すると、完了した量子タスクの結果を取得できます。

```
# parse log file for arn
with open(log_file) as openfile:
    for line in openfile:
        for part in line.split():
            if "arn:" in part:
                arn = part
                break
# remove final semicolon in logs
arn = arn[:-1]

# with this arn you can restore again task from unique arn
task_load = AwsQuantumTask(arn=arn, aws_session=AwsSession())

# get results of task
result = task_load.result()
```

Amazon Braket のセキュリティ

この章は、Amazon Braket の使用時に責任共有モデルがどのように適用されるかを理解するために役立ちます。ここでは、セキュリティやコンプライアンスに関する目標を達成できるようにAmazon Braket を設定する方法について説明します。また、Amazon Braket AWS のサービス リソースの監視と保護に役立つその他の使い方についても学びます。

AWS ではクラウドセキュリティが最優先事項です。セキュリティを最も重視する組織の要件を満たすために構築された AWS のデータセンターとネットワークアーキテクチャは、お客様に大きく貢献します。お客様は、お客様のデータの機密性、企業の要件、および適用可能な法律および規制などの他の要因についても責任を担います。

セキュリティの責任共有

セキュリティは、AWS とお客様の間の共有責任です。責任共有モデルでは、この責任がクラウドのセキュリティおよびクラウド内のセキュリティとして説明されています。

- クラウドのセキュリティ — AWS は、AWS クラウド で AWS のサービス を実行するインフラストラクチャを保護する責任を負います。また AWS は、お客様が使用するサービスを安全に提供します。サードパーティーの監査人は、AWS コンプライアンスプログラム の一環として、セキュリティの有効性を定期的にテストおよび検証します。Amazon Braket に適用されるコンプライアンスプログラムについては、「AWSコンプライアンスプログラム別の対象サービス」を参照してください。
- クラウドのセキュリティ — AWS このインフラストラクチャでホストされているコンテンツを管理する責任はお客様にあります。このコンテンツには、使用される AWS のサービスのセキュリティ構成と管理タスクが含まれます。

データ保護

AWShttps://aws.amazon.com/compliance/shared-responsibility-model/ Amazon Braket のデータ保護に適用されます。このモデルで説明されているように、AWS は、AWS クラウド のすべてを実行するグローバルインフラストラクチャを保護するがあります。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS のサービスのセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーのよくある質問」を参照してください。欧州でのデータ保護の詳細に

ついては、「AWS セキュリティブログ」に投稿された「AWS 責任共有モデルおよび GDPR」のブログ記事を参照してください。

データを保護するため、AWS アカウント の認証情報を保護し、AWS IAM Identity Center または AWS Identity and Access Management (IAM) を使用して個々のユーザーをセットアップすることをお勧めします。この方法により、それぞれのジョブを遂行するために必要な権限のみを各ユーザーに付与できます。また、次の方法でデータを保護することをおすすめします。

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。TLS 1.2 が必須です。TLS 1.3 が推奨されます。
- AWS CloudTrail で API とユーザーアクティビティロギングをセットアップします。
- AWS のサービス内でデフォルトである、すべてのセキュリティ管理に加え、AWS の暗号化ソリューションを使用します。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは API により AWS にアクセスするときに FIPS 140-2 検証済み暗号化モジュールが必要な場合は、FIPS エンドポイントを使用します。利用可能な FIPS エンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-2」を参照してください。

顧客の E メールアドレスなどの機密情報や重要情報は、タグや Name フィールドなどの自由形式のフィールドに入力しないことを強くお勧めします。これには、コンソール、API AWS CLI、または AWS SDK AWS のサービス を使用して Amazon Braket やその他のツールを操作する場合も含まれます。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

データ保持

90 日後、Amazon Braket はすべての量子タスク ID と量子タスクに関連するその他のメタデータを自動的に削除します。このデータ保持ポリシーの結果として、これらのタスクと結果は S3 バケットに保存されたままですが、Amazon Braket コンソールからの検索では取得できなくなります。

S3 バケットに 90 日以上保存されている過去の量子タスクや結果にアクセスする必要がある場合は、タスク ID とそのデータに関連するその他のメタデータを別途記録しておく必要があります。必ず 90 日前までに情報を保存してください。その保存された情報を使用して、履歴データを取得できます。

Amazon Braket へのアクセスを管理する

この章では、Amazon Braket を実行したり、特定のユーザーやロールのアクセスを制限したりするために必要な権限について説明します。必要な権限は、アカウント内のどのユーザーまたはロールにも付与 (または拒否) できます。そのためには、以下のセクションで説明するように、アカウントのそのユーザーまたはロールに適切な Amazon Braket ポリシーをアタッチします。

前提条件として、Amazon Braket を有効にします。Braket を有効にするには、必ず (1) 管理者権限を持つユーザーまたはロールとしてサインインするか、(2) AmazonBraketFullAccessポリシーが割り当てられていて Amazon Simple Storage Service (Amazon S3) バケットを作成する権限を持つユーザーまたはロールとしてサインインしてください。

このセクションの内容:

- Amazon Braket のリソース
- ノートブックとロール
- AmazonBraketFullAccessポリシーについて
- AmazonBraketJobsExecutionPolicyポリシーについて
- 特定のデバイスへのユーザーアクセスを制限する
- Amazon Braket AWS の管理ポリシーの更新
- 特定のノートブックインスタンスへのユーザーアクセスを制限します。
- 特定の S3 バケットへのユーザーアクセスを制限します。

Amazon Braket のリソース

Braket は、クォンタムタスクリソースという 1 つのタイプのリソースを作成します。このリソースタイプの Amazon リソースネーム (ARN) は次のとおりです。

- リソース名:: サービスAWS:: ブラケット
- ARN 正規表現:arn: $ {パーティション}: ブラケット:$ {地域}: $ {アカウント}: quantum-task/$ {} RandomId

ノートブックとロール

Braket ではノートブックリソースタイプを使用できます。ノートブックは Braket が共有できる Amazon SageMaker リソースです。Braket でノートブックを使用するには、で始まる名前の IAM ロールを指定する必要があります。AmazonBraketServiceSageMakerNotebook

ノートブックを作成するには、管理者権限を持つロール、または以下のインラインポリシーがアタッチされたロールを使用する必要があります。

```
{
   "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*"
                   ]
               }
```

```
            }
        }
    ]
}
```

ロールを作成するには、「ノートブックの作成」ページの手順に従うか、管理者に作成を依頼してください。AmazonBraketFullAccessポリシーが添付されていることを確認します。

ロールを作成したら、今後起動するすべてのノートブックでそのロールを再利用できます。

AmazonBraketFullAccessポリシーについて

AmazonBraketFullAccessこのポリシーは、以下のタスクに対するアクセス権限を含む Amazon Braket オペレーションのアクセス権限を付与します。

- Amazon Elastic Container レジストリからコンテナをダウンロードする — Amazon Braket ハイブリッドジョブ機能に使用されるコンテナイメージを読み込んでダウンロードします。コンテナは「arn: aws: ecr:: repository/amazon-braket」の形式に準拠している必要があります。
- AWS CloudTrail ログを保存 — クエリの開始と停止、メトリクスフィルタのテスト、ログイベントのフィルタリングに加えて、説明、取得、一覧表示のすべてのアクションに適用されます。 AWS CloudTrail ログファイルには、アカウントで発生したすべての Amazon Braket API アクティビティの記録が含まれます。
- ロールを利用してリソースを管理 — アカウントにサービスにリンクされたロールを作成します。サービスにリンクされたロールは、 AWS ユーザーに代わってリソースにアクセスできます。Amazon Braket サービスでのみ使用できます。また、IAM ロールを Amazon Braket `CreateJob` API に渡し、ロールを作成して、 AmazonBraketFullAccess そのロールにスコープが設定されたポリシーをアタッチすることもできます。
- アカウントの使用状況ログファイルを管理するために、ロググループ、ログイベント、クエリログループを作成する — アカウント内の Amazon Braket の使用状況に関するロギング情報を作成、保存、表示する。ハイブリッドジョブのロググループのクエリメトリクス。適切な Braket パスを含め、ログデータを入力できるようにする。メトリックデータを入力してください。 CloudWatch
- Amazon S3 バケットにデータを作成して保存し、すべてのバケットを一覧表示する — S3 バケットを作成するには、アカウント内の S3 バケットを一覧表示し、名前が amazon-braket-で始まるアカウントのバケットにオブジェクトを入れたり取得したりします。Braket が処理済みの量子タスクの結果を含むファイルをバケットに入れたり、バケットから取得したりするには、これらの権限が必要です。
- IAM ロールを渡す — IAM ロールをに渡します。`CreateJob` API

- Amazon SageMaker ノートブック —「arn: aws: sagemaker:: notebook-instance/amazon-braket-」SageMaker のリソースを対象とするノートブックインスタンスを作成および管理します。
- サービスクォータの検証 — SageMaker ノートブックと Amazon Braket Hybrid ジョブを作成するには、リソース数がアカウントのクォータを超えることはできません。

ポリシーの内容

```
{
    "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"
                }
            }
        }
    ]
}
```

AmazonBraketJobsExecutionPolicyポリシーについて

AmazonBraketJobsExecutionPolicyこのポリシーは、Amazon Braket ハイブリッドジョブで使用される実行ロールに次のようにアクセス権限を付与します。

- Amazon Elastic コンテナレジストリからのコンテナのダウンロード-Amazon Braket ハイブリッドジョブ機能に使用されるコンテナイメージを読み取り、ダウンロードする権限。コンテナは「arn: aws: ecr: *: repository/amazon-braket*」の形式に準拠している必要があります。
- アカウントの使用状況ログファイルを管理するために、ロググループを作成し、イベントを記録し、ロググループをクエリします。アカウントの Amazon Braket の使用状況に関するログ情報を作成、保存、表示します。ハイブリッドジョブのロググループのクエリメトリクス。適切な Braket パスを含め、ログデータを入力できるようにする。メトリックデータを入力してください。 CloudWatch
- Amazon S3 バケットにデータを保存 — アカウント内の S3 バケットを一覧表示し、名前に amazon-braket-で始まるアカウント内の任意のバケットにオブジェクトを入力、取得します。これらの権限は、処理された量子タスクの結果を含むファイルをBraketがバケットに入れたり、バケットから取得したりするために必要です。
- IAM ロールを渡す — IAM ロールをに渡します。 CreateJob APIロールは arn: aws: iam:: * * の形式に従う必要があります。: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"
    }
   }
  }
 ]
}
```

特定のデバイスへのユーザーアクセスを制限する

特定のユーザーによる特定の Braket デバイスへのアクセスを制限するには、特定のロールにアクセス権限拒否ポリシーを追加できます。 IAM

このような権限では以下のアクションを制限できます。

- `CreateQuantumTask`-指定したデバイスでの量子タスク作成を拒否する。
- `CreateJob`-指定したデバイスでのハイブリッドジョブの作成を拒否します。
- `GetDevice`-指定したデバイスの詳細情報の取得を拒否する。

次の例では、のすべての QPU へのアクセスを制限しています。 AWS アカウント `123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
  ]
}
```

このコードを適用するには、Amazon前の例で示した文字列を制限対象デバイスのリソース番号 (ARN) に置き換えます。この文字列は、リソース値を指定します。Braket では、デバイスとは、量

子タスクを実行するために呼び出すことができる QPU またはシミュレーターを表します。使用可能なデバイスは、デバイスページに一覧表示されます。これらのデバイスへのアクセスを指定するために使用するスキーマは 2 つあります。

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

さまざまなタイプのデバイスアクセスの例を次に示します。

- すべてのリージョンですべての QPU を選択するには次の手順を実行します。`arn:aws:braket:*:*:device/qpu/*`
- us-west-2 リージョンのすべての QPU のみを選択するには次の手順を実行します。`arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- 同様に、us-west-2 リージョンのすべての QPU のみを選択する場合 (デバイスはサービスリソースであり、顧客リソースではないため): `arn:aws:braket:us-west-2:* :device/qpu/*`
- すべてのオンデマンドシミュレータデバイスへのアクセスを制限するには:`arn:aws:braket:* :123456789012:device/quantum-simulator/*`
- us-east-1 IonQ Harmony リージョンのデバイスへのアクセスを制限するには:`arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- 特定のプロバイダーのデバイス (デバイスなど) へのアクセスを制限するには RigettiQPU:`arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- TN1デバイスへのアクセスを制限するには:`arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

Amazon Braket AWS の管理ポリシーの更新

次の表は、このサービスが変更の追跡を開始してからの Braket AWS の管理ポリシーの更新に関する詳細を示しています。

変更	説明	日付
AmazonBraketFullAccess-Braket のフルアクセスポリシー	ポリシーにサービスクォータ:GetServiceQuotaアクションと cloudwatch: GetMetric	2023 年 3 月 24 日

変更	説明	日付
	Data アクションを追加しました。AmazonBraketFullAccess .	
AmazonBraketFullAccess-Braket のフルアクセスポリシー	ブラケット調整済み iam: PassRole AmazonBraketFullAccess パスを含めるための権限。service-role/	2021 年 11 月 29 日
AmazonBraketJobsExecutionPolicy-Amazon Braket ハイブリッドジョブのハイブリッドジョブ実行ポリシー	Braket は、パスを含むようにハイブリッドジョブ実行ロール ARN を更新しました。service-role/	2021 年 11 月 29 日
Braket は変更の追跡を開始しました。	Braket AWS は管理ポリシーの変更を追跡し始めました。	2021 年 11 月 29 日

特定のノートブックインスタンスへのユーザーアクセスを制限します。

特定のユーザーによる特定の Braket ノートブックインスタンスへのアクセスを制限するには、特定のロール、ユーザー、またはグループにアクセス許可拒否ポリシーを追加できます。

以下の例では、ポリシー変数を使用して、内の特定のノートブックインスタンスを起動、停止、およびアクセスするための権限を効率的に制限しています。このインスタンスには AWS アカウント 123456789012、アクセス権が必要なユーザー (たとえば、という名前のノートブックインスタンスへのアクセス権が付与されます) amazon-braket-Alice に基づいて名前が付けられています。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}*"
      ]
    }
  ]
}
```

特定の S3 バケットへのユーザーアクセスを制限します。

特定のユーザーのアクセスを特定の Amazon S3 バケットに制限するには、特定のロール、ユーザー、またはグループに拒否ポリシーを追加できます。

次の例では、S3特定のバケット (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) にオブジェクトを取得して配置する権限を制限し、それらのオブジェクトのリストも制限しています。

```
{
  "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/*"
      ]
    }
  ]
}
```

特定のノートブックインスタンスのバケットへのアクセスを制限するには、前述のポリシーをノートブック実行ロールに追加します。

Amazon Braket のサービスリンクロール

Amazon Braket を有効にすると、サービスリンクロールがアカウント内に作成されます。

サービスリンクロールは、この場合、Amazon Braket に直接リンクされた特殊なタイプの IAM ロールです。Amazon Braketのサービスにリンクされたロールは、ユーザーに代わって他のユーザーに電話をかけるときにBraketが必要とするすべての権限を含むように事前に定義されています。AWS のサービス

必要な許可を手動で追加する必要がないため、サービスリンクロールは Amazon Braket のセットアップを容易にします。Amazon Braket は、サービスリンクロールのアクセス許可を定義します。これらの定義を変更しない限り Amazon Braketのみがロールを引き受けることができます。定義されたアクセス許可には、信頼ポリシーとアクセス許可ポリシーが含まれます。アクセス許可ポリシーを他の IAM エンティティにアタッチすることはできません。

Amazon Braket が設定するサービスにリンクされたロールは、AWS Identity and Access Management (IAM) サービスにリンクされたロール機能の一部です。AWS のサービスサービスにリ

ンクされたロールをサポートする他のサービスについては、「IAM AWS と連携するサービス」を参照し、「サービスにリンクされたロール」列で「はい」と表示されているサービスを探してください。サービスリンクロールに関するドキュメントをサービスで表示するには、[Yes] (はい) リンクを選択します。

Amazon Braket のサービスリンクロール許可

Amazon Braket では、braket.amazonaws.com AWSServiceRoleForAmazonBraket エンティティを信頼するサービスにリンクされたロールを使用してロールを引き受けます。

IAM エンティティ (グループやロールなど) がサービスにリンクされたロールを作成、編集、削除できるようにアクセス権限を設定する必要があります。詳細については、「サービスにリンクされたロールの権限」を参照してください。

Amazon Braket のサービスリンクロールには、デフォルトで次のアクセス許可が付与されます。

- Amazon S3 — アカウント内のバケットを一覧表示する権限。また、名前が amazon-braket-で始まるアカウント内の任意のバケットにオブジェクトを格納したり、そのバケットからオブジェクトを取得したりする権限。
- Amazon CloudWatch Logs — ロググループの一覧表示と作成、関連するログストリームの作成、Amazon Braket 用に作成されたロググループへのイベントの入力を行う権限。

AWSServiceRoleForAmazonBraketサービスにリンクされたロールには次のポリシーが添付されています。

```
{"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"
                }
            }
        }
    ]
}
```

Amazon Braket の耐障害性

AWS のグローバルインフラストラクチャは AWS リージョン とアベイラビリティーゾーンを中心として構築されます。

各リージョンには物理的に独立して、隔離されている複数のアベイラビリティーゾーンが用意されています。アベイラビリティーゾーン (AZ) は、低レイテンシー、高スループット、そして高度の冗長ネットワークで接続されています。その結果、アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AZ 間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。

アベイラビリティーゾーンの詳細については、「AWSグローバルインフラストラクチャ」を参照してください。AWS リージョン

Amazon Braket のコンプライアンス検証

第三者監査人は、Amazon Braket のセキュリティとコンプライアンス、およびサードパーティのハードウェアプロバイダーとの統合を定期的に評価しています。up-to-dateBraketのコンプライア

ンス情報のリストについては、「コンプライアンスプログラム別の対象範囲」を参照してください AWS のサービス。一般的な情報については、「AWSコンプライアンス」を参照してください。

AWS Artifact を使用して、サードパーティーの監査レポートをダウンロードできます。詳細については、の「レポートのダウンロード」を参照してくださいAWS Artifact。

> **Note**
>
> AWSコンプライアンスレポートには、独立した監査を受けることを選択できるサードパーティのハードウェアプロバイダーの QPU は対象外です。

Amazon Braket を使用する際のお客様のコンプライアンス責任は、データの機密性、会社のコンプライアンス目標、および適用される法律や規制によって決まります。 AWSコンプライアンスに役立つ以下のリソースを提供します。

- セキュリティとコンプライアンスのクイックスタートガイド - これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、機密性とコンプライアンスに焦点を当てたベースライン環境を AWS にデプロイするためのステップが示されています。
- AWS コンプライアンスのリソース - ワークブックとお客様の業界や所在地に適用される場合があるガイドのコレクション。

Amazon Braket でのインフラストラクチャセキュリティ

マネージドサービスである Amazon Braket は、「セキュリティプロセスの概要」AWS ホワイトペーパーに記載されているグローバルネットワークセキュリティ手順によって保護されていますAWS。

ネットワーク経由で Amazon Braket にアクセスするには、公開されている AWS API を呼び出します。クライアントで Transport Layer Security (TLS) 1.2 以降がサポートされている必要があります。また、Ephemeral Diffie-Hellman (DHE) や Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) などの Perfect Forward Secrecy (PFS) を使用した暗号スイートもクライアントでサポートされている必要があります。これらのモードは、Java 7 以降など、最近のほとんどのシステムでサポートされています。

また、リクエストは、アクセスキー ID と、IAM プリンシパルに関連付けられているシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Braket ハードウェアプロバイダーのセキュリティ

Amazon Braket の QPU は、サードパーティーのハードウェアプロバイダーによってホストされています。量子タスクを QPU 上で実行する場合、Amazon Braket は回路を指定された QPU に送信して処理されます。

サードパーティーハードウェアプロバイダーの 1 つが運営する量子コンピューティングハードウェアへのアクセスに Amazon Braket を使用する場合、お客様の回路とその関連データは、サードパーティーハードウェアプロバイダーによって運営する施設外のハードウェアプロバイダーによって処理されます。AWS。物理的な場所に関する情報とAWS各 QPU が利用できる地域は、以下のとおりです。デバイス詳細Amazon Braket コンソールのセクション。

コンテンツは匿名化されています。回路の処理に必要なコンテンツのみがサードパーティーに送信されます。AWS アカウント情報は第三者に送信されません。

すべてのデータは、保管時と転送時のいずれも暗号化されます。データは処理のためだけに復号されます。Amazon Braket サードパーティープロバイダーは、お客様の回路の処理以外の目的でお客様のコンテンツを保存または使用することは許可されていません。回路の完了後、結果は Amazon Braket に返され、S3 バケットに保存されます。

Amazon Braket サードパーティーの量子ハードウェアプロバイダーのセキュリティは、ネットワークセキュリティ、アクセスコントロール、データ保護、および物理的セキュリティの基準が満たされていることを確認するために、定期的に監査されます。

Amazon Braket 用の VPC エンドポイント

インターフェイス VPC エンドポイントを作成することで、VPC と Amazon Braket の間にプライベート接続を確立できます。インターフェースエンドポイントは、インターネットゲートウェイAWS PrivateLink、NAT デバイス、VPN 接続、または接続なしで Braket API にアクセスできるテクノロジーを利用しています。AWS Direct ConnectVPC のインスタンスは、パブリック IP アドレスがなくても Braket API と通信できます。

各インターフェースエンドポイントは、サブネット内の 1 つ以上の Elastic Network Interface によって表されます。

これにより PrivateLink、VPC と Braket Amazon 間のトラフィックがネットワーク外に出ることはなく、データがパブリックインターネットにさらされる機会が減るため、クラウドベースのアプリケーションと共有するデータのセキュリティが向上します。詳細については、Amazon VPC ユーザーガイドの「インターフェイス VPC エンドポイント (AWS PrivateLink)」を参照してください。

Amazon Braket VPC エンドポイントに関する考慮事項

Braket 用の VPC エンドポイントを設定する前に、Amazon VPC ユーザーガイドの「インターフェイスエンドポイントのプロパティと制限」を確認してください。

Braket は、VPC からのすべての API アクションの呼び出しをサポートしています。

デフォルトでは、VPC エンドポイントを通じた Braket へのフルアクセスが許可されています。VPC エンドポイントポリシーを指定すれば、アクセスをコントロールできます。詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

Braket をセットアップし、 PrivateLink

AmazonBraket AWS PrivateLink で使用するには、Amazon Virtual Private Cloud (Amazon VPC) エンドポイントをインターフェースとして作成し、Amazon API Braket サービスを通じてエンドポイントに接続する必要があります。

ここでは、このプロセスの一般的なステップを示します。これについては、後のセクションで詳しく説明します。

- Amazon VPC を設定して起動し、AWSリソースをホストします。VPC が既にある場合、このステップは省略できます。
- Braket 用の Amazon VPC エンドポイントを作成します
- Braket の量子タスクをエンドポイントConnect して実行する

ステップ 1: 必要に応じて Amazon VPC を起動する

アカウントに既に VPC が運用されている場合は、このステップを省略できることにご注意ください。

VPC は、IP アドレス範囲、サブネット、ルートテーブル、ネットワークゲートウェイなどのネットワーク設定をコントロールできます。基本的に、AWSカスタム仮想ネットワークでリソースを起動します。VPC の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

Amazon VPC コンソールを開いて、サブネット、セキュリティグループ、ネットワークゲートウェイを含む新しい VPC を作成します。

ステップ 2: Braket のインターフェイス VPC エンドポイントの作成

Braket サービス用の VPC エンドポイントは、Amazon VPC コンソールまたは () のいずれかを使用して作成できます。AWS Command Line Interface AWS CLI詳細については、「Amazon VPC ユーザーガイド」のインターフェースエンドポイントの作成を参照してください。

コンソールで VPC エンドポイントを作成するには、Amazon VPC コンソールを開き、エンドポイントページを開いて、新しいエンドポイントの作成に進みます。後で参照できるように、エンドポイント ID を書きとめます。Braket への特定の呼び出しを行う際には、–endpoint-urlフラグの一部として必要です。API

Braket 用の VPC エンドポイントを作成するには、次のサービス名を使用します。

- `com.amazonaws.substitute_your_region.braket`

注:エンドポイントのプライベート DNS を有効にすると、リージョンのデフォルト DNS 名 (例:) を使用して Braket API にリクエストを行うことができます。`braket.us-east-1.amazonaws.com`

詳細については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントを介したサービスへのアクセス」を参照してください。

ステップ 3: Braket 量子タスクをエンドポイントConnect して実行する

VPC エンドポイントを作成したら、次の例のように、endpoint-urlAPIまたはランタイムへのインターフェイスエンドポイントを指定するパラメータを含む CLI コマンドを実行できます。

```
aws braket search-quantum-tasks --endpoint-url
 VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

VPC エンドポイントのプライベート DNS ホスト名を有効にした場合は、CLI コマンドで URL をエンドポイントとして指定する必要はありません。代わりに、CLI と Amazon Braket SDK がデフォルトで使用する Braket API DNS ホスト名が VPC エンドポイントに解決されます。その形式は、次の例のようにします。

```
https://braket.substituteYourRegionHere.amazonaws.com
```

「AWS PrivateLinkエンドポイントを使用して Amazon VPC から Amazon SageMaker ノートブックに直接アクセスする」というブログ記事では、Braket ノートブックと同様に、 SageMaker ノートブックに安全に接続するためのエンドポイントの設定方法の例を紹介しています。Amazon

ブログ投稿の手順に従っている場合は、AmazonAmazonの代わりにBraketという名前を使用することを忘れないでください。 SageMaker[サービス名] に、リージョンが us-east-1 でない場合は、*com.amazonaws.us-east-1.braket*AWS リージョンその文字列に正しい名前を入力するか、正しい名前に置き換えてください。

エンドポイントの作成の詳細

- VPC をプライベートサブネットで作成する方法については、「プライベートサブネットを持つ VPC を作成する」を参照してください。
- Amazon VPC コンソールまたは AWS CLI を使用して、エンドポイントを作成および設定する方法については、「Amazon VPC ユーザーガイド」の「インターフェイスエンドポイントの作成」を参照してください。
- を使用してエンドポイントを作成および設定する方法についてはAWS CloudFormation、ユーザーガイドの:: EC2:AWS: VpcEndpoint リソースを参照してください。AWS CloudFormation

Amazon VPC エンドポイントポリシーによるアクセスのコントロール

AmazonBraket への接続アクセスを制御するには、AWS Identity and Access Management (IAM) エンドポイントポリシーを Amazon VPC エンドポイントにアタッチできます。このポリシーでは、以下の情報を指定します。

- アクションを実行できるプリンシパル (ユーザーまたはロール)
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「Amazon VPC ユーザーガイド」の「VPC エンドポイントでサービスへのアクセスを制御する」を参照してください。

例: Braket アクションの VPC エンドポイントポリシー

Braket の VPC エンドポイントポリシーの例を以下に示します。このポリシーは、エンドポイントに添付されると、すべてのリソースのすべてのプリンシパルに対して、登録されている Braket アクションへのアクセスを許可します。

```
{
 "Statement":[
 {
```

```
    "Principal":"*",
    "Effect":"Allow",
    "Action":[
      “braket:action-1",
      “braket:action-2",
      “braket:action-3”
      ],
    "Resource":"*"
    }
  ]
}
```

複数のエンドポイントポリシーを添付することで、複雑な IAM ルールを作成できます。詳細な説明と例については、以下を参照してください。

- Step Functions の Amazon Virtual Private Cloud エンドポイントポリシー
- 管理者以外のユーザー用の詳細な IAM アクセス許可の作成
- VPC エンドポイントによるサービスのアクセス制御

Amazon Braket のトラブルシューティング

このセクションのトラブルシューティング情報と解決策を使用して、Amazon Braket の問題を解決します。

このセクションの内容:

AccessDeniedException

Braket を有効化または使用AccessDeniedExceptionするときに を受け取った場合、制限されたロールにアクセスできないリージョンで Braket を有効化または使用しようとしている可能性があります。

このような場合は、内部 AWS 管理者に連絡して、次の条件のうちどれに該当するかを理解する必要があります。

- リージョンへのアクセスを妨げるロール制限がある場合。
- 使用しようとしているロールに Braket の使用が許可されている場合。

Braket の使用時にロールが特定のリージョンにアクセスできない場合、その特定のリージョンでデバイスを使用することはできません。

CreateQuantumTask オペレーションを呼び出すときにエラーが発生しました (ValidationException)

次のようなエラーが表示された場合は、既存の `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-braket-…` s3_folder を参照していることを確認します。Braket は、新しい Amazon S3 バケットとプレフィックスを自動的に作成しません。

API に直接アクセスしていて、次のような エラーが表示されている場合は、Amazon `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` S3 バケットパス`s3://`に を含めていないことを確認します。 Amazon S3

SDK 機能が動作しません

Python バージョンは 3.9 以降である必要があります。Amazon Braket Hybrid Jobs の場合は、Python 3.10 をお勧めします。

SDK とスキーマが であることを確認します up-to-date。ノートブックまたは Python エディタから SDK を更新するには、次のコマンドを実行します。

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

スキーマを更新するには、次のコマンドを実行します。

```
pip install amazon-braket-schemas --upgrade
```

独自のクライアントから Amazon Braket にアクセスする場合は、AWS リージョンが Amazon Braket でサポートされているリージョンに設定されていることを確認します。

ハイブリッドジョブが により失敗する ServiceQuotaExceededException

ターゲットとするシミュレーターデバイスの同時量子タスク制限を超えると、Amazon Braket シミュレーターに対して量子タスクを実行するハイブリッドジョブが作成されない可能性があります。サービスの制限の詳細については、「クォータ」トピックを参照してください。

アカウントから複数のハイブリッドジョブでシミュレーターデバイスに対して同時タスクを実行している場合、このエラーが発生する可能性があります。

特定のシミュレーターデバイスに対する同時量子タスクの数を確認するには、次のコード例に示すようにsearch-quantum-tasksAPI、 を使用します。

```
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
```

Amazon CloudWatch メトリクスを使用して、デバイスに対して作成された量子タスクを表示することもできます: Braket > Device 別。

これらのエラーが発生しないようにするには：

1. シミュレーターデバイスの同時量子タスクの数に対するサービスクォータの引き上げをリクエストします。これはSV1デバイスにのみ適用されます。
2. コード内の ServiceQuotaExceeded の例外を処理し、再試行してください。

ノートブックインスタンスでコンポーネントが動作しなくなった

ノートブックの一部のコンポーネントが機能しない場合は、以下を試してください。

1. 作成または変更したノートブックをローカルドライブにダウンロードします。
2. ノートブックインスタンスを停止します。
3. ノートブックインスタンスを削除します。
4. 別の名前で新しいノートブックインスタンスを作成します。
5. ノートブックを新しいインスタンスにアップロードします。

Amazon Braket のクォータ

次のテーブルに Amazon Braket のサービスクォータの一覧を示します。サービスクォータ (制限とも呼ばれます) は、 AWS アカウントのサービスリソースまたはオペレーションの最大数です。

一部のクォータは増やすことができます。詳細については、AWS のサービス クォータを参照してください。

- バーストレートクォータを増やすことはできません。
- 調整可能なクォータの最大レート増加 (調整できないバーストレートを除く) は、指定されたデフォルトのレート制限の2倍です。例えば、デフォルトのクォータの 60 個は最大 120 個に調整できます。
- 同時 SV1 (DM1) 量子タスクの調整可能なクォータは、 あたり最大 60 個まで可能です AWS リージョン。
- ハイブリッドジョブで許可されるコンピューティングインスタンスの最大数は 5 で、クォータは調整可能です。

リソース	説明	制限	調整可能
API リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりのリクエストの最大数。	140	はい
API リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加リクエストの最大数 (RPS) 。	600	いいえ
`CreateQuantumTask` リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの `CreateQuantumTask` リクエストの最大数。	20	はい

リソース	説明	制限	調整可能
CreateQuantumTask リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の CreateQuantumTask リクエストの最大数 (RPS)。	40	いいえ
SearchQuantumTasks リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの SearchQuantumTasks リクエストの最大数。	5	はい
SearchQuantumTasks リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の SearchQuantumTasks リクエストの最大数 (RPS)。	50	いいえ
GetQuantumTask リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの GetQuantumTask リクエストの最大数。	100	はい

リソース	説明	制限	調整可能
GetQuantumTask リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の GetQuantumTask リクエストの最大数 (RPS)。	500	いいえ
CancelQuantumTask リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの CancelQuantumTask リクエストの最大数。	2	はい
CancelQuantumTask リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の CancelQuantumTask リクエストの最大数 (RPS)。	20	いいえ
GetDevice リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの GetDevice リクエストの最大数。	5	はい

リソース	説明	制限	調整可能
GetDevice リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の GetDevice リクエストの最大数 (RPS)。	50	いいえ
SearchDevices リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの SearchDevices リクエストの最大数。	5	はい
SearchDevices リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の SearchDevices リクエストの最大数 (RPS)。	50	いいえ
CreateJob リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの CreateJob リクエストの最大数。	1	はい

リソース	説明	制限	調整可能
CreateJob リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の CreateJob リクエストの最大数 (RPS)。	5	いいえ
SearchJob リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの SearchJob リクエストの最大数。	5	はい
SearchJob リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の SearchJob リクエストの最大数 (RPS)。	50	いいえ
GetJob リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの GetJob リクエストの最大数。	5	はい
GetJob リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の GetJob リクエストの最大数 (RPS)。	25	いいえ

リソース	説明	制限	調整可能
CancelJob リクエストのレート	現在のリージョンのこのアカウントで送信できる 1 秒あたりの CancelJob リクエストの最大数。	2	はい
CancelJob リクエストのバーストレート	現在のリージョンのこのアカウントで 1 回のバーストで送信できる 1 秒あたりの追加の CancelJob リクエストの最大数 (RPS)。	5	いいえ
同時SV1量子タスクの数	現在のリージョンでステートベクトルシミュレーター (SV1) で実行されている同時量子タスクの最大数。	100 us-east-1、 50 us-west-1、 100 us-west-2、 50 eu-west-2	いいえ
同時DM1量子タスクの数	現在のリージョンで密度行列シミュレーター (DM1) で実行されている同時量子タスクの最大数。	100 us-east-1、 50 us-west-1、 100 us-west-2、 50 eu-west-2	いいえ
同時TN1量子タスクの数	現在のリージョンでテンソルネットワークシミュレーター (TN1) で実行されている同時量子タスクの最大数。	10 us-east-1、 10 us-west-2、 5 eu-west-2、	はい

リソース	説明	制限	調整可能
同時ハイブリッドジョブの数	現在のリージョンでの同時ハイブリッドジョブの最大数。	5	はい
ハイブリッドジョブのランタイム制限	ハイブリッドジョブを実行できる日数の最大時間。	5	いいえ

Hybrid Jobs のデフォルトのクラシックコンピューティングインスタンスクォータを次に示します。これらのクォータを引き上げるには、 にお問い合わせください AWS Support。さらに、使用可能なリージョンはインスタンスごとに指定されます。

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c4.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c4.xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c4.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c4.2xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c4.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c4.4xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c4.8xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c4.8xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5.xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5.2xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5.4xlarge タイプのインスタンスの最大数。	1	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5.9xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5.9xlarge タイプのインスタンスの最大数。	1	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5.18xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5.18xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5n.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5n.xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5n.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5n.2xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5n.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5n.4xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5n.9xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5n.9xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.c5n.18xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.c5n.18xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.2xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.4xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.8xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.8xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.12xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.12xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.g4dn.16xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.g4dn.16xlarge タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m4.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m4.xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m4.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m4.2xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m4.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m4.4xlarge タイプのインスタンスの最大数。	2	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m4.10› large インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m4.10› large タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m4.16› large インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m4.16› large タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.large インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.large タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.2xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.4xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.4xlarge タイプのインスタンスの最大数。	5	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.12› large インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.12› large タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.m5.24› large インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.m5.24› large タイプのインスタンスの最大数。	0	はい	はい	はい	はい	はい	はい

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p2.xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p2.xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p2.8xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p2.8xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p2.16xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p2.16xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p3.2xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p3.2xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p4d.24xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p4d.24xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p3dn.24xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p3dn.24xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	いいえ	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p3.8xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p3.8xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	はい	いいえ

リソース	説明	制限	調整可能	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
ハイブリッドジョブの ml.p3.16xlarge インスタンスの最大数	このアカウントとリージョンのすべての Amazon Braket Hybrid Jobs で許可される ml.p3.16xlarge タイプのインスタンスの最大数。	0	はい	はい	いいえ	はい	はい	いいえ

制限更新のリクエスト

インスタンスタイプの ServiceQuotaExceeded 例外を受け取り、十分なインスタンスが利用できない場合は、 AWS コンソールの Service Quotas ページから制限の引き上げをリクエストし、 AWS サービスで Amazon Braket を検索できます。

> **Note**
>
> ハイブリッドジョブがリクエストされた ML コンピューティング容量をプロビジョニングできない場合は、別のリージョンを使用します。さらに、テーブルにインスタンスが表示されない場合、ハイブリッドジョブでは使用できません。

追加のクォータと制限

- Amazon Braket 量子タスクアクションのサイズは 3MB に制限されています。
- SV1、、DM1および Rigetti デバイスで許可されるタスクあたりの最大ショット数は 100,000 です。
- で許可されるタスクあたりの最大ショット数は 1000 TN1です。
- IonQの Aria-1 および Aria-2 デバイスの場合、最大はタスクあたり 5,000 ショットです。IonQの「Harmony」、「Forte」、およびOQC「デバイス」の場合、最大値は 10,000 です。
- の場合QuEra、タスクあたりの最大許容ショット数は 1000 です。
- TN1 および QPU デバイスの場合、タスクあたりのショットは > 0 である必要があります。

OpenQASM のトラブルシューティング

このセクションでは、OpenQASM 3.0 を使用してエラーが発生した場合に役立つトラブルシューティングポインタについて説明します。

このセクションの内容:

- ステートメントを含めるエラー
- 連続しないqubitsエラー
- 物理エラーqubitsと仮想qubitsエラーの混在
- 同じプログラムエラーqubitsで結果タイプをリクエストして測定する
- Classical および qubit register limits exceeded エラー
- ボックスの前に逐語的なプラグマエラーが発生していない
- 逐語的なボックスのネイティブゲート欠落エラー
- 逐語的なボックスの欠落による物理qubitsエラー
- 逐語的なプラグマに「ブラケット」エラーがない
- 単一 はインデックス作成qubitsできないエラー
- 2 つのqubitゲートqubitsの物理 が接続されていないエラー
- GetDevice は OpenQASM 結果エラーを返しません
- Local Simulator のサポートに関する警告

ステートメントを含めるエラー

Braket には、現在、OpenQASM プログラムに含める標準のゲートライブラリファイルはありません。例えば、次の例ではパーサーエラーが発生します。

```
OPENQASM 3;
include "standardlib.inc";
```

このコードはエラーメッセージを生成します。`No terminal matches '"' in the current parser context, at line 2 col 17.`

連続しないqubitsエラー

デバイス機能qubitsで にrequiresContiguousQubitIndices設定されたデバイスで連続しない`true` を使用すると、エラーが発生します。

シミュレーターと で量子タスクを実行するとIonQ、次のプログラムによってエラーがトリガーされます。

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

このコードはエラーメッセージを生成します。`Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

物理エラーqubitsと仮想qubitsエラーの混在

同じプログラムqubitsで物理qubitsと仮想を混在させることは許可されず、エラーが発生します。次のコードはエラーを生成します。

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

このコードはエラーメッセージを生成します。`[line 4] mixes physical qubits and qubits registers.`

同じプログラムエラーqubitsで結果タイプをリクエストして測定する

同じプログラムで明示的に測定qubitsされる結果タイプ と をリクエストすると、エラーが発生します。次のコードはエラーを生成します。

```
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])
```

このコードはエラーメッセージを生成します。`Qubits should not be explicitly measured when result types are requested.`

Classical および qubit register limits exceeded エラー

許可されるクラシックレジスターは 1 つだけで、qubitレジスターは 1 つだけです。次のコードはエラーを生成します。

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

このコードはエラーメッセージを生成します。`[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

ボックスの前に逐語的なプラグマエラーが発生していない

すべてのボックスの前に逐語的なプラグマを付ける必要があります。次のコードはエラーを生成します。

```
box{
```

```
rx(0.5) $0;
}
```

このコードはエラーメッセージを生成します。 `In verbatim boxes, native gates are required. x is not a device native gate.`

逐語的なボックスのネイティブゲート欠落エラー

逐語的なボックスにはネイティブゲートと物理 が必要ですqubits。次のコードはネイティブゲートエラーを生成します。

```
#pragma braket verbatim
box{
x $0;
}
```

このコードはエラーメッセージを生成します。 `In verbatim boxes, native gates are required. x is not a device native gate.`

逐語的なボックスの欠落による物理qubitsエラー

逐語的なボックスには物理 が必要ですqubits。次のコードは、欠落している物理qubitsエラーを生成します。

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

このコードはエラーメッセージを生成します。 `Physical qubits are required in verbatim box.`

逐語的なプラグマに「ブラケット」エラーがない

逐語的なプラグマには「ブラケット」を含める必要があります。次のコードはエラーを生成します。

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

このコードはエラーメッセージを生成します。 `You must include “braket” in the verbatim pragma`

単一 はインデックス作成qubitsできないエラー

単一 はインデックス作成qubitsできません。次のコードはエラーを生成します。

```
OPENQASM 3;

qubit q;
h q[0];
```

このコードはエラーを生成します。 `[line 4] single qubit cannot be indexed.`

ただし、単一qubit配列は次のようにインデックスを作成できます。

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

2 つのqubitゲートqubitsの物理 が接続されていないエラー

物理 を使用するにはqubits、まずデバイスが物理 を使用していることを確認し qubitsdevice.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits、次に `device.properties.paradigm.connectivity.connectivityGraph`または を確認して接続グラフを確認しますdevice.properties.paradigm.connectivity.fullyConnected。

```
OPENQASM 3;

cnot $0, $14;
```

このコードはエラーメッセージを生成します。 `[line 3] has disconnected qubits 0 and 14`

GetDevice は OpenQASM 結果エラーを返しません

Braket SDK の使用時に OpenQASM の結果が GetDevice レスポンスに表示されない場合は、AWS_EXECUTION_ENV 環境変数を設定してユーザーエージェントを設定する必要があります。Go および Java SDKs でこれを行う方法については、以下のコード例を参照してください。

AWS_EXECUTION_ENV 環境変数を設定して、 の使用時にユーザーエージェントを設定するには 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]
```

Boto3 を使用するときにユーザーエージェントを設定するように AWS_EXECUTION_ENV 環境変数を設定するには：

```
import boto3
import botocore

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

JavaScript/TypeScript (SDK v2) を使用するときに AWS_EXECUTION_ENV 環境変数を設定してユーザーエージェントを設定するには：

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

JavaScript/TypeScript (SDK v3) を使用するときにユーザーエージェントを設定するように AWS_EXECUTION_ENV 環境変数を設定するには：

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

Go SDK を使用するときにユーザーエージェントを設定するように AWS_EXECUTION_ENV 環境変数を設定するには：

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

Java SDK を使用するときにユーザーエージェントを設定するように AWS_EXECUTION_ENV 環境変数を設定するには：

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

Local Simulator のサポートに関する警告

は OpenQASM の高度な機能LocalSimulatorをサポートしていますが、QPUsやオンデマンドシミュレーターでは使用できない場合があります。次の例に示すようにLocalSimulator、プログラムに のみに固有の言語機能が含まれている場合、警告が表示されます。

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

このコードは警告を生成します。`このプログラムは、 でのみサポートされている OpenQASM 言語機能を使用します LocalSimulator。これらの機能の一部は、QPUsまたはオンデマンドシミュレーターではサポートされていない場合があります。

サポートされている OpenQASM 機能の詳細については、こちら をクリックします。

アマゾンブラケットの API と SDK リファレンスガイド

Amazon Braket には、ノートブックインスタンスの作成と管理、モデルのトレーニングとデプロイに使用できる API、SDK、コマンドラインインターフェイスが用意されています。

- アマゾンブラケットパイソン SDK (推奨)
- アマゾンブラケット API リファレンス
- 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

Amazon Braket GitHub チュートリアルのリポジトリからコード例を入手することもできます。

- ブラケットチュートリアル GitHub

ドキュメント履歴

以下の表は、Amazon Braket の今回のリリースのドキュメント内容をまとめたものです。

- API バージョン： 2022 年 4 月 28 日
- API リファレンスの最終更新日： 2023 年 9 月 25 日
- ドキュメントの最終更新日： 2024 年 5 月 22 日

変更	説明	日付
新しいデバイスIQM Garnetとリージョン Europe North 1	I"" Garnet デバイスのサポートが追加されました。正方形の格子トポロジーを備えた 20 量子ビットデバイス。Braket がサポートするリージョンを欧州北部 1 (ストックホルム) に拡張しました。	2024 年 5 月 22 日
ローカルデチューニングのリリース	実験的な機能には、 の QuEraAquila QPU のローカル調整機能が含まれるようになりました。	2024 年 4 月 11 日
ノートブックの非アクティブマネージャーのリリース	ノートブックインスタンスを作成するときは、非アクティブマネージャーを有効にし、アイドル時間を設定して Braket ノートブックインスタンスを自動的にリセットします。	2024 年 3 月 27 日
目次の再作業	Amazon Braket の目次を再編成して、 AWS スタイルガイドの要件に準拠し、カスタマーエクスペリエンスのコ	2023 年 12 月 12 日

	ンテンツフローを改善しました。	
Braket Direct のリリース	Braket direct 機能のサポートが追加されました。以下が含まれます。 • 予約 • 専門家のアドバイス • 実験的な機能	2023 年 11 月 27 日
「Amazon Braket ノートブックインスタンスを作成する」を更新	ドキュメントを更新して、新規および既存の Amazon Braket のお客様向けのノートブックインスタンスを作成するための情報を追加しました。	2023 年 11 月 27 日
「独自のコンテナ」を更新	ドキュメントを更新して、いつ BYOC にするか、レシピを BYOC にするか、コンテナで Braket Hybrid Jobs を実行するかに関する情報を追加しました。	2023 年 10 月 18 日

ハイブリッドジョブデコレータのリリース	ローカルコードをハイブリッドジョブとして実行する ページを追加しました。例を示します。 • ローカル Python コードからハイブリッドジョブを作成する • 追加の Python パッケージとソースコードをインストールする • ハイブリッドジョブインスタンスにデータを保存してロードする • ハイブリッドジョブデコレータのベストプラクティス	2023 年 10 月 16 日
キューの可視性を追加	デベロッパーガイドのドキュメントを更新して、 queue depthと を含めましたqueue position。 キューの可視性のための新しい API の変更を反映するように API ドキュメンテーションを更新しました。	2023 年 9 月 25 日
ドキュメントの命名を標準化する	「ジョブ」のインスタンスを「ハイブリッドジョブ」に変更し、「タスク」を「量子タスク」に変更するようにドキュメントを更新しました。	2023 年 9 月 11 日
新しいデバイス IonQ Aria 2	IonQ Aria 2 デバイスのサポートを追加	2023 年 9 月 8 日

ネイティブゲートの更新	ドキュメントを更新して、Rigetti からネイティブゲートへのプログラムによるアクセスに関する情報を追加しました。	2023 年 8 月 16 日
Xanadu 出発	ドキュメントを更新してすべてのXanaduデバイスを削除	2023 年 6 月 2 日
新しいデバイス IonQ Aria	IonQ Aria デバイスのサポートを追加	2023 年 5 月 16 日
リタイアしたRigettiデバイス	のサポートの中止 Rigetti Aspen-M-2	2023 年 5 月 2 日
AmazonBraketFullAccess 更新されたポリシー情報	servicequotas: および cloudwatch:GetServiceQuota アクションと、クォータに関する制限に関する情報を含めるようにAmazonBraketFullAccessポリシーの内容を定義するスクリプトを更新しました。 GetMetricData	2023 年 4 月 19 日
ガイド付きジャーニーの起動	Braket オンボーディングのより最新のシンプルな方法を反映するようにドキュメントを変更しました。	2023 年 4 月 5 日
新しいデバイス Rigetti Aspen-M-3	Rigetti Aspen-M-3 デバイスのサポートを追加	2023 年 1 月 17 日
新しい結合勾配機能	が提供する結合勾配機能に関する情報を追加しました。 SV1	2022 年 12 月 7 日

アルゴリズムライブラリの新しい機能	事前に構築された量子アルゴリズムのカタログを提供する Braket アルゴリズムライブラリに関する情報を追加しました。	2022 年 11 月 28 日
D-Wave 出発	すべてのD-Waveデバイスの削除に対応するためにドキュメントを更新しました	2022 年 11 月 17 日
新しいデバイス QuEra Aquila	QuEra Aquila デバイスのサポートを追加	2022 年 10 月 31 日
Braket Pulse のサポート	Braket Pulse のサポートが追加されました。これにより、Rigettiおよび OQC デバイスでの Pulse 制御の使用が可能になります。	2022 年 10 月 20 日
IonQ ネイティブゲートのサポート	IonQ デバイスが提供するネイティブゲートセットのサポートを追加	2022 年 9 月 13 日
新しいインスタンスクォータ	Hybrid Jobs に関連付けられたデフォルトのクラシックコンピューティングインスタンスクォータを更新しました	2022 年 8 月 22 日
新しいサービスダッシュボード	サービスダッシュボードを含むようにコンソールのスクリーンショットを更新	2022 年 8 月 17 日
新しいデバイス Rigetti Aspen-M-2	Rigetti Aspen-M-2 デバイスのサポートを追加	2022 年 8 月 12 日

OpenQASM の新機能	ローカルシミュレーター (braket_sv および braket_dm) の OpenQASM 機能のサポートを追加しました。	2022 年 8 月 4 日
新しいコスト追跡手順	シミュレーターとハードウェアワークロードの最大コストをほぼリアルタイムで見積もる方法を追加しました。	2022 年 7 月 18 日
新しいXanadu Borealisデバイス	Xanadu Borealis デバイスのサポートを追加	2022 年 6 月 2 日
新しいオンボーディングの簡素化手順	新しいオンボーディング手順と簡略化されたオンボーディング手順の仕組みに関する情報を追加しました。	2022 年 5 月 16 日
新しいデバイス D-Wave Advantage_system6.1	D-Wave Advantage_system6.1 デバイスのサポートを追加	2022 年 5 月 12 日
埋め込みシミュレーターのサポート	ハイブリッドジョブで埋め込みシミュレーションを実行する方法と PennyLane 、Lightning Simulator を使用する方法を追加しました。	2022 年 5 月 4 日
AmazonBraketFullAccess - Amazon Braket のフルアクセスポリシー	ユーザーが Amazon Braket 用に作成および使用されたバケットを表示および検査できるようにする s3:ListAllMyBuckets permissions を追加しました	2022 年 3 月 31 日

OpenQASM のサポート	ゲートベースの量子デバイスとシミュレーターに対する OpenQASM 3.0 サポートを追加	2022 年 3 月 7 日
新しい量子ハードウェアプロバイダーOxford Quantum Circuits、新しいリージョン、eu-west-2	OQC と eu-west-2 のサポートを追加	2022 年 2 月 28 日
新しいRigettiデバイス	Rigetti Aspen M-1 のサポートの追加	2022 年 2 月 15 日
新しいリソース制限	同時DM1タスクと SV1タスクの最大数を 55 から 100 に増やしました	2022 年 1 月 5 日
新しいRigettiデバイス	Rigetti Aspen-11 のサポートの追加	2021 年 12 月 20 日
リタイアしたRigettiデバイス	Rigetti Aspen-10 デバイスのサポートを終了しました	2021 年 12 月 20 日
新しい結果タイプ	局所密度行列シミュレーターとDM1デバイスでサポートされる低密度行列の結果タイプ	2021 年 12 月 20 日
ポリシーの説明を更新しました	Amazon Braket は、`servicerole/` ロール ARN を更新してパスを含めました。ポリシーの更新の詳細については、Amazon Braket による マネージドポリシーの更新 AWS」表を参照してください。	2021 年 11 月 29 日

Amazon Braket Jobs	Amazon Braket Hybrid Jobs のユーザーガイドを追加APIしました	2021 年 11 月 29 日
新しいRigettiデバイス	Rigetti Aspen-10 のサポートの追加	2021 年 11 月 20 日
リタイアしたD-Waveデバイス	D-Wave QPU のサポートを終了しました。 Advantage_system1	2021 年 11 月 4 日
新しいD-Waveデバイス	追加の D-Wave QPU のサポートが追加されました。 Advantage_system4	2021 年 10 月 5 日
新しいノイズシミュレーター	最大 17 の回路をシミュレートできる密度行列シミュレーター (DM1) qubitsとローカルノイズシミュレーター braket_dm のサポートを追加	2021 年 5 月 25 日
PennyLane サポート	Amazon Braket PennyLane での のサポートを追加	2020 年 12 月 8 日
新しいシミュレーター	より大きな回路を可能にする Tensor Network Simulator (TN1) のサポートを追加	2020 年 12 月 8 日
タスクバッチ処理	Braket はカスタマータスクのバッチ処理をサポート	2020 年 11 月 24 日
手動qubit割り当て	Braket がRigettiデバイスへの手動qubit割り当てをサポート	2020 年 11 月 24 日
調整可能なクォータ	Braket は、タスクリソースのセルフサービス調整可能クォータをサポートします。	2020 年 10 月 30 日

のサポート PrivateLink	Braket ジョブ用のプライベート VPC エンドポイントを設定できます。	2020 年 10 月 30 日
タグのサポート	Braket は量子タスクリソースの APIベースのタグをサポートします	2020 年 10 月 30 日
新しいD-Waveデバイス	追加の D-Wave QPU のサポートが追加されました。Advantage_system1	2020 年 9 月 29 日
初回リリース	Amazon Braket ドキュメントの初回リリース	2020 年 8 月 12 日

AWS 用語集

AWS最新の用語については、AWS用語集リファレンスの用語集を参照してください。AWS

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。