Amazon Braket
Developer Guide
## Table of Contents

What Is Amazon Braket? .......................................................................................................................... 1
  Amazon Braket terms and concepts .................................................................................................................. 1
  AWS terminology and tips for Amazon Braket ................................................................................................. 2
How it works .................................................................................................................................................. 5
  Amazon Braket task flow ................................................................................................................................ 6
  Third-party data processing ............................................................................................................................. 6
Supported devices ........................................................................................................................................... 8
  IonQ ............................................................................................................................................................... 9
  D-Wave ....................................................................................................................................................... 10
  Local state vector simulator (braket_sv) ........................................................................................................ 10
  Local density matrix simulator (braket_dm) .................................................................................................... 10
  State vector simulator (SV1) ......................................................................................................................... 11
  Density matrix simulator (DM1) .................................................................................................................... 11
  Tensor network simulator (TN1) .................................................................................................................... 12
  Compare simulators ...................................................................................................................................... 13
Regions and endpoints ........................................................................................................................................ 16
Pricing .............................................................................................................................................................. 17
  Best practices for cost savings .......................................................................................................................... 17
Quotas ............................................................................................................................................................. 18
  Additional quotas and limits ............................................................................................................................ 20
When will my task run? .................................................................................................................................... 21
  Status change notifications in email or SMS .................................................................................................. 21
  QPU availability windows and status ............................................................................................................. 21
Enable Amazon Braket ..................................................................................................................................... 22
  Prerequisites ................................................................................................................................................ 22
  Steps to enable Amazon Braket ..................................................................................................................... 22
  Common causes of failure when enabling Amazon Braket ........................................................................ 22
Get started ....................................................................................................................................................... 24
  Create an Amazon Braket notebook instance ............................................................................................. 24
  Run your first circuit using the Amazon Braket Python SDK ..................................................................... 25
  Run your first annealing problem with Ocean ............................................................................................. 28
Work with Amazon Braket ............................................................................................................................. 31
  Construct circuits in the SDK .......................................................................................................................... 32
    Gates and circuits ....................................................................................................................................... 32
    Manual qubit allocation ................................................................................................................................. 35
    Verbatim compilation .................................................................................................................................. 36
    Noise simulation ......................................................................................................................................... 36
    Inspecting the circuit ................................................................................................................................. 37
    Result types ................................................................................................................................................ 38
  Submitting tasks to QPUs and simulators .................................................................................................... 40
    Example tasks on Amazon Braket .............................................................................................................. 41
    Submitting tasks to a QPU .......................................................................................................................... 44
    Running a task with the local simulator ...................................................................................................... 45
    Task batching ............................................................................................................................................ 46
    Set up SNS notifications (optional) ........................................................................................................... 47
  Monitoring and tracking tasks ....................................................................................................................... 47
    Tracking tasks from the Braket SDK ......................................................................................................... 47
    Advanced logging ....................................................................................................................................... 49
    Monitoring tasks through the Amazon Braket console .............................................................................. 50
Working with Boto3 ........................................................................................................................................ 53
  Enable the Amazon Braket Boto3 client ....................................................................................................... 54
  Configure AWS CLI profiles for Boto3 and the Amazon Braket SDK ............................................................ 56
Run a hybrid algorithm .................................................................................................................................... 58
What Is Amazon Braket?

Amazon Braket is a fully-managed AWS service that helps researchers, scientists, and developers get started with quantum computing. Quantum computing has the potential to solve computational problems that are beyond the reach of classical computers, because it harnesses the laws of quantum mechanics to process information in new ways.

Gaining access to quantum computing hardware can be expensive and inconvenient. Limited access makes it difficult to run algorithms, optimize designs, evaluate the current state of the technology, and plan for when to invest your resources for maximum benefit. Amazon Braket helps you overcome these challenges.

Amazon Braket offers a single point of access to a variety of quantum computing technologies. It enables you to:

- Explore and design quantum and hybrid algorithms
- Test algorithms on different quantum circuit simulators
- Run algorithms on different types of quantum computers
- Create proof of concept applications

Defining quantum problems and programming quantum computers to solve them requires a new set of skills. To help you gain these skills, Amazon Braket offers different environments to simulate and run your quantum algorithms. You can find the approach that best suits your requirements, and you can get started quickly with a set of example environments, called notebooks.

Amazon Braket development has three aspects — Build, Test, and Run:

**Build** - Amazon Braket provides fully-managed Jupyter notebook environments that make it easy to get started. Amazon Braket notebooks are pre-installed with sample algorithms, resources, and developer tools, including the Amazon Braket SDK. With the Amazon Braket SDK, you can build quantum algorithms, then test and run them on different quantum computers and simulators by changing a single line of code.

**Test** - Amazon Braket provides access to fully-managed, high-performance, quantum circuit simulators. You can test and validate your circuits. Amazon Braket handles all the underlying software components and EC2 clusters to take away the burden of simulating quantum circuits on classical HPC infrastructure.

**Run** - Amazon Braket provides secure, on-demand access to different types of quantum computers. You have access to gate-based quantum computers from IonQ and Rigetti, as well as a quantum annealer from D-Wave. You have no upfront commitment, and no need to procure access with individual providers.

**About quantum computing and Amazon Braket**

Quantum computing is in early developmental stages. It's important to understand that no universal, fault-tolerant quantum computer exists at present. Therefore, certain types of quantum hardware are better suited for certain use cases, and it is crucial to have access to a variety of computing hardware. Amazon Braket offers a variety of hardware, through third-party providers.

Existing quantum hardware is limited due to noise, which introduces errors. The industry is in the Noisy Intermediate Scale Quantum (NISQ) era. In the NISQ era, quantum computing devices are too noisy to sustain pure quantum algorithms, such as Shor's algorithm or Grover's algorithm. Until better quantum
error correction is available, the most practical quantum computing requires the combination of classical (traditional) computing resources with quantum computers, to create hybrid algorithms. Amazon Braket helps you work with hybrid quantum algorithms.

In hybrid quantum algorithms, quantum processing units (QPUs) are used as co-processors for CPUs, thus speeding up specific calculations in a classical algorithm. These algorithms utilize iterative processing, in which computation moves between classical and quantum computers. For example, current applications of quantum computing in chemistry, optimization, and machine learning are based on variational quantum algorithms, which are a type of hybrid quantum algorithm. In variational quantum algorithms, classical optimization routines adjust the parameters of a parameterized quantum circuit iteratively, much in the same way the weights of a neural network are adjusted iteratively, based on the error in a machine learning training set. Amazon Braket offers access to the PennyLane open source software library, which assists you with variational quantum algorithms.

Quantum computing is gaining traction for computations in four main areas:

- **Number theory** — including factoring and cryptography. (For example, Shor’s algorithm is a primary quantum method for number theory computations.)
- **Optimization** — including constraint satisfaction, solving linear systems, and machine learning.
- **Oracular computing** — including search, hidden subgroups, and order finding. (For example, Grover’s algorithm is a primary quantum method for oracular computations.)
- **Simulation** — including direct simulation, knot invariants, and quantum approximate optimization algorithm (QAOA) applications.

Applications for these categories of computations can be found in financial services, biotechnology, manufacturing, and pharmaceuticals, to name a few. Amazon Braket offers capabilities and notebook examples that can apply to many proof of concept problems, and certain practical problems, today.

Amazon Braket terms and concepts

The following terms and concepts are used in Amazon Braket:

**Braket**

We named our service after the bra-ket notation, a standard notation in quantum mechanics. It was introduced by Paul Dirac in 1939 to describe the state of quantum systems, and it is also known as the Dirac notation.

**Quantum computer**

A quantum computer is a physical device that uses quantum-mechanical phenomena such as superposition and entanglement to perform computations. There are different paradigms to quantum computing (QC), such as, gate-based QC or quantum annealing.

**Qubit**

The basic unit of information in a quantum computer is called a qubit (quantum bit), in analogy to classical bits. A qubit is a two-level quantum system that can be realized by different physical implementations, such as superconducting circuits, or individual ions and atoms. Other qubit types are based on photons, electronic or nuclear spins, or more exotic quantum systems.

**Gate-based Quantum Computing**

In gate-based QC (also called circuit-based QC), computations are broken down into elementary operations (gates). It can be shown that certain sets of gates are universal, meaning that every computation can be expressed as a finite sequence of those gates. Gates are the building blocks of quantum circuits, in analogy to the logic gates of classical digital circuits.
Quantum Annealing

Quantum annealing is a form of special purpose quantum computing that tries to utilize quantum fluctuations to find global minima of an objective function. In most approaches, the objective function that is encoded directly in the physical couplings parameters of the qubits. Quantum annealing is mainly used for combinatorial optimization problems (e.g., QUBO problems), where one has a finite and discrete search space.

Device

In Amazon Braket, a device is a backend that can execute quantum tasks. A device can be a QPU or a quantum circuit simulator. To learn more, see Amazon Braket supported devices (p. 8).

Quantum Circuit Simulator

A quantum circuit simulator is a computer program that runs on classical computers and calculates the measurement outcomes of a quantum circuit. For general circuits, the resource requirements of a quantum simulation grows exponentially with the number of qubits to simulate. Amazon Braket provides access to both managed (accessed through the Braket API) and local (part of the Amazon Braket SDK) quantum circuit simulators.

Quantum Processing Unit (QPU)

A QPU is a physical quantum computing device that can execute a quantum task. QPUs can be based on different QC paradigms, e.g., gate-based QC or quantum annealing. To learn more, see Amazon Braket supported devices (p. 8).

Quantum Circuit

A quantum circuit is the instruction set that defines a computation on a gate-based quantum computer. A quantum circuit is a sequence of quantum gates (which are reversible transformations on a qubit register) together with measurement instructions.

Shots

Since quantum computing is inherently probabilistic, any circuit (or annealing schedule) needs to be evaluated multiple times to get an accurate results. A single circuit execution and measurement is called a shot. The number of shots (repeated executions) for circuit is chosen based on the desired accuracy for the result. The number of shots can range from 10 to 100,000 shots per task.

Quantum Task

In Amazon Braket, a quantum task is the atomic request to a device. For gate-based QC devices, this includes the quantum circuit (including the measurement instructions and number of shots), and other request metadata. You can create quantum tasks through Amazon Braket SDK or by using the CreateQuantumTask API operation directly. After you create a task, it will be queued until the requested device becomes available. You can view your quantum tasks on the Tasks page of the Amazon Braket console, or by using the GetQuantumTask or SearchQuantumTasks API operations.

QPU supported gates

QPU supported gates are the gates accepted by the QPU device. These gates might not be able to directly run on the QPU, meaning that they might need to be decomposed into native gates. You can find the supported gates of a device on the Devices page in the Amazon Braket console and through the Braket SDK.

QPU native gates

QPU native gates are the gates that can be directly mapped to control pulses by the QPU control system. Native gates can be run on the QPU device without further compilation. Subset of QPU supported gates. You can find the native gates of a device on the Devices page in the Amazon Braket console and through the Braket SDK.
AWS terminology and tips for Amazon Braket

IAM users

An IAM user is an identity that you create in AWS. It represents the person or application that interacts with AWS services and resources. It consists of a name and credentials. By default, when you create a new IAM user in AWS, it has no permissions associated with it. To allow the IAM user to perform specific actions in AWS, such as launching an Amazon EC2 instance or creating an Amazon S3 bucket, you must grant the IAM user the necessary permissions.

- **Best practice:** We recommend that you create an individual IAM user for each person who needs access to AWS. Even if you have multiple employees who require the same level of access, create individual IAM users for each of them. This approach provides additional security by allowing each IAM user to have a unique set of security credentials.

IAM policies

An IAM policy is a document that allows or denies permissions to AWS services and resources. IAM policies enable you to customize users’ levels of access to resources. For example, you can allow users access to all of the Amazon S3 buckets within your AWS account, or only a specific bucket.

- **Best practice:** Follow the security principle of least privilege when granting permissions. By following this principle, you help to prevent users or roles from having more permissions than needed to perform their tasks. For example, if an employee needs access to only a specific bucket, specify the bucket in the IAM policy. Do this instead of granting the employee access to all of the buckets in your AWS account.

IAM roles

An IAM role is an identity that you can assume to gain temporary access to permissions. Before an IAM user, application, or service can assume an IAM role, they must be granted permissions to switch to the role. When someone assumes an IAM role, they abandon all previous permissions that they had under a previous role and assume the permissions of the new role.

- **Best practice:** IAM roles are ideal for situations in which access to services or resources needs to be granted temporarily, instead of long-term.

Amazon S3 bucket

Amazon Simple Storage Service (Amazon S3) is an AWS service that lets you store data as objects in buckets. Amazon S3 buckets offer unlimited storage space. The maximum size for an object in an Amazon S3 bucket is 5 TB. You can upload any type of file data to an Amazon S3 bucket, such as images, videos, text files, backup files, media files for a website, archived documents, and your Braket task results.

- **Best practice:** You can set permissions to control access to your S3 bucket. For more information, see Bucket policies and user policies in the Amazon S3 documentation.
How Amazon Braket works

Amazon Braket provides on-demand access to quantum computing devices, including managed circuit simulators and different types of QPUs. In Amazon Braket, the atomic request to a device is a task. For gate-based QC devices, this includes the quantum circuit (including the measurement instructions and number of shots), and other request metadata. For annealing devices it includes the problem definition, the number of shots, and other optional parameters.

In this section, we are going to learn about the high-level flow of executing tasks on Amazon Braket.
To make it easy for customers to define, submit, and monitor their tasks, Amazon Braket provides managed Jupyter notebooks (1) that come pre-installed with the Amazon Braket SDK. You can build
your quantum circuits directly in the SDK or, for annealing devices, define the annealing problem and parameter. The Amazon Braket SDK also provides a plugin for D-Wave's Ocean tool suite, so you can natively program the D-Wave device. After your task is defined, you can choose a device to execute it on, and submit it to the Amazon Braket API (2). Depending on the device you chose, the task is queued until the device becomes available and the task is sent to the QPU or simulator for execution (3). Amazon Braket gives you access to 3 different types of QPUs (D-Wave, IonQ, Rigetti) and one managed Simulator, SV1. To learn more, see Amazon Braket supported devices (p. 8).

After your task is processed, Amazon Braket returns the results to an Amazon S3 bucket, where the data is stored in your AWS account (4). At the same time, the SDK polls for the results in the background and loads them into the Jupyter notebook at task completion. You can also view and manage your tasks on the Tasks page in the Amazon Braket console, or by using the GetQuantumTask operation of the Amazon Braket API.

Of course, Amazon Braket is integrated with Amazon Identity and Access Management (IAM), Amazon CloudWatch, Amazon CloudTrail and Amazon EventBridge for user access management, monitoring and logging, as well as, for event based processing (5).

Third-party data processing

Tasks that are submitted to a QPU device, process on quantum computers located in facilities operated by third party providers. To learn more about Security and third-party processing in Amazon Braket, see Security of Amazon Braket Hardware Providers (p. 68).
Amazon Braket supported devices

In Amazon Braket, a device represents a QPU or simulator that you can call to run quantum tasks. That is, a device refers to a set of circuits for gate-based quantum computing, or it refers to an annealing problem for a quantum annealer device.

Amazon Braket provides access to four QPU devices—from D-Wave, IonQ, and Rigetti—and three simulator devices. For all devices, you can find further device properties, such as device topology, calibration data, and native gate sets in the Amazon Braket console in the Devices tab or by means of the GetDevice API.

If you are working with the Amazon Braket SDK, you have access to device properties as shown in the following code example:

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

device = AwsDevice('arn:aws:braket:::device/quantum-simulator/amazon/sv1')  # SV1
# device = LocalSimulator()  # Local
# device = LocalSimulator("default")  # Local
# device = LocalSimulator(backend="default")  # Local
# device = LocalSimulator(backend="braket_sv")  # Local
# device = LocalSimulator(backend="braket_dm")  # Local
# device = AwsDevice('arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6')  # D-Wave 2000Q
# device = AwsDevice('arn:aws:braket:::device/qpu/d-wave/Advantage_system1')  # D-Wave Advantage_system
# device = AwsDevice('arn:aws:braket:::device/qpu/ionq/ionQdevice')  # IonQ
# device = AwsDevice('arn:aws:braket:::device/qpu/rigetti/Aspen-9')  # Aspen-9

# get device properties
device.properties
```

**Supported QPUs:**
- IonQ
- Rigetti Aspen-9
- D-Wave 2000Q
- D-Wave Advantage_system

**Supported Simulators:**
- Local simulator ('Default Simulator')
- State vector simulator (SV1)
- Density matrix simulator (DM1)
- Tensor network simulator (TN1)
Note
To view the available AWS Regions for each device, you can scroll the following table horizontally toward the right.

Amazon Braket devices

<table>
<thead>
<tr>
<th>Provider</th>
<th>Device Name</th>
<th>Paradigm</th>
<th>Type</th>
<th>Device ARN</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-Wave</td>
<td>Advantage_system1</td>
<td>quantum annealer</td>
<td>QPU</td>
<td>arn:aws:braket:::device/qpu/d-wave/Advantage_system1</td>
<td>us-west-2</td>
</tr>
<tr>
<td>IonQ</td>
<td>ionQdevice</td>
<td>gate-based</td>
<td>QPU</td>
<td>arn:aws:braket:::device/qpu/ionq/ionQdevice</td>
<td>us-east-1</td>
</tr>
<tr>
<td>Rigetti</td>
<td>Aspen-9</td>
<td>gate-based</td>
<td>QPU</td>
<td>arn:aws:braket:::device/qpu/rigetti/Aspen-9</td>
<td>us-west-1</td>
</tr>
<tr>
<td>AWS</td>
<td>braket_sv</td>
<td>gate-based</td>
<td>Simulator</td>
<td>N/A (local simulator in Braket SDK)</td>
<td>N/A</td>
</tr>
<tr>
<td>AWS</td>
<td>braket_dm</td>
<td>gate-based</td>
<td>Simulator</td>
<td>N/A (local simulator in Braket SDK)</td>
<td>N/A</td>
</tr>
<tr>
<td>AWS</td>
<td>SV1</td>
<td>gate-based</td>
<td>Simulator</td>
<td>arn:aws:braket:::device/quantum-simulator/amazon/sv1</td>
<td>All Regions where Amazon Braket is available.</td>
</tr>
<tr>
<td>AWS</td>
<td>DM1</td>
<td>gate-based</td>
<td>Simulator</td>
<td>arn:aws:braket:::device/quantum-simulator/amazon/dm1</td>
<td>All Regions where Amazon Braket is available.</td>
</tr>
<tr>
<td>AWS</td>
<td>TN1</td>
<td>gate-based</td>
<td>Simulator</td>
<td>arn:aws:braket:::device/quantum-simulator/amazon/tn1</td>
<td>us-west-2 and us-east-1</td>
</tr>
</tbody>
</table>

To view additional details about the QPUs you can use with Amazon Braket, see Amazon Braket Hardware Providers.

IonQ

IonQ offers a gate-based QPU based on ion trap technology. IonQ's trapped ion QPUs are built on a chain of trapped 171Yb+ ions, spatially confined by means of a microfabricated surface electrode trap within a vacuum chamber.

Quantum gates supported by the IonQ device:
Rigetti

Rigetti quantum processors are universal, gate-model machines based on all-tunable superconducting qubits. The Rigetti Aspen-9 system is based on scalable 32-qubit node technology.

Quantum gates supported by the Aspen-9 device:

```
'cz', 'xy', 'cnot', 'cnot', 'cphaseshift', 'cphaseshift00', 'cphaseshift01',
'cphaseshift10', 'cswap', 'h', 'i', 'iswap', 'phaseshift', 'pswap', 'rx', 'ry', 'rz', 's',
'si', 'swap', 't', 'ti', 'x', 'y', 'z'
```

D-Wave

D-Wave offers quantum annealers based on superconducting qubits. Quantum annealing processors naturally return low-energy solutions. This type of QPU is a specific-purpose machine, designed and best suited to solve problems belonging to the class of Quadratic Unconstrained Optimization (QUBO) problems, such as optimization problems and probabilistic sampling problems.

Quantum annealers do not have to meet the strict engineering requirements that universal gate-based machines have to meet. Already today this technology features approximately 5000 superconducting qubits, compared to less than 100 qubits on gate-model quantum computers. Amazon Braket offers access to the superconducting quantum annealers provided by D-Wave Systems that can be programmed using the high-level, open source tool suite called Ocean.

For more information, see the quantum annealing example notebooks.

Local state vector simulator (braket_sv)

The local state vector simulator ("braket_sv") is part of the Amazon Braket SDK that runs locally in your environment. It is well-suited for rapid prototyping on small circuits, up to 25 qubits, depending on the hardware specifications of your Braket notebook instance or your local environment.

The simulator supports all gates in the Amazon Braket SDK, but QPU devices support a smaller subset. You can find the supported gates of a device in the device properties.

For more information about how to work with simulators, see the Amazon Braket examples.

Local density matrix simulator (braket_dm)

The local density matrix simulator ("braket_dm") is part of the Amazon Braket SDK that runs locally in your environment. It is well-suited for rapid prototyping on small circuits with noise, up to 12 qubits, depending on the hardware specifications of your Braket notebook instance or your local environment.

You can build common noisy circuits from the ground up using gate noise operations such as bit-flip and depolarizing error. You can also apply noise operations to specific qubits and gates of existing circuits that are intended to run both with and without noise.
To learn more about the local density matrix simulator, see the Braket introductory noise simulator example.

State vector simulator (SV1)

SV1 is a fully-managed, high-performance, universal state vector simulator. It can simulate circuits of up to 34 qubits. You can expect a 34-qubit, dense, and square circuit (circuit depth = 34) to take approximately 1 to 2 hours to complete, depending on the type of gates used and other factors. Circuits with all-to-all gates are well suited for SV1. It returns results in forms such as a full state vector or an array of amplitudes.

SV1 has a maximum runtime of 6 hours. It has a default of 35 concurrent tasks, and a maximum of 50 concurrent tasks.

SV1 Results

SV1 can provide the following results, given the specified number of shots:

- Sample: Shots > 0
- Expectation: Shots >= 0
- Variance: Shots >= 0
- Probability: Shots > 0
- Amplitude: Shots = 0

For more about results, see Result types.

SV1 is always available, it executes your circuits on demand, and it can run multiple circuits in parallel. The runtime scales linearly with the number of operations and exponential with the number of qubits. The number of shots has a small impact on the runtime. To learn more, visit Compare simulators.

Simulators support all gates in the Braket SDK, but QPU devices support a smaller subset. You can find the supported gates of a device in the device properties.

Density matrix simulator (DM1)

DM1 is a fully-managed, high-performance, density matrix simulator. It can simulate circuits of up to 17 qubits.

DM1 has a maximum runtime of 6 hours. It has a default of 35 concurrent tasks, and a maximum of 50 concurrent tasks.

DM1 Results

DM1 can provide the following results, given the specified number of shots:

- Sample: Shots > 0
- Expectation: Shots >= 0
- Variance: Shots >= 0
- Probability: Shots > 0

For more about results, see Result types (https://docs.aws.amazon.com/braket/latest/developerguide/braket-result-types.html).
DM1 is always available, it executes your circuits on demand, and it can run multiple circuits in parallel. The runtime scales linearly with the number of operations and exponential with the number of qubits. The number of shots has a small impact on the runtime. To learn more, visit Compare simulators.

Noise Gates and Limitations

<table>
<thead>
<tr>
<th>Gate</th>
<th>Probability Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmplitudeDamping</td>
<td>within [0,1]</td>
</tr>
<tr>
<td>BitFlip</td>
<td>within [0,0.5]</td>
</tr>
<tr>
<td>Depolarizing</td>
<td>within [0,0.75]</td>
</tr>
<tr>
<td>GeneralizedAmplitudeDamping</td>
<td>within [0,1]</td>
</tr>
<tr>
<td>PauliChannel</td>
<td>The sum of probabilities</td>
</tr>
<tr>
<td></td>
<td>has to be within [0,1]</td>
</tr>
<tr>
<td>Kraus</td>
<td>at most 2 qubits</td>
</tr>
<tr>
<td></td>
<td>at most 4 (16) Kraus</td>
</tr>
<tr>
<td></td>
<td>matrices for 1 (2) qubit</td>
</tr>
<tr>
<td>PhaseDamping</td>
<td>within [0,1]</td>
</tr>
<tr>
<td>PhaseFlip</td>
<td>within [0,0.5]</td>
</tr>
<tr>
<td>TwoQubitDephasing</td>
<td>within [0,0.75]</td>
</tr>
<tr>
<td>TwoQubitDepolarizing</td>
<td>within [0,0.9375]</td>
</tr>
</tbody>
</table>

Tensor network simulator (TN1)

TN1 is a fully-managed, high-performance, tensor network simulator. TN1 can simulate certain circuit types with up to 50 qubits, and a circuit depth of 100 or smaller. TN1 is particularly powerful for sparse circuits, circuits with local gates, and other circuits with special structure — for example, quantum Fourier transform (QFT) circuits. TN1 operates in two phases. First, the rehearsal phase attempts to identify an efficient computational path for your circuit, so TN1 can estimate the runtime of the next stage, which is called the contraction phase. If the estimated contraction time exceeds the TN1 simulation runtime limit, TN1 does not attempt contraction.

TN1 has a runtime limit of 6 hours. It is limited to a maximum of 10 concurrent tasks.

TN1 Results

The contraction phase consists of a series of matrix multiplications. The series of multiplications continues, until a result is reached, or until it is determined that a result cannot be reached.

**Note:** Shots must be > 0 for the TN1 simulator.

Result types include:

- Sample
- Expectation
- Variance

For more about results, see Result types.

TN1 is always available, it executes your circuits on demand, and it can run multiple circuits in parallel. To learn more, visit Compare simulators.
Simulators support all gates in the Braket SDK, but QPU devices support a smaller subset. You can find the supported gates of a device in the device properties.

Visit the Amazon Braket GitHub repository for a TN1 example notebook to help you get started with TN1.

**Best practices for working with the TN1 simulator**

- Avoid all-to-all circuits.
- Test a new circuit or class of circuits with a small number of shots, to learn the circuit’s "hardness" for TN1.
- Split large shot simulations over multiple tasks.

**Compare simulators**

This section helps you select the Amazon Braket simulator that’s best suited for your task, by describing some concepts, limitations, and use cases.

**What is a state vector simulator?**

The Amazon Braket state vector simulator (SV1) is a universal state vector simulator. It stores the full wave function of the quantum state and sequentially applies gate operations to the state. It stores all possibilities, even the extremely unlikely ones. The SV1 simulator’s run time for a task increases linearly, with the number of gates in the circuit.

**What is a density matrix simulator?**

The Amazon Braket density matrix simulator (DM1) simulates quantum circuits with noise. It stores the full density matrix of the system and sequentially applies gates and noise operations of the circuit. The final density matrix contains complete information of the quantum state after execution of the circuit. The runtime generally scales linearly with the number of operations and exponential with the number of qubits.

**What is a tensor network simulator?**

The Amazon Braket tensor network simulator (TN1) encodes quantum circuits into a structured graph.

- The nodes of the graph consist of quantum gates, or qubits.
- The edges of the graph represent connections between gates.

As a result of this structure, TN1 can find simulated solutions for relatively large and complex quantum circuits.

**The TN1 simulator requires two phases**

Typically, TN1 operates in a two-phase approach to simulating quantum computation.

- **The rehearsal phase:** In this phase, TN1 comes up with a way to traverse the graph in an efficient manner, which involves visiting every node, so that you can obtain the measurement you desire. As a customer, you do not see this phase because TN1 performs both phases together for you. It completes the first phase and determines whether to perform the second phase, on its own, based on practical constraints. You have no input into that decision after the simulation has begun.

- **The contraction phase:** This phase is analogous to the execution phase of a computation in a classical computer. The phase consists of a series of matrix multiplications. The order of these multiplications has a great effect on the difficulty of the computation. Therefore, the rehearsal phase is accomplished...
first, to find the most effective computation paths across the graph. After it finds the contraction path during the rehearsal phase, TN1 contracts together the gates of your circuit to produce the results of the simulation.

**TN1 Concept: It's like reading a map**

Metaphorically, you can compare the underlying TN1 graph to the streets of a city. In a city with a planned grid, it is easy to find a route to your destination using a map. In a city with unplanned streets, duplicate street names, and so forth, it can be difficult to find a route to your destination by looking at a map.

If TN1 did not perform the rehearsal phase, it would be like walking around the streets of the city to find your destination, instead of looking at a map first. It can really pay off in terms of walking time to spend more time looking at the map. Similarly, the rehearsal phase provides valuable information.

You might say that the TN1 has a certain “awareness” of the structure of the underlying circuit that it traverses. It gains this awareness during the rehearsal phase.

**Types of problems best suited for each of these types of simulators**

For SV1, any class of problems that rely primarily on having a certain number of qubits and gates is well-suited. The number of qubits and gates matters most. Generally, the time required grows linearly with the number of gates, and it does not depend on the number of shots. SV1 is generally faster than TN1 for circuits under 28 qubits.

The SV1 simulator can be slower for higher qubit numbers, because it actually simulates all possibilities, even the extremely unlikely ones. It has no way to determine which outcomes are likely. Thus, for a 30-qubit evaluation, SV1 must calculate $2^{30}$ configurations. The limit of 34 qubits for the Amazon Braket SV1 simulator is a practical constraint due to memory and storage limitations. You can think of it like this: Each time you add a qubit to the SV1 simulator, the problem becomes twice as hard.

For many classes of problems, the TN1 simulator can evaluate much larger circuits in realistic time than the SV1 simulator, because TN1 takes advantage of the structure of the graph. It essentially tracks the evolution of solutions from its starting place, and it retains only the configurations that contribute to an efficient traversal; that is, it saves the configurations to create an ordering of matrix multiplication that results in a simpler evaluation process.

For TN1, the number of qubits and gates matters, but the structure of the graph matters a lot more. For example, TN1 is very good at evaluating circuits (graphs) in which the gates are short-range (that is, each qubit is connected by gates only to its nearest neighbor qubits), and circuits (graphs) in which the connections (or gates) have similar range, for example, if each qubit talks only to other qubits that are 5 qubits away. If most of the structure can be decomposed into simpler relationships such as these, which can be represented in more, smaller, or more uniform matrices, TN1 performs the evaluation easily.

**Limitations of the TN1 simulator**

The TN1 simulator can be slower than the SV1 simulator, depending on the graph's structural complexity. For certain graphs, TN1 terminates the simulation after the rehearsal stage, and shows a status of FAILED, for either of these two reasons:

- **Cannot find a path** — If the graph is too complex, it is too difficult to find a good traversal path, and you can give up on the computation, practically speaking. TN1 cannot perform the contraction. You may see an error message similar to this one: No viable contraction path found.

- **Contraction stage is too difficult** — In some graphs, TN1 can find a traversal path, but it is very long, and extremely time-consuming to evaluate. In this case, the contraction is so expensive that the cost would be prohibitive. Instead, TN1 exits after the rehearsal phase. You may see an error message similar to this one: Predicted runtime based on best contraction path found exceeds TN1 limit.
Note: You are billed for the rehearsal stage of TN1 even if contraction is not performed and you see a FAILED status.

The predicted runtime also depends on the shot count. In worst-case scenarios, TN1 contraction time depends linearly on the shot count. The circuit may be contractable with fewer shots. For example, you might submit a task with 100 shots, which TN1 decides is uncontractable, but if you resubmit with only 10, the contraction proceeds. In this situation, to attain 100 samples, you could submit 10 tasks of 10 shots for the same circuit, combining the results in the end.

As a best practice, we recommend that you always test your circuit or circuit class with a few shots (for example, 10) to find out how hard your circuit is for TN1, before you proceed with a higher number of shots.

A note for the curious: The series of multiplications that forms the contraction phase begins with small, NxN matrices. For example, a 2-qubit gate requires a 4x4 matrix. The intermediate matrices required during a contraction that is adjudged to be too difficult are gigantic. Such a computation would require days to complete. That’s why Amazon Braket does not attempt extremely complex contractions.

Concurrency

- The SV1 (and DM1) simulator can perform a default of 35 tasks concurrently, and a maximum of 50 tasks concurrently.
- The DM1 simulator can perform a default (and a maximum number) of 35 tasks concurrently.
- The TN1 simulator can perform a default (and a maximum number) of 10 tasks concurrently.

More information is available on the Quotas page.

Example notebooks

Amazon Braket provides a variety of example notebooks showing the types of circuits that can either work well for, or challenge, the TN1 and SV1 simulators, such as the quantum Fourier transformation (QFT).
Amazon Braket Regions and endpoints

Amazon Braket is available in the following AWS Regions:

**Region availability of Amazon Braket**

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Braket Endpoint</th>
<th>QPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>braket.us-east-1.amazonaws.com</td>
<td>IonQ</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td>braket.us-west-1.amazonaws.com</td>
<td>Rigetti</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>braket.us-west-2.amazonaws.com</td>
<td>D-Wave</td>
</tr>
</tbody>
</table>

You can run Amazon Braket from any Region in which it is available, but each QPU is available only in a single Region. Tasks that run on a QPU device can be viewed in the Amazon Braket console, in the Region of that device. If you are using the Amazon Braket SDK, you can submit tasks to any QPU device, regardless of the Region in which you are working. The SDK automatically creates a session to the Region for the QPU specified, as shown in the following image.

**The Amazon Braket SDK automatically routes device requests to the correct region**

For general information about how AWS works with Regions and endpoints, see [AWS service endpoints](#) in the [AWS General Reference](#).
Amazon Braket pricing

With Amazon Braket, you have access to quantum computing resources on demand, without upfront commitment. You pay only for what you use. To learn more about pricing, please visit our pricing page.

Best practices for cost savings

Consider the following best practices for using Amazon Braket. Save time, minimize costs, and avoid common errors.

Verify with simulators

- Verify your circuits using a simulator before you run it on a QPU, so you can fine-tune your circuit without incurring charges for QPU usage.
- Although the results from running the circuit on a simulator may not be identical to the results from running the circuit on a QPU, you can identify coding errors or configuration issues using a simulator.

Restrict user access to certain devices

- You can set up restrictions that keep unauthorized users from submitting tasks on certain devices. The recommended method for restricting access is with AWS IAM. For more information about how to do that, see Restrict access.
- We recommend that you do not use your admin account as a way to give or restrict user access to Amazon Braket devices.

Set billing alarms

- You can set a billing alarm to notify you when your bill reaches a preset limit. The recommended way to set up an alarm is through AWS Budgets. You can set custom budgets and receive alerts when your costs or usage may exceed your budgeted amount. Information is available at AWS Budgets.

Test TN1 simulator tasks with low shot counts

- Simulators cost less than QHPs, but certain simulators can be expensive if tasks are run with high shot counts. We recommend that you test your TN1 simulator tasks with a low shot count. Shot count does not affect the cost for SV1 and local simulator tasks.

Check all Regions for tasks

- The console displays tasks only for your current AWS Region. When looking for billable tasks that have been submitted, be sure to check all Regions.
- You can view a list of devices and their associated Regions on the Supported Devices (p. 8) documentation page.
Amazon Braket Quotas

The following table lists the service quotas for Amazon Braket. Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

Some quotas can be increased. For more information, see AWS service quotas.

- Burst rate quotas cannot be increased.
- The maximum rate increase for adjustable quotas (except burst rate, which cannot be adjusted) is 2X the specified default rate limit. For example, a default quota of 60 can be adjusted to a maximum of 120.
- The adjustable quota for concurrent SV1 (DM1) tasks allows a maximum of 50 per AWS Region.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Limit</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rate of API requests</td>
<td>The maximum number of requests per second that you can send in this account in the current Region.</td>
<td>140</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of API requests</td>
<td>The maximum number of additional requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>600</td>
<td>No</td>
</tr>
<tr>
<td>Rate of CreateQuantumTask requests</td>
<td>The maximum number of CreateQuantumTask requests you can send per second in this account in the current Region.</td>
<td>20</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of CreateQuantumTask requests</td>
<td>The maximum number of additional CreateQuantumTask requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>40</td>
<td>No</td>
</tr>
<tr>
<td>Rate of SearchQuantumTasks requests</td>
<td>The maximum number of SearchQuantumTasks requests you can send per second in this account in the current Region.</td>
<td>5</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of SearchQuantumTasks requests</td>
<td>The maximum number of additional SearchQuantumTasks requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>Resource</td>
<td>Description</td>
<td>Limit</td>
<td>Adjustable</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>Rate of GetQuantumTask requests</td>
<td>The maximum number of GetQuantumTask requests you can send per second in this account per Region.</td>
<td>100</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of GetQuantumTask requests</td>
<td>The maximum number of additional GetQuantumTask requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>500</td>
<td>No</td>
</tr>
<tr>
<td>Rate of CancelQuantumTask requests</td>
<td>The maximum number of CancelQuantumTask requests you can send per second in this account per Region.</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of CancelQuantumTask requests</td>
<td>The maximum number of additional CancelQuantumTask requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>20</td>
<td>No</td>
</tr>
<tr>
<td>Rate of GetDevice requests</td>
<td>The maximum number of GetDevice requests you can send per second in this account per Region.</td>
<td>5</td>
<td>Yes</td>
</tr>
<tr>
<td>Burst rate of GetDevice requests</td>
<td>The maximum number of additional GetDevice requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>Rate of SearchDevices requests</td>
<td>The maximum number of SearchDevices requests you can send per second in this account per Region.</td>
<td>5</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Additional quotas and limits

- The Amazon Braket quantum task action (for example, a circuit or annealing problem) is limited to 3MB in size.
- The maximum number of shots allowed for the SV1 managed simulator, DM1 managed simulator and Rigetti device is 100,000.
- The maximum number of shots allowed for the TN1 managed simulator is 1000.
- For D-Wave and IonQ devices, the maximum is 10,000 shots.
- For TN1 and the QPU devices, shots must be > 0.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
<th>Limit</th>
<th>Adjustable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Burst rate of SearchDevices requests</td>
<td>The maximum number of additional SearchDevices requests per second (RPS) that you can send in one burst in this account in the current Region.</td>
<td>50</td>
<td>No</td>
</tr>
<tr>
<td>Number of concurrent SV1 tasks</td>
<td>The maximum number of concurrent tasks running on the state vector simulator (SV1) in the current Region.</td>
<td>35</td>
<td>Yes</td>
</tr>
<tr>
<td>Number of concurrent DM1 tasks</td>
<td>The maximum number of concurrent tasks running on the density matrix simulator (DM1) in the current Region.</td>
<td>35</td>
<td>No</td>
</tr>
<tr>
<td>Number of concurrent TN1 tasks</td>
<td>The maximum number of concurrent tasks running on the tensor network simulator (TN1) in the current Region.</td>
<td>10</td>
<td>No</td>
</tr>
</tbody>
</table>
When will my task run?

When you submit a circuit, Amazon Braket sends it to the device you specify. QPU and simulator tasks are queued and processed in the order they are received. The time required to process your task after you submit it varies, depending on the number and complexity of tasks submitted by other Amazon Braket customers, and on the availability of the QPU you selected.

Status change notifications in email or SMS

Amazon Braket sends events to Amazon EventBridge when the availability of a QPU changes, or when your task’s state changes. Follow these steps to receive device and task status change notifications by email or SMS message:

1. Create an Amazon SNS topic and a subscription to email or SMS. Availability of email or SMS depends on your Region. See Getting started with Amazon SNS and Sending SMS messages.
2. Create a rule in EventBridge that triggers the notifications to your SNS topic. See Monitoring Amazon Braket with Amazon EventBridge (p. 80).

Task completion alerts

You can set up notifications through the Amazon Simple Notification Service (SNS), so that you receive an alert when your Amazon Braket task is complete. Active notifications are useful if you expect a long wait time — for example, when you submit a large task, or when you submit a task outside of a device’s availability window. If you do not want to wait for the task to complete, you can set up an SNS notification.

An Amazon Braket notebook walks you through the setup steps. For more information, see the Amazon Braket example notebook for setting up notifications.

QPU availability windows and status

QPU availability varies among the Quantum Hardware Providers (QHPs).

In the Devices page of the Amazon Braket console, you can see the current and upcoming availability windows for each device. You also can view the status of each device.

A device is considered offline if is not available to customers, regardless of availability window. For example, it could be offline due to scheduled maintenance, upgrades, or operational issues.

The Announcements page in the Amazon Braket console shows scheduled downtime in advance, to all Amazon Braket customers.
Enable Amazon Braket

You can enable Amazon Braket in your account through the AWS console.

Prerequisites

To enable and run Amazon Braket, you must have an IAM user or role with permission to initiate Amazon Braket actions. These permissions are included in the AmazonBraketFullAccess IAM policy.

In particular, your IAM user or role must have permission to create an S3 bucket in the AWS account, because Amazon Braket requires an Amazon S3 bucket to store results from your tasks. Results from all users and roles in the account are stored in the same bucket location, which you specify when you enable Amazon Braket.

The next section shows how to create or select an S3 bucket associated with the AWS account in which you enable Amazon Braket.

Note

If you are an administrator:

To enable Amazon Braket for other IAM users, for IAM roles, or for an IAM group in an account, you must grant permissions to each user, role, or group. You can grant these permissions by attaching the AmazonBraketFullAccess policy or by attaching a custom policy that you create.

To learn more about the permissions necessary to use Amazon Braket, see Managing access to Amazon Braket.

Steps to enable Amazon Braket

1. Log in to your AWS account and then open the Amazon Braket console.
2. Choose Go to Braket.
3. To specify an Amazon S3 bucket to use for your results, do one of the following tasks:
   a. Choose Create new bucket to have Amazon Braket create a bucket in your account named amazon-braket-uniqueString.
   b. Choose Specify bucket name and enter a name to append to the bucket name after amazon-braket-. For example, if you enter mycircuits, the resulting bucket name is amazon-braket-mycircuits.
   c. Choose Select an existing bucket to use an existing bucket from your account. The bucket name must start with amazon-braket-. You also can specify a folder within the bucket to store your task results.
4. Review the permission policy attached to the service-linked role that Amazon Braket creates in your account.
5. Review the Terms and Conditions for using Amazon Braket, then select the check box to confirm that you have read and accept these terms.
6. Choose Enable Amazon Braket.

Common causes of failure when enabling Amazon Braket

Here are some error messages you might see when you enable or run Amazon Braket.
Access denied due to lacking S3 permissions

This error means that the user or role tried to create or manage access to an Amazon S3 bucket, without the required permissions. Be sure that you’re signed in with a user or role has the AmazonBraketFullAccess policy enabled.
Get started with Amazon Braket

After you have followed the instructions in Enable Amazon Braket (p. 22), you can get started with Amazon Braket.

**The steps to get started include:**

- Create an Amazon Braket notebook instance (p. 24)
- Run your first circuit using the Amazon Braket Python SDK (p. 25)
- Run your first annealing problem with Ocean (p. 28)

Create an Amazon Braket notebook instance

Amazon Braket provides fully-managed Jupyter notebooks to get you started. The Amazon Braket notebook instances are based on Amazon SageMaker notebook instances. You can learn more about notebook instances. To get started with Braket, follow these steps to create an Amazon Braket notebook instance.

1. Open the Amazon Braket console.
2. Choose Notebooks in the left pane, then choose Create notebook.
3. In Notebook instance settings, enter a Notebook instance name using only alphanumeric and hyphen characters.
4. Select the Notebook instance type. Choose the smallest type you need. To get started, choose a cost-effective instance type, such as ml.t3.medium.

The instance types are Amazon SageMaker notebook instances. To learn more, see Amazon SageMaker pricing.
5. In Permissions and encryption, select Create a new role (a new role with a name that begins with `AmazonBraketServiceSageMakerNotebook` is created).
6. Choose Create notebook instance.

It takes several minutes to create the notebook. The notebook is displayed on the Notebooks page with a status of Pending. When the notebook instance is ready to use, the status changes to InService. You may need to refresh the page to display the updated status for the notebook.

**Note**

You can view and manage your Amazon Braket notebook instances in the Amazon Braket and Amazon SageMaker consoles. Additional Amazon Braket notebook settings are available through the SageMaker console. Open the SageMaker console at https://console.aws.amazon.com/sagemaker/.

If you’re working in the Amazon Braket console within AWS, as given previously, the Amazon Braket SDK and plugins are preloaded in the notebooks you just created. If you want to run on your own machine, you can install the SDK and plugins when you run the command pip install amazon-braket-sdk or when you run the command pip install amazon-braket-pennylane-plugin (the latter for use with PennyLane plugins).
Run your first circuit using the Amazon Braket Python SDK

After your notebook instance has launched, open the instance with a standard Jupyter interface by choosing the notebook you just created.

Amazon Braket notebook instances are pre-installed with the Amazon Braket SDK and all its dependencies. Start by creating a new notebook with conda_braket kernel.
You can start with a simple “Hello, world!” example. First, construct a circuit that prepares a Bell state, and then run that circuit on different devices to obtain the results.

Begin by importing the Amazon Braket SDK modules and defining a simple Bell State circuit.

```python
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)
```

You can visualize the circuit with this command:

```python
print(bell)
```

Run your circuit on the local simulator
Next, choose the quantum device on which to execute the circuit. The Amazon Braket SDK comes with a local simulator for rapid prototyping and testing. We recommend using the local simulator for smaller circuits up to 25 qubits (depending on your local hardware).

Here's how to instantiate the local simulator:

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

and run the circuit:

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

You should see a result something like this:

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

The specific Bell state you have prepared is an equal superposition of $|00\rangle$ and $|11\rangle$, and you'll find a roughly equal (up to shot noise) distribution of 00 and 11 as measurement outcomes, as expected.

**Run your circuit on a managed simulator**

Amazon Braket also provides access to a fully-managed, high-performance simulator, SV1, for running larger circuits. SV1 is a state-vector simulator that allows for simulation of quantum circuits of up to 34 qubits. You can find more information on SV1 in the [Supported Devices](#) section and in the AWS console. When running tasks on SV1 (and on TN1 or any QPU), the results of your task are stored in an S3 bucket in your account.

For now, we will use the S3 bucket you created in [Enable Amazon Braket](#), but you can choose any S3 bucket that Amazon Braket has permissions to access. To learn more, see [Managing access to Amazon Braket](#).

**Note**

Fill in your actual, existing bucket name where the following example shows `example-bucket` as your bucket name. Bucket names for Amazon Braket always begin with `amazon-braket-` followed by other identifying characters you add.

```python
# 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_location = (my_bucket, my_prefix)
```

To run a circuit on SV1, you must provide the location of the S3 bucket you previously selected, as a positional argument in the `.run()` call.

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

# execute the circuit
task = device.run(bell, s3_location, shots=100)
# display the results
```
Run your first annealing problem with Ocean

Quantum annealers are special-purpose quantum computers designed to solve combinatorial optimization problems. In particular, quantum annealers solve problems belonging to the class of Quadratic Unconstrained Optimization (QUBO). Amazon Braket allows you to program the D-Wave QPUs natively, using D-Wave's Ocean software through the Braket-Ocean plugin. Amazon Braket notebook instances are pre-installed with Ocean and the Braket-Ocean plugin.

Get started with a simple example of solving the Minimum Vertex Cover (MVC) problem. Here's the problem definition:

Given an undirected graph with a vertex set and an edge set, a vertex cover is a subset of the vertices (nodes) such that each edge in the graph is incident to at least one vertex in the subset. The Minimum Vertex Cover problem seeks to find a cover with a minimum number of vertices in the subset. In other words, in a graph like this:
The goal is to color nodes in red such that every edge touches at least one red node. And you want to do it with as little paint as possible. The optimal solution is to paint the central node red, as shown in the figure that follows.

How to solve such a problem with D-Wave's 2000Q QPU

To begin, import the following dependencies and specify an S3 location.

**Note**
Fill in your actual, existing bucket name where the following example shows `example-bucket` as your bucket name. Bucket names for Amazon Braket always begin with `amazon-braket-` followed by other identifying characters you add.
# import relevant modules
import boto3
from braket.ocean_plugin import BraketSampler, BraketDWaveSampler
import networkx as nx
import dwave_networkx as dnx
from dwave.system.composites import EmbeddingComposite

# Please enter the S3 bucket you created during onboarding
# (or any other S3 bucket starting with 'amazon-braket-' in your account) in the code below
my_bucket = f"amazon-braket-Your-Bucket-Name" # the name of the bucket
my_prefix = "Your-Folder-Name" # the name of the folder in the bucket
s3_folder = (my_bucket, my_prefix)

Now, set the sampler. Then use EmbeddingComposite to minor-embed a problem automatically into a structured sampler, such as a D-Wave system.

# set sampler using BraketSampler
sampler = BraketSampler(s3_folder,'arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6')
# or alternatively using BraketDWaveSampler
sampler = BraketDWaveSampler(s3_folder,'arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6')

# EmbeddingComposite automatically maps the problem to the structure of the solver.
embedded_sampler = EmbeddingComposite(sampler)

You can create the graph from the family of random Erdos-Renyi graphs. Such a graph can be generated using the networkx library. As input, set the desired number of vertices and edges connecting pairs of vertices.

# setup Erdos Renyi graph
# 5 nodes
n = 5
# 10 edges
m = 10
# generate graph
graph = nx.gnm_random_graph(n, m, seed=42)

Finally, run the problem in D-Wave and print the results.

# run the problem on D-Wave using BraketSampler
result = dnx.min_vertex_cover(graph, embedded_sampler, resultFormat="HISTOGRAM")
# or alternatively using BraketDWaveSampler
result = dnx.min_vertex_cover(graph, embedded_sampler, answer_mode="histogram")
print('Result to MVC problem:', result)
print('Size of the vertex cover:', len(result))

Result to MVC problem: [0, 1, 3, 4]
Size of the vertex cover: 4

Notice that the result to the MVC problem is a list containing the vertices [0, 1, 3, 4]. These vertices form a minimum vertex cover such that the subset can reach every edge in the graph.
Work with Amazon Braket

This section shows you how to design quantum circuits and annealing problems, submit these problems as tasks to devices, and monitor the tasks with the Amazon Braket SDK.

The main ways to interact with resources on Amazon Braket are by means of:

- the Amazon Braket Console — The Amazon Braket console provides device information and status to help you create, manage, and monitor your resources and tasks.
- the Amazon Braket SDK — You can submit and run quantum tasks through the Amazon Braket Python SDK as well as through the console. The SDK is easily accessible through preconfigured Amazon Braket notebooks.
- the Amazon Braket API — The Amazon Braket API is accessible through the Amazon Braket Python SDK and notebooks. You can make calls directly to the API if you’re building applications that work with quantum computing programmatically.

The examples throughout this section demonstrate how you can work with the Amazon Braket API directly, using the Amazon Braket Python SDK along with the AWS Python SDK for Braket (Boto3).

More about the Amazon Braket Python SDK

To work with the Amazon Braket Python SDK, first you must install the AWS Python SDK for Braket (Boto3) so that you can communicate with the AWS API. You can think of the Amazon Braket Python SDK as a convenient wrapper around Boto3, for quantum customers.

- Boto3 contains interfaces you’ll need to tap into the AWS API. (Note that Boto3 is a large Python SDK that talks to the AWS API. Most AWS services support a Boto3 interface.)
- The Amazon Braket Python SDK contains software modules for circuits, gates, devices, result types, and other parts of a quantum task. Each time you create a program, you’ll import the modules you need for that task.
- The Amazon Braket Python SDK is easily accessible through notebooks, which are pre-loaded with all of the modules and dependencies you need for running quantum tasks.
- You can import modules from the Amazon Braket Python SDK into any Python script, if you do not wish to work with notebooks.

After you've installed Boto3, an overview of steps for creating a task through the Amazon Braket Python SDK is something like this:

- (Optionally) Open your notebook
- Import the SDK modules you need for your circuits
- Specify a QPU or simulator
- Instantiate the circuit
- Run the circuit
- Collect the results

The examples in this section show details of each step.

Many more examples are available in the Amazon Braket Examples repository on GitHub.

In this section:

- Construct circuits in the SDK (p. 32)
Construct circuits in the SDK

This section provides examples of defining a circuit, viewing available gates, extending a circuit, and viewing gates that each device supports. It also contains instructions on how to manually allocate qubits, instruct the compiler to run your circuits exactly as defined, and build noisy circuits with a noise simulator.

In this section:
- Gates and circuits (p. 32)
- Manual qubit allocation (p. 35)
- Verbatim compilation (p. 36)
- Noise simulation (p. 36)
- Inspecting the circuit (p. 37)
- Result types (p. 38)

Gates and circuits

Quantum gates and circuits are defined in the SDK’s `braket.circuits` class. From the SDK, you can instantiate a new circuit object by calling `Circuit()`.

Example: Define a circuit

The example starts by defining a sample circuit of four qubits (labelled `q0`, `q1`, `q2`, and `q3`) consisting of standard, single-qubit Hadamard gates and two-qubit CNOT gates. You can visualize this circuit by calling the `print` function, as the example shows.

```python
# 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 |
```

Example: See all available gates

The following example shows how to look at all the available gates in Amazon Braket.

```python
import string
```
from braket.circuits import Gate
# print all available gates in Amazon Braket
gate_set = [attr for attr in dir(Gate) if attr[0] in string.ascii_uppercase]
print(gate_set)

['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']

Any of these gates can be appended to a circuit by calling the method for that type of circuit. For
example, you’d call circ.h(0), to add a Hadamard gate to the first qubit.

**Note**
Gates are appended in place, and the example that follows adds all of the gates listed in the
previous example to the same circuit.

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((1,1,exp(1j*phi),1))
circ.cphaseshift01(0, 1, 0.15)
# controlled-phase gate that phases the |10> state, cphaseshift10(phi) =
# diag((1,1,1,exp(1j*phi)))
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)
# 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(i(XX+YY) theta/4), where theta=0.15 in the examples below
circ.xx(0, 1, 0.15)
# exp(-iXY theta/2)
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)

Apart from the pre-defined gate set, you also can apply self-defined unitary gates to the circuit. These can be single-qubit gates (as shown in the following source code) or multi-qubit gates applied to the qubits defined by the targets parameter.

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

**Example: Extend existing circuits**

You can extend existing circuits by adding instructions. An Instruction is a quantum directive that describes the task to perform on a quantum device. Instruction operators include objects of type Gate only.

```python
# 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})
```
Example: View the gates that each device supports

Simulators support all gates in the Braket SDK, but QPU devices support a smaller subset. You can find the supported gates of a device in the device properties.

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

device = AwsDevice("arn:aws:braket:::device/qpu/ionq/ionQdevice")

# get device name
device_name = device.name

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

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

device = AwsDevice("arn:aws:braket:::device/qpu/rigetti/Aspen-9")

# get device name
device_name = device.name

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

Quantum Gates supported by Aspen-9:
['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']
```

Supported gates may need to be compiled into native gates before they can run on quantum hardware. When you submit a circuit, Amazon Braket performs this compilation automatically.

Manual qubit allocation

When you run a quantum circuit on quantum computers from Rigetti, you can optionally use manual qubit allocation to get control over which qubits are used for your algorithm. The Amazon Braket Console and the Amazon Braket SDK help you to inspect the most recent calibration data of your selected quantum processing unit (QPU) device, so you can select the best qubits for your experiment.

Manual qubit allocation enables you to run circuits with greater accuracy and to investigate individual qubit properties. Researchers and advanced users optimize their circuit design based on the latest device calibration data, and thus can obtain more accurate results.

Here's an example of how to allocate qubits explicitly:

```python
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)
```

For more information, see the Amazon Braket examples on GitHub, or more specifically, this notebook: Allocating Qubits on QPU Devices.
Verbatim compilation

When you run a quantum circuit on quantum computers from Rigetti, you have the ability to direct the compiler to run your circuits exactly as defined, without any modifications. Using verbatim compilation, you can specify either that an entire circuit be preserved precisely as specified or that only specific parts of it be preserved. When developing algorithms for hardware benchmarking or error mitigation protocols, you need to be able to specify the gates and circuit layouts that are to be executed on the hardware exactly. Verbatim compilation gives you direct control over the compilation process by disabling certain optimization steps, thereby ensuring that your circuits are executed exactly as designed.

Verbatim compilation is currently supported on Rigetti devices and requires the use of native gates. When using verbatim compilation, it is advisable to check the topology of the device to ensure that gates are called on connected qubits and that the circuit uses the native gates supported on the hardware. The following example shows how to programmatically access the list of native gates supported by a device:

```python
rigetti.properties.paradigm.nativeGateSet
```

Also, qubit rewiring must be disabled by setting `disableQubitRewiring=True` for use with verbatim compilation. If `disableQubitRewiring=False` is set when using verbatim boxes in a compilation, the quantum circuit will fail validation and not run.

If verbatim compilation is enabled for a circuit and run on a QPU that does not support it, an error is generated indicating that an unsupported operation has caused the task to fail. As more quantum hardware natively support compiler functions, this feature will be expanded to include these devices. Devices that support verbatim compilation include it as a supported operation when queried with the following code.

```python
from braket.aws import AwsDevice
from braket.device_schema.device_action_properties import DeviceActionType
device = AwsDevice("arn:aws:braket:::device/qpu/rigetti/Aspen-9")
device.properties.action[DeviceActionType.JAQCD].supportedOperations
```

There is no additional cost associated with using verbatim compilation. You will continue to be charged for tasks executed on Braket QPU devices, notebook instances, and managed simulators based on our current rates as specified on the Amazon Braket Pricing page. For more information, see the Verbatim compilation example notebook.

Noise simulation

To instantiate the local noise simulator you can change the backend as follows:

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

You can build noisy circuits in two ways: (i) Build the noisy circuit from the bottom up. (ii) Take an existing, noise-free circuit and inject noise throughout. The following shows the approaches using a simple circuit with depolarizing noise and a custom Kraus channel:

```python
# 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
```
# 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)

Running a circuit is the same user experience as before:

task = device.run(circ, s3_location)

Or

task = device.run(circ_noise, s3_location)

For more examples, see the Braket introductory noise simulator example

Inspecting the circuit

Quantum circuits in Amazon Braket have a pseudo-time concept called Moments. Each qubit can experience a single gate per Moment. The purpose of Moments is to make circuits and their gates easier to address, and to provide a temporal structure.

**Note**

Moments generally do not correspond to the real time at which gates are executed on a QPU.

The depth of a circuit is given by the total number of Moments in that circuit. You can view the circuit depth calling the method `circuit.depth` as shown in the example that follows.

# 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)

<table>
<thead>
<tr>
<th>T</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>q0</td>
<td>-Rx(0.15)-C----------X-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q1</td>
<td>-Ry(0.2)--</td>
<td>--ZZ(0.15)---</td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>-------------------X</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>-------------------ZZ(0.15)---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total circuit depth: 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The total circuit depth of the circuit above is three, shown as moments 0, 1, and 2. You can check the gate operation for each moment.

Moments functions as a dictionary of *key, value* pairs.

- The key is `MomentsKey()`, which contains pseudo-time and qubit information.
- The value is assigned in the type of `Instructions()`.  


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

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

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

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

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

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

You can also add gates to a circuit through Moments.

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

T  : |0|1|2|
q0 : -S-Z---
|  
q1 : ---C-H-
T  : |0|1|2|

Result types

Amazon Braket can return different types of results when a circuit is measured using ResultType. Types of results that a circuit can return are as follows:

- **Amplitude** — returns the amplitude of specified quantum states in the output wave function. It is available on the SV1 and local simulators only.
- **Expectation** — returns the expectation value of a given observable, which can be specified with the Observable class introduced later in this chapter. The target qubits used to measure the observable must be specified, and the number of specified targets must equal the number of qubits on which the observable acts. If no targets are specified, the observable must operate only on 1 qubit, and it is applied to all qubits in parallel.
- **Probability** — returns the probabilities of measuring computational basis states. If no targets are specified, Probability returns the probability of measuring all basis states. If targets are specified, only the marginal probabilities of the basis vectors on the specified qubits are returned.
- **StateVector** — returns the full state vector. It is available on the local simulator.
- **Sample** — returns the measurement counts of a specified target qubit set and observable. If no targets are specified, the observable must operate only on 1 qubit, and it is applied to all qubits in parallel. If targets are specified, the number of specified targets must equal the number of qubits on which the observable acts.

- **Variance** — returns the variance \((\text{mean}_x - x.\text{mean}(^2))\) of specified target qubit set and observable as the requested result type. If no targets are specified, the observable must operate only on 1 qubit and it is applied to all qubits in parallel. Otherwise, the number of targets specified must equal the number of qubits to which the observable can be applied.

### The supported result types for different devices:

<table>
<thead>
<tr>
<th>Result Type</th>
<th>Local sim</th>
<th>SV1</th>
<th>DM1</th>
<th>TN1</th>
<th>Rigetti</th>
<th>IonQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplitude</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Expectation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Probability</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>State vector</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Sample</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Variance</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

You can check the supported result types by examining the device properties, as shown in the example.

```python
device = AwsDevice("arn:aws:braket:::device/qpu/rigetti/Aspen-9")
# print the result types supported by this device
for iter in device.properties.action['braket.ir.jaqcd.program'].supportedResultTypes:
    print(iter)
```

```python
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
```

To call a *ResultType*, append it to a circuit. For example:

```python
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])

### Observables

Amazon Braket includes an *Observable* class, which can be used to specify an observable to be measured.
You can apply at most one unique non-identity observable to each qubit. If you specify two or more different non-identity observables to the same qubit, you will see an error. For this purpose, each factor of a tensor product counts as an individual observable, so it is permissible to have multiple tensor products acting on the same qubit, provided that the factor acting on that qubit is the same.

The Observable class includes the following observables:

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

```python
# 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:
", Observable.Z().to_matrix())
print("The matrix form of the tensor product:
", 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]])))
```

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.+1.j  0.+0.j]
 [ 0.+1.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j -0.+0.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]])

## Submitting tasks to QPUs and simulators

Amazon Braket provides access to several devices that can execute quantum tasks. You can submit tasks individually, or you can set up task batching.

### QPUs

You can submit tasks to QPUs at any time, but the task runs within certain availability windows that are displayed on the Devices page of the Amazon Braket Console. The results of the task can be retrieved with the task ID, which is introduced in the next section.

- **IonQ**: arn:aws:braket:::device/qpu/ionq/ionQdevice
- **Rigetti**: arn:aws:braket:::device/qpu/riqetti/Aspen-9
- **D-Wave 2000Q**: arn:aws:braket:::device/qpu/d-wave/DW_2000Q_6
**Simulators**

- **D-Wave Advantage_system**: `arn:aws:braket:::device/gpu/d-wave/Advantage_system1`
- **Managed state vector simulator, SV1**: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- **Managed density matrix simulator, DM1**: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- **Managed tensor network simulator, TN1**: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- **The local simulator**: `LocalSimulator()`

**Note**

Tasks in the CREATED state can be cancelled for QPUs and managed simulators. Tasks in the QUEUED state can be cancelled on a best effort basis for managed simulators and QPUs. Note that QPU QUEUED tasks are unlikely to be cancelled successfully during QPU availability windows.

---

**Example tasks on Amazon Braket**

This section walks through the stages of running an example task, from selecting the device, to viewing the result. As a best practice for Amazon Braket, we recommend to begin by running the circuit on a simulator, such as SV1.

**Specify the device**

First, select and specify the device for your task. This example shows how to choose the simulator, SV1.

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

You can view some of the properties of this device as follows:

```python
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)])
```

---

41
Submit an example task

Submit an example task to run on the managed simulator.

```python
# 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 task to run
my_task = device.run(circ, s3_location, shots=1000, poll_timeout_seconds = 100,
                      poll_interval_seconds = 10)

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

The `device.run()` command creates a task through the CreateQuantumTask API. After a short initialization time, the task is queued until capacity exists to execute the task on a device. In this case, the device is the managed simulator SV1. After the device completes the computation, Amazon Braket writes the results to the S3 location specified in the call. The positional argument `s3_location` is required for all devices except the local simulator.

**Note**
The Braket quantum task action (for example, a circuit or annealing problem) is limited to 3MB in size.

Specify shots

The `shots` argument refers to the number of desired measurement shots. Simulators such as SV1 support two simulation modes.

- For `shots = 0`, the simulator performs an exact simulation, returning the true values for all result types. (Not available on TN1.)
- For non-zero values of `shots`, the simulator samples from the output distribution to emulate the shot noise of real QPUs. QPU devices only allow `shots > 0`.

The maximum number of shots allowed for the the managed simulator and Rigetti device is 100,000. For D-Wave and IonQ devices, the maximum is 10,000 shots.

Polling for results

When executing `my_task.result()`, the SDK begins polling for a result, with the parameters you define upon task creation:

- `poll_timeout_seconds` is the number of seconds to poll the task before it times out when running the task on the managed simulator and or QPU devices. The default value is 432,000 seconds, which is 5 days.
- **Note**: For QPU devices such as Rigetti and IonQ, we recommend that you allow a few days. If your polling timeout is too short, results may not be returned within the polling time. For example, when a QPU is unavailable, a local timeout error is returned.
poll_interval_seconds is the frequency with which the task is polled. It specifies how often you call the Braket API to get the status, when the task is run on the managed simulator and on QPU devices. The default value is 1 second.

This asynchronous execution facilitates the interaction with QPU devices that are not always available. For example, a device could be unavailable during a regular maintenance window.

The returned result contains a range of metadata associated with the task. You can check the measurement result with the following commands:

```python
print('Measurement results:
', result.measurements)
print('Counts for collapsed states:
', result.measurement_counts)
print('Probabilities for collapsed states:
', 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}

View the example results

Because you've also specified the ResultType, you can view the returned results. The result types appear in the order in which they were added to the circuit.

```python
print('Result types include:
', result.result_types)
print('Variance=', result.values[0])
print('Probability=', result.values[1])
```

```
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]
```
Submitting tasks to a QPU

Amazon Braket allows you to run a quantum circuit on a QPU device. The following example shows how to submit a task to the Rigetti or the IonQ device.

First, here’s how to choose the Rigetti device, then look at the associated connectivity graph.

```python
# import the QPU module
from braket.aws import AwsDevice
# choose the Rigetti device
device = AwsDevice("arn:aws:braket:::device/qpu/rigetti/Aspen-9")
# take a look at the device connectivity graph
device.properties.dict()['paradigm']['connectivity']
```
The dictionary `connectivityGraph` shown above contains information about the connectivity of the current Rigetti device.

**Here's how to choose the IonQ device**

For the IonQ device, as shown below, the `connectivityGraph` is empty, because the device offers all-to-all connectivity. Therefore, a detailed `connectivityGraph` is not needed.

```python
# or choose the IonQ device
device = AwsDevice("arn:aws:braket:::device/qpu/ionq/ionQdevice")

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

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

For any QPU device, when you launch a task, remember to specify the S3 bucket in which to store your results. You have the option to adjust the `shots` (default=1000), the `poll_timeout_seconds` (default = 432000 = 5 days), and the `poll_interval_seconds` (default = 1) when you submit the task. The following example shows how.

```python
my_task = device.run(circ, s3_location, shots=100, poll_timeout_seconds = 100,
poll_interval_seconds = 10)
```

The IonQ and Rigetti devices compile the provided circuit into their respective native gate sets automatically, and they map the abstract qubit indices to physical qubits on the respective QPU.

**Note**

QPU devices have limited capacity. You can expect longer wait times when capacity is reached.

Amazon Braket can execute QPU tasks within certain availability windows. Still, you can submit tasks any time (24/7), because all corresponding data and metadata are stored reliably in your S3 bucket. As shown in the next section, you can recover your task using `AwsQuantumTask` and your unique task ID.

**Running a task with the local simulator**

You can send tasks directly to a local simulator for rapid prototyping and testing. This simulator runs in your local environment, so you do not need to specify an S3 location. The results are computed directly in your session. To run a task on the local simulator, you must only specify the `shots` parameter.

**Note**

The execution speed and maximum number of qubits the local simulator can process depends on the Amazon Braket Notebook instance type, or on your local hardware specifications.
The following commands are all identical and will instantiate the state vector (noise free) local simulator

```
# 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")
```

Then run a task with

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

To instantiate the local density matrix (noise) simulator customers change the backend as follows:

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

## Task batching

Task batching is available on every Amazon Braket device, except the local simulator. Batching is especially useful for tasks you run on the managed simulators, TN1 or SV1, because they can process multiple tasks in parallel. To help you set up various tasks, Amazon Braket provides [example notebooks](https://github.com/aws/aws-braket-sdk-python).

Batching allows you to launch tasks in parallel. For example, if you wish to make a calculation that requires 10 tasks, and the circuits in those tasks are independent of each other, it is a good idea to use batching. That way, you don’t have to wait for one task to be complete before another task begins.

The following example shows how to run a batch of tasks:

```
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 task in the batch
```

For more information, see the [Amazon Braket examples on GitHub](https://github.com/aws/aws-braket-sdk-python), or more specifically about batching, see [Quantum task batching](https://github.com/aws/aws-braket-sdk-python/blob/main/docs/quantum_tasks.md).

### About task batching and costs

A few caveats to keep in mind regarding task batching and billing costs:

- By default, task batching retries all timed out or failed tasks 3 times.
- A batch of long running tasks, such as 34 qubits for SV1, can incur large costs. Be sure to double check the ‘run_batch’ assignment values carefully before you start a batch of tasks. We do not recommend using TN1 with `run_batch`.
- TN1 can incur costs for failed rehearsal phase tasks. See the [TN1 description](https://github.com/aws/aws-braket-sdk-python/blob/main/docs/quantum_tasks.md) for more information. Automatic retries can add to the cost. We recommend to set the number of ‘max_retries’ on batching to 0 when using TN1. See [Quantum Task Batching, Line 186](https://github.com/aws/aws-braket-sdk-python/blob/main/docs/quantum_tasks.md).

### Task batching and PennyLane

It is easy to take advantage of batching when you’re using PennyLane on Amazon Braket. Set `parallel = True` when you instantiate an Amazon Braket device:
You can find more information about batching with PennyLane in Parallelized optimization of quantum circuits.

**Set up SNS notifications (optional)**

You can set up notifications through the Amazon Simple Notification Service (SNS), so that you receive an alert when your Amazon Braket task is complete. Active notifications are useful if you expect a long wait time — for example, when you submit a large task, or when you submit a task outside of a device's availability window. If you do not want to wait for the task to complete, you can set up an SNS notification.

An Amazon Braket notebook walks you through the setup steps. For more information, see the Amazon Braket examples on GitHub, and specifically, the example notebook for setting up notifications.

**Monitoring and tracking tasks**

After a task is submitted, you can keep track of its status through the Amazon Braket SDK and console. When the task completes, Braket saves the results in your specified S3 location. Completion may take some time, especially for QPU devices, depending on the length of the queue. Status types include:

- CREATED
- RUNNING
- COMPLETED
- FAILED
- CANCELLED

**Tracking tasks from the Braket SDK**

The command `device.run(...)` defines a task with a unique task ID. You can query and track the status with `task.state()` as shown in the following example.

*Note* that `task = device.run()` is an asynchronous operation, which means that you can keep working while the system processes your task in the background.

**Retrieve a result**

When you call `task.result()`, the SDK begins polling Amazon Braket, to see whether the task is complete. The SDK uses the polling parameters you defined in `.run()`. After the task is complete, the SDK retrieves the result from the S3 bucket and returns it as a `QuantumTaskResult` object.
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

Cancel a task

To cancel a task, call the `cancel()` method, as shown in the following example.

```python
# cancel task
task.cancel()
status = task.state()
print('Status of task:', status)
```

Status of task: CANCELLING

Check the metadata

You can check the metadata of the finished task, as shown in the following example.

```python
# get the metadata of the 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
Retrieve a task or result

If your kernel dies after you submit the task, or if you close your notebook or computer, you can reconstruct the task object with its unique ARN (task ID). Then you can call `task.result()` to get the result from the S3 bucket where it is stored.

```python
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()
```

Advanced logging

You can record the whole task-processing process using a logger. These advanced logging techniques allow you to see the background polling and create a record for later debugging.

To use the logger, we recommend changing the `poll_timeout_seconds` and `poll_interval_seconds` parameters, so that a task can be long-running and the task status is logged continuously, with results saved to a file. You can transfer this code to a Python script instead of a Jupyter notebook, so that the script can run as a process in the background.

Configure the logger

First, configure the logger so that all logs are written into a text file automatically, as shown in the following example lines.

```python
# 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
```

Create and run the circuit

Now you can create a circuit, submit it to a device to run, and see what happens, as shown in this example.

```python
# 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
    )

Check the log file

You can check what is written into the file, by entering the following command.

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

```
Counter({'00001': 493, '00011': 493, '01001': 5, '10111': 4, '01011': 3, '10101': 2})
```

Get the ARN from the log file

From the log file output that’s returned, as shown in the previous example, you can obtain the ARN information. With the ARN ID, you can retrieve the result of the completed task.

```
# 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()
```

Monitoring tasks through the Amazon Braket console

Amazon Braket offers a convenient way of monitoring the task through the Amazon Braket console. All submitted tasks are listed in the Tasks field, as shown in the following figure. This service is region-specific, which means that you can view only those tasks created in the specific AWS Region.
### Tasks (18)

<table>
<thead>
<tr>
<th>Task id</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1fd4a807-b8bc-4cd2-b5e2-482db2f74fc7</td>
<td>✔ COMPLETED</td>
</tr>
<tr>
<td>1aefe2d1-beef-4564-8b4d-4222301a536f</td>
<td>✔ COMPLETED</td>
</tr>
<tr>
<td>479d27d9-b6d2-418e-9efa-55a54b4bc526</td>
<td>✔ COMPLETED</td>
</tr>
<tr>
<td>542e7d34-d6a3-45e9-9f77-734180d86869</td>
<td>✔ COMPLETED</td>
</tr>
<tr>
<td>250295bb-fd09-4312-97cf-e3bb85b83b88</td>
<td>✔ COMPLETED</td>
</tr>
<tr>
<td>002389de-f860-43b7-a39d-922b62a0175</td>
<td>✔ COMPLETED</td>
</tr>
</tbody>
</table>
You can search for particular tasks through the navigation bar. The search can be based on Task ARN (ID), Status, Device, and Creation time. The options appear automatically when you select the navigation bar, as shown in the next example.

Here is an example of searching for a task based on its unique task ID, which can be obtained by calling `task.id`. 
Working with Boto3

Boto3 is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as Amazon Braket. Boto3 provides an easy-to-use, object-oriented API, as well as low-level access to Amazon Braket. Getting started with Boto3 is easy, but it requires a few steps.

Follow the instructions in the Boto3 Quickstart guide to learn how to install and configure Boto3.

Boto3 provides the core functionality that works along with the Amazon Braket Python SDK to help you configure and run your quantum tasks. Python customers always need to install Boto3, because that is the core implementation. If you want to make use of additional helper methods, you also need to install the Amazon Braket SDK.

For example, when you call CreateQuantumTask, the Amazon Braket SDK will submit the request to Boto3, which then calls the AWS API.
Enable the Amazon Braket Boto3 client

To use Boto3 with Amazon Braket, you must import Boto3 and then define a client that you’ll be using to connect to the Amazon Braket API. In the example that follows, the Boto3 client is named `braket`.

```python
import boto3
# Create the braket client
braket = boto3.client('braket')
```

Now that you have a `braket` client established, you can make requests and process responses from the Amazon Braket service. You can get more detail on request and response data in the API Reference.

The following examples show more about working with devices and sessions. Examples include these tasks:

- Search for devices
- Retrieve devices
- Create a quantum task
- Retrieve a quantum task
- Search for quantum tasks
- Cancel quantum tasks

**Example**

**Example: Search for devices**

```python
• search_devices(**kwargs)

Searches for devices using the specified filters.
```

```python
# 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'])
```

**Example: Retrieve devices**

```python
• get_device(deviceArn)

Retrieves the devices available in Amazon Braket.
```

```python
# Pass the device ARN when sending the request and capture the response
response = braket.get_device(deviceArn='arn:aws:braket:::device/quantum-simulator/amazon/sv1')
print(f"Device {response['deviceName']} is {response['deviceStatus']}")
```
Example

Example: Create a quantum task

• `create_quantum_task(**kwargs)`

Creates a quantum task.

```python
# 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")
```

Example: Retrieve a quantum task

• `get_quantum_task(quantumTaskArn)`

Retrieves the specified quantum task.

```python
# 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'])
```

Example

Example: Search for quantum tasks

• `search_quantum_tasks(**kwargs)`

Searches for tasks that match the specified filter values.

```python
# Pass search filters and optional parameters when sending the request and capture the response
response = braket.search_quantum_tasks(filters=[
    'deviceArn',
    'operator': 'EQUAL',
])
```
Configure AWS CLI profiles for Boto3 and the Amazon Braket SDK

The Amazon Braket SDK relies upon the default AWS CLI credentials, unless you explicitly specify otherwise. We recommend that you keep the default when you run on a managed Amazon Braket notebook, because you must provide an IAM role that has permissions to launch the notebook instance.

Optionally, if you run your code locally, on an EC2 instance, for example, you can establish named AWS CLI profiles. You can give each profile a different permission set, rather than regularly overwriting the default profile.

This section provides a brief explanation of how to configure such a CLI profile, and how to incorporate that profile into Amazon Braket so that API calls are made with the permissions from that profile.

Step 1: Configure a local AWS CLI profile

It is beyond the scope of this document to explain how to create IAM Users and how to configure a non-default profile. For information on these topics, see:

- Configure an IAM User
- Establish a CLI profile

To use Amazon Braket, you must provide this IAM user — and the associated CLI profile — with necessary Braket permissions. For instance, you can attach the AmazonBraketFullAccess policy.

Step 2: Establish a Boto3 Session object

```python
from boto3 import Session

# Insert CLI profile name here
boto_sess = Session(profile_name='profile')
```
Note
If the expected API calls have region-based restrictions that are not aligned with your profile default Region, you can specify a Region for the Boto3 session, as shown in the following example.

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

Substitute a value of `region` that corresponds to one of the AWS Regions in which Amazon Braket is available, such as `us-east-1`, `us-west-1`, and so forth.

**Step 3: Incorporate the Boto3 Session into the Braket AwsSession**

The example shows how to initialize a Boto3 Braket session and instantiate a device in that session.

```python
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)
```

After this setup is complete, you can submit quantum tasks to that instantiated `AwsDevice` object, for example, by calling the `device.run(...)` command. All API calls made by that device can leverage the IAM credentials associated with the CLI profile that you previously designated as `profile`. 

---

57
Run hybrid algorithms with Amazon Braket

Hybrid algorithms are algorithms that contain classical and quantum instructions. The classical instructions are executed on classical hardware (an EC2 instance or your laptop), and the quantum instructions are executed either on a simulator or on a quantum computer.

Amazon Braket enables you to set up and run hybrid quantum algorithms with the assistance of the Amazon Braket PennyLane plugin, or with the Amazon Braket Python SDK and example notebook repositories. Amazon Braket example notebooks, based on the SDK, enable you to set up and run certain hybrid algorithms without the PennyLane plugin. However, we recommend PennyLane because it provides a much easier and richer experience.

About hybrid quantum algorithms

Hybrid quantum algorithms are important to the industry today, because contemporary quantum computing devices generally produce noise, or errors. Every quantum gate added to a computation increases the chance of adding noise; therefore, long-running algorithms can be overwhelmed by noise, and lost.

Pure quantum algorithms such as Shor’s (QPE example) or Grover’s (Grover’s example) require thousands, or millions, of operations. For this reason, they can be impractical for existing quantum devices, which are generally referred to as noisy intermediate-scale quantum (NISQ) devices.

In hybrid quantum algorithms, quantum processing units (QPUs) work as co-processors for classic CPUs, specifically to speed up certain calculations in a classical algorithm. Circuit executions become much shorter, within reach of the capabilities of today’s devices.

Amazon Braket with PennyLane

Amazon Braket provides support for PennyLane, an open-source software framework built around the concept of quantum differentiable programming. This framework allows you to train quantum circuits in the same way that you would train a neural network to find solutions for computational problems in quantum chemistry, quantum machine learning, and optimization.

The PennyLane library provides interfaces to familiar machine learning tools, including PyTorch and TensorFlow, to make training quantum circuits fast, easy and intuitive.

• **The PennyLane Library** — PennyLane is pre-installed in Amazon Braket notebooks. For access to Amazon Braket devices from PennyLane, open a notebook, and import the PennyLane library with this command:

```python
import pennylane as qml
```

Tutorial notebooks help you get started quickly. Alternatively, you can use PennyLane on Amazon Braket from any IDE of your choice.

• **The Amazon Braket PennyLane plugin** — To use your own IDE, you can install the Amazon Braket PennyLane plugin manually. The plugin connects PennyLane with the Amazon Braket Python SDK, so
you can run circuits in PennyLane on Amazon Braket devices. To install the PennyLane plugin, use this command:

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

The following example demonstrates how to set up access to Amazon Braket devices in PennyLane:

```python
# to use SV1
import pennylane as qml
s3 = ("my-bucket", "my-prefix")
sv1 = qml.device("braket.aws.qubit", device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1", s3_destination_folder=s3, 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)
```

You can find information and PennyLane tutorial examples in the Amazon Braket examples repository.

The Amazon Braket PennyLane plugin enables you to switch between Amazon Braket QPU and simulator devices in PennyLane with a single line of code. It offers two Amazon Braket quantum devices to work with:

- `braket.aws.qubit` for running with the Amazon Braket service's quantum devices, including QPUs and simulators
- `braket.local.qubit` for running with the Amazon Braket SDK's local simulator

The Amazon Braket PennyLane plugin is open source. It can be installed from the PennyLane Plugin GitHub repository.

To find out more, visit the PennyLane Documentation.

## Hybrid algorithms in Amazon Braket example notebooks

Amazon Braket does provide a variety of example notebooks that do not rely on the PennyLane plugin for running hybrid algorithms. You can get started with any of these Amazon Braket hybrid example notebooks that illustrate variational methods, such as the Quantum Approximate Optimization Algorithm (QAOA) or Variational Quantum Eigensolver (VQE).

The Amazon Braket example notebooks rely on the Amazon Braket Python SDK. The SDK provides a framework to interact with quantum computing hardware devices through Amazon Braket. It is an open source library, designed to assist you with the quantum portion of your hybrid workflow.
You can explore Amazon Braket further, with any of these other example notebooks.
Security in Amazon Braket

This chapter helps you understand how to apply the shared responsibility model when using Amazon Braket. It shows you how to configure Amazon Braket to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Braket resources.

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. You are responsible for other factors, including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

Shared responsibility for security

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon Braket, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – You are responsible for maintaining control over your content that is hosted on this AWS infrastructure. This content includes the security configuration and management tasks for the AWS services that you use.

Data protection

For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
- If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put sensitive identifying information, such as your customer account numbers, into free-form fields such as a Name field. This includes when you work with Braket or
other AWS services using the console, API, CLI, or AWS SDKs. Any data that you enter into Braket or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, do not include credentials information in the URL to validate your request to that server.

Data retention

After 90 days, Amazon Braket automatically removes all task IDs and other metadata associated with your tasks. As a result of this data retention policy, these tasks and results are no longer retrievable by search from the Amazon Braket console, although they remain stored in your S3 bucket.

If you need access to historical tasks and results that are stored in your S3 bucket for longer than 90 days, you must keep a separate record of your task ID and other metadata associated with that data. Be sure to save the information prior to 90 days. You can use that saved information to retrieve the historical data.

Managing access to Amazon Braket

This chapter describes the permissions that are required to run Amazon Braket, or to restrict the access of specific IAM users and roles. You can grant (or deny) the required permissions to any IAM user or role in your account. To do so, attach the appropriate Amazon Braket policy to that user or role in the account, as given in this chapter.

As a prerequisite, you must enable Amazon Braket. To enable Amazon Braket, be sure to sign in as a user or role that has (1) administrator permissions or (2) is assigned the AmazonBraketFullAccess policy and has permissions to create S3 buckets.

Add the AmazonBraketFullAccess policy to a user

You can skip this sequence if you already have created a user with the AmazonBraketFullAccess role and policy attached.

Step 1. Set up a user with the correct IAM access type

- Navigate to IAM console, select Users and choose Add User or select the user from the list of existing users.
- Fill in details for the user and access type.
- Choose Next:Permissions.
- Choose Attach existing policies directly.
- Select AmazonBraketFullAccess.
- Proceed through the Next buttons to create the user.

Step 2. Add the permissions policy to the user

- In the IAM console, choose Users.
- Select the user you created in Step 1.
- Under permissions, choose Add inline policy.
- Select JSON and replace the JSON string in the text box with the permissions policy given in following example code:
{  
"Version": "2012-10-17",
"Statement": [
  
  "Effect": "Allow",
  "Action": [
    "iam:CreateRole",
    "iam:CreatePolicy",
    "iam:AttachRolePolicy",
    "iam:UpdateRolePolicy"
  ],
  "Resource": [
    "arn:aws:iam::012345678901:role/service-role/AmazonBraketServiceSageMakerNotebookRole-*",
    "arn:aws:iam::012345678901:policy/service-role/AmazonBraketServiceSageMakerNotebookAccess-*"
  ]
}

**Note:** Replace the example account ID 012345678901 with the actual AWS account ID you are signed into.

**Step 3. Save the permissions policy you just created**

- Choose **Review Policy** and save it with a descriptive name, such as **AmazonBraketCreateRolePermissions**.

**Amazon Braket resources**

Amazon Braket creates one type of resource, which is the *quantum-task* resource. Here is the form of the ARN for that resource type:

- **Resource Name:** AWS::Service::Braket
- **ARN Regex:** arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}

**Notebooks and roles**

Notebooks are another type of resource that Amazon Braket utilizes on your behalf. A notebook is an Amazon SageMaker resource, which Braket is able to share. The notebooks require a specific IAM role to function: a role with a name that begins with **AmazonBraketServiceSageMakerNotebook**.

When you are creating a notebook in the Amazon Braket console, you'll see the option to create this new role. Your IAM user must have the **AmazonBraketFullAccess** role and **permissions policy** assigned and enabled to create the role.

To create the role, follow the steps given in Create a notebook, or have your administrator create it for you. Ensure that the **AmazonBraketFullAccess** policy is attached.

After you've created the role, you can reuse that role for all notebooks you launch in the future.

**About the AmazonBraketFullAccess policy**

The **AmazonBraketFullAccess** policy grants permissions for all Amazon Braket operations, including permissions for these tasks:
- **Store data in Amazon S3 buckets** – list the S3 buckets in your account, put objects into and get objects from any bucket in your account that starts with `amazon-braket-` in its name. These permissions are required for Amazon Braket to put files containing results from processed tasks into the bucket, and to retrieve them from the bucket.

- **Keep AWS CloudTrail logs** – all `describe`, `get`, and `list` actions, as well as starting and stopping queries, testing metrics filters, and filter log events. The AWS CloudTrail log file contains a record of all Amazon Braket API activity that occurs in your account.

- **Utilize roles to control resources** – Create a service-linked role in your account. The service-linked role has access to AWS resources on your behalf. It can be used only by the Amazon Braket service.

- **Maintain usage log files for your account** – Create, store, and view logging information about Amazon Braket usage in your account.

The **AmazonBraketFullAccess** policy artifact is shown in this example code:

```json
{
  "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": [
        "iam:ListRoles",
        "iam:ListRolePolicies",
        "iam:GetRole",
        "iam:GetRolePolicy",
        "iam:ListAttachedRolePolicies"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:ListNotebookInstances"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:CreatePresignedNotebookInstanceUrl"
      ],
      "Resource": "*"
    }
  ]
}
```
Restrict user access to certain devices

To restrict access for certain users to certain Amazon Braket devices, you can add a deny permissions policy to a specific IAM role.

The following example actually restricts access to all devices, for AWS account 012345678901.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:DeleteNotebookInstance"
      ],
      "Resource": "arn:aws:sagemaker:*:*:notebook-instance/amazon-braket-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker:DeleteNotebookInstanceLifecycleConfig"
      ],
      "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"
        }
      }
    },
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:iam::*:role/service-role/AmazonBraketServiceSageMakerNotebookRole*",
      "Condition": {
        "StringLike": {
          "iam:PassedToService": [
            "sagemaker.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```
Service-linked role

Amazon Braket service-linked role

When you enable Amazon Braket, a service-linked role is created in your account.

A service-linked role is a unique type of IAM role that, in this case, is linked directly to Amazon Braket. The Amazon Braket service-linked role is predefined to include all the permissions that Braket requires when calling other AWS services on your behalf.

A service-linked role makes setting up Amazon Braket easier because you don’t have to add the necessary permissions manually. Amazon Braket defines the permissions of its service-linked roles. Unless you change these definitions, only Amazon Braket can assume its roles. The defined permissions include the trust policy and the permissions policy. The permissions policy cannot be attached to any other IAM entity.

The service-linked role that Amazon Braket sets up is part of the AWS Identity and Access Management (IAM) service-linked roles capability. For information about other AWS services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have Yes in the Service-Linked Role column. Choose a Yes with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon Braket

Amazon Braket uses the service-linked role named with the prefix AWSServiceRoleForAmazonBraket.
The `AWSServiceRoleForAmazonBraket` service-linked role trusts the following services to assume the role:

- Amazon Braket

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the IAM User Guide.

The service-linked role in Amazon Braket is granted the following permissions by default:

- **Amazon S3** – permissions to list the buckets in your account, and put objects into and get objects from any bucket in your account with a name that starts with `amazon-braket-`.
- **Amazon CloudWatch Logs** – permissions to list and create log groups and the associated log streams, and put events into the log group created for Amazon Braket.

The following policy is attached to the `AWSServiceRoleForAmazonBraket` service-linked role:

```json
{"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"}}
  }
]
}
```

**Resilience in Amazon Braket**

The AWS global infrastructure is built around AWS Regions and Availability Zones.
Each Region provides multiple availability zones that are physically separated and isolated. These availability zones (AZs) are connected through low-latency, high-throughput, and highly redundant networking. As a result, availability zones are more highly available, fault tolerant, and scalable than traditional single- or multiple-datacenter infrastructures.

You can design and operate applications and databases that fail over between AZs automatically, without interruption.

For more information about AWS Regions and availability zones, see AWS Global Infrastructure.

Compliance validation for Amazon Braket

Your compliance responsibility when using Amazon Braket is determined by the sensitivity of your data, your company’s compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

Infrastructure Security in Amazon Braket

As a managed service, Amazon Braket is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

For access to Amazon Braket through the network, you make calls to published AWS APIs. Clients must support Transport Layer Security (TLS) 1.2 or later. Clients also must support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Security of Amazon Braket Hardware Providers

QPs on Amazon Braket are hosted by third-party hardware providers. When you run your task on a QPU, Amazon Braket uses the DeviceARN as an identifier when sending the circuit to the specified QPU for processing.

If you use Amazon Braket for access to quantum computing hardware operated by one of the third-party hardware providers, your circuit and its associated data are processed by hardware providers outside of facilities operated by AWS. Information about the physical location and AWS Region where each QPU is available can be found in the Device Details section of the Amazon Braket console.

Your content is anonymized. Only the content necessary to process the circuit is sent to third parties. AWS account information is not transmitted to third parties.
All data is encrypted at rest and in transit. Data is decrypted for processing only. Amazon Braket third-party providers are not permitted to store or use your content for purposes other than processing your circuit. Once the circuit completes, the results are returned to Amazon Braket and stored in your S3 bucket.

The security of Amazon Braket third-party quantum hardware providers is audited periodically, to ensure that standards of network security, access control, data protection, and physical security are met.
Troubleshooting

Solve common problems you might find when working with Amazon Braket.

Topics

In this section:

- A task is failing creation (p. 70)
- An SDK feature does not work (p. 70)
- Something stopped working in your notebook instance (p. 70)

A task is failing creation

If you receive an error along the lines of "An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-braket-..." make sure you are referring to an existing s3_folder since we do not auto create new Amazon S3 buckets and prefixes for you.

If you are accessing the API directly and getting an error like "Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET" make sure you are not including s3:// in the Amazon S3 bucket path.

An SDK feature does not work

Make sure your SDK (and schemas) are up-to-date. From the notebook or your python editor run

```bash
!pip install --upgrade amazon-braket-sdk
```

Make sure your SDK and schemas are up-to-date. To update the SDK from the notebook or your python editor run the following:

```bash
pip install --upgrade amazon-braket-schemas
```

If you are accessing Amazon Braket from your own client make sure your AWS region is set to one supported by Amazon Braket.

Something stopped working in your notebook instance

If some components of your notebook stop working, try the following:

1. Download any notebooks you created or modified to a local drive.
2. Stop your notebook instance.
3. Delete your notebook instance.
4. Create new notebook instance with a different name.
5. Upload the notebooks to the new instance.
Amazon VPC endpoints for Amazon Braket

You can establish a private connection between your VPC and Amazon Braket by creating an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology that enables access to Braket APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with Braket APIs.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

With PrivateLink, traffic between your VPC and Braket does not leave the Amazon network, which increases the security of data that you share with cloud-based applications, because it reduces your data's exposure to the public internet. For more information, see Interface VPC endpoints (AWS PrivateLink) in the Amazon VPC User Guide.

Considerations for Amazon Braket VPC endpoints

Before you set up an interface VPC endpoint for Braket, ensure that you review Interface endpoint properties and limitations in the Amazon VPC User Guide.

Braket supports making calls to all of its API actions from your VPC.

By default, full access to Braket is allowed through the VPC endpoint. You can control access if you specify VPC endpoint policies. For more information, see Controlling access to services with VPC endpoints in the Amazon VPC User Guide.

Set up Braket and PrivateLink

To use AWS PrivateLink with Amazon Braket, you must create an Amazon Virtual Private Cloud (Amazon VPC) endpoint as an interface, and then connect to the endpoint through the Amazon Braket API service.

Here are the general steps of this process, which are explained in detail in later sections.

- Configure and launch an Amazon VPC to host your AWS resources. If you already have a VPC, you can skip this step.
- Create an Amazon VPC endpoint for Braket
- Connect and run Braket tasks through your endpoint

Step 1: Launch an Amazon VPC if needed

Remember that you can skip this step if your account already has a VPC in operation.

A VPC controls your network settings, such as the IP address range, subnets, route tables, and network gateways. Essentially, you are launching your AWS resources in a custom virtual network. For more information about VPCs, see the Amazon VPC User Guide.
Open the Amazon VPC console and create a new VPC with subnets, security groups, and network gateways.

**Step 2: Create an interface VPC endpoint for Braket**

You can create a VPC endpoint for the Braket service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the Amazon VPC User Guide.

To create a VPC endpoint in the console, open the Amazon VPC console, open the Endpoints page, and proceed to create the new endpoint. Make note of the endpoint ID for later reference. It is required as part of the –endpoint-url flag when you are making certain calls to the Braket API.

Create the VPC endpoint for Braket using the following service name:

- `com.amazonaws.substitute_your_region.braket`

**Note:** If you enable private DNS for the endpoint, you can make API requests to Braket using its default DNS name for the Region, for example, `braket.us-east-1.amazonaws.com`.

For more information, see Accessing a service through an interface endpoint in the Amazon VPC User Guide.

**Step 3: Connect and run Braket tasks through your endpoint**

After you have created a VPC endpoint, you can run CLI commands that include the endpoint-url parameter to specify interface endpoints to the API or runtime, such as the following example:

```bash
aws braket search-quantum-tasks --endpoint-url VPC_Endpoint_ID.braket.substituteYourRegionHere.vpce.amazonaws.com
```

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint as a URL in your CLI commands. Instead, the Amazon Braket API DNS hostname, which the CLI and Braket SDK use by default, resolves to your VPC endpoint. It has the form shown in the following example:

`https://braket.substituteYourRegionHere.amazonaws.com`

The blog post called Direct access to Amazon SageMaker notebooks from Amazon VPC by using an AWS PrivateLink endpoint provides an example of how to set up an endpoint to make secure connections to SageMaker notebooks, which are similar to Amazon Braket notebooks.

If you’re following the steps in the blog post, remember to substitute the name Amazon Braket for Amazon SageMaker. For Service Name enter `com.amazonaws.us-east-1.braket` or substitute your correct AWS Region name into that string, if your Region is not `us-east-1`.

**More about creating an endpoint**

- For information about how to create a VPC with private subnets, see Create a VPC with private subnets
- For information about creating and configuring an endpoint using the Amazon VPC console or the AWS CLI, see Creating an Interface Endpoint in the Amazon VPC User Guide.
Control access with Amazon VPC endpoint policies

To control connectivity access to Amazon Braket, you can attach an AWS Identity and Access Management (IAM) endpoint policy to your Amazon VPC endpoint. The policy specifies the following information:

- The principal (user or role) that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the Amazon VPC User Guide.

Example: VPC endpoint policy for Braket actions

The following example shows an endpoint policy for Braket. When attached to an endpoint, this policy grants access to the listed Braket actions for all principals on all resources.

```
{
    "Statement": [
        {
            "Principal":"*",
            "Effect":"Allow",
            "Action": [
                "braket:action-1",
                "braket:action-2",
                "braket:action-3"
            ],
            "Resource":"*"
        }
    ]
}
```

You can create complex IAM rules by attaching multiple endpoint policies. For more information and examples, see:

- Amazon Virtual Private Cloud Endpoint Policies for Step Functions
- Creating Granular IAM Permissions for Non-Admin Users
- Controlling Access to Services with VPC Endpoints
Tagging Amazon Braket resources

A tag is a custom attribute label that you assign or that AWS assigns to an AWS resource. A tag is metadata that tells more about your resource. Each tag consists of a key and a value. Together these are known as key-value pairs. For tags that you assign, you define the key and value.

In the Amazon Braket console, you can navigate to a task or a notebook and view the list of tags associated with it. You can add a tag, remove a tag, or modify a tag. You can tag a task or notebook upon creation, and then manage associated tags through the console, AWS CLI, or API.

Using tags

Tags can organize your resources into categories that are useful to you. For example, you can assign a "Department" tag to specify the department that owns this resource.

Each tag has two parts:

• A tag key (for example, CostCenter, Environment, or Project). Tag keys are case sensitive.
• An optional field known as a tag value (for example, 111122223333 or Production). Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Tags help you do the following things:

• Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related.
• Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use cost allocation tags in the AWS Billing and Cost Management User Guide.
• Control access to your AWS resources. For more information, see Controlling access using tags in the IAM User Guide.

More about AWS and tags

• For general information on tagging, including naming and usage conventions, see Tagging AWS Resources in the AWS General Reference.
• For information about restrictions on tagging, see Tag naming limits and requirements in the AWS General Reference.
• For best practices and tagging strategies, see Tagging best practices and AWS Tagging Strategies.
• For a list of services that support using tags, see the Resource Groups Tagging API Reference.

The following sections provide more specific information about tags for Amazon Braket.

Supported resources in Amazon Braket

The following resource type in Amazon Braket supports tagging:
Tag restrictions

The following basic restrictions apply to tags on Amazon Braket resources:

- Maximum number of tags that you can assign to a resource: 50
- Maximum key length: 128 Unicode characters
- Maximum value length: 256 Unicode characters
- Valid characters for key and value: a–z, A–Z, 0–9, space, and these characters: _ . : / = + - , @
- Keys and values are case sensitive
- Don’t use aws as a prefix for keys; it’s reserved for AWS use.

Managing tags in Amazon Braket

You set tags as properties on a resource. You can view, add, modify, list, and delete tags through the Amazon Braket console, the Amazon Braket API, or the AWS CLI. For more information, see the Amazon Braket API reference.

Add tags

You can add tags to taggable resources at the following times:

- **When you create the resource:** Use the console, or include the Tags parameter with the Create operation in the AWS API.
- **After you create the resource:** Use the console to navigate to the task or notebook resource, or call the TagResource operation in the AWS API.

To add tags to a resource when you create it, you also need permission to create a resource of the specified type.

View tags

You can view the tags on any of the taggable resources in Amazon Braket by using the console to navigate to the task or notebook resource, or by calling the AWS ListTagsForResource API operation.

You can use the following AWS API command to view tags on a resource:

- **AWS API:** ListTagsForResource
Edit tags

You can edit tags by using the console to navigate to the task or notebook resource, or you can use the following command to modify the value for a tag attached to a taggable resource. When you specify a tag key that already exists, the value for that key is overwritten:

- **AWS API**: TagResource

Remove tags

You can remove tags from a resource by specifying the keys to remove, by using the console to navigate to the task or notebook resource, or when calling the UntagResource operation.

- **AWS API**: UntagResource

Example of CLI tagging in Amazon Braket

If you're working with the AWS CLI, here is an example command showing how to create a tag that applies to a quantum task you create for the SV1 simulator, with parameter settings of the Rigetti QPU. Notice that the tag is specified at the end of the example command. In this case, **Key** is given the value **state** and **Value** is given the value Washington.

```
aws braket create-quantum-task --action /""braketSchemaHeader": {"name": "braket.ir.jaqcd.program","/ 
"version": "1"}, / 
"instructions": [{"angle": 0.15, "target": 0, "type": "rz"}], 
"results": null, / 
"basis_rotation_instructions": null"} / 
--device-arn "arn:aws:braket:::device/quantum-simulator/amazon/sv1" / 
--output-s3-bucket "my-example-braket-bucket-name" / 
--output-s3-key-prefix "my-example-username" / 
--shots 100 / 
--device-parameters / 
"{"braketSchemaHeader": {"name": "braket.device_schema.rigetti.rigetti_device_parameters","/ 
"version": "1"}, "paradigmParameters": {"braketSchemaHeader": {"name": "braket.device_schema.gate_model_parameters","/ 
"version": "1"}, "qubitCount": 2}="/ 
--tags {"state":"Washington"}
```

Tagging with the Amazon Braket API

- If you’re using the Amazon Braket API to set up tags on a resource, call the TagResource API.

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

- To remove tags from a resource, call the UntagResource API.

  `aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN`
• To list all tags that are attached to a particular resource, call the `ListTagsForResource` API.

```bash
aws braket tag-resource --resource-arn $YOUR_TASKARN --tag-keys "["city", "state"]"
```
Monitoring Amazon Braket with Amazon CloudWatch

You can monitor Amazon Braket using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. You view historical information up to 15 months, or search metrics that have been updated in the last 2 weeks in the Amazon CloudWatch console to gain a better perspective on how Amazon Braket is performing. To learn more, see Using CloudWatch metrics.

Amazon Braket Metrics and Dimensions

Metrics are the fundamental concept in CloudWatch. A metric represents a time-ordered set of data points that are published to CloudWatch. Every metric is characterized by a set of dimensions. To learn more about metrics dimensions in CloudWatch, see CloudWatch dimensions.

Amazon Braket sends the following metric data, specific to Amazon Braket, into the Amazon CloudWatch metrics:

Task Metrics

Metrics are available if tasks exist. They are displayed under AWS/Braket/By Device in the CloudWatch console.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Number of tasks.</td>
</tr>
<tr>
<td>Latency</td>
<td>This metric is emitted when a task has completed. It represents the total</td>
</tr>
<tr>
<td></td>
<td>time from task initialization to completion.</td>
</tr>
</tbody>
</table>

Dimensions for Task Metrics

The task metrics are published with a dimension based on the deviceArn parameter, which has the form arn:aws:braket:::device/xxx.

Supported Devices

For a list of supported devices and device ARNs, see Braket devices.

Note

You can view the CloudWatch log streams for Amazon Braket notebooks by navigating to the Notebook detail page on the Amazon SageMaker console. Additional Amazon Braket notebook settings are available through the SageMaker console. Open the SageMaker console at https://console.aws.amazon.com/sagemaker/.
Events and automated actions for Amazon Braket with Amazon EventBridge

Amazon EventBridge monitors status change events in Amazon Braket tasks. Events from Amazon Braket are delivered to EventBridge, almost in real time. You can write simple rules that indicate which events interest you, including automated actions to take when an event matches a rule. Automatic actions that can be triggered include these:

- Invoking an AWS Lambda function
- Activating an AWS Step Functions state machine
- Notifying an Amazon SNS topic

EventBridge monitors these Amazon Braket status change events:

- The state of task changes

Amazon Braket guarantees delivery of task status change events. These events are delivered at least once, but possibly out of order.

For more information, see the Events and Event Patterns in EventBridge.

Monitor task status with EventBridge

With EventBridge, you can create rules that define actions to take when Amazon Braket sends notification of a status change regarding a Braket task. For example, you can create a rule that sends you an email message each time the status of a task changes.

- Log in to AWS using an account that has permissions to use EventBridge and Amazon Braket.
- Open the Amazon EventBridge console at https://console.aws.amazon.com/events/.
- Choose Create rule.
- Enter a Name for the rule, and, optionally, a description.
- Under Define pattern choose Event pattern.
- Under Event matching pattern, choose Custom pattern.
- In the Event pattern box, add the following pattern and then choose Save.

```json
{
    "source": [
        "aws.braket"
    ],
    "detail-type": [
        "Braket Task State Change"
    ]
}
```
In the **Select event bus** section, choose the event bus to use. If you have not created a custom event bus, choose **AWS default event bus**.

Confirm that **Enable the rule on the selected event bus** is toggled on.

Under **Select targets**, choose the target action to take when a task state change event is received from Amazon Braket.

For example, use an Amazon Simple Notification Service (SNS) topic to send an email or text message when an event occurs. To do that, first create an Amazon SNS topic using the Amazon SNS console. To learn more, see [Using Amazon SNS for user notifications](#).

Optionally, choose **Add target** to specify an additional target action for the event rule.

Choose **Create**.

To capture all events from Amazon Braket, exclude the `detail-type` section as shown in the following code:

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

### Example Amazon Braket event

The following example shows a task status change event:

```json
{
   "version": "0",
   "id": "foobar",
   "detail-type": "Braket Task State Change",
   "source": "aws.braket",
   "account": "foobar",
   "time": "2020-08-06T05:10:45Z",
   "region": "us-east-1",
   "resources": [ 
      "foobar"
   ],
   "detail": {
      "quantumTaskArn": "foobar",
      "created": "foobar",
      "irType": "GA-MODEL",
      "shots": "100",
      "resultsS3ObjectKey": "Aug2020/sanity/24de4823-9688-4b7d-b916-32b547ab6454",
      "resultsS3Bucket": "braket-load-tests-013039061202",
      "modified": "foobar",
      "backendArn": "foobar",
      "status": "COMPLETED"
   }
}
```
Amazon Braket API logging with CloudTrail

Amazon Braket is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Braket. CloudTrail captures all API calls for Amazon Braket as events. The calls captured include calls from the Amazon Braket console and code calls to the Amazon Braket API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon Braket. If you do not configure a trail, you can still view the most recent events in the CloudTrail console in Event history. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Braket, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon Braket Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon Braket, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for Amazon Braket, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

All Amazon Braket actions are logged by CloudTrail. For example, calls to the GetQuantumTask or GetDevice actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.
Understanding Amazon Braket Log File Entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example is a log entry for the `GetQuantumTask` action, which gets the details of a quantum task.

```
{
  "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"
      }
    },
    "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.metall.x86_64 botocore/1.17.33",
  "requestParameters": {
    "quantumTaskArn": "foobar"
  },
  "responseElements": null,
  "requestID": "20e8000c-29b8-4137-9c7b-af77d1dd12f7",
  "eventID": "4a2f2b22-a73d-414a-b30f-c0797c088f7c",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```

The following shows a log entry for the `GetDevice` action, which returns the details of a device event.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
    "account": "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"
      }
    },
    "webIdFederationData": {},
    "attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2020-08-07T00:56:57Z"
    }
  },
  "eventTime": "2020-08-07T01:00:08Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetDevice",
  "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.metall.x86_64 botocore/1.17.33",
  "requestParameters": {
    "deviceArn": "foobar"
  },
  "responseElements": null,
  "requestID": "20e8000c-29b8-4137-9c7b-af77d1dd12f7",
  "eventID": "4a2f2b22-a73d-414a-b30f-c0797c088f7c",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "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"}
Document history

The following table describes the documentation for this release of Amazon Braket.

- **API version**: 2019-09-01
- **Latest API Reference update**: November 9, 2020
- **Latest documentation update**: December 8, 2020

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>PennyLane support</td>
<td>Added support for PennyLane on Amazon Braket</td>
<td>December 8, 2020</td>
</tr>
<tr>
<td>New simulator</td>
<td>Added support for a Tensor Network Simulator, which allows larger circuits</td>
<td>December 8, 2020</td>
</tr>
<tr>
<td>Task batching</td>
<td>Braket supports customer task batching</td>
<td>November 24, 2020</td>
</tr>
<tr>
<td>Manual qubit allocation</td>
<td>Braket supports manual qubit allocation on the Rigetti device</td>
<td>November 24, 2020</td>
</tr>
<tr>
<td>Adjustable quotas</td>
<td>Braket supports self-service adjustable quotas for your task resources</td>
<td>October 30, 2020</td>
</tr>
<tr>
<td>Support for PrivateLink</td>
<td>You can set up private VPC endpoints for your Braket jobs</td>
<td>October 30, 2020</td>
</tr>
<tr>
<td>Support for tags</td>
<td>Braket supports API-based tags for the <code>quantum-task</code> resource</td>
<td>October 30, 2020</td>
</tr>
<tr>
<td>New D-Wave device</td>
<td>Added support for an additional D-Wave QPU, Advantage_system1</td>
<td>September 29, 2020</td>
</tr>
<tr>
<td>Initial release</td>
<td>Initial release of the Amazon Braket documentation</td>
<td>August 12, 2020</td>
</tr>
</tbody>
</table>