aws

Panduan Developer

Amazon Braket

Amazon Braket: Panduan Developer

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan kekayaan masing-masing pemiliknya, yang mungkin atau mungkin tidak berafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu Amazon Braket?

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

AmazonBraket sepenuhnya dikelola Layanan AWS yang membantu para peneliti, ilmuwan, dan pengembang memulai dengan komputasi kuantum. Komputasi kuantum memiliki potensi untuk memecahkan masalah komputasi yang berada di luar jangkauan komputer klasik karena memanfaatkan hukum mekanika kuantum untuk memproses informasi dengan cara baru.

Mendapatkan akses ke perangkat keras komputasi kuantum bisa mahal dan merepotkan. Akses terbatas membuat sulit untuk menjalankan algoritme, mengoptimalkan desain, mengevaluasi keadaan teknologi saat ini, dan merencanakan kapan harus menginvestasikan sumber daya Anda untuk keuntungan maksimal. Braket membantu Anda mengatasi tantangan ini.

Braket menawarkan satu titik akses ke berbagai teknologi komputasi kuantum. Dengan Braket, Anda dapat:

- Jelajahi dan rancang algoritma kuantum dan hibrida.
- Uji algoritma pada simulator sirkuit kuantum yang berbeda.
- Jalankan algoritma pada berbagai jenis komputer kuantum.
- Buat bukti aplikasi konsep.

Mendefinisikan masalah kuantum dan pemrograman komputer kuantum untuk menyelesaikannya membutuhkan seperangkat keterampilan baru. Untuk membantu Anda mendapatkan keterampilan ini, Braket menawarkan lingkungan yang berbeda untuk mensimulasikan dan menjalankan algoritma kuantum Anda. Anda dapat menemukan pendekatan yang paling sesuai dengan kebutuhan Anda dan memulai dengan cepat dengan serangkaian contoh lingkungan yang disebut notebook.

Pengembangan Braket memiliki tiga tahap - membangun, menguji, dan menjalankan:

Build - Braket menyediakan lingkungan notebook Jupyter yang dikelola sepenuhnya yang memudahkan untuk memulai. Notebook Braket sudah diinstal sebelumnya dengan algoritma sampel,

sumber daya, dan alat pengembang, termasuk Braket SDK. Amazon Dengan Amazon Braket SDK, Anda dapat membangun algoritma kuantum dan kemudian menguji dan menjalankannya pada komputer kuantum dan simulator yang berbeda dengan mengubah satu baris kode.

Tes - Braket menyediakan akses ke simulator sirkuit kuantum kinerja tinggi yang dikelola sepenuhnya. Anda dapat menguji dan memvalidasi sirkuit Anda. Braket menangani semua komponen perangkat lunak yang mendasarinya dan cluster Amazon Elastic Compute Cloud (Amazon EC2) untuk menghilangkan beban simulasi sirkuit kuantum pada infrastruktur komputasi kinerja tinggi klasik (HPC).

Run - Braket menyediakan akses yang aman dan sesuai permintaan ke berbagai jenis komputer kuantum. Anda memiliki akses ke komputer kuantum berbasis gerbang dariIonQ,, dan OQCRigetti, serta Simulator Hamiltonian Analog dari. QuEra Anda juga tidak memiliki komitmen di muka, dan tidak perlu mendapatkan akses melalui penyedia individu.

Tentang komputasi kuantum dan Braket

Komputasi kuantum sedang dalam tahap perkembangan awal. Penting untuk dipahami bahwa tidak ada komputer kuantum yang universal dan memiliki toleransi kesalahan yang ada saat ini. Oleh karena itu, beberapa jenis perangkat keras kuantum lebih cocok untuk setiap kasus penggunaan dan sangat penting untuk memiliki akses ke berbagai perangkat keras komputasi. Braket menawarkan berbagai perangkat keras melalui penyedia pihak ketiga.

Perangkat keras kuantum yang ada terbatas karena kebisingan, yang memperkenalkan kesalahan. Industri ini berada di era Noisy Intermediate Scale Quantum (NISQ). Di era NISQ, perangkat komputasi kuantum terlalu berisik untuk mempertahankan algoritme kuantum murni, seperti Algoritme Shor atau Algoritma Grover. Sampai koreksi kesalahan kuantum yang lebih baik tersedia, komputasi kuantum yang paling praktis membutuhkan kombinasi sumber daya komputasi klasik (tradisional) dengan komputer kuantum untuk membuat algoritma hibrida. Braket membantu Anda bekerja dengan algoritma kuantum hibrida.

Dalam algoritme kuantum hibrid, unit pemrosesan kuantum (QPU) digunakan sebagai co-processor untuk CPU, sehingga mempercepat penghitungan spesifik dalam algoritme klasik. Algoritme ini memanfaatkan pengolahan berulang, di mana komputasi bergerak antara komputer klasik dan kuantum. Misalnya, aplikasi komputasi kuantum saat ini di bidang kimia, optimasi, dan machine learning didasarkan pada algoritme kuantum variasional, yang merupakan jenis Algoritme kuantum hibrid. Dalam algoritme kuantum variasional, rutinitas optimasi klasik menyesuaikan parameter sirkuit kuantum berparameter secara iteratif, dengan cara yang sama bobot jaringan saraf disesuaikan secara iteratif berdasarkan kesalahan dalam set pelatihan pembelajaran mesin. Braket menawarkan

akses ke pustaka perangkat lunak PennyLane open source, yang membantu Anda dengan algoritma kuantum variasional.

Komputasi kuantum mendapatkan traksi untuk penghitungan di empat bidang utama:

- Teori bilangan — termasuk anjak piutang dan kriptografi (misalnya, algoritma Shor adalah metode kuantum utama untuk perhitungan teori bilangan)
- Optimalisasi — termasuk kepuasan kendala, pemecahan sistem linier, dan pembelajaran mesin
- Komputasi orakular — termasuk pencarian, subkelompok tersembunyi, dan pencarian urutan (misalnya, algoritma Grover adalah metode kuantum utama untuk komputasi orakular)
- Simulasi — termasuk simulasi langsung, invarian simpul, dan aplikasi algoritma optimasi perkiraan kuantum (QAOA)

Aplikasi untuk kategori komputasi ini dapat ditemukan, misalnya, di layanan keuangan, bioteknologi, manufaktur, dan farmasi,. Braket menawarkan kemampuan dan contoh notebook yang sudah dapat diterapkan pada banyak bukti masalah konsep selain masalah praktis tertentu.

Syarat dan konsep Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Istilah dan konsep berikut digunakan dalam Braket:

Simulasi Hamiltonian Analog

Analog Hamiltonian Simulation (AHS) adalah paradigma komputasi kuantum yang berbeda untuk simulasi langsung dinamika kuantum yang bergantung pada waktu dari sistem banyak benda. Dalam AHS, pengguna secara langsung menentukan Hamiltonian yang bergantung pada waktu dan komputer kuantum disetel sedemikian rupa sehingga secara langsung meniru evolusi waktu berkelanjutan di bawah Hamiltonian ini. Perangkat AHS biasanya perangkat tujuan khusus dan bukan komputer kuantum universal seperti perangkat berbasis gerbang. Mereka terbatas pada kelas Hamiltonian yang dapat mereka simulasikan. Namun, karena Hamiltonian ini secara alami

diimplementasikan pada perangkat, AHS tidak mengalami overhead yang diperlukan untuk merumuskan algoritme sebagai sirkuit dan mengimplementasikan operasi gerbang.

Braket

Kami menamai layanan Braket setelah notasi bra-ket, notasi standar dalam mekanika kuantum. Ini diperkenalkan oleh Paul Dirac pada tahun 1939 untuk menggambarkan keadaan sistem kuantum, dan juga dikenal sebagai notasi Dirac.

Pekerjaan hybrid braket

AmazonBraket memiliki fitur yang disebut Amazon Braket Hybrid Jobs yang menyediakan eksekusi algoritma hybrid yang dikelola sepenuhnya. Pekerjaan hybrid Braket terdiri dari tiga komponen:

1. Definisi algoritma Anda, yang dapat disediakan sebagai skrip, modul Python, atau wadah Docker.
2. Instans pekerjaan hybrid, berdasarkan Amazon EC2, untuk menjalankan algoritme Anda. Defaultnya adalah instance ml.m5.xlarge.
3. Perangkat kuantum untuk menjalankan tugas kuantum yang merupakan bagian dari algoritme Anda. Pekerjaan hibrida tunggal biasanya berisi kumpulan banyak tugas kuantum.

Perangkat

Di Amazon Braket, perangkat adalah backend yang dapat menjalankan tugas kuantum. Perangkat dapat berupa QPU atau simulator sirkuit kuantum. Untuk mempelajari selengkapnya, lihat Perangkat yang didukung Amazon Braket.

Komputasi kuantum berbasis gerbang

Dalam komputasi kuantum berbasis gerbang (QC), juga disebut QC berbasis sirkuit, komputasi dipecah menjadi operasi dasar (gerbang). Set gerbang tertentu bersifat universal, artinya setiap perhitungan dapat dinyatakan sebagai urutan terbatas dari gerbang tersebut. Gerbang adalah blok bangunan sirkuit kuantum dan analog dengan gerbang logika sirkuit digital klasik.

Hamiltonian

Dinamika kuantum sistem fisik ditentukan oleh Hamiltonian-nya, yang mengkodekan semua informasi tentang interaksi antara konstituen sistem dan efek kekuatan pendorong eksogen. Hamiltonian dari sistem N-qubit umumnya direpresentasikan sebagai matriks $2^N \times 2^N$ bilangan kompleks pada mesin klasik.

Dengan menjalankan Simulasi Hamiltonian Analog pada perangkat kuantum, Anda dapat menghindari persyaratan sumber daya eksponensial ini.

Denyut nadi

Pulsa adalah sinyal fisik sementara yang ditransmisikan ke qubit. Ini dijelaskan oleh bentuk gelombang yang dimainkan dalam bingkai yang berfungsi sebagai dukungan untuk sinyal pembawa dan terikat ke saluran perangkat keras atau port. Pelanggan dapat merancang pulsa mereka sendiri dengan menyediakan amplop analog yang memodulasi sinyal pembawa sinusoidal frekuensi tinggi. Bingkai secara unik dijelaskan oleh frekuensi dan fase yang sering dipilih untuk berada pada resonansi dengan pemisahan energi antara tingkat energi untuk |0# dan |1# dari qubit. Gerbang dengan demikian diberlakukan sebagai pulsa dengan bentuk yang telah ditentukan dan parameter yang dikalibrasi seperti amplitudo, frekuensi, dan durasinya. Kasus penggunaan yang tidak tercakup oleh bentuk gelombang templat akan diaktifkan melalui bentuk gelombang khusus yang akan ditentukan pada resolusi sampel tunggal dengan memberikan daftar nilai yang dipisahkan oleh waktu siklus fisik yang tetap.

Sirkuit kuantum

Sirkuit kuantum adalah serangkaian instruksi yang mendefinisikan komputasi pada komputer kuantum berbasis gerbang. Sirkuit kuantum adalah urutan gerbang kuantum, yang merupakan transformasi reversibel pada qubit register, bersama dengan instruksi pengukuran.

Simulator sirkuit kuantum

Simulator sirkuit kuantum adalah program komputer yang berjalan pada komputer klasik dan menghitung hasil pengukuran dari sirkuit kuantum. Untuk sirkuit umum, kebutuhan sumber daya simulasi kuantum tumbuh secara eksponensial dengan jumlah simulasi. qubits Braket menyediakan akses ke simulator sirkuit kuantum terkelola (diakses melalui BraketAPI) dan lokal (bagian dari Amazon Braket SDK).

Komputer kuantum

Komputer kuantum adalah perangkat fisik yang menggunakan fenomena mekanika kuantum, seperti superposisi dan keterikatan, untuk melakukan perhitungan. Ada paradigma yang berbeda untuk komputasi kuantum (QC), seperti QC berbasis gerbang.

Unit pemrosesan kuantum (QPU)

QPU adalah perangkat komputasi kuantum fisik yang dapat berjalan pada tugas kuantum. QPU dapat didasarkan pada paradigma QC yang berbeda, seperti QC berbasis gerbang. Untuk mempelajari selengkapnya, lihat Perangkat yang didukung Amazon Braket.

Gerbang asli QPU

Gerbang asli QPU dapat langsung dipetakan untuk mengontrol pulsa oleh sistem kontrol QPU. Gerbang asli dapat dijalankan pada perangkat QPU tanpa kompilasi lebih lanjut. Subset dari gerbang yang didukung QPU. Anda dapat menemukan gerbang asli perangkat di halaman Perangkat di konsol Braket dan melalui Amazon Braket SDK.

Gerbang yang didukung QPU

Gerbang yang didukung QPU adalah gerbang yang diterima oleh perangkat QPU. Gerbang ini mungkin tidak dapat langsung berjalan di QPU, yang berarti bahwa gerbang tersebut mungkin perlu didekomposisi menjadi gerbang asli. Anda dapat menemukan gerbang perangkat yang didukung pada halaman Perangkat di konsol Braket dan melalui Amazon Braket Amazon SDK.

Tugas kuantum

Di Braket, tugas kuantum adalah permintaan atom ke perangkat. Untuk perangkat QC berbasis gerbang, ini termasuk sirkuit kuantum (termasuk instruksi pengukuran dan jumlahshots) dan metadata permintaan lainnya. Anda dapat membuat tugas kuantum melalui Amazon Braket SDK atau dengan menggunakan CreateQuantumTask API operasi secara langsung. Setelah Anda membuat tugas kuantum, itu akan antri sampai perangkat yang diminta tersedia. Anda dapat melihat tugas kuantum Anda di halaman Quantum Tasks konsol Amazon Braket atau dengan menggunakan GetQuantumTask atau SearchQuantumTasks API operasi.

Qubit

Unit dasar informasi dalam komputer kuantum disebut qubit (bit kuantum), seperti bit dalam komputasi klasik. A qubit adalah sistem kuantum dua tingkat yang dapat diwujudkan dengan implementasi fisik yang berbeda, seperti sirkuit superkonduktor atau ion dan atom individu. qubitJenis lain didasarkan pada foton, putaran elektronik atau nuklir, atau sistem kuantum yang lebih eksotis.

Queue depth

Queue depthmengacu pada jumlah tugas kuantum dan pekerjaan hibrida yang diantrian untuk perangkat tertentu. Tugas kuantum perangkat dan jumlah antrian pekerjaan hibrida dapat diakses melalui Braket Software Development Kit (SDK) atauAmazon Braket Management Console.

1. Kedalaman antrian tugas mengacu pada jumlah total tugas kuantum yang saat ini menunggu untuk dijalankan dalam prioritas normal.
2. Kedalaman antrian tugas prioritas mengacu pada jumlah total tugas kuantum yang dikirimkan yang menunggu untuk dijalankan. Amazon Braket Hybrid Jobs Tugas-tugas ini mendapatkan prioritas daripada tugas mandiri setelah pekerjaan hibrida dimulai.

3. Kedalaman antrian pekerjaan hibrida mengacu pada jumlah total pekerjaan hibrida yang saat ini mengantri di perangkat. Quantum tasksdiajukan sebagai bagian dari pekerjaan hibrida memiliki prioritas, dan digabungkan dalam. Priority Task Queue

Queue position

Queue positionmengacu pada posisi tugas kuantum Anda saat ini atau pekerjaan hibrida dalam antrian perangkat masing-masing. Ini dapat diperoleh untuk tugas kuantum atau pekerjaan hibrida melalui Braket Software Development Kit (SDK) atauAmazon Braket Management Console.

Shots

Karena komputasi kuantum secara inheren bersifat probabilistik, sirkuit apa pun perlu dievaluasi beberapa kali untuk mendapatkan hasil yang akurat. Eksekusi sirkuit tunggal dan pengukuran disebut tembakan. Jumlah tembakan (eksekusi berulang) untuk sirkuit dipilih berdasarkan akurasi yang diinginkan untuk hasilnya.

AWS terminologi dan tips untuk Amazon Braket

Kebijakan IAM

Kebijakan IAM adalah dokumen yang mengizinkan atau menolak izin dan sumber daya. Layanan AWS Kebijakan IAM memungkinkan Anda menyesuaikan tingkat akses pengguna ke sumber daya. Misalnya, Anda dapat mengizinkan pengguna mengakses semua bucket Amazon S3 di dalam bucket Anda Akun AWS, atau hanya bucket tertentu.

- Praktik terbaik: Ikuti prinsip keamanan setidaknya hak istimewa saat memberikan izin. Dengan mengikuti prinsip ini, Anda membantu mencegah pengguna atau peran memiliki lebih banyak izin daripada yang diperlukan untuk melakukan tugas kuantum mereka. Misalnya, jika seorang karyawan hanya membutuhkan akses ke bucket tertentu, tentukan bucket dalam kebijakan IAM alih-alih memberikan akses kepada karyawan ke semua bucket di dalam ember Anda. Akun AWS

Peran IAM

IAM role adalah identitas yang dapat Anda gunakan untuk mendapatkan akses sementara ke izin. Sebelum pengguna, aplikasi, atau layanan dapat mengambil peran IAM, mereka harus diberikan izin untuk beralih ke peran tersebut. Ketika seseorang menggunakan IAM role, mereka meninggalkan semua izin sebelumnya yang mereka miliki berdasarkan peran sebelumnya dan menggunakan izin peran baru.

- Praktik terbaik: IAM role ideal untuk situasi di mana akses ke layanan atau sumber daya perlu diberikan sementara, bukan jangka panjang.

Ember Amazon S3

Amazon Simple Storage Service (Amazon S3) adalah layanan Layanan AWS yang memungkinkan Anda menyimpan data sebagai objek dalam bucket. Bucket Amazon S3 menawarkan ruang penyimpanan tak terbatas. Ukuran objek maksimum di bucket Amazon S3 adalah 5 TB. Anda dapat mengunggah semua jenis data file ke bucket Amazon S3, seperti gambar, video, file teks, file cadangan, file media untuk situs web, dokumen yang diarsipkan, dan hasil tugas kuantum Braket Anda.

- Praktik terbaik: Anda dapat mengatur izin untuk mengontrol akses ke bucket S3 Anda. Untuk informasi lebih lanjut, lihat Kebijakan bucket dan kebijakan pengguna dalam dokumentasi Amazon S3.

Harga Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Dengan Amazon Braket, Anda memiliki akses ke sumber daya komputasi kuantum sesuai permintaan tanpa komitmen di muka. Pembayaran dilakukan sesuai penggunaan. Untuk mempelajari lebih lanjut tentang harga, kunjungi laman harga.

Pelacakan biaya mendekati waktu nyata

Braket SDK menawarkan opsi untuk menambahkan pelacakan biaya mendekati waktu nyata ke beban kerja kuantum Anda. Setiap contoh notebook kami menyertakan kode pelacakan biaya untuk memberi Anda perkiraan biaya maksimum pada unit pemrosesan kuantum (QPU) Braket dan simulator sesuai permintaan. Perkiraan biaya maksimum akan ditampilkan dalam USD dan tidak termasuk kredit atau diskon apa pun.

> **Note**
>
> Biaya yang ditampilkan adalah perkiraan berdasarkan simulator Amazon Braket dan penggunaan tugas unit pemrosesan kuantum (QPU) Anda. Perkiraan biaya yang ditampilkan mungkin berbeda dari biaya aktual Anda. Perkiraan biaya tidak memperhitungkan diskon atau kredit apa pun dan Anda mungkin mengalami biaya tambahan berdasarkan penggunaan layanan lain seperti Amazon Elastic Compute Cloud (Amazon EC2).

Pelacakan biaya untuk SV1

Untuk mendemonstrasikan bagaimana fungsi pelacakan biaya dapat digunakan, kami akan membangun sirkuit Bell State dan menjalankannya di simulator SV1 kami. Mulailah dengan mengimpor modul Braket SDK, mendefinisikan Bell State dan menambahkan `Tracker()` fungsi ke sirkuit kami:

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

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

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

Saat menjalankan Notebook, Anda dapat mengharapkan output berikut untuk simulasi Bell State. Fungsi pelacak akan menunjukkan jumlah bidikan yang dikirim, tugas kuantum yang diselesaikan, durasi eksekusi, durasi eksekusi yang ditagih, dan biaya maksimum Anda dalam USD. Waktu eksekusi Anda dapat bervariasi untuk setiap simulasi.

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

```
  'billed_execution_duration': datetime.timedelta(seconds=3)}}

tracker.simulator_tasks_cost()
$0.00375
```

Menggunakan pelacak biaya untuk mengatur biaya maksimum

Anda dapat menggunakan pelacak biaya untuk menetapkan biaya maksimum pada suatu program. Anda mungkin memiliki ambang batas maksimum untuk berapa banyak yang ingin Anda belanjakan untuk program tertentu. Dengan cara ini, Anda dapat menggunakan pelacak biaya untuk membangun logika kontrol biaya dalam kode eksekusi Anda. Contoh berikut mengambil sirkuit yang sama pada Rigetti QPU dan membatasi biaya hingga 1 USD. Biaya untuk menjalankan satu iterasi sirkuit dalam kode kami adalah 0,37 USD. Kami telah menetapkan logika untuk mengulangi iterasi sampai total biaya melebihi 1 USD; karenanya, cuplikan kode akan berjalan tiga kali sampai iterasi berikutnya melebihi 1 USD. Umumnya, sebuah program akan terus iterasi hingga mencapai biaya maksimum yang Anda inginkan, dalam hal ini - tiga iterasi.

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

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

> **Note**
>
> Pelacak biaya tidak akan melacak durasi untuk tugas TN1 kuantum yang gagal. Selama TN1 simulasi, jika latihan Anda selesai, tetapi langkah kontraksi gagal, biaya latihan Anda tidak akan ditampilkan di pelacak biaya.

Praktik terbaik untuk menghemat biaya

Pertimbangkan praktik terbaik berikut untuk menggunakan Amazon Braket. Hemat waktu, minimalkan biaya, dan hindari kesalahan umum.

Verifikasi dengan simulator

- Verifikasi sirkuit Anda menggunakan simulator sebelum Anda menjalankannya pada QPU, sehingga Anda dapat menyetel sirkuit Anda tanpa mengeluarkan biaya untuk penggunaan QPU.
- Meskipun hasil dari menjalankan sirkuit pada simulator mungkin tidak identik dengan hasil dari menjalankan sirkuit pada QPU, Anda dapat mengidentifikasi kesalahan coding atau masalah konfigurasi menggunakan simulator.

Batasi akses pengguna ke perangkat tertentu

- Anda dapat mengatur pembatasan yang mencegah pengguna yang tidak sah mengirimkan tugas kuantum pada perangkat tertentu. Metode yang disarankan untuk membatasi akses adalah dengan AWS IAM. Untuk informasi lebih lanjut tentang cara melakukannya, lihat Batasi Akses.
- Kami menyarankan agar Anda tidak menggunakan akun admin Anda sebagai cara untuk memberikan atau membatasi akses pengguna ke perangkat Amazon Braket.

Atur alarm penagihan

- Anda dapat mengatur alarm penagihan untuk memberi tahu Anda ketika tagihan mencapai batas preset. Cara yang disarankan untuk mengatur alarm adalah melalui AWS Budgets. Anda dapat mengatur anggaran khusus dan menerima peringatan ketika biaya atau penggunaan Anda mungkin melebihi jumlah yang dianggarkan. Informasi tersedia di AWS Budgets.

Uji tugas TN1 kuantum dengan jumlah tembakan rendah

- Simulator harganya lebih murah daripada QHP, tetapi simulator tertentu bisa mahal jika tugas kuantum dijalankan dengan jumlah tembakan tinggi. Kami menyarankan Anda menguji TN1 tugas Anda dengan shot hitungan rendah. Shothitungan tidak mempengaruhi biaya untuk SV1 dan tugas simulator lokal.

Periksa semua Wilayah untuk tugas kuantum

- Konsol menampilkan tugas kuantum hanya untuk Anda saat ini Wilayah AWS. Saat mencari tugas kuantum yang dapat ditagih yang telah dikirimkan, pastikan untuk memeriksa semua Wilayah.
- Anda dapat melihat daftar perangkat dan Wilayah terkaitnya di laman dokumentasi Perangkat yang didukung.

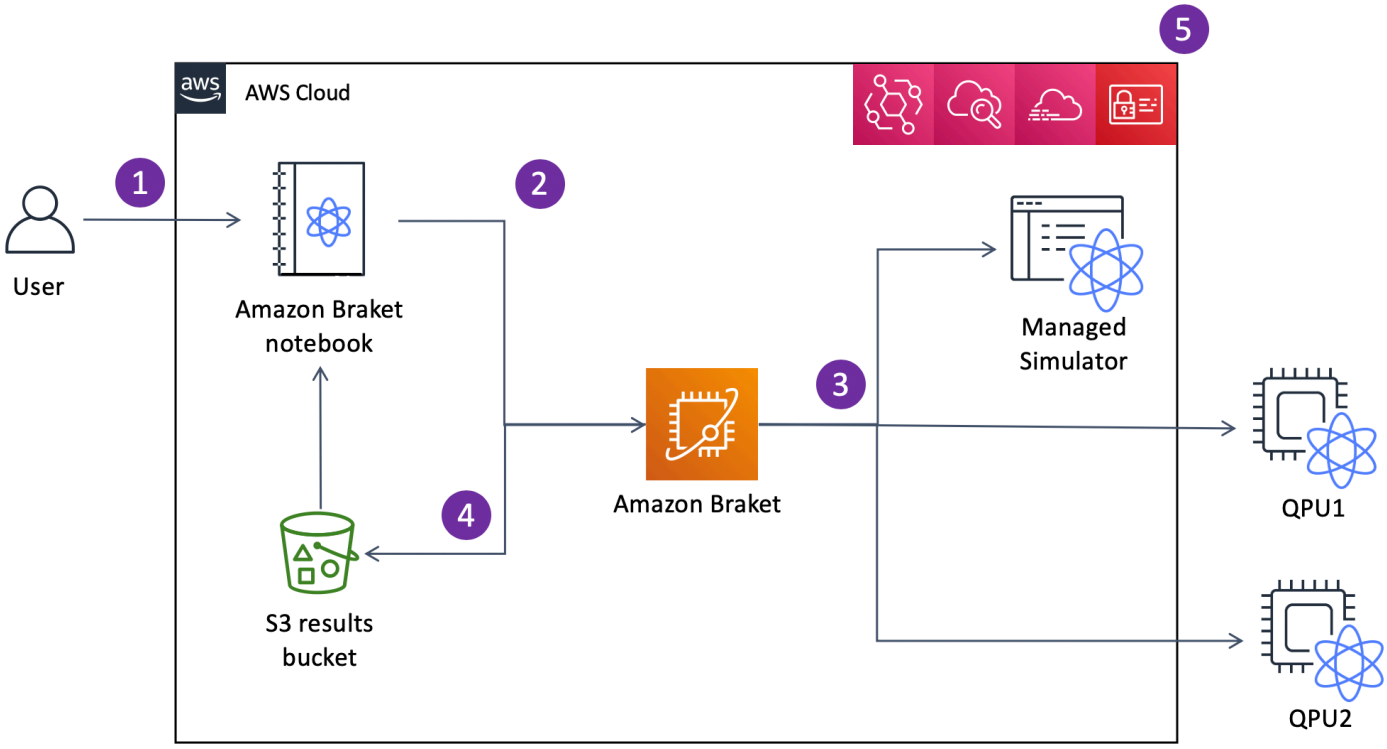
Cara kerja Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Amazon Braket menyediakan akses sesuai permintaan ke perangkat komputasi kuantum, termasuk simulator sirkuit sesuai permintaan dan berbagai jenis QPU. Di Amazon Braket, permintaan atom ke perangkat adalah tugas kuantum. Untuk perangkat QC berbasis gerbang, permintaan ini mencakup sirkuit kuantum (termasuk instruksi pengukuran dan jumlah bidikan) dan metadata permintaan lainnya. Untuk Simulator Hamiltonian Analog, tugas kuantum berisi tata letak fisik register kuantum dan ketergantungan waktu dan ruang dari medan manipulasi.

Pada bagian ini, kita akan belajar tentang aliran tingkat tinggi menjalankan tugas kuantum di Amazon Braket.

Alur tugas kuantum Amazon Braket



Dengan Jupyter notebook, Anda dapat dengan mudah menentukan, mengirimkan, dan memantau tugas kuantum Anda dari Konsol Amazon Braket atau menggunakan Amazon Braket SDK. Anda dapat membangun sirkuit kuantum Anda langsung di SDK. Namun, untuk Simulator Hamiltonian Analog, Anda menentukan tata letak register dan bidang pengontrol. Setelah tugas kuantum Anda ditentukan, Anda dapat memilih perangkat untuk menjalankannya dan mengirimkannya ke Amazon Braket API (2). Bergantung pada perangkat yang Anda pilih, tugas kuantum diantrian hingga perangkat tersedia dan tugas dikirim ke QPU atau simulator untuk implementasi (3). Amazon Braket memberi Anda akses ke berbagai jenis QPU (IonQ,,,Rigetti) Oxford Quantum Circuits (OQC)QuEra, tiga simulator sesuai permintaan (,,TN1), dua simulator lokal SV1DM1, dan satu simulator tertanam. Untuk mempelajari lebih lanjut, lihat Perangkat yang didukung Amazon Braket.

Setelah memproses tugas kuantum Anda, Amazon Braket mengembalikan hasilnya ke bucket Amazon S3, tempat data disimpan di Akun AWS (4). Pada saat yang sama, SDK melakukan polling untuk hasil di latar belakang dan memuatnya ke notebook Jupyter pada penyelesaian tugas kuantum. Anda juga dapat melihat dan mengelola tugas kuantum Anda di halaman Quantum Tasks di konsol Amazon Braket atau dengan menggunakan `GetQuantumTask` pengoperasian Amazon API Braket.

AmazonBraket terintegrasi dengan AWS Identity and Access Management (IAM) CloudWatch, Amazon, dan AWS CloudTrail Amazon EventBridge untuk manajemen akses pengguna, pemantauan dan pencatatan serta untuk pemrosesan berbasis peristiwa (5).

Pengolahan data pihak ketiga

Tugas kuantum yang dikirimkan ke perangkat QPU diproses pada komputer kuantum yang terletak di fasilitas yang dioperasikan oleh penyedia pihak ketiga. Untuk mempelajari lebih lanjut tentang keamanan dan pemrosesan pihak ketiga di Amazon Braket, lihat Keamanan Penyedia Perangkat Keras Amazon Braket.

Repositori inti dan plugin untuk Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Repositori inti

Berikut ini menampilkan daftar repositori inti yang berisi paket kunci yang digunakan untuk Braket:

- Braket Python SDK - Gunakan SDK Python Braket untuk mengatur kode Anda pada Jupyter notebook dalam bahasa pemrograman Python. Setelah Jupyter notebook Anda diatur, Anda dapat menjalankan kode Anda pada perangkat Braket dan simulator
- Skema Braket - Kontrak antara Braket SDK dan layanan Braket.
- Braket Default Simulator - Semua simulator kuantum lokal kami untuk Braket (vektor keadaan dan matriks kepadatan).

Plugin

Lalu ada berbagai plugin yang digunakan bersama dengan berbagai perangkat dan alat pemrograman. Ini termasuk plugin yang didukung Braket serta plugin yang didukung oleh pihak ketiga seperti yang ditunjukkan di bawah ini.

Amazon Braket didukung:

- Pustaka algoritma Amazon Braket - Katalog algoritme kuantum pra-bangun yang ditulis dengan Python. Jalankan mereka apa adanya atau gunakan sebagai titik awal untuk membangun algoritma yang lebih kompleks.
- Braket- PennyLane Plugin - Gunakan PennyLane sebagai kerangka kerja QMLpada Braket.

Pihak ketiga (tim Braket memantau dan berkontribusi):

- Penyedia Qiskit-Braket - Gunakan Qiskit SDK untuk mengakses sumber daya Braket.
- Braket-Julia SDK - (EKSPERIMENTAL) Versi asli Julia dari Braket SDK

Perangkat yang didukung Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Di Amazon Braket, perangkat mewakili QPU atau simulator yang dapat Anda hubungi untuk menjalankan tugas kuantum. Amazon Braket menyediakan akses ke perangkat QPU dariIonQ,,,,IQM, dan Oxford Quantum CircuitsQuEra, tiga simulator sesuai permintaanRigetti, tiga simulator lokal, dan satu simulator tertanam. Untuk semua perangkat, Anda dapat menemukan properti perangkat lebih lanjut, seperti topologi perangkat, data kalibrasi, dan set gerbang asli, di tab Perangkat di konsol Amazon Braket atau melalui API. `GetDevice` Saat membangun sirkuit dengan simulator, Amazon Braket saat ini mengharuskan Anda menggunakan qubit atau indeks yang berdekatan. Jika Anda bekerja dengan Amazon Braket SDK, Anda memiliki akses ke properti perangkat seperti yang ditunjukkan pada contoh kode berikut.

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

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

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

# get device properties
device.properties
```

Penyedia perangkat keras kuantum yang didukung

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

- Rigetti

Simulator yang didukung

- Simulator vektor negara bagian lokal (braket_sv) ('Simulator Default')
- Simulator matriks kepadatan lokal (`braket_dm`)
- Simulator AHS lokal
- Simulator vektor negara (SV1)
- Simulator matriks kepadatan (DM1)
- Simulator jaringan tensor () TN1
- PennyLaneSimulator Petir

Pilih simulator terbaik untuk tugas kuantum Anda

- Bandingkan simulator

> **Note**
>
> Untuk melihat yang tersedia Wilayah AWS untuk setiap perangkat, gulir ke kanan di tabel berikut.

Perangkat Amazon Braket

Penyedia	Nama perangkat	Paradigma	Tipe	Perangkat ARN	Wilayah
IonQ	Aria 1	berbasis gerbang	QPU	arn:aws:rem: us-timur-1: :perangkat/qpu/ionq/aria-1	us-east-1
IonQ	Aria 2	berbasis gerbang	QPU	arn:aws:rem: us-timur-1: :perangkat/qpu/ionq/aria-2	us-east-1

Penyedia	Nama perangkat	Paradigma	Tipe	Perangkat ARN	Wilayah
IonQ	Forte 1	berbasis gerbang	QPU (hanya reservasi)	arn:aws:rem: us-timur-1: :perangkat/qpu/ionq/forte-1	us-east-1
IonQ	Harmony	berbasis gerbang	QPU	arn:aws:rem: us-timur-1: :perangkat/QPU/ionq/harmoni	us-east-1
IQM	Garnet	berbasis gerbang	QPU	arn:aws:rem: eu-utara-1: :perangkat/QPU/IQM/Garnet	eu-north-1
Oxford Quantum Circuits	Lucy	berbasis gerbang	QPU	arn:aws:rem: eu-barat-2: :perangkat/QPU/OQC/Lucy	eu-west-2
QuEra	Aquila	Simulasi Hamiltonian Analog	QPU	arn:aws:braket:us-east-1: :perangkat/QPU/quera/Aquila	us-east-1
Rigetti	Aspen M-3	berbasis gerbang	QPU	arn:aws:rem: us-barat-1: :perangkat/qpu/rigetti/aspen-m-3	us-west-1
AWS	braket_sv	berbasis gerbang	Simulator lokal	N/A (simulator lokal di SDK Braket)	N/A
AWS	braket_dm	berbasis gerbang	Simulator lokal	N/A (simulator lokal di SDK Braket)	N/A

Penyedia	Nama perangkat	Paradigma	Tipe	Perangkat ARN	Wilayah
AWS	SV1	berbasis gerbang	Simulator sesuai permintaan	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Semua Wilayah di mana Amazon Braket tersedia.
AWS	DM1	berbasis gerbang	Simulator sesuai permintaan	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Semua Wilayah di mana Amazon Braket tersedia.
AWS	TN1	berbasis gerbang	Simulator sesuai permintaan	arn:aws:braket:::device/quantum-simulator/amazon/tn1	us-west-2, us-east-1, dan eu-west-2

> **Note**
>
> QPU tertentu hanya dapat diakses menggunakan reservasi melalui Braket Direct, lihat Reservasi.

Untuk melihat detail tambahan tentang QPU yang dapat Anda gunakan dengan Amazon Braket, lihat Penyedia Perangkat Keras Amazon Braket.

IonQ

IonQmenawarkan QPU berbasis gerbang berdasarkan teknologi perangkap ion. IonQ'sQPU ion terperangkap dibangun di atas rantai ion 171Yb+yang terperangkap yang dibatasi secara spasial melalui perangkap elektroda permukaan mikrofabrikasi di dalam ruang vakum.

IonQperangkat mendukung gerbang kuantum berikut.

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

Dengan kompilasi kata demi kata, IonQ QPU mendukung gerbang asli berikut.

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

Jika Anda hanya menentukan dua parameter fase saat menggunakan gerbang MS asli, gerbang MS yang sepenuhnya terjerat berjalan. Gerbang MS yang terjerat sepenuhnya selalu melakukan rotasi π/2. Untuk menentukan sudut yang berbeda dan menjalankan gerbang MS yang terjerat sebagian, Anda menentukan sudut yang diinginkan dengan menambahkan parameter ketiga. Untuk informasi selengkapnya, lihat modul braket.circuits.gate.

Gerbang asli ini hanya dapat digunakan dengan kompilasi kata demi kata. Untuk mempelajari lebih lanjut tentang kompilasi kata demi kata, lihat Kompilasi Verbatim.

IQM

IQMprosesor kuantum adalah perangkat universal dan model gerbang berdasarkan qubit transmon superkonduktor. IQM GarnetPerangkat ini adalah perangkat 20-qubit dengan topologi kisi persegi.

IQMPerangkat mendukung gerbang kuantum berikut.

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

Dengan kompilasi kata demi kata, IQM perangkat mendukung gerbang asli berikut.

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

Rigetti

Rigettiprosesor kuantum adalah mesin model gerbang universal yang didasarkan pada superkonduktor yang dapat disetel semua. qubits Aspen-M-3Perangkat 79-qubit memanfaatkan teknologi multi-chip milik mereka dan dirakit dari 2 prosesor 40-qubit.

RigettiPerangkat ini mendukung gerbang kuantum berikut.

```
'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'
```

Dengan kompilasi kata demi kata, perangkat Rigetti mendukung gerbang asli berikut.

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

Rigettiprosesor kuantum superkonduktor dapat menjalankan gerbang 'rx' hanya dengan sudut ±π/2 atau ±π.

Kontrol tingkat pulsa tersedia di Rigetti perangkat, yang mendukung serangkaian bingkai yang telah ditentukan dari jenis berikut:

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

Untuk informasi selengkapnya tentang frame ini, lihat Peran frame dan port.

Oxford Quantum Circuits (OQC)

OQCprosesor kuantum adalah mesin model gerbang universal, dibangun menggunakan teknologi Coaxmon yang dapat diskalakan. OQC LucySistem ini adalah 8-qubit perangkat dengan topologi cincin di mana masing-masing qubit terhubung ke dua tetangga terdekatnya.

LucyPerangkat ini mendukung gerbang kuantum berikut.

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

Dengan kompilasi kata demi kata, perangkat OQC mendukung gerbang asli berikut.

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

Kontrol tingkat pulsa tersedia di perangkat. OQC OQCPerangkat mendukung satu set frame yang telah ditentukan dari jenis berikut:

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

OQCperangkat mendukung deklarasi dinamis frame asalkan Anda menyediakan pengenal port yang valid. Untuk informasi selengkapnya tentang frame dan port ini, lihat Peran frame dan port.

> **Note**
>
> Saat menggunakan kontrol pulsa denganOQC, panjang program Anda tidak dapat melebihi maksimum 90 mikrodetik. Durasi maksimum adalah sekitar 50 nanodetik untuk gerbang qubit tunggal dan 1 mikrodetik untuk gerbang dua qubit. Angka-angka ini dapat bervariasi tergantung pada qubit yang digunakan, kalibrasi perangkat saat ini, dan kompilasi sirkuit.

QuEra

QuEramenawarkan perangkat berbasis atom netral yang dapat menjalankan tugas kuantum Analog Hamiltonian Simulation (AHS). Perangkat tujuan khusus ini dengan setia mereproduksi dinamika kuantum yang bergantung pada waktu dari ratusan qubit yang berinteraksi secara bersamaan.

Seseorang dapat memprogram perangkat ini dalam paradigma Simulasi Hamiltonian Analog dengan menentukan tata letak register qubit dan ketergantungan temporal dan spasial dari bidang manipulasi. Amazon Braket menyediakan utilitas untuk membangun program tersebut melalui modul AHS dari python SDK,. `braket.ahs`

Untuk informasi lebih lanjut, lihat contoh buku catatan Simulasi Hamiltonian Analog atau halaman Kirim program analog menggunakan halaman Aquila. QuEra

Simulator vektor negara bagian lokal (`braket_sv`)

Simulator vektor status lokal (`braket_sv`) adalah bagian dari Amazon Braket SDK yang berjalan secara lokal di lingkungan Anda. Ini sangat cocok untuk pembuatan prototipe cepat pada sirkuit kecil (hingga 25qubits) tergantung pada spesifikasi perangkat keras instance notebook Braket Anda atau lingkungan lokal Anda.

Simulator lokal mendukung semua gerbang di Amazon Braket SDK, tetapi perangkat QPU mendukung subset yang lebih kecil. Anda dapat menemukan gerbang perangkat yang didukung di properti perangkat.

> **Note**
>
> Simulator lokal mendukung fitur OpenQASM canggih yang mungkin tidak didukung pada perangkat QPU atau simulator lainnya. Untuk informasi selengkapnya tentang fitur yang didukung, lihat contoh yang disediakan di notebook Simulator Lokal OpenQASM.

Untuk informasi lebih lanjut tentang cara bekerja dengan simulator, lihat Contoh Amazon Braket.

Simulator matriks kepadatan lokal (**`braket_dm`**)

Simulator matriks kepadatan lokal (`braket_dm`) adalah bagian dari Amazon Braket SDK yang berjalan secara lokal di lingkungan Anda. Ini sangat cocok untuk pembuatan prototipe cepat pada sirkuit kecil dengan kebisingan (hingga 12qubits) tergantung pada spesifikasi perangkat keras instance notebook Braket Anda atau lingkungan lokal Anda.

Anda dapat membangun sirkuit bising umum dari bawah ke atas menggunakan operasi gerbang kebisingan seperti bit-flip dan kesalahan depolarisasi. Anda juga dapat menerapkan operasi kebisingan ke spesifik qubits dan gerbang sirkuit yang ada yang dimaksudkan untuk berjalan baik dengan maupun tanpa kebisingan.

Simulator `braket_dm` lokal dapat memberikan hasil berikut, mengingat jumlah yang ditentukanshots:

- Mengurangi matriks densitas: Shots = 0

> **Note**
>
> Simulator lokal mendukung fitur OpenQASM canggih, yang mungkin tidak didukung pada perangkat QPU atau simulator lainnya. Untuk informasi selengkapnya tentang fitur yang didukung, lihat contoh yang disediakan di notebook Simulator Lokal OpenQASM.

Untuk mempelajari lebih lanjut tentang simulator matriks densitas lokal, lihat contoh simulator bunyi pengantar Braket.

Simulator AHS lokal () **`braket_ahs`**

Simulator AHS (Analog Hamiltonian Simulation) lokal (`braket_ahs`) adalah bagian dari SDK Amazon Braket yang berjalan secara lokal di lingkungan Anda. Ini dapat digunakan untuk

mensimulasikan hasil dari program AHS. Ini sangat cocok untuk membuat prototipe pada register kecil (hingga 10-12 atom) tergantung pada spesifikasi perangkat keras instance notebook Braket Anda atau lingkungan lokal Anda.

Simulator lokal mendukung program AHS dengan satu bidang mengemudi yang seragam, satu bidang pergeseran (tidak seragam), dan pengaturan atom sewenang-wenang. Untuk detailnya, silakan merujuk ke kelas Braket AHS dan skema program Braket AHS.

Untuk mempelajari lebih lanjut tentang simulator AHS lokal, lihat Hello AHS: Jalankan halaman Simulasi Hamiltonian Analog pertama Anda dan contoh notebook Simulasi Hamiltonian Analog.

Simulator vektor negara (SV1)

SV1adalah simulator vektor keadaan universal sesuai permintaan, berkinerja tinggi. Itu dapat mensimulasikan sirkuit hingga 34qubits. Anda dapat mengharapkan rangkaian34-qubit, padat, dan persegi (kedalaman sirkuit = 34) membutuhkan waktu sekitar 1-2 jam untuk menyelesaikannya, tergantung pada jenis gerbang yang digunakan dan faktor lainnya. Sirkuit dengan all-to-all gerbang sangat cocok untukSV1. Ini mengembalikan hasil dalam bentuk seperti vektor keadaan penuh atau array amplitudo.

SV1memiliki runtime maksimum 6 jam. Ini memiliki default 35 tugas kuantum bersamaan, dan maksimum 100 (50 di us-west-1 dan eu-west-2) tugas kuantum bersamaan.

SV1hasil

SV1dapat memberikan hasil sebagai berikut, mengingat jumlah yang ditentukanshots:

- Contoh: Shots > 0
- Harapan: Shots >= 0
- Varians: Shots >= 0
- Probabilitas: Shots > 0
- Amplitudo: Shots = 0
- Gradien Bersebelahan: = 0 Shots

Untuk hasil lebih lanjut, lihat Jenis hasil.

SV1selalu tersedia, ia menjalankan sirkuit Anda sesuai permintaan, dan dapat menjalankan beberapa sirkuit secara paralel. Skala runtime secara linier dengan jumlah operasi dan secara

eksponensial dengan jumlah. qubits Jumlah shots memiliki dampak kecil pada runtime. Untuk mempelajari lebih lanjut, kunjungi Membandingkan simulator.

Simulator mendukung semua gerbang di SDK Braket, namun perangkat QPU mendukung subset yang lebih kecil. Anda dapat menemukan gerbang perangkat yang didukung di properti perangkat.

Simulator matriks kepadatan (DM1)

DM1adalah simulator matriks kepadatan sesuai permintaan, kinerja tinggi. Itu dapat mensimulasikan sirkuit hingga 17qubits.

DM1memiliki runtime maksimum 6 jam, default 35 tugas kuantum bersamaan, dan maksimum 50 tugas kuantum bersamaan.

DM1hasil

DM1dapat memberikan hasil sebagai berikut, mengingat jumlah yang ditentukanshots:

- Contoh: Shots > 0
- Harapan: Shots >= 0
- Varians: Shots >= 0
- Probabilitas: Shots > 0
- Matriks kepadatan berkurang: Shots = 0, hingga maks 8 qubits

Untuk informasi selengkapnya tentang hasil, lihat Jenis hasil.

DM1selalu tersedia, ia menjalankan sirkuit Anda sesuai permintaan, dan dapat menjalankan beberapa sirkuit secara paralel. Skala runtime secara linier dengan jumlah operasi dan secara eksponensial dengan jumlah. qubits Jumlah shots memiliki dampak kecil pada runtime. Untuk mempelajari lebih lanjut, lihat Bandingkan simulator.

Gerbang kebisingan dan keterbatasan

```
AmplitudeDamping
    Probability has to be within [0,1]
BitFlip
    Probability has to be within [0,0.5]
Depolarizing
```

```
    Probability has to be within [0,0.75]
GeneralizedAmplitudeDamping
    Probability has to be within [0,1]
PauliChannel
    The sum of the probabilities has to be within [0,1]
Kraus
    At most 2 qubits
    At most 4 (16) Kraus matrices for 1 (2) qubit
PhaseDamping
    Probability has to be within [0,1]
PhaseFlip
    Probability has to be within [0,0.5]
TwoQubitDephasing
    Probability has to be within [0,0.75]
TwoQubitDepolarizing
    Probability has to be within [0,0.9375]
```

Simulator jaringan tensor () TN1

TN1adalah simulator jaringan tensor sesuai permintaan, kinerja tinggi. TN1dapat mensimulasikan jenis sirkuit tertentu hingga 50 qubits dan kedalaman sirkuit 1.000 atau lebih kecil. TN1sangat kuat untuk sirkuit jarang, sirkuit dengan gerbang lokal, dan sirkuit lain dengan struktur khusus, seperti sirkuit transformasi Fourier kuantum (QFT). TN1beroperasi dalam dua tahap. Pertama, fase latihan mencoba mengidentifikasi jalur komputasi yang efisien untuk sirkuit Anda, sehingga TN1 dapat memperkirakan runtime tahap berikutnya, yang disebut fase kontraksi. Jika perkiraan waktu kontraksi melebihi batas runtime TN1 simulasi, TN1 tidak mencoba kontraksi.

TN1memiliki batas runtime 6 jam. Ini terbatas pada maksimum 10 (5 di eu-west-2) tugas kuantum bersamaan.

TN1hasil

Fase kontraksi terdiri atas serangkaian perkalian matriks. Rangkaian perkalian berlanjut sampai hasil tercapai atau sampai ditentukan bahwa suatu hasil tidak dapat dicapai.

Catatan: Shots harus > 0.

Jenis hasil meliputi:

- Sampel
- Perkiraan

- Varians

Untuk hasil lebih lanjut, lihat Jenis hasil.

TN1selalu tersedia, ia menjalankan sirkuit Anda sesuai permintaan, dan dapat menjalankan beberapa sirkuit secara paralel. Untuk mempelajari lebih lanjut, lihat Bandingkan simulator.

Simulator mendukung semua gerbang di SDK Braket, namun perangkat QPU mendukung subset yang lebih kecil. Anda dapat menemukan gerbang perangkat yang didukung di properti perangkat.

Kunjungi GitHub repositori Amazon Braket untuk notebook contoh TN1 untuk membantu Anda memulai. TN1

Praktik terbaik untuk bekerja dengan TN1

- Hindari all-to-all sirkuit.
- Uji sirkuit atau kelas sirkuit baru dengan sejumlah kecilshots, untuk mempelajari “kekerasan” sirkuit untukTN1.
- Pisahkan shot simulasi besar pada beberapa tugas kuantum.

Simulator tertanam

Simulator tertanam bekerja dengan memiliki simulasi tertanam dengan kode algoritma dalam wadah yang sama dan mengeksekusi simulasi pada instance pekerjaan hybrid secara langsung. Ini dapat berguna untuk menghilangkan kemacetan yang terkait dengan komunikasi simulasi dengan perangkat jarak jauh. Hal ini dapat mengakibatkan penggunaan memori yang jauh lebih rendah, mengurangi jumlah eksekusi sirkuit untuk mencapai hasil yang diinginkan, dan peningkatan kinerja sepuluh kali atau lebih. Untuk informasi selengkapnya tentang simulator tertanam, lihat halaman Jalankan pekerjaan hybrid dengan Amazon Braket Hybrid Jobs.

PennyLanesimulator petir

Anda dapat menggunakan PennyLane simulator petir sebagai simulator tertanam di Braket. Dengan PennyLane simulator petir, Anda dapat memanfaatkan metode komputasi gradien lanjutan, seperti diferensiasi adjoint, untuk mengevaluasi gradien lebih cepat. Simulator lightning.qubit tersedia sebagai perangkat melalui Braket NBI dan sebagai simulator tertanam, sedangkan simulator lightning.gpu perlu dijalankan sebagai simulator tertanam dengan instance GPU. Lihat simulator Tertanam di notebook Braket Hybrid Jobs untuk contoh penggunaan lightning.gpu.

Membandingkan simulator

Bagian ini membantu Anda memilih simulator Amazon Braket yang paling cocok untuk tugas kuantum Anda, dengan menjelaskan beberapa konsep, batasan, dan kasus penggunaan.

Memilih antara simulator lokal dan simulator sesuai permintaan (SV1,,) TN1 DM1

Kinerja simulator lokal tergantung pada perangkat keras yang menghosting lingkungan lokal, seperti instance notebook Braket, yang digunakan untuk menjalankan simulator Anda. Simulator sesuai permintaan berjalan di AWS cloud dan dirancang untuk skala di luar lingkungan lokal yang khas. Simulator sesuai permintaan dioptimalkan untuk sirkuit yang lebih besar, tetapi menambahkan beberapa overhead latensi per tugas kuantum atau kumpulan tugas kuantum. Ini dapat menyiratkan trade-off jika banyak tugas kuantum terlibat. Dengan karakteristik kinerja umum ini, panduan berikut dapat membantu Anda memilih cara menjalankan simulasi, termasuk yang berisik.

Untuk simulasi:

- Saat menggunakan kurang dari 18qubits, gunakan simulator lokal.
- Saat mempekerjakan 18-24qubits, pilih simulator berdasarkan beban kerja.
- Saat menggunakan lebih dari 24qubits, gunakan simulator sesuai permintaan.

Untuk simulasi kebisingan:

- Saat menggunakan kurang dari 9qubits, gunakan simulator lokal.
- Saat menggunakan 9-12qubits, pilih simulator berdasarkan beban kerja.
- Saat mempekerjakan lebih dari 12qubits, gunakanDM1.

Apa itu simulator vektor negara?

SV1adalah simulator vektor negara universal. Ini menyimpan fungsi gelombang penuh dari keadaan kuantum dan secara berurutan menerapkan operasi gerbang ke keadaan. Ini menyimpan semua kemungkinan, bahkan yang sangat tidak mungkin. Waktu berjalan SV1 simulator untuk tugas kuantum meningkat secara linier dengan jumlah gerbang di sirkuit.

Apa itu simulator matriks kepadatan?

DM1mensimulasikan sirkuit kuantum dengan noise. Ini menyimpan matriks kepadatan penuh dari sistem dan secara berurutan menerapkan gerbang dan operasi kebisingan sirkuit. Matriks kerapatan akhir berisi informasi lengkap tentang keadaan kuantum setelah rangkaian berjalan. Runtime

umumnya menskalakan secara linier dengan jumlah operasi dan secara eksponensial dengan jumlah. qubits

Apa itu simulator jaringan tensor?

TN1mengkodekan sirkuit kuantum menjadi grafik terstruktur.

- Simpul grafik terdiri dari gerbang kuantum, atauqubits.
- Tepi grafik menunjukkan koneksi antargerbang.

Sebagai hasil dari struktur ini, TN1 dapat menemukan solusi simulasi untuk sirkuit kuantum yang relatif besar dan kompleks.

TN1membutuhkan dua fase

Biasanya, TN1 beroperasi dalam pendekatan dua fase untuk mensimulasikan komputasi kuantum.

- Fase latihan: Pada fase ini, TN1 muncul cara untuk melintasi grafik dengan cara yang efisien, yang melibatkan mengunjungi setiap node sehingga Anda dapat memperoleh pengukuran yang Anda inginkan. Sebagai pelanggan, Anda tidak melihat fase ini karena TN1 melakukan kedua fase bersama untuk Anda. Ini menyelesaikan fase pertama dan menentukan apakah akan melakukan fase kedua sendiri berdasarkan kendala praktis. Anda tidak memiliki masukan ke dalam keputusan itu setelah simulasi dimulai.
- Fase kontraksi: Fase ini analog dengan fase eksekusi komputasi dalam komputer klasik. Fase kontraksi terdiri atas serangkaian perkalian matriks. Urutan perkalian ini memiliki efek yang besar pada kesulitan komputasi. Oleh karena itu, fase latihan dilakukan terlebih dahulu untuk menemukan jalur komputasi yang paling efektif di seluruh grafik. Setelah menemukan jalur kontraksi selama fase latihan, TN1 kontraksikan gerbang sirkuit Anda untuk menghasilkan hasil simulasi.

TN1grafik analog dengan peta

Secara metaforis, Anda dapat membandingkan TN1 grafik yang mendasarinya dengan jalan-jalan kota. Di kota dengan grid yang direncanakan, mudah untuk menemukan rute ke tujuan Anda menggunakan peta. Di kota dengan jalan-jalan yang tidak direncanakan, nama jalan duplikat, dan sebagainya, sulit untuk menemukan rute ke tujuan Anda dengan melihat peta.

Jika TN1 tidak melakukan fase latihan, itu akan seperti berjalan di sekitar jalan-jalan kota untuk menemukan tujuan Anda, daripada melihat peta terlebih dahulu. Ini benar-benar lebih menghemat

waktu dari segi waktu berjalan dibandingkan waktu yang dihabiskan untuk melihat peta. Demikian pula, fase latihan memberikan informasi berharga.

Anda mungkin mengatakan bahwa TN1 memiliki “kesadaran” tertentu tentang struktur sirkuit yang mendasarinya yang dilaluinya. Ini mendapatkan kesadaran ini selama fase latihan.

Jenis masalah yang paling cocok untuk masing-masing jenis simulator

SV1sangat cocok untuk setiap kelas masalah yang bergantung terutama pada memiliki sejumlah qubits dan gerbang tertentu. Umumnya, waktu yang dibutuhkan tumbuh secara linier dengan jumlah gerbang, sementara itu tidak tergantung pada jumlah. shots SV1umumnya lebih cepat daripada TN1 sirkuit di bawah 28qubits.

SV1bisa lebih lambat untuk qubit angka yang lebih tinggi karena sebenarnya mensimulasikan semua kemungkinan, bahkan yang sangat tidak mungkin. Ini tidak memiliki cara untuk menentukan kemungkinan hasil. Jadi, untuk 30-qubit evaluasi, SV1 harus menghitung 2^30 konfigurasi. Batas 34 qubits untuk SV1 simulator Amazon Braket adalah kendala praktis karena keterbatasan memori dan penyimpanan. Anda dapat memikirkannya seperti ini: Setiap kali Anda qubit menambahkanSV1, masalahnya menjadi dua kali lebih sulit.

Untuk banyak kelas masalah, TN1 dapat mengevaluasi sirkuit yang jauh lebih besar dalam waktu realistis daripada SV1 karena TN1 mengambil keuntungan dari struktur grafik. Ini pada dasarnya melacak evolusi solusi dari tempat awalnya dan hanya mempertahankan konfigurasi yang berkontribusi pada traversal yang efisien. Dengan kata lain, ini menyimpan konfigurasi untuk membuat urutan perkalian matriks yang menghasilkan proses evaluasi yang lebih sederhana.

SebabTN1, jumlah qubits dan gerbang penting, tetapi struktur grafik lebih penting. Misalnya, TN1 sangat baik dalam mengevaluasi sirkuit (grafik) di mana gerbang jarak pendek (yaitu, masing-masing qubit dihubungkan oleh gerbang hanya ke tetangga terdekatnyaqubits), dan sirkuit (grafik) di mana koneksi (atau gerbang) memiliki jangkauan yang sama. Rentang khas untuk TN1 adalah memiliki masing-masing qubit berbicara hanya dengan qubits yang lain yang qubits berjarak 5. Jika sebagian besar struktur dapat didekomposisi menjadi hubungan yang lebih sederhana seperti ini, yang dapat direpresentasikan dalam matriks yang lebih banyak, lebih kecil, atau lebih seragam, TN1 lakukan evaluasi dengan mudah.

Keterbatasan TN1

TN1bisa lebih lambat daripada SV1 tergantung pada kompleksitas struktural grafik. Untuk grafik tertentu, TN1 akhiri simulasi setelah tahap latihan, dan menunjukkan status, karena salah satu dari dua `FAILED` alasan ini:

- Tidak dapat menemukan jalur — Jika grafiknya terlalu rumit, terlalu sulit untuk menemukan jalur traversal yang baik dan simulator menyerah pada komputasi. TN1tidak dapat melakukan kontraksi. Anda mungkin melihat pesan kesalahan yang mirip dengan pesan ini: `No viable contraction path found.`
- Tahap kontraksi terlalu sulit — Dalam beberapa grafik, TN1 dapat menemukan jalur traversal, tetapi sangat panjang dan sangat memakan waktu untuk mengevaluasi. Dalam hal ini, kontraksi sangat mahal sehingga biayanya menjadi penghalang dan sebaliknya, TN1 keluar setelah fase latihan. Anda mungkin melihat pesan kesalahan yang mirip dengan pesan ini: `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> Anda ditagih untuk tahap latihan TN1 bahkan jika kontraksi tidak dilakukan dan Anda melihat status. `FAILED`

Runtime yang diprediksi juga tergantung pada shot hitungan. Dalam skenario terburuk, waktu TN1 kontraksi tergantung secara linier pada hitungan. shot Sirkuit mungkin dapat dikontrak dengan lebih sedikitshots. Misalnya, Anda mungkin mengirimkan tugas kuantum dengan 100shots, yang TN1 memutuskan tidak dapat dikontrak, tetapi jika Anda mengirim ulang hanya dengan 10, kontraksi berlanjut. Dalam situasi ini, untuk mencapai 100 sampel, Anda dapat mengirimkan 10 tugas kuantum 10 shots untuk sirkuit yang sama dan menggabungkan hasilnya pada akhirnya.

Sebagai praktik terbaik, kami menyarankan Anda untuk selalu menguji sirkuit atau kelas sirkuit Anda dengan beberapa shots (misalnya, 10) untuk mengetahui seberapa keras sirkuit AndaTN1, sebelum Anda melanjutkan dengan jumlah yang lebih tinggishots.

> **Note**
>
> Rangkaian perkalian yang membentuk fase kontraksi dimulai dengan matriks kecil NxN. Misalnya, 2-qubit gerbang membutuhkan matriks 4x4. Matriks menengah yang diperlukan selama kontraksi yang dinilai terlalu sulit adalah yang berukuran raksasa. Komputasi tersebut akan membutuhkan waktu berhari-hari untuk menyelesaikan. Itu sebabnya Amazon Braket tidak mencoba kontraksi yang sangat kompleks.

Konkurensi

Semua simulator Braket memberi Anda kemampuan untuk menjalankan beberapa sirkuit secara bersamaan. Batas konkurensi bervariasi menurut simulator dan wilayah. Untuk informasi selengkapnya tentang batas konkurensi, lihat halaman Kuota.

Wilayah dan titik akhir Amazon Braket

Amazon Braket tersedia sebagai berikut: Wilayah AWS

Ketersediaan wilayah Amazon Braket

Nama Wilayah	Wilayah	Titik Akhir Braket	QPU
US East (N. Virginia)	as-east-1	braket.us-east-1.amazonaws.com	IonQ
US East (Northern Virginia)	as-east-1	braket.us-east-1.amazonaws.com	QuEra
US West (Northern California)	as-west-1	braket.us-west-1.amazonaws.com	Rigetti
Uni Eropa Utara 1 (Stockholm)	eu-north-1	braket.eu-north-1.amazonaws.com	IQM
Uni Eropa Barat 2 (London)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Anda dapat menjalankan Amazon Braket dari setiap Wilayah yang menyediakan, tetapi setiap QPU hanya tersedia satu Wilayah. Tugas kuantum yang berjalan pada perangkat QPU dapat dilihat di konsol Amazon Braket di Wilayah perangkat itu. Jika Anda menggunakan Amazon Braket SDK, Anda dapat mengirimkan tugas kuantum ke perangkat QPU apa pun, terlepas dari Wilayah tempat Anda bekerja. SDK secara otomatis membuat sesi ke Wilayah untuk QPU yang ditentukan.

Untuk informasi umum tentang cara AWS kerja dengan Wilayah dan titik akhir, lihat Layanan AWS titik akhir di Referensi AWS Umum.

Kapan tugas kuantum saya akan berjalan?

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Ketika Anda mengirimkan sebuah sirkuit, Amazon Braket mengirimkannya ke perangkat yang Anda tentukan. Quantum Processing Unit (QPU) dan tugas kuantum simulator sesuai permintaan diantrian dan diproses sesuai urutan penerimaannya. Waktu yang diperlukan untuk memproses tugas kuantum Anda setelah Anda mengirimkannya bervariasi tergantung pada jumlah dan kompleksitas tugas yang diajukan oleh pelanggan Amazon Braket lainnya dan ketersediaan QPU yang dipilih.

Notifikasi perubahan status di email atau SMS

Amazon Braket mengirimkan peristiwa ke Amazon EventBridge saat ketersediaan QPU berubah atau saat status tugas kuantum Anda berubah. Ikuti langkah-langkah ini untuk menerima pemberitahuan perubahan status tugas perangkat dan kuantum melalui email atau pesan SMS:

1. Buat topik Amazon SNS dan berlangganan ke email atau SMS. Ketersediaan email atau SMS tergantung pada Wilayah Anda. Untuk informasi selengkapnya, lihat Memulai Amazon SNS dan Mengirim pesan SMS.
2. Buat aturan EventBridge yang memicu notifikasi ke topik SNS Anda. Untuk informasi selengkapnya, lihat Memantau Amazon Braket dengan Amazon. EventBridge

Peringatan penyelesaian tugas kuantum

Anda dapat mengatur notifikasi melalui Amazon Simple Notification Service (SNS) sehingga Anda menerima peringatan saat tugas kuantum Amazon Braket Anda selesai. Pemberitahuan aktif berguna jika Anda mengharapkan waktu tunggu yang lama — misalnya, saat Anda mengirimkan tugas besar atau saat Anda mengirimkan tugas di luar jendela ketersediaan perangkat. Jika Anda tidak ingin menunggu tugas selesai, Anda dapat mengatur notifikasi SNS.

Notebook Amazon Braket memandu Anda melalui langkah-langkah pengaturan. Untuk informasi lebih lanjut, lihat Notebook contoh Amazon Braket untuk mengatur notifikasi.

Windows dan status ketersediaan QPU

Ketersediaan QPU bervariasi dari perangkat ke perangkat.

Di halaman Perangkat konsol Amazon Braket, Anda dapat melihat jendela ketersediaan saat ini dan yang akan datang serta status perangkat. Selain itu, setiap halaman perangkat menunjukkan kedalaman antrian individu untuk tugas kuantum dan pekerjaan hibrida.

Perangkat dianggap offline jika tidak tersedia untuk pelanggan, terlepas dari window ketersediaan. Misalnya, ini mungkin offline karena masalah pemeliharaan terjadwal, upgrade, atau operasional.

Visibilitas antrian

Sebelum mengirimkan tugas kuantum atau pekerjaan hibrida, Anda dapat melihat berapa banyak tugas kuantum atau pekerjaan hibrida di depan Anda dengan memeriksa kedalaman antrian perangkat.

Kedalaman antrian

Queue depthmengacu pada jumlah tugas kuantum dan pekerjaan hibrida yang diantrian untuk perangkat tertentu. Tugas kuantum perangkat dan jumlah antrian pekerjaan hibrida dapat diakses melalui Braket Software Development Kit (SDK) atauAmazon Braket Management Console.

1. Kedalaman antrian tugas mengacu pada jumlah total tugas kuantum yang saat ini menunggu untuk dijalankan dalam prioritas normal.
2. Kedalaman antrian tugas prioritas mengacu pada jumlah total tugas kuantum yang dikirimkan yang menunggu untuk dijalankan. Amazon Braket Hybrid Jobs Tugas-tugas ini berjalan sebelum tugas mandiri.
3. Kedalaman antrian pekerjaan hibrida mengacu pada jumlah total pekerjaan hibrida yang saat ini mengantri di perangkat. Quantum tasksdiajukan sebagai bagian dari pekerjaan hibrida memiliki prioritas, dan digabungkan dalam. Priority Task Queue

Pelanggan yang ingin melihat kedalaman antrian melalui Braket SDK dapat memodifikasi cuplikan kode berikut untuk mendapatkan posisi antrian tugas kuantum atau pekerjaan hibrida mereka:

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

# returns the number of quantum tasks queued on the device
print(device.queue_depth().quantum_tasks)
```

```
{<QueueType.NORMAL: 'Normal'>: '0', <QueueType.PRIORITY: 'Priority'>: '0'}


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

Mengirimkan tugas kuantum atau pekerjaan hibrida ke QPU dapat mengakibatkan beban kerja Anda berada dalam keadaan. QUEUED Amazon Braket memberikan visibilitas pelanggan ke tugas kuantum dan posisi antrian pekerjaan hibrida mereka.

Posisi antrian

Queue positionmengacu pada posisi tugas kuantum Anda saat ini atau pekerjaan hibrida dalam antrian perangkat masing-masing. Ini dapat diperoleh untuk tugas-tugas kuantum atau pekerjaan hibrida melalui Braket Software Development Kit (SDK) atauAmazon Braket Management Console.

Pelanggan yang ingin melihat posisi antrian melalui Braket SDK dapat memodifikasi cuplikan kode berikut untuk mendapatkan posisi antrian tugas kuantum atau pekerjaan hibrida mereka:

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

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

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

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


from braket.aws import AwsQuantumJob

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

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

Memulai dengan Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Setelah Anda mengikuti petunjuk di Aktifkan Amazon Braket, Anda dapat memulai dengan Amazon Braket.

Langkah-langkah untuk memulai meliputi:

- Aktifkan Amazon Braket
- Buat instans notebook Amazon Braket
- Jalankan sirkuit pertama Anda menggunakan SDK Python Amazon Braket
- Jalankan algoritma kuantum pertama Anda

Aktifkan Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Anda dapat mengaktifkan Amazon Braket di akun Anda melalui konsol.AWS

Prasyarat

Untuk mengaktifkan dan menjalankan Amazon Braket, Anda harus memiliki pengguna atau peran dengan izin untuk memulai tindakan Amazon Braket. Izin ini disertakan dalam kebijakan IAM (AmazonBraketFullAccess arn:aws:iam: :aws:policy/). AmazonBraket FullAccess

> **Note**
>
> Jika Anda seorang administrator:
> Untuk memberi pengguna lain akses ke Amazon Braket, berikan izin kepada pengguna dengan melampirkan AmazonBraketFullAccesskebijakan atau dengan melampirkan kebijakan khusus yang Anda buat. Untuk mempelajari lebih lanjut tentang izin yang diperlukan untuk menggunakan Amazon Braket, lihat Mengelola akses ke Amazon Braket.

Langkah-langkah untuk mengaktifkan Amazon Braket

1. Masuk ke konsol Amazon Braket dengan Anda. Akun AWS
2. Buka konsol Amazon Braket.
3. Dari halaman landing Braket, klik Memulai untuk dibawa ke halaman Dasbor Layanan. Peringatan di bagian atas dasbor layanan Anda akan memandu Anda melalui tiga langkah berikut:
   a. Membuat peran terkait layanan (SLR)
   b. Mengaktifkan akses ke komputer kuantum pihak ketiga
   c. Membuat instance notebook Jupyter baru

Untuk menggunakan perangkat kuantum pihak ketiga, Anda harus menyetujui kondisi tertentu terkait transfer data antara Anda AWS, dan perangkat tersebut. Syarat dan ketentuan perjanjian ini disediakan pada tab Umum pada halaman Izin dan pengaturan di konsol Amazon Braket.

> **Note**
>
> Perangkat kuantum yang tidak melibatkan pihak ketiga, seperti simulator lokal Braket atau simulator sesuai permintaan, dapat digunakan tanpa menyetujui perjanjian Aktifkan perangkat pihak ketiga.
> Menerima persyaratan ini untuk mengaktifkan penggunaan perangkat pihak ketiga hanya perlu dilakukan sekali per akun jika Anda mengakses perangkat keras pihak ketiga.

Buat instans notebook Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Amazon Braket menyediakan notebook Jupyter terkelola penuh agar Anda bisa mulai. Instans notebook Amazon Braket didasarkan pada instance notebook Amazon SageMaker . Petunjuk berikut menguraikan langkah-langkah yang diperlukan untuk membuat instance notebook baru untuk pelanggan baru dan yang sudah ada.

Pelanggan Amazon Braket baru

1. Buka konsol Amazon Braket dan arahkan ke halaman Dasbor di panel kiri.
2. Klik Memulai pada modal Selamat Datang di Amazon Braket, yang terletak di tengah halaman dasbor Anda, untuk memberikan nama buku catatan. Ini akan membuat notebook Jupyter default.
3. Diperlukan beberapa menit untuk membuat notebook Anda. Buku catatan Anda akan terdaftar di halaman Notebook dengan status Tertunda. Saat instance notebook Anda siap digunakan, statusnya berubah menjadi InService. Anda mungkin perlu me-refresh halaman untuk menampilkan status yang diperbarui untuk buku catatan.

Pelanggan Amazon Braket yang ada

1. Buka konsol Amazon Braket, pilih Notebook di panel kiri, pilih Buat instance notebook. Jika Anda memiliki nol buku catatan, pilih Pengaturan Standar untuk membuat buku catatan Jupyter default dan masukkan nama instance Notebook hanya menggunakan karakter alfanumerik dan tanda hubung dan pilih mode visual pilihan Anda. Kemudian, aktifkan atau nonaktifkan manajer ketidakaktifan untuk buku catatan Anda.
   a. Jika diaktifkan, pilih waktu durasi idle yang diinginkan sebelum notebook diatur ulang. Saat notebook disetel ulang, biaya komputasi akan berhenti timbul, tetapi biaya penyimpanan akan terus berlanjut.
   b. Untuk melihat sisa waktu idle di instance notebook Anda, navigasikan ke bilah perintah dan pilih tab Braket, diikuti oleh tab Inactivity Manager.

> **Note**
>
> Untuk menyimpan pekerjaan Anda agar tidak hilang, pertimbangkan untuk mengintegrasikan instance SageMaker notebook Anda dengan repositori git. Sebagai alternatif, memindahkan pekerjaan Anda ke luar `/Braket Examples` folder `/Braket Algorithms` dan akan mencegah file ditimpa oleh restart instance notebook.

2. (Opsional) Dengan Pengaturan lanjutan Anda dapat membuat buku catatan dengan izin akses, konfigurasi tambahan, dan pengaturan akses jaringan:
   a. Dalam konfigurasi Notebook pilih jenis instans Anda. Jenis instans standar dan hemat biaya, ml.t3.medium dipilih secara default. Untuk mempelajari lebih lanjut tentang harga instans, lihat SageMaker harga Amazon. Jika Anda ingin mengaitkan repositori Github publik dengan instance notebook Anda, klik dropdown repositori Git dan pilih Clone repositori git publik dari url dari menu tarik-turun Repositori. Masukkan URL repo di bilah teks URL repositori Git.
   b. Dalam Izin, konfigurasikan peran IAM opsional, akses root, dan kunci enkripsi apa pun.
   c. Di Jaringan, konfigurasikan pengaturan jaringan dan akses khusus untuk Jupyter Notebook instans Anda.
3. Tinjau pengaturan Anda, setel tag apa pun untuk mengidentifikasi instance buku catatan Anda, dan klik Luncurkan.

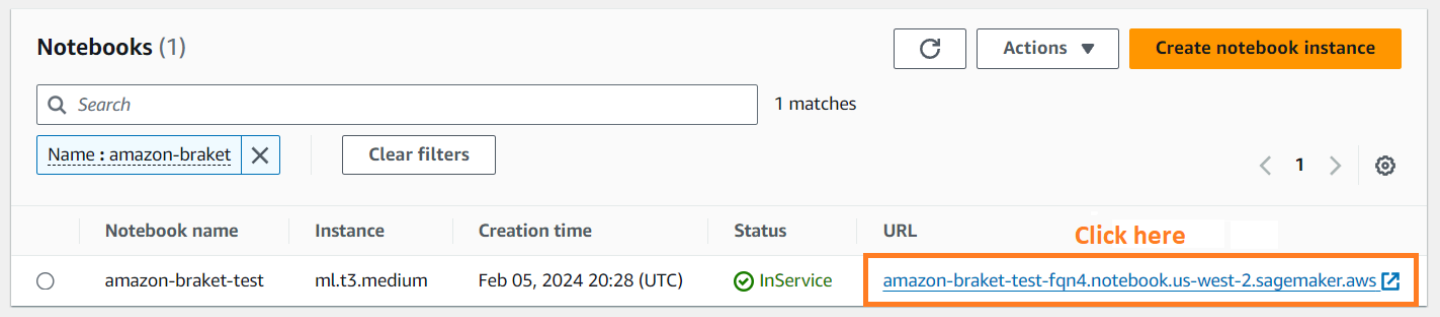
> **Note**
>
> Anda dapat melihat dan mengelola instans notebook Amazon Braket Anda di Amazon Braket dan konsol Amazon. SageMaker Pengaturan notebook Amazon Braket tambahan tersedia melalui konsol. SageMaker

Jika Anda bekerja di konsol Amazon Braket dalam AWS SDK Amazon Braket dan plugin dimuat sebelumnya di buku catatan yang Anda buat. Jika Anda ingin menjalankan pada mesin Anda sendiri, Anda dapat menginstal SDK dan plugin ketika Anda menjalankan perintah `pip install amazon-braket-sdk` atau ketika Anda menjalankan perintah `pip install amazon-braket-pennylane-plugin` untuk digunakan dengan PennyLane plugin.

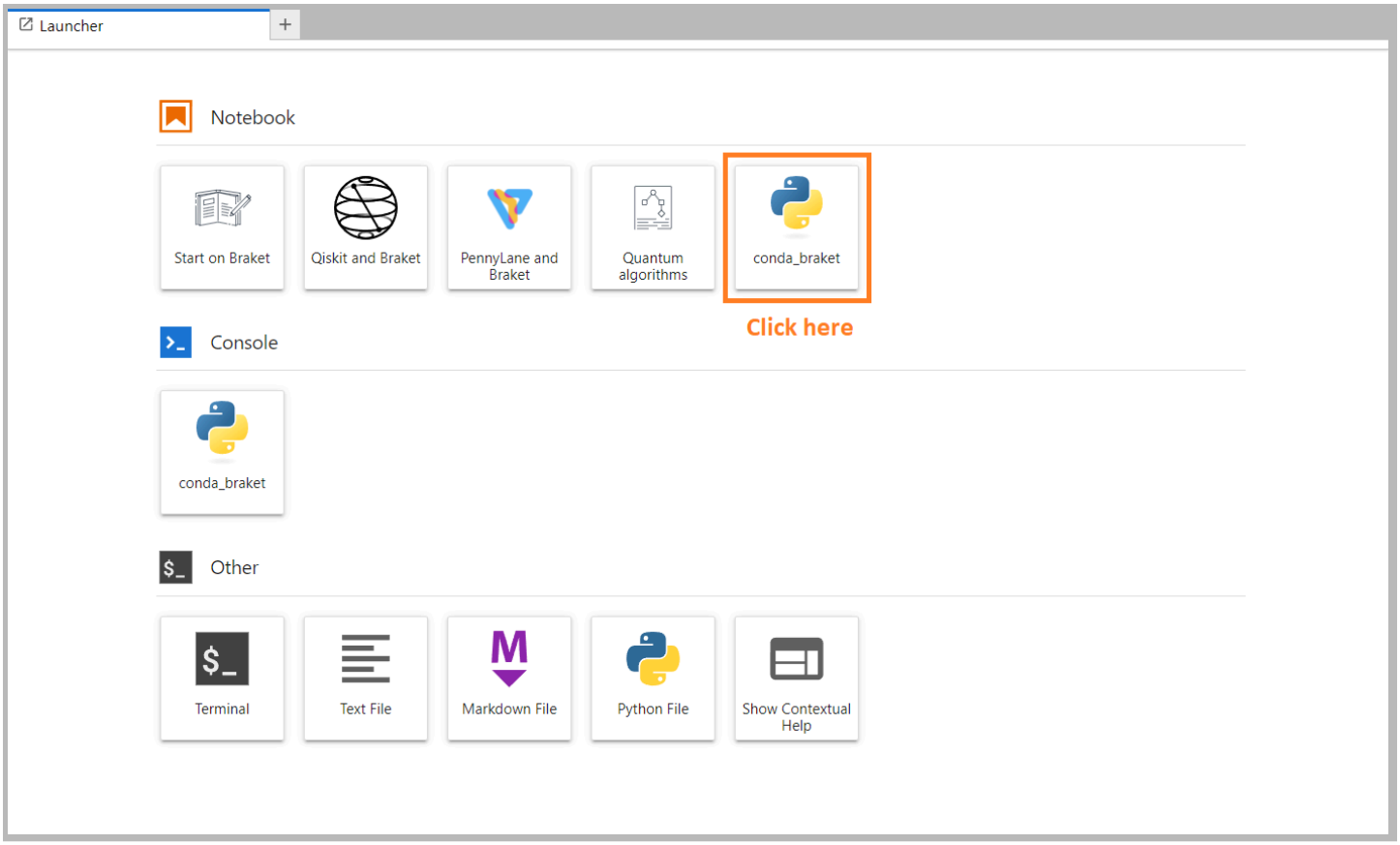
Jalankan sirkuit pertama Anda menggunakan SDK Python Amazon Braket

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Setelah instans notebook Anda diluncurkan, buka instans dengan antarmuka Jupyter standar dengan memilih notebook yang baru saja Anda buat.



AmazonInstans notebook Braket sudah diinstal sebelumnya dengan Amazon Braket SDK dan semua dependensinya. Mulailah dengan membuat notebook baru dengan Kernel `conda_braket`.



Anda bisa mulai dengan yang sederhana seperti “Halo, dunia!” contoh. Pertama, bangun sirkuit yang mempersiapkan keadaan Bell, dan kemudian jalankan sirkuit pada perangkat yang berbeda untuk mendapatkan hasil.

Mulailah dengan mengimpor modul Amazon Braket SDK dan mendefinisikan sirkuit Bell State sederhana.

```
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)
```

Anda dapat memvisualisasikan sirkuit dengan perintah ini:

```
print(bell)
```

Jalankan sirkuit Anda di simulator lokal

Selanjutnya, pilih perangkat kuantum untuk menjalankan sirkuit. AmazonBraket SDK hadir dengan simulator lokal untuk pembuatan prototipe dan pengujian cepat. Kami merekomendasikan menggunakan simulator lokal untuk sirkuit yang lebih kecil, yang bisa mencapai 25 qubits (tergantung pada perangkat keras lokal Anda).

Berikut cara untuk instantiate simulator lokal:

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

dan menjalankan sirkuit:

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

Anda seharusnya melihat hasil seperti ini:

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

Status Bell spesifik yang telah Anda siapkan adalah superposisi yang sama dari |00> dan |11>, dan Anda akan menemukan distribusi 00 dan 11 yang kira-kira sama (hingga shot noise) sebagai hasil pengukuran, seperti yang diharapkan.

Jalankan sirkuit Anda pada simulator sesuai permintaan

AmazonBraket juga menyediakan akses ke simulator berkinerja tinggi sesuai permintaanSV1, untuk menjalankan sirkuit yang lebih besar. SV1adalah simulator vektor negara sesuai permintaan yang memungkinkan simulasi sirkuit kuantum hingga 34. qubits Anda dapat menemukan informasi selengkapnya SV1 di bagian Perangkat yang Didukung dan di AWS konsol. Saat menjalankan tugas kuantum pada SV1 (dan di TN1 atau QPU apa pun), hasil tugas kuantum Anda disimpan dalam bucket S3 di akun Anda. Jika Anda tidak menentukan bucket, Braket SDK akan membuat bucket default `amazon-braket-{region}-{accountID}` untuk Anda. Untuk mempelajari lebih lanjut, lihat Mengelola akses ke Amazon Braket.

> **Note**
>
> Isi nama bucket sebenarnya yang ada di mana contoh berikut menunjukkan `example-bucket` sebagai nama bucket Anda. Nama bucket untuk Amazon Braket selalu dimulai dengan `amazon-braket-` diikuti oleh karakter pengenal lain yang Anda tambahkan. Jika Anda memerlukan informasi tentang cara menyiapkan bucket S3, lihat Memulai Amazon S3.

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

Untuk menjalankan sirkuitSV1, Anda harus memberikan lokasi bucket S3 yang sebelumnya Anda pilih sebagai argumen posisi dalam panggilan. `.run()`

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

# run the circuit
task = device.run(bell, s3_folder, shots=100)
# display the results
print(task.result().measurement_counts)
```

Konsol Amazon Braket memberikan informasi lebih lanjut tentang tugas kuantum Anda. Arahkan ke tab Quantum Tasks di konsol dan tugas kuantum Anda harus berada di bagian atas daftar. Atau, Anda dapat mencari tugas kuantum Anda menggunakan ID tugas kuantum unik atau kriteria lainnya.

> **Note**
>
> Setelah 90 hari, Amazon Braket secara otomatis menghapus semua ID tugas kuantum dan metadata lain yang terkait dengan tugas kuantum Anda. Untuk informasi lebih lanjut, lihat Retensi data.

Berjalan di QPU

Dengan Amazon Braket, Anda dapat menjalankan contoh sirkuit kuantum sebelumnya pada komputer kuantum fisik hanya dengan mengubah satu baris kode. Amazon Braket menyediakan akses ke QPU perangkat dariIonQ,, Oxford Quantum CircuitsQuEra, dan. Rigetti Anda dapat menemukan informasi tentang berbagai perangkat dan jendela ketersediaan di bagian Perangkat yang Didukung, dan di AWS konsol di bawah tab Perangkat. Contoh berikut menunjukkan cara membuat instance perangkat. Rigetti

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

Pilih IonQ perangkat dengan kode ini:

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

Setelah memilih perangkat dan sebelum menjalankan beban kerja, Anda dapat menanyakan kedalaman antrian perangkat dengan kode berikut untuk menentukan jumlah tugas kuantum atau pekerjaan hibrida. Selain itu, pelanggan dapat melihat kedalaman antrian khusus perangkat di halaman Perangkat. Amazon Braket Management Console

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

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

Saat Anda menjalankan tugas, SDK Amazon Braket melakukan polling untuk hasil (dengan batas waktu default 5 hari). Anda dapat mengubah default ini dengan memodifikasi `poll_timeout_seconds` parameter dalam `.run()` perintah seperti yang ditunjukkan pada contoh berikut. Perlu diingat bahwa jika batas waktu polling Anda terlalu pendek, hasil mungkin tidak dikembalikan dalam waktu polling, seperti ketika QPU tidak tersedia dan kesalahan batas waktu lokal dikembalikan. Anda dapat merestart pemungutan suara dengan memanggil fungsi `task.result()`.

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

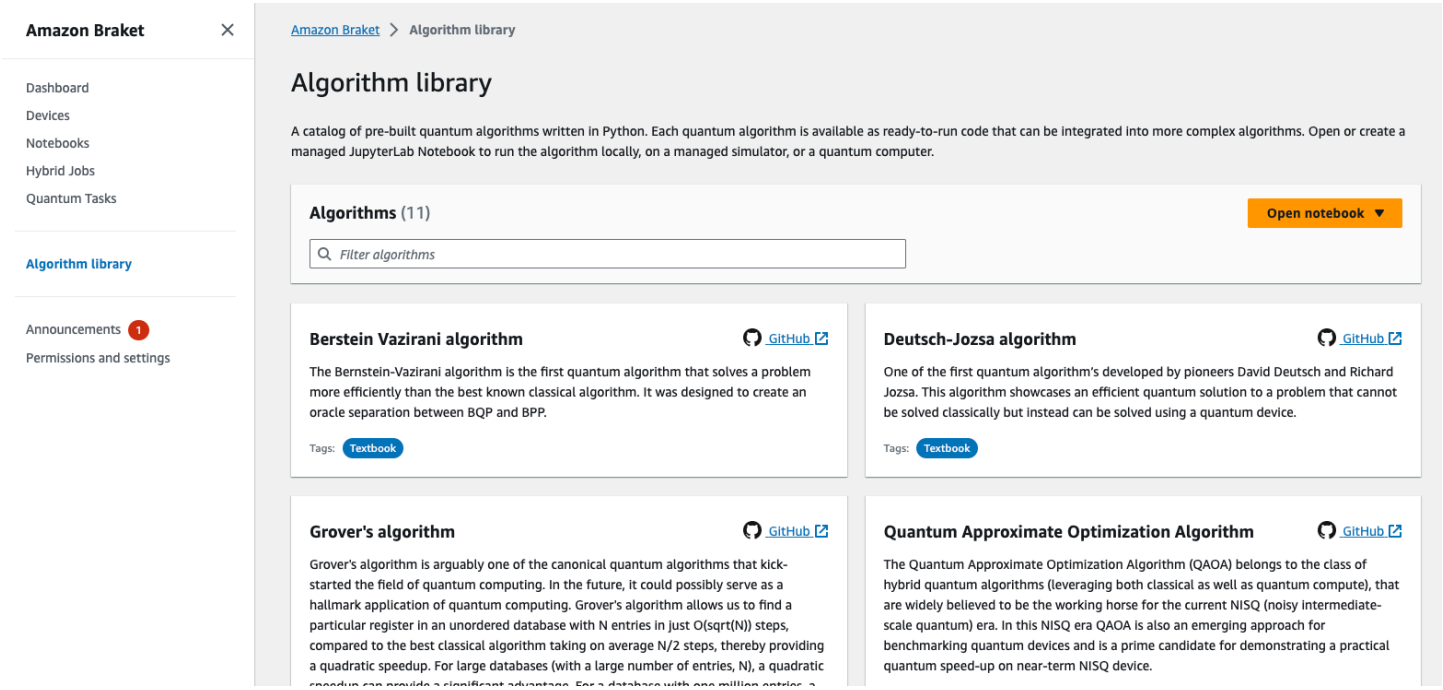
Selain itu, setelah mengirimkan tugas kuantum atau pekerjaan hibrida Anda, Anda dapat memanggil `queue_position()` fungsi untuk memeriksa posisi antrian Anda.

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

Jalankan algoritma kuantum pertama Anda

> **Tip**
>
> Pelajari dasar-dasar komputasi kuantum dengan AWS! Daftar di Amazon Braket Digital Learning Plan dan dapatkan lencana Digital Anda sendiri setelah menyelesaikan serangkaian kursus pembelajaran dan penilaian digital.

Pustaka algoritma Amazon Braket adalah katalog algoritma kuantum pra-bangun yang ditulis dengan Python. Anda dapat menjalankan algoritma ini sebagaimana adanya atau menggunakannya sebagai titik awal untuk membangun algoritma yang lebih kompleks. Anda dapat mengakses pustaka algoritma dari konsol Braket. Anda juga dapat mengakses pustaka algoritma Braket di Github: https://github.com/aws-samples/amazon-braket-algorithm-library.



Konsol Braket memberikan deskripsi dari setiap algoritma yang tersedia di perpustakaan algoritma. Pilih GitHub tautan untuk melihat detail setiap algoritme, atau pilih Buka buku catatan untuk membuka atau membuat buku catatan yang berisi semua algoritme yang tersedia. Jika Anda memilih opsi notebook, Anda kemudian dapat menemukan pustaka algoritma Braket di folder root notebook Anda.

Bekerja dengan Amazon Braket

Bagian ini menunjukkan kepada Anda cara merancang sirkuit kuantum, mengirimkan masalah ini sebagai tugas kuantum ke perangkat, dan memantau tugas kuantum dengan Amazon Braket SDK.

Berikut ini adalah sarana utama berinteraksi dengan sumber daya di Amazon Braket.

- Konsol Amazon Braket menyediakan informasi dan status perangkat untuk membantu Anda membuat, mengelola, dan memantau sumber daya dan tugas kuantum Anda.
- Kirim dan jalankan tugas kuantum melalui Amazon Braket Python SDK, serta melalui konsol. SDK dapat diakses melalui notebook Amazon Braket yang telah dikonfigurasi sebelumnya.
- Amazon Braket API dapat diakses melalui Amazon Braket Python SDK dan notebook. Anda dapat melakukan panggilan langsung ke API jika Anda sedang membangun aplikasi yang bekerja dengan komputasi kuantum secara terprogram.

Contoh di seluruh bagian ini menunjukkan bagaimana Anda dapat bekerja dengan Amazon Braket API secara langsung menggunakan SDK Python Amazon Braket bersama dengan SDK Python untuk Braket (AWS Boto3).

Lebih lanjut tentang SDK Python Amazon Braket

Untuk bekerja dengan SDK Python Amazon Braket, pertama instal SDK AWS Python untuk Braket (Boto3) sehingga Anda dapat berkomunikasi dengan. AWS API Anda dapat menganggap Amazon Braket Python SDK sebagai pembungkus yang nyaman di sekitar Boto3 untuk pelanggan kuantum.

- Boto3 berisi antarmuka yang perlu Anda manfaatkan. AWS API (Perhatikan bahwa Boto3 adalah SDK Python besar yang berbicara dengan. AWS API Sebagian besar Layanan AWS mendukung antarmuka Boto3.)
- SDK Python Amazon Braket berisi modul perangkat lunak untuk sirkuit, gerbang, perangkat, jenis hasil, dan bagian lain dari tugas kuantum. Setiap kali Anda membuat program, Anda mengimpor modul yang Anda butuhkan untuk tugas kuantum itu.
- SDK Python Amazon Braket dapat diakses melalui notebook, yang sudah dimuat sebelumnya dengan semua modul dan dependensi yang Anda butuhkan untuk menjalankan tugas kuantum.
- Anda dapat mengimpor modul dari SDK Python Amazon Braket ke skrip Python apa pun jika Anda tidak ingin bekerja dengan notebook.

Setelah Anda menginstal Boto3, ikhtisar langkah-langkah untuk membuat tugas kuantum melalui SDK Python Amazon Braket menyerupai yang berikut:

1. (Opsional) Buka buku catatan Anda.
2. Impor modul SDK yang Anda butuhkan untuk sirkuit Anda.
3. Tentukan QPU atau simulator.
4. Instantiasikan sirkuit.
5. Jalankan sirkuit.
6. Kumpulkan hasilnya.

Contoh di bagian ini menunjukkan detail setiap langkah.

Untuk contoh lainnya, lihat repositori Contoh Amazon Braket di. GitHub

Di bagian ini:

- Halo AHS: Jalankan Simulasi Hamiltonian Analog pertama Anda
- Bangun sirkuit di SDK
- Mengirimkan tugas kuantum ke QPU dan simulator
- Jalankan sirkuit Anda dengan OpenQASM 3.0
- Kirim program analog menggunakan QuEra Aquila
- Bekerja dengan Boto3

Halo AHS: Jalankan Simulasi Hamiltonian Analog pertama Anda

AHS

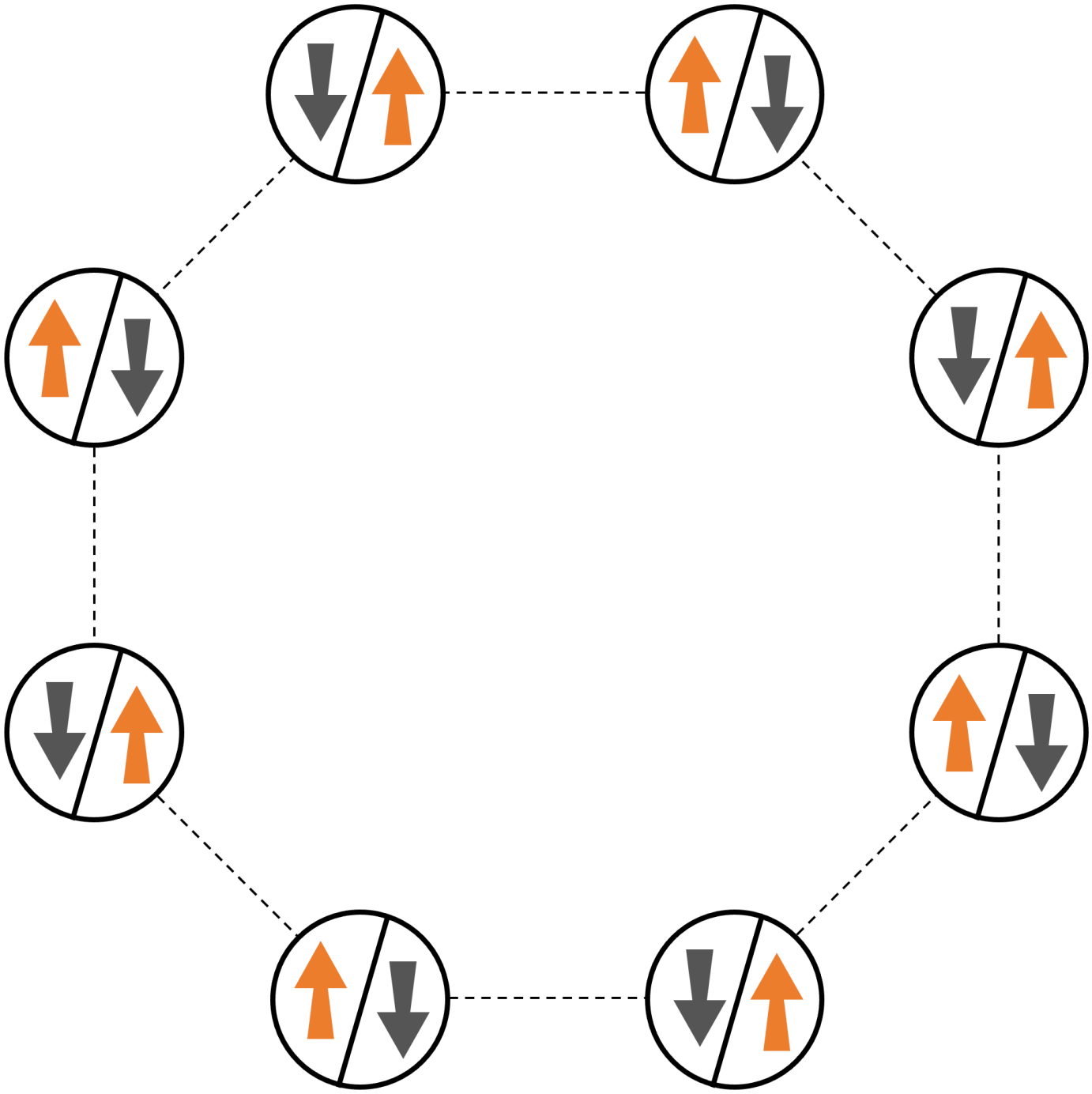
Simulasi Hamiltonian Analog (AHS) adalah paradigma komputasi kuantum yang berbeda dari sirkuit kuantum: alih-alih urutan gerbang, masing-masing bertindak hanya pada beberapa qubit pada satu waktu, program AHS ditentukan oleh parameter yang bergantung pada waktu dan ruang dari Hamiltonian yang bersangkutan. Hamiltonian dari suatu sistem mengkodekan tingkat energinya dan efek kekuatan eksternal, yang bersama-sama mengatur evolusi waktu dari keadaannya. Untuk sistem N-qubit, Hamiltonian dapat diwakili oleh matriks kuadrat $2^N X 2^N$ dari bilangan kompleks.

Perangkat kuantum yang mampu melakukan AHS akan menyetel parameternya (misalnya amplitudo dan detuning medan penggerak yang koheren) untuk mendekati evolusi waktu sistem kuantum di

bawah Hamiltonian khusus. Paradigma AHS cocok untuk mensimulasikan sifat statis dan dinamis dari sistem kuantum dari banyak partikel yang berinteraksi. QPU yang dibuat khusus, seperti perangkat Aquila dari QuEra dapat mensimulasikan evolusi waktu sistem dengan ukuran yang tidak layak pada perangkat keras klasik.

Rantai spin yang berinteraksi

Untuk contoh kanonik dari sistem banyak partikel yang berinteraksi, mari kita pertimbangkan cincin delapan putaran (yang masing-masing dapat berada dalam keadaan “atas”). Meskipun kecil, sistem model ini sudah menunjukkan beberapa fenomena menarik dari bahan magnetik yang terjadi secara alami. Dalam contoh ini, kami akan menunjukkan bagaimana menyiapkan apa yang disebut urutan anti-feromagnetik, di mana putaran berturut-turut mengarah ke arah yang berlawanan.



Pengaturan

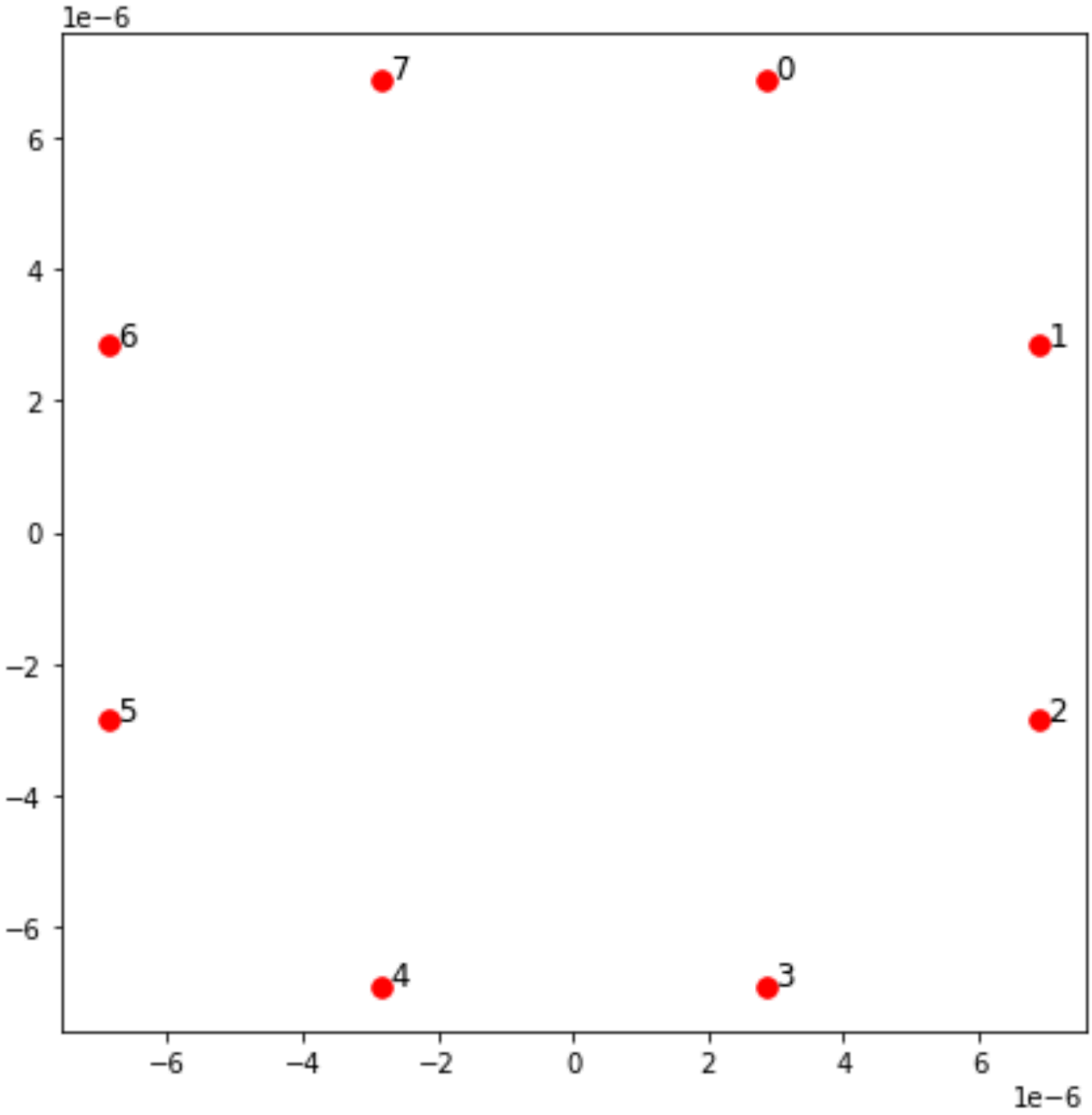
Kita akan menggunakan satu atom netral untuk mewakili setiap putaran, dan status putaran “atas” dan “bawah” akan dikodekan dalam keadaan Rydberg yang tereksitasi dan keadaan dasar atom, masing-masing. Pertama, kami membuat pengaturan 2-d. Kita dapat memprogram cincin putaran di atas dengan kode berikut.

Prasyarat: Anda perlu menginstal Braket SDK. (Jika Anda menggunakan instance notebook yang dihosting Braket, SDK ini sudah diinstal sebelumnya dengan notebook.) Untuk mereproduksi plot, Anda juga perlu menginstal matplotlib secara terpisah dengan perintah shell. `pip install matplotlib`

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

from braket.ahs.atom_arrangement import AtomArrangement

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

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

yang juga bisa kita rencanakan

```
fig, ax = plt.subplots(1, 1, figsize=(7,7))
xs, ys = [register.coordinate_list(dim) for dim in (0, 1)]
ax.plot(xs, ys, 'r.', ms=15)
for idx, (x, y) in enumerate(zip(xs, ys)):
    ax.text(x, y, f" {idx}", fontsize=12)
plt.show()  # this will show the plot below in an ipython or jupyter session
```



Interaksi

Untuk mempersiapkan fase anti-feromagnetik, kita perlu menginduksi interaksi antara putaran tetangga. Kami menggunakan interaksi van der Waals untuk ini, yang secara native diimplementasikan oleh perangkat atom netral (seperti Aquila perangkat dariQuEra). Menggunakan

spin-representasi, istilah Hamiltonian untuk interaksi ini dapat dinyatakan sebagai jumlah atas semua pasangan spin (j, k).

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

Di sini, nj=↑ $_j$ #↑ adalah $_j$ operator yang mengambil nilai 1 hanya jika spin j berada dalam keadaan "naik", dan 0 sebaliknya. Kekuatannya adalah V $_{j,k}$ =C_6/($d_{j,k}$) ^{6}, dimana C $_6$ adalah koefisien tetap, dan d $_{j,k}$ adalah jarak Euclidean antara spin j dan k. Efek langsung dari istilah interaksi ini adalah bahwa setiap keadaan di mana spin j dan spin k "naik" memiliki energi yang meningkat (dengan jumlah $V_{j,k}$). Dengan hati-hati merancang sisa program AHS, interaksi ini akan mencegah putaran tetangga dari keduanya berada dalam keadaan "naik", efek yang umumnya dikenal sebagai "blokade Rydberg."

Bidang mengemudi

Pada awal program AHS, semua putaran (secara default) dimulai dalam keadaan "turun", mereka berada dalam apa yang disebut fase feromagnetik. Mengawasi tujuan kami untuk mempersiapkan fase anti-feromagnetik, kami menentukan medan penggerak koheren yang bergantung pada waktu yang dengan lancar mentransisikan putaran dari keadaan ini ke keadaan banyak tubuh di mana keadaan "naik" lebih disukai. Hamiltonian yang sesuai dapat ditulis sebagai

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

di mana Ω (t), φ (t), Δ (t) adalah amplitudo global yang bergantung pada waktu (alias frekuensi Rabi), fase, dan detuning medan penggerak yang mempengaruhi semua putaran secara seragam. Di sini S $_{-,k}$ =↓ $_k$ #↑ $_k$ ^and S $_{+,k}$ =($S_{-,k}$) † =↑ $_k$ #↓ $_k$ adalah operator penurun dan peningkatan spin k, masing-masing, dan n $_k$ =↑ #↑ adalah operator yang sama seperti sebelumnya. $_k$ $_k$ Bagian Ω dari medan mengemudi secara koheren menggabungkan status "turun" dan "naik" dari semua putaran secara bersamaan, sedangkan bagian Δ mengontrol hadiah energi untuk keadaan "naik".

Untuk memprogram transisi yang mulus dari fase feromagnetik ke fase anti-feromagnetik, kami menentukan bidang penggerak dengan kode berikut.

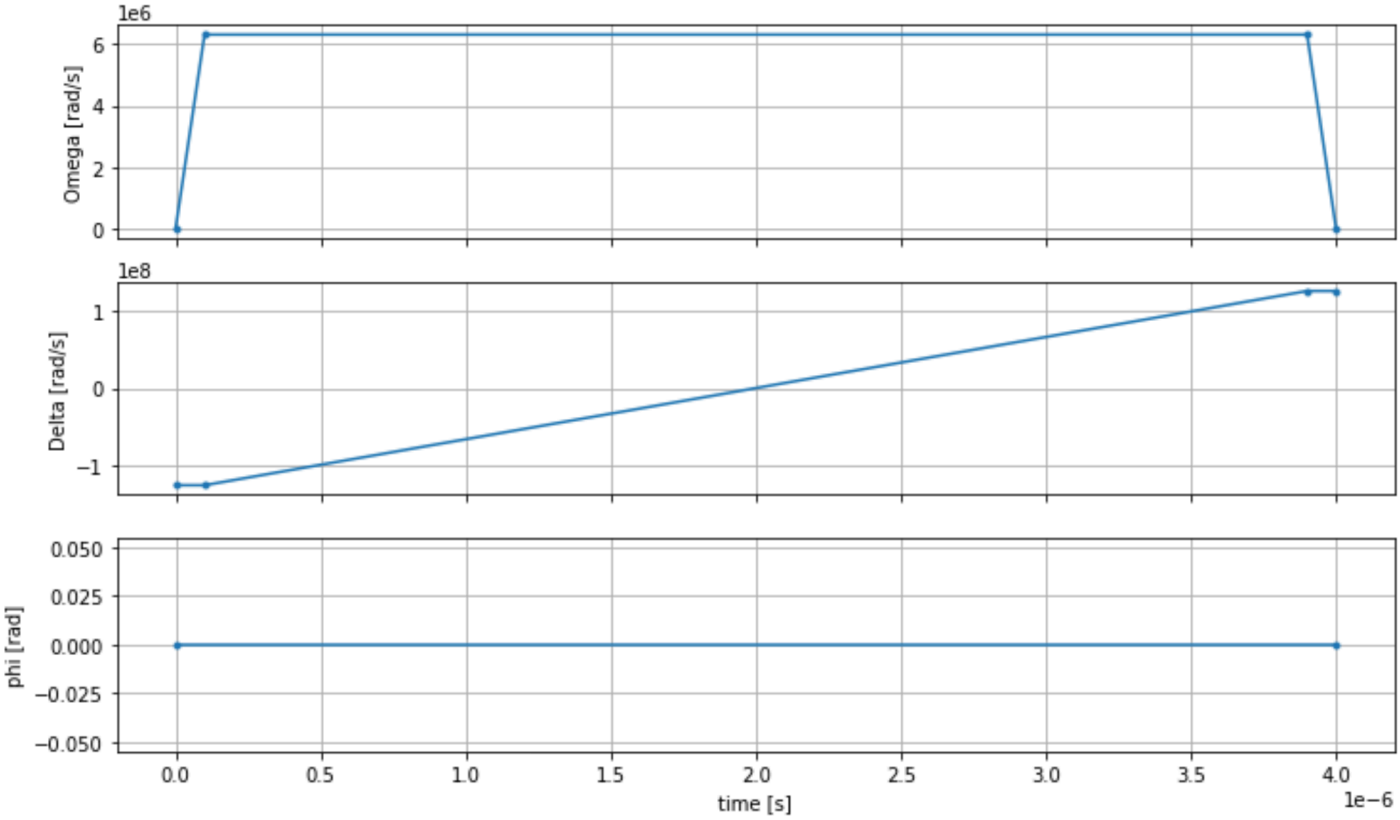
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField

# smooth transition from "down" to "up" state
time_max = 4e-6  # seconds
```

```
time_ramp = 1e-7  # seconds
omega_max = 6300000.0  # rad / sec
delta_start = -5 * omega_max
delta_end = 5 * omega_max

omega = TimeSeries()
omega.put(0.0, 0.0)
omega.put(time_ramp, omega_max)
omega.put(time_max - time_ramp, omega_max)
omega.put(time_max, 0.0)

delta = TimeSeries()
delta.put(0.0, delta_start)
delta.put(time_ramp, delta_start)
delta.put(time_max - time_ramp, delta_end)
delta.put(time_max, delta_end)

phi = TimeSeries().put(0.0, 0.0).put(time_max, 0.0)

drive = DrivingField(
    amplitude=omega,
    phase=phi,
    detuning=delta
)
```

Kita dapat memvisualisasikan deret waktu bidang mengemudi dengan skrip berikut.

```
fig, axes = plt.subplots(3, 1, figsize=(12, 7), sharex=True)

ax = axes[0]
time_series = drive.amplitude.time_series
ax.plot(time_series.times(), time_series.values(), '.-');
ax.grid()
ax.set_ylabel('Omega [rad/s]')

ax = axes[1]
time_series = drive.detuning.time_series
ax.plot(time_series.times(), time_series.values(), '.-');
ax.grid()
ax.set_ylabel('Delta [rad/s]')

ax = axes[2]
time_series = drive.phase.time_series
```

```
# Note: time series of phase is understood as a piecewise constant function
ax.step(time_series.times(), time_series.values(), '.-', where='post');
ax.set_ylabel('phi [rad]')
ax.grid()
ax.set_xlabel('time [s]')

plt.show()  # this will show the plot below in an ipython or jupyter session
```



Program AHS

Register, bidang mengemudi, (dan interaksi van der Waals implisit) membentuk program Simulasi Hamiltonian Analog. `ahs_program`

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

Berjalan di simulator lokal

Karena contoh ini kecil (kurang dari 15 putaran), sebelum menjalankannya pada QPU yang kompatibel dengan AHS, kita dapat menjalankannya di simulator AHS lokal yang dilengkapi dengan Braket SDK. Karena simulator lokal tersedia secara gratis dengan Braket SDK, ini adalah praktik terbaik untuk memastikan bahwa kode kami dapat dijalankan dengan benar.

Di sini, kita dapat mengatur jumlah bidikan ke nilai tinggi (katakanlah, 1 juta) karena simulator lokal melacak evolusi waktu status kuantum dan mengambil sampel dari keadaan akhir; karenanya, meningkatkan jumlah bidikan, sambil meningkatkan total runtime hanya sedikit.

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

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

Menganalisis hasil simulator

Kami dapat menggabungkan hasil bidikan dengan fungsi berikut yang menyimpulkan status setiap putaran (yang mungkin “d” untuk “bawah”, “u” untuk “naik”, atau “e” untuk situs kosong), dan menghitung berapa kali setiap konfigurasi terjadi di seluruh bidikan.

```
from collections import Counter

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

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

    Args:
       result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

    Returns
```

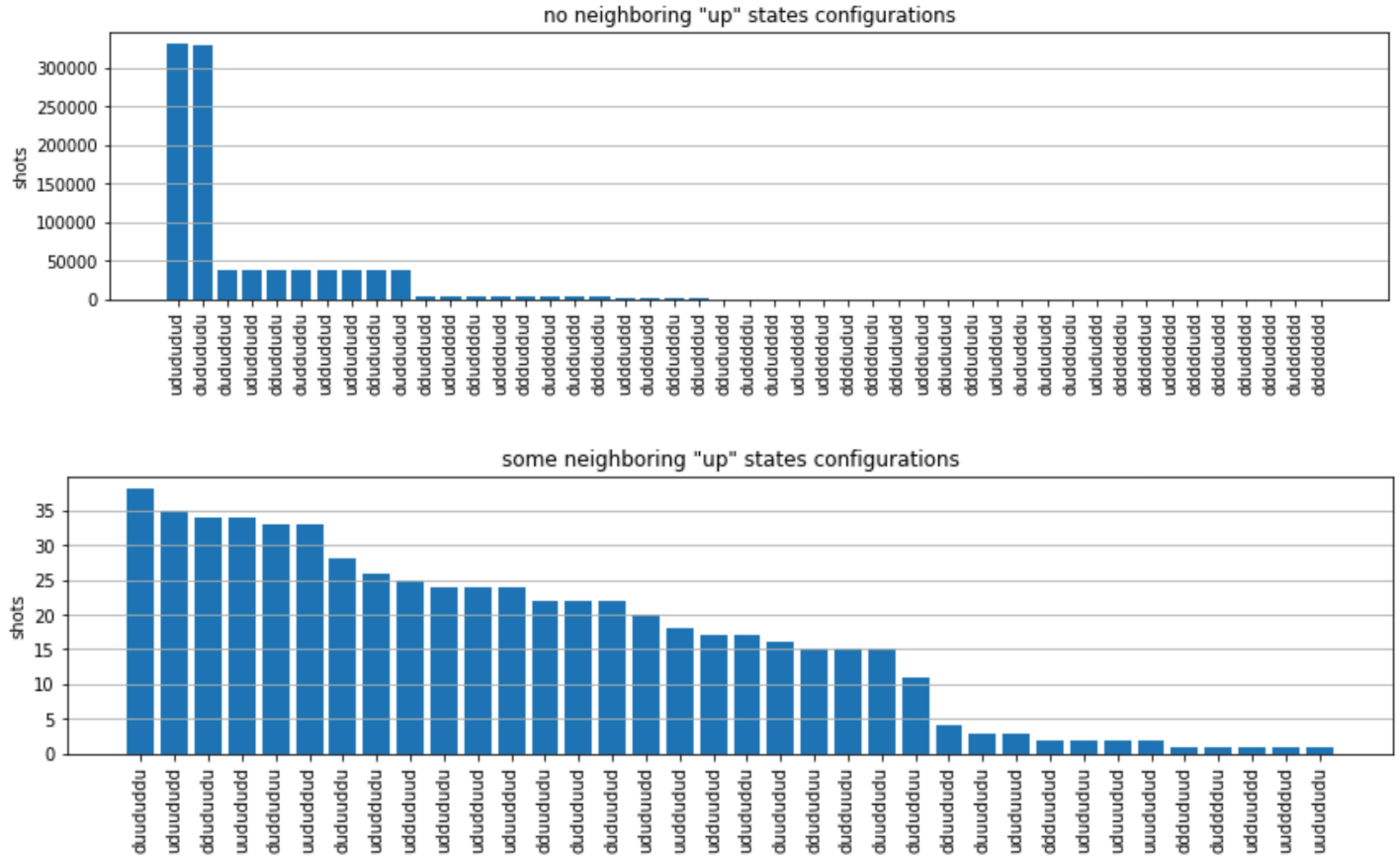
```
        dict: number of times each state configuration is measured

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

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

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

Berikut `counts` adalah kamus yang menghitung berapa kali setiap konfigurasi status diamati di seluruh bidikan. Kami juga dapat memvisualisasikannya dengan kode berikut.

```
from collections import Counter

def has_neighboring_up_states(state):
    if 'uu' in state:
        return True
    if state[0] == 'u' and state[-1] == 'u':
        return True
    return False

def number_of_up_states(state):
    return Counter(state)['u']

def plot_counts(counts):
    non_blockaded = []
    blockaded = []
    for state, count in counts.items():
        if not has_neighboring_up_states(state):
            collection = non_blockaded
        else:
            collection = blockaded
        collection.append((state, count, number_of_up_states(state)))
```

```
    blockaded.sort(key=lambda _: _[1], reverse=True)
    non_blockaded.sort(key=lambda _: _[1], reverse=True)

    for configurations, name in zip((non_blockaded,
                                     blockaded),
                                    ('no neighboring "up" states',
                                     'some neighboring "up" states')):
        plt.figure(figsize=(14, 3))
        plt.bar(range(len(configurations)), [item[1] for item in configurations])
        plt.xticks(range(len(configurations)))
        plt.gca().set_xticklabels([item[0] for item in configurations], rotation=90)
        plt.ylabel('shots')
        plt.grid(axis='y')
        plt.title(f'{name} configurations')
        plt.show()

plot_counts(counts_simulator)
```



Dari plot, kita dapat membaca pengamatan berikut verifikasi bahwa kita berhasil menyiapkan fase anti-feromagnetik.

1. Umumnya, negara bagian yang tidak diblokade (di mana tidak ada dua putaran tetangga yang berada dalam keadaan “naik”) lebih umum daripada negara bagian di mana setidaknya satu pasang putaran tetangga keduanya berada dalam keadaan “naik”.
2. Umumnya, status dengan lebih banyak eksitasi “naik” lebih disukai, kecuali konfigurasi diblokade.
3. Keadaan yang paling umum memang merupakan keadaan anti-feromagnetik yang sempurna dan. `"dudududu" "udududud"`
4. Keadaan paling umum kedua adalah keadaan di mana hanya ada 3 eksitasi “naik” dengan pemisahan berturut-turut 1, 2, 2. Ini menunjukkan bahwa interaksi van der Waals memiliki pengaruh (meskipun jauh lebih kecil) pada tetangga terdekat berikutnya juga.

Berjalan di QuEra Aquila QPU

Prasyarat: Selain pip menginstal Braket SDK, jika Anda baru mengenal Amazon Braket, pastikan Anda telah menyelesaikan langkah-langkah Memulai yang diperlukan.

> **Note**
>
> Jika Anda menggunakan instance notebook yang dihosting Braket, Braket SDK sudah diinstal sebelumnya dengan instance tersebut.

Dengan semua dependensi diinstal, kita dapat terhubung ke QPU. Aquila

```
from braket.aws import AwsDevice

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

Untuk membuat program AHS kami cocok untuk QuEra mesin, kami perlu membulatkan semua nilai untuk memenuhi tingkat presisi yang diizinkan oleh Aquila QPU. (Persyaratan ini diatur oleh parameter perangkat dengan “Resolusi” dalam namanya. Kita bisa melihatnya dengan mengeksekusi `aquila_qpu.properties.dict()` di notebook. Untuk detail lebih lanjut tentang kemampuan dan persyaratan Aquila, lihat buku catatan Pengantar Aquila.) Kita bisa melakukan ini dengan memanggil `discretize` metode.

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

Sekarang kita dapat menjalankan program (hanya menjalankan 100 tembakan untuk saat ini) pada Aquila QPU.

> **Note**
>
> Menjalankan program ini pada Aquila prosesor akan dikenakan biaya. Amazon Braket SDK menyertakan Cost Tracker yang memungkinkan pelanggan untuk menetapkan batas biaya serta melacak biaya mereka dalam waktu dekat.

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

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

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

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

Karena varians yang besar tentang berapa lama tugas kuantum dapat dijalankan (tergantung pada jendela ketersediaan dan pemanfaatan QPU), ada baiknya untuk mencatat ARN tugas kuantum, sehingga kami dapat memeriksa statusnya di lain waktu dengan cuplikan kode berikut.

```
# Optionally, in a new python session

from braket.aws import AwsQuantumTask

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

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

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

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

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

Setelah status SELESAI (yang juga dapat diperiksa dari halaman tugas kuantum konsol Amazon Braket), kami dapat menanyakan hasilnya dengan:

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

Menganalisis hasil QPU

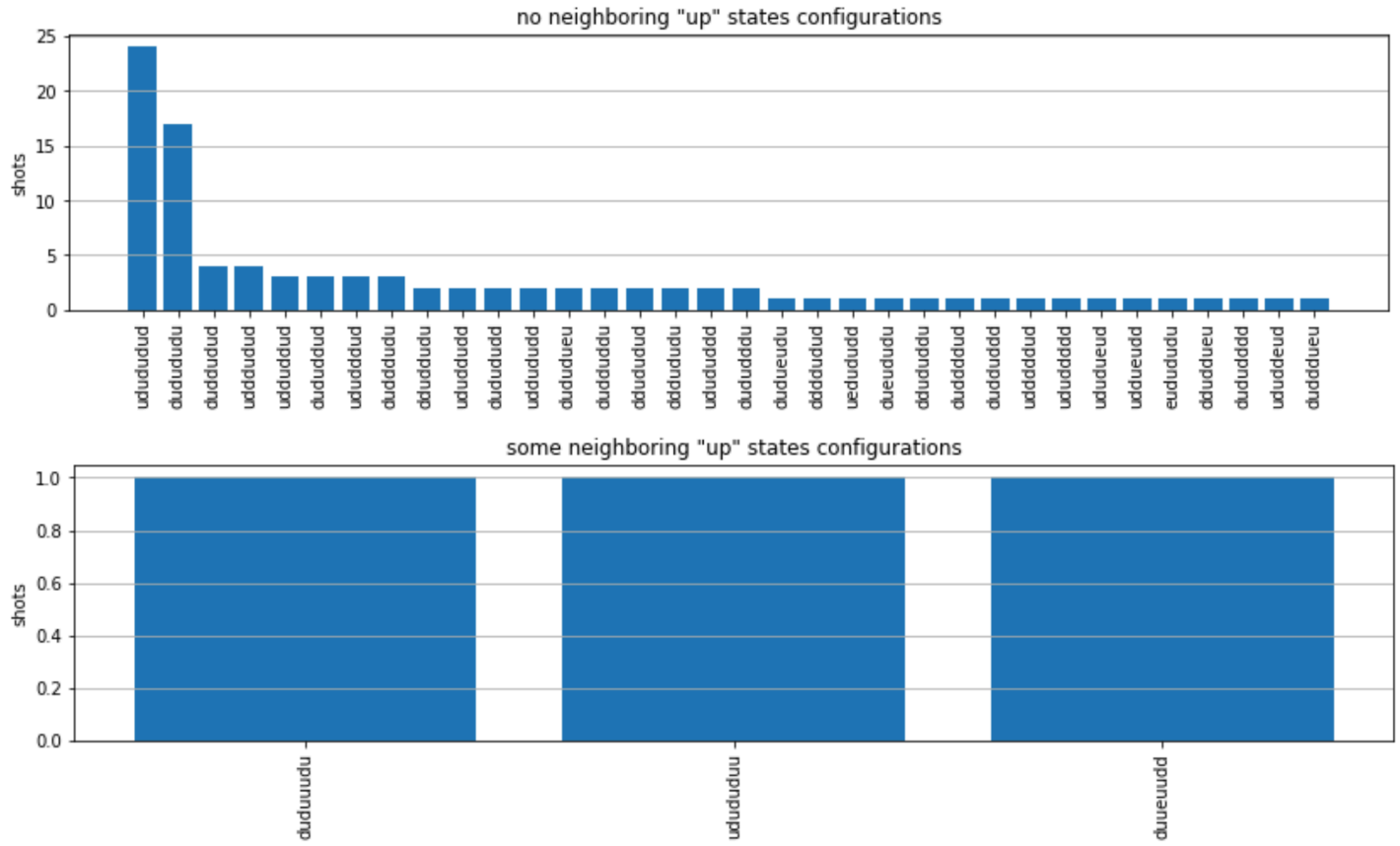
Menggunakan `get_counts` fungsi yang sama seperti sebelumnya, kita dapat menghitung hitungan:

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

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

dan plot mereka dengan`plot_counts`:

```
plot_counts(counts_aquila)
```



Perhatikan bahwa sebagian kecil bidikan memiliki situs kosong (ditandai dengan “e”). Hal ini disebabkan oleh ketidaksempurnaan persiapan atom 1-2% per atom dari QPU. Aquila Selain itu, hasilnya cocok dengan simulasi dalam fluktuasi statistik yang diharapkan karena sejumlah kecil tembakan.

Selanjutnya

Selamat, Anda sekarang telah menjalankan beban kerja AHS pertama Anda di Amazon Braket menggunakan simulator AHS lokal dan QPU. Aquila

Untuk mempelajari lebih lanjut tentang fisika Rydberg, Simulasi Hamiltonian Analog dan Aquila perangkat, lihat contoh notebook kami.

Bangun sirkuit di SDK

Bagian ini memberikan contoh mendefinisikan sirkuit, melihat gerbang yang tersedia, memperluas sirkuit, dan melihat gerbang yang didukung setiap perangkat. Ini juga berisi instruksi tentang cara

mengalokasikan secara manualqubits, menginstruksikan kompiler untuk menjalankan sirkuit Anda persis seperti yang ditentukan, dan membangun sirkuit bising dengan simulator kebisingan.

Anda juga dapat bekerja pada tingkat pulsa di Braket untuk berbagai gerbang dengan QPU tertentu. Untuk informasi selengkapnya, lihat Kontrol Pulsa di Amazon Braket.

Di bagian ini:

- Gerbang dan sirkuit
- Pengukuran Sebagian
- Alokasi manual qubit
- Kompilasi kata demi kata
- Simulasi kebisingan
- Memeriksa sirkuit
- Tipe Hasil

Gerbang dan sirkuit

Gerbang kuantum dan sirkuit didefinisikan dalam `braket.circuits`kelas Amazon Braket Python SDK. Dari SDK, Anda dapat memberi contoh objek sirkuit baru dengan memanggil `Circuit()`.

Contoh: Tentukan sirkuit

Contoh dimulai dengan mendefinisikan rangkaian sampel empat qubits (berlabelq0,,q1, danq3) yang terdiri dari gerbang q2 Hadamard qubit tunggal standar dan gerbang CNOT dua-qubit. Anda dapat memvisualisasikan rangkaian ini dengan memanggil `print` fungsi seperti yang ditunjukkan contoh berikut.

```
# 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 |
```

Contoh: Tentukan sirkuit berparameter

Dalam contoh ini, kami mendefinisikan sirkuit dengan gerbang yang bergantung pada parameter bebas. Kita dapat menentukan nilai parameter ini untuk membuat sirkuit baru, atau, saat mengirimkan sirkuit, untuk dijalankan sebagai tugas kuantum pada perangkat tertentu.

```
from braket.circuits import Circuit, FreeParameter

#define a FreeParameter to represent the angle of a gate
alpha = FreeParameter("alpha")

#define a circuit with three qubits
my_circuit = Circuit().h(range(3)).cnot(control=0, target=2).rx(0, alpha).rx(1, alpha)
print(my_circuit)
```

Anda dapat membuat sirkuit non-parametris baru dari yang diparametris dengan memasok satu `float` (yang merupakan nilai yang akan diambil semua parameter bebas) atau argumen kata kunci yang menentukan nilai setiap parameter ke rangkaian sebagai berikut.

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

Perhatikan bahwa tidak `my_circuit` dimodifikasi, sehingga Anda dapat menggunakannya untuk membuat instance banyak sirkuit baru dengan nilai parameter tetap.

Contoh: Memodifikasi gerbang di sirkuit

Contoh berikut mendefinisikan sirkuit dengan gerbang yang menggunakan kontrol dan pengubah daya. Anda dapat menggunakan modifikasi ini untuk membuat gerbang baru, seperti `Ry` gerbang yang dikendalikan.

```
from braket.circuits import Circuit
```

```
# Create a bell circuit with a controlled x gate
my_circuit = Circuit().h(0).x(control=0, target=1)

# Add a multi-controlled Ry gate of angle .13
my_circuit.ry(angle=.13, target=2, control=(0, 1))

# Add a 1/5 root of X gate
my_circuit.x(0, power=1/5)

print(my_circuit)
```

Pengubah gerbang hanya didukung pada simulator lokal.

Contoh: Lihat semua gerbang yang tersedia

Contoh berikut menunjukkan bagaimana melihat semua gerbang yang tersedia di Amazon Braket.

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

Output dari kode ini mencantumkan semua gerbang.

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

Setiap gerbang ini dapat ditambahkan ke sirkuit dengan memanggil metode untuk jenis sirkuit. Misalnya, Anda akan meneleponcirc.h(0), untuk menambahkan gerbang Hadamard ke yang pertama. qubit

> **Note**
>
> Gerbang ditambahkan di tempat, dan contoh yang mengikutinya menambahkan semua gerbang yang tercantum dalam contoh sebelumnya ke sirkuit yang sama.

```
circ = Circuit()
# toffoli gate with q0, q1 the control qubits and q2 the target.
```

```
circ.ccnot(0, 1, 2)
# cnot gate
circ.cnot(0, 1)
# controlled-phase gate that phases the |11> state, cphaseshift(phi) =
 diag((1,1,1,exp(1j*phi))), where phi=0.15 in the examples below
circ.cphaseshift(0, 1, 0.15)
# controlled-phase gate that phases the |00> state, cphaseshift00(phi) =
 diag([exp(1j*phi),1,1,1])
circ.cphaseshift00(0, 1, 0.15)
# controlled-phase gate that phases the |01> state, cphaseshift01(phi) =
 diag([1,exp(1j*phi),1,1])
circ.cphaseshift01(0, 1, 0.15)
# controlled-phase gate that phases the |10> state, cphaseshift10(phi) =
 diag([1,1,exp(1j*phi),1])
circ.cphaseshift10(0, 1, 0.15)
# controlled swap gate
circ.cswap(0, 1, 2)
# swap gate
circ.swap(0,1)
# phaseshift(phi)= diag([1,exp(1j*phi)])
circ.phaseshift(0,0.15)
# controlled Y gate
circ.cy(0, 1)
# controlled phase gate
circ.cz(0, 1)
# Echoed cross-resonance gate applied to q0, q1
circ = Circuit().ecr(0,1)
# X rotation with angle 0.15
circ.rx(0, 0.15)
# Y rotation with angle 0.15
circ.ry(0, 0.15)
# Z rotation with angle 0.15
circ.rz(0, 0.15)
# Hadamard gates applied to q0, q1, q2
circ.h(range(3))
# identity gates applied to q0, q1, q2
circ.i([0, 1, 2])
# iswap gate, iswap = [[1,0,0,0],[0,0,1j,0],[0,1j,0,0],[0,0,0,1]]
circ.iswap(0, 1)
# pswap gate, PSWAP(phi) = [[1,0,0,0],[0,0,exp(1j*phi),0],[0,exp(1j*phi),0,0],
[0,0,0,1]]
circ.pswap(0, 1, 0.15)
# X gate applied to q1, q2
circ.x([1, 2])
```

```
# Y gate applied to q1, q2
circ.y([1, 2])
# Z gate applied to q1, q2
circ.z([1, 2])
# S gate applied to q0, q1, q2
circ.s([0, 1, 2])
# conjugate transpose of S gate applied to q0, q1
circ.si([0, 1])
# T gate applied to q0, q1
circ.t([0, 1])
# conjugate transpose of T gate applied to q0, q1
circ.ti([0, 1])
# square root of not gate applied to q0, q1, q2
circ.v([0, 1, 2])
# conjugate transpose of square root of not gate applied to q0, q1, q2
circ.vi([0, 1, 2])
# exp(-iXX theta/2)
circ.xx(0, 1, 0.15)
# exp(i(XX+YY) theta/4), where theta=0.15 in the examples below
circ.xy(0, 1, 0.15)
# exp(-iYY theta/2)
circ.yy(0, 1, 0.15)
# exp(-iZZ theta/2)
circ.zz(0, 1, 0.15)
# IonQ native gate GPi with angle 0.15 applied to q0
circ.gpi(0, 0.15)
# IonQ native gate GPi2 with angle 0.15 applied to q0
circ.gpi2(0, 0.15)
# IonQ native gate MS with angles 0.15, 0.15, 0.15 applied to q0, q1
circ.ms(0, 1, 0.15, 0.15, 0.15)
```

Terlepas dari gerbang set yang telah ditentukan sebelumnya, Anda juga dapat menerapkan gerbang kesatuan yang didefinisikan sendiri ke sirkuit. Ini bisa berupa gerbang qubit tunggal (seperti yang ditunjukkan pada kode sumber berikut) atau gerbang multi-qubit yang diterapkan pada yang qubits ditentukan oleh parameter. `targets`

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

Contoh: Perluas sirkuit yang ada

Anda dapat memperpanjang sirkuit yang ada dengan menambahkan instruksi. An `Instruction` adalah direktif kuantum yang menggambarkan tugas kuantum yang harus dilakukan pada perangkat kuantum. `Instruction`operator menyertakan objek tipe `Gate` saja.

```
# 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})
```

Contoh: Lihat gerbang yang didukung setiap perangkat

Simulator mendukung semua gerbang di SDK Braket, namun perangkat QPU mendukung subset yang lebih kecil. Anda dapat menemukan gerbang perangkat yang didukung di properti perangkat. Berikut ini menunjukkan contoh dengan perangkat ionQ:

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

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

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

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

Gerbang yang didukung mungkin perlu dikompilasi ke gerbang asli sebelum mereka dapat berjalan pada perangkat keras kuantum. Saat Anda mengirimkan sirkuit, Amazon Braket melakukan kompilasi ini secara otomatis.

Contoh: Secara terprogram mengambil kesetiaan gerbang asli yang didukung oleh perangkat

Anda dapat melihat informasi kesetiaan di halaman Perangkat konsol Braket. Terkadang sangat membantu untuk mengakses informasi yang sama secara terprogram. Kode berikut menunjukkan cara mengekstrak kesetiaan dua qubit gerbang antara dua gerbang QPU.

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

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

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

Pengukuran Sebagian

Mengikuti contoh sebelumnya, kami telah mengukur semua qubit di sirkuit kuantum. Namun, dimungkinkan untuk mengukur qubit individu atau subset qubit.

Contoh: Ukur subset qubit

Dalam contoh ini, kami mendemonstrasikan pengukuran sebagian dengan menambahkan `measure` instruksi dengan qubit target ke ujung rangkaian.

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

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

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

# Get the results
result = task.result()

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

Alokasi manual qubit

Ketika Anda menjalankan sirkuit kuantum pada komputer kuantum dariRigetti, Anda dapat secara opsional menggunakan qubit alokasi manual untuk mengontrol mana yang qubits digunakan untuk algoritme Anda. Konsol Amazon Braket dan SDK Amazon Braket membantu Anda memeriksa data kalibrasi terbaru dari perangkat unit pemrosesan kuantum (QPU) yang Anda pilih, sehingga Anda dapat memilih yang terbaik untuk eksperimen Anda. qubits

qubitAlokasi manual memungkinkan Anda untuk menjalankan sirkuit dengan akurasi yang lebih besar dan untuk menyelidiki qubit properti individu. Peneliti dan pengguna tingkat lanjut mengoptimalkan desain sirkuit mereka berdasarkan data kalibrasi perangkat terbaru dan dapat memperoleh hasil yang lebih akurat.

Contoh berikut menunjukkan bagaimana qubits mengalokasikan secara eksplisit.

```
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)
```

Untuk informasi selengkapnya, lihat contoh Amazon Braket di GitHub, atau lebih khusus lagi, buku catatan ini: Mengalokasikan Qubit di Perangkat QPU.

> **Note**
>
> OQCKompiler tidak mendukung pengaturan`disable_qubit_rewiring=True`. Menyetel bendera ini untuk `True` menghasilkan kesalahan berikut:`An error occurred (ValidationException) when calling the CreateQuantumTask operation: Device arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy does not support disabled qubit rewiring`.

Kompilasi kata demi kata

Saat Anda menjalankan sirkuit kuantum pada komputer kuantum dariRigetti,IonQ, atau Oxford Quantum Circuits (OQC), Anda dapat mengarahkan kompiler untuk menjalankan sirkuit Anda persis seperti yang ditentukan tanpa modifikasi apa pun. Dengan menggunakan kompilasi kata demi kata, Anda dapat menentukan bahwa seluruh rangkaian dipertahankan secara tepat (didukung olehRigetti,IonQ, danOQC) seperti yang ditentukan atau hanya bagian tertentu saja yang dipertahankan (hanya didukung olehRigetti). Saat mengembangkan algoritme untuk pembandingan perangkat keras atau protokol mitigasi kesalahan, Anda perlu memiliki opsi untuk menentukan dengan tepat gerbang dan tata letak sirkuit yang Anda jalankan pada perangkat keras. Kompilasi verbatim memberi Anda kontrol langsung atas proses kompilasi dengan mematikan langkah-langkah pengoptimalan tertentu, sehingga memastikan bahwa sirkuit Anda berjalan persis seperti yang dirancang.

Kompilasi kata demi kata saat ini didukung pada perangkat RigettiIonQ,, dan Oxford Quantum Circuits (OQC) dan memerlukan penggunaan gerbang asli. Saat menggunakan kompilasi kata demi kata, disarankan untuk memeriksa topologi perangkat untuk memastikan bahwa gerbang dipanggil terhubung qubits dan bahwa sirkuit menggunakan gerbang asli yang didukung pada perangkat keras. Contoh berikut menunjukkan cara mengakses daftar gerbang asli yang didukung oleh perangkat secara terprogram.

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

UntukRigetti, qubit rewiring harus dimatikan dengan pengaturan `disableQubitRewiring=True` untuk digunakan dengan kompilasi kata demi kata. Jika `disableQubitRewiring=False` disetel saat menggunakan kotak kata demi kata dalam kompilasi, rangkaian kuantum gagal validasi dan tidak berjalan.

Jika kompilasi kata demi kata diaktifkan untuk sirkuit dan dijalankan pada QPU yang tidak mendukungnya, kesalahan dihasilkan yang menunjukkan bahwa operasi yang tidak didukung telah menyebabkan tugas gagal. Karena semakin banyak perangkat keras kuantum yang secara native mendukung fungsi kompiler, fitur ini akan diperluas untuk menyertakan perangkat ini. Perangkat yang mendukung kompilasi kata demi kata menyertakannya sebagai operasi yang didukung saat ditanyakan dengan kode berikut.

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

Tidak ada biaya tambahan yang terkait dengan penggunaan kompilasi kata demi kata. Anda terus dikenakan biaya untuk tugas kuantum yang dijalankan pada perangkat Braket QPU, instans notebook, dan simulator sesuai permintaan berdasarkan tarif saat ini sebagaimana ditentukan pada halaman Harga Amazon Braket. Untuk informasi selengkapnya, lihat buku catatan contoh kompilasi Verbatim.

> **Note**
>
> Jika Anda menggunakan OpenQASM untuk menulis sirkuit Anda untuk IonQ perangkat OQC dan, dan Anda ingin memetakan sirkuit Anda langsung ke qubit fisik, Anda perlu menggunakan `#pragma braket verbatim` sebagai `disableQubitRewiring` bendera benar-benar diabaikan oleh OpenQASM.

Simulasi kebisingan

Untuk membuat instance simulator kebisingan lokal, Anda dapat mengubah backend sebagai berikut.

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

Anda dapat membangun sirkuit berisik dengan dua cara:

1. Bangun sirkuit berisik dari bawah ke atas.
2. Ambil sirkuit bebas kebisingan yang ada dan suntikkan kebisingan ke seluruh bagian.

Contoh berikut menunjukkan pendekatan menggunakan sirkuit sederhana dengan derau depolarisasi dan saluran Kraus khusus.

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

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

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

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

Menjalankan sirkuit adalah pengalaman pengguna yang sama seperti sebelumnya, seperti yang ditunjukkan dalam dua contoh berikut.

Contoh 1

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

Atau

Contoh 2

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

Untuk contoh lainnya, lihat contoh simulator bising pengantar Braket

Memeriksa sirkuit

Sirkuit kuantum di Amazon Braket memiliki konsep pseudo-time yang disebut. `Moments` Masing-masing qubit dapat mengalami satu gerbang per`Moment`. Tujuannya `Moments` adalah untuk membuat sirkuit dan gerbangnya lebih mudah diatasi dan menyediakan struktur temporal.

> **Note**
>
> Momen umumnya tidak sesuai dengan waktu nyata di mana gerbang dieksekusi di QPU.

Kedalaman sirkuit diberikan oleh jumlah total Momen di sirkuit tersebut. Anda dapat melihat kedalaman sirkuit memanggil metode `circuit.depth` seperti yang ditunjukkan pada contoh berikut.

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

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

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

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

Kedalaman rangkaian total rangkaian di atas adalah 3 (ditunjukkan sebagai momen`0`,`1`, dan`2`). Anda dapat memeriksa operasi gerbang untuk setiap momen.

`Moments`berfungsi sebagai kamus pasangan kunci-nilai.

- Kuncinya adalah`MomentsKey()`, yang berisi pseudo-time dan qubit informasi.
- Nilai yang ditetapkan pada jenis `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)]))
```

Anda juga dapat menambahkan gerbang ke sirkuit melalui `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|
```

Tipe Hasil

AmazonBraket dapat mengembalikan berbagai jenis hasil ketika rangkaian diukur menggunakan`ResultType`. Sirkuit dapat mengembalikan jenis hasil berikut.

- `AdjointGradient`mengembalikan gradien (turunan vektor) dari nilai ekspektasi dari observable yang disediakan. Observable ini bekerja pada target yang diberikan sehubungan dengan parameter yang ditentukan menggunakan metode diferensiasi adjoint. Anda hanya dapat menggunakan metode ini ketika shots=0.
- `Amplitude`mengembalikan amplitudo keadaan kuantum tertentu dalam fungsi gelombang keluaran. Ini hanya tersedia di simulator lokal SV1 dan lokal.
- `Expectation`mengembalikan nilai ekspektasi dari observable yang diberikan, yang dapat ditentukan dengan `Observable` kelas diperkenalkan nanti dalam chapter ini. Target yang qubits digunakan untuk mengukur yang dapat diamati harus ditentukan, dan jumlah target yang ditentukan harus sama dengan jumlah qubits tindakan yang dapat diamati. Jika tidak ada target yang ditentukan, observable harus beroperasi hanya pada 1 qubit dan diterapkan ke semua secara qubits paralel.
- `Probability`mengembalikan probabilitas pengukuran keadaan basis komputasi. Jika target tidak ditentukan, `Probability` mengembalikan probabilitas mengukur semua keadaan dasar. Jika target ditentukan, hanya probabilitas marjinal dari vektor dasar pada yang ditentukan yang dikembalikan. qubits
- `Reduced density matrix`mengembalikan matriks kepadatan untuk subsistem target tertentu qubits dari sistem. qubits Untuk membatasi ukuran jenis hasil ini, Braket membatasi jumlah target qubits hingga maksimal 8.
- `StateVector`mengembalikan vektor status penuh. Ini tersedia di simulator lokal.
- `Sample`mengembalikan jumlah pengukuran dari qubit set target tertentu dan dapat diamati. Jika tidak ada target yang ditentukan, observable harus beroperasi hanya pada 1 qubit dan diterapkan ke semua secara qubits paralel. Jika target ditentukan, jumlah target yang ditentukan harus sama dengan jumlah qubits tindakan yang dapat diamati.
- `Variance`mengembalikan varians (`mean([x-mean(x)]`^{2}`)`) dari qubit set target yang ditentukan dan dapat diamati sebagai jenis hasil yang diminta. Jika tidak ada target yang ditentukan, observable harus beroperasi hanya pada 1 qubit dan diterapkan ke semua secara qubits paralel. Jika tidak, jumlah target yang ditentukan harus sama dengan jumlah qubits yang dapat diamati dapat diterapkan.

Jenis hasil yang didukung untuk perangkat yang berbeda:

	Sim lokal	SV1	DM1	TN1	Rigetti	IonQ	OQC
Gradien Bersebelahan	T	T	T	T	T	T	T
Amplitudo	T	T	T	T	T	T	T
Perkiraan	T	Y	Y	Y	Y	Y	T
probabilitas	T	Y	T	T	Y*	T	T
Mengurangi matriks kepadatan	T	T	T	T	T	T	T
Vektor keadaan	T	T	T	T	T	T	T
Sampel	T	Y	Y	Y	Y	Y	T
Varians	T	Y	Y	Y	Y	Y	T

> **Note**
>
> * Rigetti hanya mendukung jenis hasil probabilitas hingga 40qubits.

Anda dapat memeriksa jenis hasil yang didukung dengan memeriksa properti perangkat, seperti yang ditunjukkan pada contoh berikut.

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

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

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

Untuk memanggil a`ResultType`, tambahkan ke sirkuit, seperti yang ditunjukkan pada contoh berikut.

```
from braket.circuits import Observable

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

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

> **Note**
>
> Beberapa perangkat memberikan pengukuran (misalnyaRigetti) sebagai hasil dan yang lain memberikan probabilitas sebagai hasil (misalnya IonQ danOQC). SDK menyediakan properti pengukuran pada hasil, tetapi untuk perangkat yang mengembalikan probabilitas, itu adalah pasca-dihitung. Dengan demikian, perangkat seperti yang disediakan oleh IonQ dan OQC memiliki hasil pengukuran yang ditentukan oleh probabilitas karena pengukuran per tembakan tidak dikembalikan. Anda dapat memeriksa apakah hasilnya dihitung pasca dengan melihat objek hasil seperti yang ditunjukkan dalam file ini. `measurements_copied_from_device`

Dapat diamati

AmazonBraket termasuk `Observable` kelas, yang dapat digunakan untuk menentukan observable untuk diukur.

Anda dapat menerapkan paling banyak satu non-identitas unik yang dapat diamati untuk masing-masing. qubit Jika Anda menentukan dua atau lebih pengamatan non-identitas yang berbeda ke yang samaqubit, Anda akan melihat kesalahan. Untuk tujuan ini, setiap faktor produk tensor dihitung

sebagai individu yang dapat diamati, sehingga diperbolehkan untuk memiliki beberapa produk tensor yang bekerja pada yang samaqubit, asalkan faktor yang bekerja pada itu qubit sama.

Anda juga dapat menskalakan yang dapat diamati dan menambahkan yang dapat diamati (diskalakan atau tidak). Ini menciptakan `Sum` yang dapat digunakan dalam tipe `AdjointGradient` hasil.

`Observable`Kelas ini mencakup observable berikut.

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

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

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

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

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

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

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

```
 [ 0.+1.j  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j -0.-1.j  0.+0.j -0.+0.j]]
Factorize an observable: (Y('qubit_count': 1), Z('qubit_count': 1))
Self-defined Hermitian: Hermitian('qubit_count': 1, 'matrix': [[0.+0.j 1.+0.j], [1.+0.j
 0.+0.j]])
Sum of other (scaled) observables: Sum(TensorProduct(X('qubit_count': 1),
 X('qubit_count': 1)), TensorProduct(Z('qubit_count': 1), Z('qubit_count': 1)))
```

Parameter

Sirkuit dapat mencakup parameter bebas, yang dapat Anda gunakan dengan cara “membangun sekali - jalankan berkali-kali” dan untuk menghitung gradien. Parameter gratis memiliki nama yang disandikan string yang dapat Anda gunakan untuk menentukan nilainya atau untuk menentukan apakah akan membedakannya.

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

Untuk parameter yang ingin Anda bedakan, tentukan keduanya dengan menggunakan namanya (sebagai string) atau dengan referensi langsung. Perhatikan bahwa menghitung gradien menggunakan tipe `AdjointGradient` hasil dilakukan sehubungan dengan nilai ekspektasi yang dapat diamati.

Catatan: Jika Anda telah memperbaiki nilai parameter bebas dengan meneruskannya sebagai argumen ke sirkuit berparameter, menjalankan sirkuit dengan `AdjointGradient` jenis hasil dan parameter yang ditentukan akan menghasilkan kesalahan. Ini karena parameter yang kita gunakan untuk membedakan dengan tidak lagi ada. Lihat contoh berikut ini.

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

Mengirimkan tugas kuantum ke QPU dan simulator

AmazonBraket menyediakan akses ke beberapa perangkat yang dapat menjalankan tugas kuantum. Anda dapat mengirimkan tugas kuantum satu per satu atau Anda dapat mengatur batch tugas kuantum.

QPU

Anda dapat mengirimkan tugas kuantum ke qPU kapan saja, tetapi tugas berjalan dalam jendela ketersediaan tertentu yang ditampilkan di halaman Perangkat konsol Amazon Braket. Anda dapat mengambil hasil tugas kuantum dengan ID tugas kuantum, yang diperkenalkan di bagian berikutnya.

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

Simulator

- Simulator matriks kepadatan, DM1: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- Simulator vektor negara, SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Simulator jaringan tensor, TN1: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- Simulator lokal : `LocalSimulator()`

> **Note**
>
> Anda dapat membatalkan tugas kuantum di `CREATED` negara bagian untuk QPU dan simulator sesuai permintaan. Anda dapat membatalkan tugas kuantum di `QUEUED` negara bagian dengan upaya terbaik untuk simulator dan QPU sesuai permintaan. Perhatikan bahwa tugas `QUEUED` kuantum QPU tidak mungkin berhasil dibatalkan selama jendela ketersediaan QPU.

Di bagian ini:

Contoh tugas kuantum di Amazon Braket

Bagian ini berjalan melalui tahapan menjalankan contoh tugas kuantum, mulai dari memilih perangkat hingga melihat hasilnya. Sebagai praktik terbaik untuk Amazon Braket, kami sarankan Anda mulai dengan menjalankan sirkuit pada simulator, sepertiSV1.

Di bagian ini:

Tentukan perangkat

Pertama, pilih dan tentukan perangkat untuk tugas kuantum Anda. Contoh ini menunjukkan bagaimana memilih simulator,SV1.

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

Anda dapat melihat beberapa properti perangkat ini sebagai berikut:

```
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)])
```

Kirimkan contoh tugas kuantum

Kirimkan contoh tugas kuantum untuk dijalankan di simulator sesuai permintaan.

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

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

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

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

`device.run()`Perintah membuat tugas kuantum melalui `CreateQuantumTask` API. Setelah waktu inisialisasi yang singkat, tugas kuantum diantrian sampai ada kapasitas untuk menjalankan tugas kuantum pada perangkat. Dalam hal ini, perangkatnya`SV1`. Setelah perangkat menyelesaikan

perhitungan, Amazon Braket menulis hasilnya ke lokasi Amazon S3 yang ditentukan dalam panggilan. Argumen posisi `s3_location` diperlukan untuk semua perangkat kecuali simulator lokal.

> **Note**
>
> Tindakan tugas kuantum Braket dibatasi hingga 3MB.

Kirim tugas parametris

Amazon Braket on-demand dan simulator lokal serta QPU juga mendukung menentukan nilai parameter gratis pada pengiriman tugas. Anda dapat melakukan ini dengan menggunakan `inputs` argumen untuk`device.run()`, seperti yang ditunjukkan pada contoh berikut. `inputs`Harus berupa kamus pasangan string-float, di mana kuncinya adalah nama parameter.

Kompilasi parametrik dapat meningkatkan kinerja mengeksekusi sirkuit parametrik pada QPU tertentu. Saat mengirimkan sirkuit parametrik sebagai tugas kuantum ke QPU yang didukung, Braket akan mengkompilasi rangkaian sekali, dan menyimpan hasilnya. Tidak ada kompilasi ulang untuk pembaruan parameter berikutnya ke sirkuit yang sama, menghasilkan runtime yang lebih cepat untuk tugas yang menggunakan sirkuit yang sama. Braket secara otomatis menggunakan data kalibrasi yang diperbarui dari penyedia perangkat keras saat menyusun sirkuit Anda untuk memastikan hasil dengan kualitas terbaik.

> **Note**
>
> Kompilasi parametrik didukung pada semua QPU superkonduktor berbasis gerbang dari Rigetti Computing dan Oxford Quantum Circuits dengan pengecualian program tingkat pulsa.

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

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

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

```
circ.probability(target=[0, 2])
# submit the quantum task to run
my_task = device.run(circ, inputs={'alpha': 0.1, 'beta':0.2})
```

Tentukan shots

shotsArgumen mengacu pada jumlah pengukuran yang diinginkanshots. Simulator seperti SV1 mendukung dua mode simulasi.

- Untuk shots = 0, simulator melakukan simulasi yang tepat, mengembalikan nilai sebenarnya untuk semua jenis hasil. (Tidak tersedia diTN1.)
- Untuk nilai bukan nolshots, sampel simulator dari distribusi output untuk meniru shot kebisingan QPU nyata. Perangkat QPU hanya shots mengizinkan> 0.

Untuk informasi tentang jumlah maksimum bidikan per tugas kuantum, silakan merujuk ke Kuota Braket.

Polling untuk hasil

Saat mengeksekusi`my_task.result()`, SDK memulai polling untuk hasil dengan parameter yang Anda tentukan pada pembuatan tugas kuantum:

- `poll_timeout_seconds`adalah jumlah detik untuk polling tugas kuantum sebelum waktu habis saat menjalankan tugas kuantum pada simulator on-demand dan atau perangkat QPU. Nilai default adalah 432,000 detik, yaitu 5 hari.
- Catatan: Untuk perangkat QPU seperti Rigetti danIonQ, kami sarankan Anda mengizinkan beberapa hari. Jika waktu jajak pendapat terlalu singkat, hasil mungkin tidak dikembalikan dalam waktu jajak pendapat. Sebagai contoh, ketika QPU tidak tersedia, kesalahan waktu habis lokal dikembalikan.
- `poll_interval_seconds`adalah frekuensi tugas kuantum disurvei. Ini menentukan seberapa sering Anda memanggil Braket API untuk mendapatkan status ketika tugas kuantum dijalankan pada simulator sesuai permintaan dan pada perangkat QPU. Nilai default adalah 1 detik.

Eksekusi asinkron ini memfasilitasi interaksi dengan perangkat QPU yang tidak selalu tersedia. Misalnya, perangkat mungkin tidak tersedia selama window perawatan rutin.

Hasil yang dikembalikan berisi berbagai metadata yang terkait dengan tugas kuantum. Anda dapat memeriksa hasil pengukuran dengan perintah berikut:

```
print('Measurement results:\n',result.measurements)
print('Counts for collapsed states:\n',result.measurement_counts)
print('Probabilities for collapsed states:\n',result.measurement_probabilities)
```

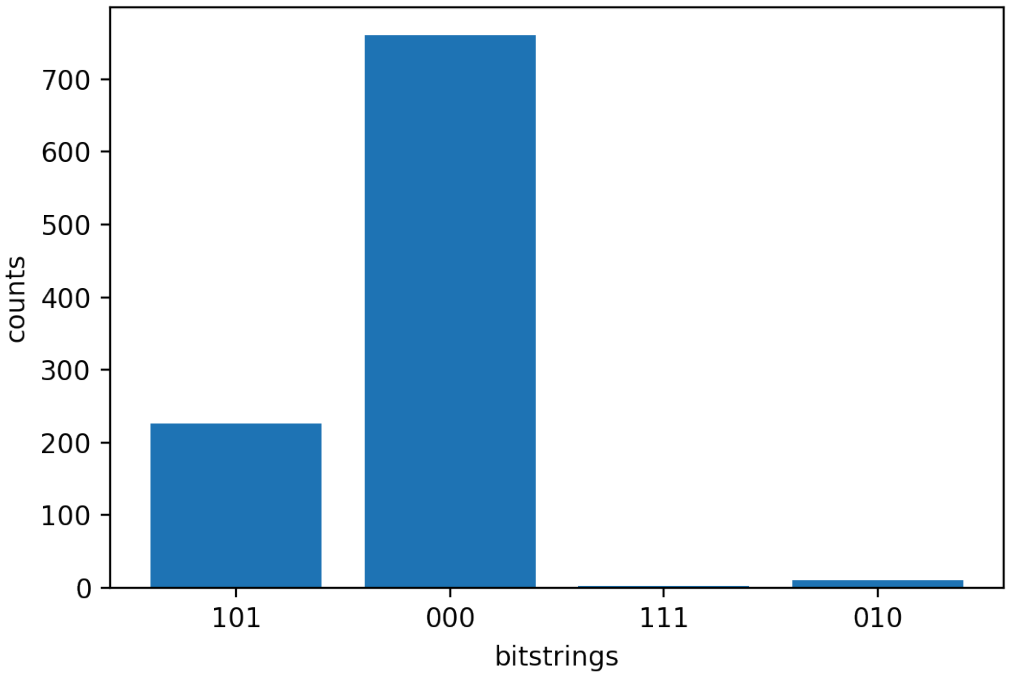
```
Measurement results:
 [[1 0 1]
 [0 0 0]
 [1 0 1]
 ...
 [0 0 0]
 [0 0 0]
 [0 0 0]]
Counts for collapsed states:
 Counter({'000': 761, '101': 226, '010': 10, '111': 3})
Probabilities for collapsed states:
 {'101': 0.226, '000': 0.761, '111': 0.003, '010': 0.01}
```

Contoh: Lihat hasilnya

Karena Anda juga telah menentukan `ResultType`, Anda dapat melihat hasil yang dikembalikan. Jenis hasil muncul dalam urutan penambahannya ke sirkuit.

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

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

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



Mengirimkan tugas kuantum ke QPU

AmazonBraket memungkinkan Anda menjalankan sirkuit kuantum pada perangkat QPU. Contoh berikut menunjukkan cara mengirimkan tugas kuantum ke Rigetti atau IonQ perangkat.

Pilih Rigetti Aspen-M-3 perangkat, lalu lihat grafik konektivitas terkait

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

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

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

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

Kamus sebelumnya `connectivityGraph` berisi informasi tentang konektivitas perangkat saat iniRigetti.

Pilih IonQ Harmony perangkat

Untuk IonQ Harmony perangkat, kosong, seperti yang ditunjukkan pada contoh berikut, karena perangkat menawarkan konektivitas all-to-all. `connectivityGraph` Oleh karena itu, `connectivityGraph` rinci tidak diperlukan.

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

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

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

Seperti yang ditunjukkan pada contoh berikut, Anda memiliki opsi untuk menyesuaikan shots (default=1000), `poll_timeout_seconds` (default = 432000 = 5 hari), `poll_interval_seconds` (default = 1), dan lokasi bucket S3 (`s3_location`) tempat hasil Anda akan disimpan jika Anda memilih untuk menentukan lokasi selain bucket default.

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

RigettiPerangkat IonQ dan mengkompilasi sirkuit yang disediakan ke dalam set gerbang asli masing-masing secara otomatis, dan mereka memetakan qubit indeks abstrak ke fisik qubits pada QPU masing-masing.

> **Note**
>
> Perangkat QPU memiliki kapasitas terbatas. Anda dapat mengharapkan waktu tunggu lebih lama saat kapasitas tercapai.

AmazonBraket dapat menjalankan tugas kuantum QPU dalam jendela ketersediaan tertentu, tetapi Anda masih dapat mengirimkan tugas kuantum kapan saja (24/7) karena semua data dan metadata yang sesuai disimpan dengan andal di bucket S3 yang sesuai. Seperti yang ditunjukkan di bagian berikutnya, Anda dapat memulihkan tugas kuantum Anda menggunakan `AwsQuantumTask` dan ID tugas kuantum unik Anda.

Menjalankan tugas kuantum dengan simulator lokal

Anda dapat mengirim tugas kuantum langsung ke simulator lokal untuk pembuatan prototipe dan pengujian cepat. Simulator ini berjalan di lingkungan lokal Anda, jadi Anda tidak perlu menentukan lokasi Amazon S3. Hasilnya dihitung langsung di sesi Anda. Untuk menjalankan tugas kuantum pada simulator lokal, Anda hanya harus menentukan shots parameter.

> **Note**
>
> Kecepatan eksekusi dan jumlah qubits maksimum simulator lokal dapat memproses tergantung pada jenis instance notebook Amazon Braket, atau pada spesifikasi perangkat keras lokal Anda.

Perintah berikut semuanya identik dan membuat instance simulator lokal vektor status (bebas kebisingan).

```
# 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")
```

Kemudian jalankan tugas kuantum dengan yang berikut ini.

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

Untuk membuat instance simulator matriks kepadatan lokal (noise), pelanggan mengubah backend sebagai berikut.

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

Mengukur qubit tertentu pada simulator lokal

Simulator vektor keadaan lokal dan simulator matriks kepadatan lokal mendukung sirkuit yang sedang berjalan di mana subset dari qubit sirkuit dapat diukur, yang sering disebut pengukuran paral.

Misalnya, dalam kode berikut Anda dapat membuat sirkuit dua-qubit dan hanya mengukur qubit pertama dengan menambahkan `measure` instruksi dengan qubit target ke akhir rangkaian.

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

# Use the local simulator device
device = LocalSimulator()

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

# Run the circuit
```

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

# Get the results
result = task.result()

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

Pengelompokan tugas kuantum

Batching tugas kuantum tersedia di setiap perangkat Amazon Braket, kecuali simulator lokal. Batching sangat berguna untuk tugas kuantum yang Anda jalankan pada simulator sesuai permintaan (TN1atauSV1) karena mereka dapat memproses beberapa tugas kuantum secara paralel. Untuk membantu Anda mengatur berbagai tugas kuantum, Amazon Braket menyediakan contoh notebook.

Batching memungkinkan Anda untuk meluncurkan tugas kuantum secara paralel. Misalnya, jika Anda ingin membuat perhitungan yang membutuhkan 10 tugas kuantum dan sirkuit dalam tugas kuantum tersebut tidak tergantung satu sama lain, sebaiknya gunakan batching. Dengan begitu, Anda tidak perlu menunggu satu tugas kuantum selesai sebelum tugas lain dimulai.

Contoh berikut menunjukkan cara menjalankan sekumpulan tugas kuantum:

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

Untuk informasi selengkapnya, lihat contoh Amazon Braket on GitHub atau Quantum task batching, yang memiliki informasi lebih spesifik tentang batching.

Tentang pengelompokan tugas kuantum dan biaya

Beberapa peringatan yang perlu diingat mengenai biaya batching dan penagihan tugas kuantum:

- Secara default, pengelompokan tugas kuantum mencoba ulang sepanjang waktu atau gagal tugas kuantum 3 kali.
- Sejumlah tugas kuantum yang berjalan lama, seperti 34 qubits untukSV1, dapat menimbulkan biaya besar. Pastikan untuk memeriksa ulang nilai `run_batch` penetapan dengan hati-hati

sebelum Anda memulai serangkaian tugas kuantum. Kami tidak merekomendasikan menggunakan TN1 dengan`run_batch`.

- TN1dapat menimbulkan biaya untuk tugas fase latihan yang gagal (lihat deskripsi TN1 untuk informasi lebih lanjut). Percobaan ulang otomatis dapat menambah biaya sehingga kami merekomendasikan pengaturan jumlah 'max_retries' pada batching ke 0 saat menggunakan TN1 (lihat Quantum Task Batching, Line 186).

Pengelompokan tugas kuantum dan PennyLane

Manfaatkan batching saat Anda menggunakan PennyLane Amazon Braket dengan menyetel `parallel = True` saat Anda membuat instance perangkat Amazon Braket, seperti yang ditunjukkan pada contoh berikut.

```
device = qml.device("braket.aws.qubit",device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",wires=wires,s3_destination_folder=s3_folder,parallel=True,)
```

Untuk informasi lebih lanjut tentang batching dengan PennyLane, lihat Optimalisasi paralel sirkuit kuantum.

Pengelompokan tugas dan sirkuit parametris

Saat mengirimkan batch tugas kuantum yang berisi sirkuit parametris, Anda dapat menyediakan `inputs` kamus, yang digunakan untuk semua tugas kuantum dalam batch, atau kamus input, dalam hal ini kamus -th dipasangkan dengan tugas `i` -th, seperti yang ditunjukkan pada contoh berikuti. `list`

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

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

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

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

```
# use the same inputs for both circuits in one batch

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

# or provide each task its own set of inputs

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

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

Anda juga dapat menyiapkan daftar kamus input untuk rangkaian parametrik tunggal dan mengirimkannya sebagai kumpulan tugas kuantum. Jika ada kamus masukan N dalam daftar, batch berisi N tugas kuantum. Tugas kuantum `i` ke--sesuai dengan rangkaian yang dijalankan dengan kamus masukan `i` ke--th.

```
from braket.circuits import Circuit, FreeParameter

# create a parametric circuit
circ = Circuit().rx(0, FreeParameter('alpha'))

# provide a list of inputs to execute with the circuit
inputs_list = [{'alpha': 0.1}, {'alpha': 0.2}, {'alpha': 0.3}]

tasks = device.run_batch(circ, inputs=inputs_list)
```

Mengatur notifikasi SNS (opsional)

Anda dapat mengatur notifikasi melalui Amazon Simple Notification Service (SNS) sehingga Anda menerima peringatan saat tugas kuantum Amazon Braket Anda selesai. Pemberitahuan aktif berguna jika Anda mengharapkan waktu tunggu yang lama; misalnya, saat Anda mengirimkan tugas kuantum besar atau saat Anda mengirimkan tugas kuantum di luar jendela ketersediaan perangkat. Jika Anda tidak ingin menunggu tugas kuantum selesai, Anda dapat mengatur pemberitahuan SNS.

Notebook Amazon Braket memandu Anda melalui langkah-langkah pengaturan. Untuk informasi selengkapnya, lihat contoh Amazon Braket GitHub dan, khususnya, buku catatan contoh untuk menyiapkan notifikasi.

Memeriksa sirkuit yang dikompilasi

Ketika sirkuit berjalan pada perangkat keras, itu harus dikompilasi dalam format yang dapat diterima seperti transpiling sirkuit ke gerbang asli yang didukung oleh QPU. Memeriksa output yang

dikompilasi yang sebenarnya bisa sangat berguna untuk tujuan debugging. Anda dapat melihat sirkuit ini untuk keduanya Rigetti dan OQC perangkat menggunakan kode di bawah ini.

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

> **Note**
>
> Hari ini Anda tidak dapat melihat sirkuit yang dikompilasi untuk IonQ perangkat.

Jalankan sirkuit Anda dengan OpenQASM 3.0

AmazonBraket sekarang mendukung OpenQASM 3.0 untuk perangkat kuantum berbasis gerbang dan simulator. Panduan pengguna ini memberikan informasi tentang subset OpenQASM 3.0 yang didukung oleh Braket. Pelanggan Braket sekarang memiliki pilihan untuk mengirimkan sirkuit Braket dengan SDK atau dengan langsung menyediakan string OpenQASM 3.0 ke semua perangkat berbasis gerbang dengan Amazon Braket API dan Amazon Braket Python SDK.

Topik dalam panduan ini memandu Anda melalui berbagai contoh cara menyelesaikan tugas kuantum berikut.

- Buat dan kirimkan tugas kuantum OpenQASM pada perangkat Braket yang berbeda
- Akses operasi dan jenis hasil yang didukung
- Simulasikan kebisingan dengan OpenQASM
- Gunakan kompilasi kata demi kata dengan OpenQASM
- Memecahkan masalah OpenQASM

Panduan ini juga memberikan pengenalan fitur khusus perangkat keras tertentu yang dapat diimplementasikan dengan OpenQASM 3.0 pada Braket dan tautan ke sumber daya lebih lanjut.

Di bagian ini:

- Apa itu OpenQASM 3.0?
- Kapan menggunakan OpenQASM 3.0

Apa itu OpenQASM 3.0?

Open Quantum Assembly Language (OpenQASM) adalah representasi perantara untuk instruksi kuantum. OpenQASM adalah kerangka kerja sumber terbuka dan banyak digunakan untuk spesifikasi program kuantum untuk perangkat berbasis gerbang. Dengan OpenQASM, pengguna dapat memprogram gerbang kuantum dan operasi pengukuran yang membentuk blok bangunan komputasi kuantum. Versi sebelumnya dari OpenQASM (2.0) digunakan oleh sejumlah perpustakaan pemrograman kuantum untuk menggambarkan program sederhana.

Versi baru OpenQASM (3.0) memperluas versi sebelumnya untuk menyertakan lebih banyak fitur, seperti kontrol tingkat pulsa, waktu gerbang, dan aliran kontrol klasik untuk menjembatani kesenjangan antara antarmuka pengguna akhir dan bahasa deskripsi perangkat keras. Detail dan spesifikasi pada versi 3.0 saat ini tersedia di GitHub OpenQASM 3.x Live Specification. Pengembangan future OpenQASM diatur oleh OpenQASM 3.0 Technical Steering Committee, yang merupakan AWS anggota bersama IBM, Microsoft, dan University of Innsbruck.

Kapan menggunakan OpenQASM 3.0

OpenQASM menyediakan kerangka kerja ekspresif untuk menentukan program kuantum melalui kontrol tingkat rendah yang tidak spesifik arsitektur, membuatnya cocok sebagai representasi di beberapa perangkat berbasis gerbang. Dukungan Braket untuk OpenQASM memajukan adopsi sebagai pendekatan yang konsisten untuk mengembangkan algoritma kuantum berbasis gerbang,

mengurangi kebutuhan pengguna untuk belajar dan memelihara perpustakaan dalam berbagai kerangka kerja.

Jika Anda memiliki pustaka program yang ada di OpenQASM 3.0, Anda dapat menyesuaikannya untuk digunakan dengan Braket daripada menulis ulang sirkuit ini sepenuhnya. Peneliti dan pengembang juga harus mendapat manfaat dari peningkatan jumlah perpustakaan pihak ketiga yang tersedia dengan dukungan untuk pengembangan algoritma di OpenQASM.

Bagaimana OpenQASM 3.0 bekerja

Support untuk OpenQASM 3.0 dari Braket memberikan paritas fitur dengan Representasi Menengah saat ini. Ini berarti bahwa apa pun yang dapat Anda lakukan hari ini pada perangkat keras dan simulator sesuai permintaan dengan Braket, Anda dapat melakukannya dengan OpenQASM menggunakan Braket. API Anda dapat menjalankan program OpenQASM 3.0 dengan langsung memasok string OpenQASM ke semua perangkat berbasis gerbang dengan cara yang mirip dengan bagaimana sirkuit saat ini dipasok ke perangkat di Braket. Pengguna Braket juga dapat mengintegrasikan pustaka pihak ketiga yang mendukung OpenQASM 3.0. Sisa panduan ini merinci bagaimana mengembangkan representasi OpenQASM untuk digunakan dengan Braket.

Prasyarat

Untuk menggunakan OpenQASM 3.0 di Amazon Braket, Anda harus memiliki versi v1.8.0 dari Skema Python Amazon Braket dan v1.17.0 atau lebih tinggi dari Amazon Braket Python SDK.

Jika Anda adalah pengguna pertama kali Amazon Braket, Anda harus mengaktifkan Amazon Braket. Untuk petunjuk, lihat Mengaktifkan Amazon Braket.

Fitur OpenQASM apa yang didukung Braket?

Bagian berikut mencantumkan tipe data OpenQASM 3.0, pernyataan, dan instruksi pragma yang didukung oleh Braket.

Di bagian ini:

- Tipe data OpenQASM yang didukung
- Pernyataan OpenQASM yang didukung
- Braket OpenQASM pragma
- Dukungan fitur lanjutan untuk OpenQASM di Simulator Lokal
- Operasi dan tata bahasa yang didukung dengan OpenPulse

Tipe data OpenQASM yang didukung

Tipe data OpenQASM berikut didukung oleh Braket. Amazon

- Bilangan bulat non-negatif digunakan untuk indeks qubit (virtual dan fisik):
  - `cnot q[0], q[1];`
  - `h $0;`
- Angka atau konstanta floating-point dapat digunakan untuk sudut rotasi gerbang:
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi adalah konstanta bawaan di OpenQASM dan tidak dapat digunakan sebagai nama parameter.

- Array bilangan kompleks (dengan `im` notasi openQASM untuk bagian imajiner) diizinkan dalam pragma tipe hasil untuk mendefinisikan hermitian umum yang dapat diamati dan dalam pragma kesatuan:
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

Pernyataan OpenQASM yang didukung

Pernyataan OpenQASM berikut didukung oleh Braket. Amazon

- `Header: OPENQASM 3;`
- Deklarasi bit klasik:
  - `bit b1;`(setara,) `creg b1;`
  - `bit[10] b2;`(setara,) `creg b2[10];`
- Deklarasi Qubit:
  - `qubit b1;`(setara,) `qreg b1;`
  - `qubit[10] b2;`(setara,) `qreg b2[10];`

- Pengindeksan dalam array: `q[0]`
- Masukan: `input float alpha;`
- spesifikasi fisikqubits: `$0`
- Gerbang dan operasi yang didukung pada perangkat:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> Gerbang yang didukung perangkat dapat ditemukan di properti perangkat untuk tindakan OpenQASM; tidak ada definisi gerbang yang diperlukan untuk menggunakan gerbang ini.

- Pernyataan kotak kata demi kata. Saat ini, kami tidak mendukung notasi durasi kotak. Gerbang asli dan fisik qubits diperlukan dalam kotak kata demi kata.

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

- Penugasan pengukuran dan pengukuran pada qubits atau seluruh qubit register.
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`

> **Note**
>
> pi adalah konstanta bawaan di OpenQASM dan tidak dapat digunakan sebagai nama parameter.

Braket OpenQASM pragma

Instruksi pragma OpenQASM berikut didukung oleh Braket. Amazon

- Pragma kebisingan
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- Pragma kata demi kata
  - `#pragma braket verbatim`
- Jenis hasil pragma
  - Jenis hasil invarian dasar:
    - Vektor negara: `#pragma braket result state_vector`
    - Matriks kepadatan: `#pragma braket result density_matrix`
  - Pragma komputasi gradien:
    - Gradien bersebelahan: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Jenis hasil dasar Z:
    - Amplitudo: `#pragma braket result amplitude "01"`
    - Probabilitas: `#pragma braket result probability q[0], q[1]`
  - Jenis hasil yang diputar dasar
    - Harapan: `#pragma braket result expectation x(q[0]) @ y([q1])`
    - Varians: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
    - Sampel: `#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0 kompatibel dengan OpenQASM 2.0, sehingga program yang ditulis menggunakan 2.0 dapat berjalan di Braket. Namun fitur OpenQASM 3.0 yang didukung oleh Braket memang memiliki beberapa perbedaan sintaks kecil, seperti vs dan vs. `qreg` `creg`

`qubit` `bit` Ada juga perbedaan dalam sintaks pengukuran, dan ini perlu didukung dengan sintaks yang benar.

Dukungan fitur lanjutan untuk OpenQASM di Simulator Lokal

Ini `LocalSimulator` mendukung fitur OpenQASM canggih yang tidak ditawarkan sebagai bagian dari simulator QPU atau on-demand Braket. Daftar fitur berikut hanya didukung di`LocalSimulator`:

- Pengubah gerbang
- Gerbang bawaan OpenQASM
- Variabel klasik
- Operasi klasik
- Gerbang kustom
- Kontrol klasik
- File QASM
- Subrutin

Untuk contoh setiap fitur lanjutan, lihat contoh buku catatan ini. Untuk spesifikasi OpenQASM lengkap, lihat situs web OpenQASM.

Operasi dan tata bahasa yang didukung dengan OpenPulse

Tipe OpenPulse Data yang Didukung

Blok Cal:

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

Blok Defcal:

```
// 1 qubit
defcal x $0 {
...
}
```

```
// 1 qubit w. input parameters as constants
defcal my_rx(pi) $0 {
...
}

// 1 qubit w. input parameters as free parameters
defcal my_rz(angle theta) $0 {
...
}

// 2 qubit (above gate args are also valid)
defcal cz $1, $0 {
...
}
```

Bingkai:

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

Bentuk gelombang:

```
// prebuilt
waveform my_waveform_1 = constant(1e-6, 1.0);

//arbitrary
waveform my_waveform_2 = {0.1 + 0.1im, 0.1 + 0.1im, 0.1, 0.1};
```

Contoh Kalibrasi Gerbang Kustom:

```
cal {
    waveform wf1 = constant(1e-6, 0.25);
}

defcal my_x $0 {
   play(wf1, q0_rf_frame);
}

defcal my_cz $1, $0 {
    barrier q0_q1_cz_frame, q0_rf_frame;
    play(q0_q1_cz_frame, wf1);
    delay[300ns] q0_rf_frame
    shift_phase(q0_rf_frame, 4.366186381749424);
```

```
    delay[300ns] q0_rf_frame;
    shift_phase(q0_rf_frame.phase, 5.916747563126659);
    barrier q0_q1_cz_frame, q0_rf_frame;
    shift_phase(q0_q1_cz_frame, 2.183093190874712);
}

bit[2] ro;
my_x $0;
my_cz $1,$0;
c[0] = measure $0;
```

Contoh pulsa sewenang-wenang:

```
bit[2] ro;
cal {
    waveform wf1 = {0.1 + 0.1im, 0.1 + 0.1im, 0.1, 0.1};
    barrier q0_drive, q0_q1_cross_resonance;
    play(q0_q1_cross_resonance, wf1);
    delay[300ns] q0_drive;
    shift_phase(q0_drive, 4.366186381749424);
    delay[300dt] q0_drive;
   barrier q0_drive, q0_q1_cross_resonance;
   play(q0_q1_cross_resonance, wf1);
    ro[0] = capture_v0(r0_measure);
    ro[1] = capture_v0(r1_measure);
}
```

Buat dan kirimkan contoh tugas kuantum OpenQASM 3.0

Anda dapat menggunakan Amazon Braket Python SDK, Boto3, atau AWS CLI untuk mengirimkan tugas kuantum OpenQASM 3.0 ke perangkat Braket. Amazon

Di bagian ini:

- Contoh program OpenQASM 3.0
- Gunakan Python SDK untuk membuat tugas kuantum OpenQASM 3.0
- Gunakan Boto3 untuk membuat tugas kuantum OpenQASM 3.0
- Gunakan AWS CLI untuk membuat tugas OpenQASM 3.0

Contoh program OpenQASM 3.0

Untuk membuat tugas OpenQASM 3.0, Anda dapat memulai dengan program OpenQASM 3.0 sederhana (ghz.qasm) yang menyiapkan status GHZ seperti yang ditunjukkan pada contoh berikut.

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

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

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

c = measure q;
```

Gunakan Python SDK untuk membuat tugas kuantum OpenQASM 3.0

Anda dapat menggunakan Amazon Braket Python SDK untuk mengirimkan program ini ke perangkat Amazon Braket dengan kode berikut.

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

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

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

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

```
)
```

Gunakan Boto3 untuk membuat tugas kuantum OpenQASM 3.0

Anda juga dapat menggunakan AWS Python SDK untuk Braket (Boto3) untuk membuat tugas kuantum menggunakan string OpenQASM 3.0, seperti yang ditunjukkan pada contoh berikut. Cuplikan kode berikut referensi ghz.qasm yang menyiapkan status GHZ seperti yang ditunjukkan di atas.

```
import boto3
import json

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

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

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

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

Gunakan AWS CLI untuk membuat tugas OpenQASM 3.0

AWS Command Line Interface (CLI) juga dapat digunakan untuk mengirimkan program OpenQASM 3.0, seperti yang ditunjukkan pada contoh berikut.

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

Support untuk OpenQASM pada Perangkat Braket yang berbeda

Untuk perangkat yang mendukung OpenQASM 3.0, `action` bidang ini mendukung tindakan baru melalui `GetDevice` respons, seperti yang ditunjukkan pada contoh berikut untuk perangkat danRigetti. IonQ

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

```
}

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

Untuk perangkat yang mendukung kontrol pulsa, `pulse` bidang ditampilkan dalam `GetDevice` respons. Contoh berikut menunjukkan `pulse` bidang ini untuk Rigetti dan OQC perangkat.

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

```
            "optional": false
          }
        ]
      },
      ...
    },
    "ports": {
      "q0_ff": {
        "portId": "q0_ff",
        "direction": "tx",
        "portType": "ff",
        "dt": 1e-9,
        "centerFrequencies": [
          375000000
        ]
      },
      ...
    },
    "supportedFunctions": {
      "shift_phase": {
        "functionName": "shift_phase",
        "arguments": [
          {
            "name": "frame",
            "type": "frame",
            "optional": false
          },
          {
            "name": "phase",
            "type": "float",
            "optional": false
          }
        ]
      },
     ...
    },
    "frames": {
      "q0_q1_cphase_frame": {
        "frameId": "q0_q1_cphase_frame",
        "portId": "q0_ff",
        "frequency": 462475694.24460185,
        "centerFrequency": 375000000,
        "phase": 0,
        "associatedGate": "cphase",
```

```
        "qubitMappings": [
          0,
          1
        ]
      },
      ...
    },
    "supportsLocalPulseElements": false,
    "supportsDynamicFrames": false,
    "supportsNonNativeGatesWithPulses": false,
    "validationParameters": {
      "MAX_SCALE": 4,
      "MAX_AMPLITUDE": 1,
      "PERMITTED_FREQUENCY_DIFFERENCE": 400000000
    }
  }
}

// OQC

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

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

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

Bidang sebelumnya merinci hal-hal berikut:

Pelabuhan:

Menjelaskan port perangkat eksternal (`extern`) yang telah dibuat sebelumnya yang dideklarasikan pada QPU selain properti terkait dari port yang diberikan. Semua port yang tercantum dalam struktur ini dideklarasikan sebelumnya sebagai pengidentifikasi yang valid dalam `OpenQASM 3.0` program yang dikirimkan oleh pengguna. Properti tambahan untuk port meliputi:

- Id port (PortID)
  - Nama port dinyatakan sebagai identifier di OpenQASM 3.0.
- Arah (arah)
  - Arah pelabuhan. Port drive mengirimkan pulsa (arah “tx”), sedangkan port pengukuran menerima pulsa (arah “rx”).

- Jenis port (PortType)
  - Jenis tindakan yang menjadi tanggung jawab port ini (misalnya, drive, capture, atau ff - fast-flux).
- Dt (dt)
  - Waktu dalam detik yang mewakili satu langkah waktu sampel pada port yang diberikan.
- Pemetaan Qubit (QubitMappings)
  - Qubit yang terkait dengan port yang diberikan.
- Frekuensi tengah (CenterFrequencies)
  - Daftar frekuensi pusat terkait untuk semua frame yang telah dideklarasikan atau ditentukan pengguna pada port. Untuk informasi lebih lanjut, lihat Frames.
- Properti Khusus QHP (SpecificPropertiesqhp)
  - Peta opsional yang merinci properti yang ada tentang port khusus untuk QHP.

Bingkai:

Menjelaskan frame eksternal pra-dibuat yang dideklarasikan pada QPU serta properti terkait tentang frame. Semua frame yang tercantum dalam struktur ini telah dideklarasikan sebelumnya sebagai pengidentifikasi yang valid dalam `OpenQASM 3.0` program yang dikirimkan oleh pengguna. Properti tambahan untuk bingkai meliputi:

- Id Bingkai (FrameID)
  - Nama frame dinyatakan sebagai identifier di OpenQASM 3.0.
- Port Id (PortID)
  - Port perangkat keras terkait untuk bingkai.
- Frekuensi (frekuensi)
  - Frekuensi awal default dari frame.
- Frekuensi Pusat (Frekuensi Tengah)
  - Pusat bandwidth frekuensi untuk frame. Biasanya, frame hanya dapat disesuaikan dengan bandwidth tertentu di sekitar frekuensi tengah. Akibatnya, penyesuaian frekuensi harus tetap berada dalam delta tertentu dari frekuensi pusat. Anda dapat menemukan nilai bandwidth dalam parameter validasi.
- Fase (fase)
  - Fase awal default dari frame.
- Gerbang Terkait (AssociatedGate)

    - Gerbang yang terkait dengan bingkai yang diberikan.
- Pemetaan Qubit (QubitMappings)
    - Qubit yang terkait dengan frame yang diberikan.
- Properti Khusus QHP (SpecificPropertiesqhp)
    - Peta opsional yang merinci properti yang ada tentang bingkai khusus untuk QHP.

SupportsDynamicBingkai:

Menjelaskan apakah bingkai dapat dideklarasikan `cal` atau `defcal` diblokir melalui OpenPulse `newframe` fungsi. Jika ini salah, hanya bingkai yang tercantum dalam struktur bingkai yang dapat digunakan dalam program.

SupportedFunctions:

Menjelaskan OpenPulse fungsi yang didukung untuk perangkat selain argumen terkait, tipe argumen, dan tipe pengembalian untuk fungsi yang diberikan. Untuk melihat contoh penggunaan OpenPulse fungsi, lihat OpenPulsespesifikasinya. Pada saat ini, Braket mendukung:

- shift_phase
    - Menggeser fase frame dengan nilai yang ditentukan
- set_phase
    - Mengatur fase frame ke nilai yang ditentukan
- shift_frequency
    - Menggeser frekuensi frame dengan nilai yang ditentukan
- set_frequency
    - Mengatur frekuensi frame ke nilai yang ditentukan
- pementasan
    - Menjadwalkan bentuk gelombang
- menangkap_v0
    - Mengembalikan nilai pada frame capture ke register bit

SupportedQhpTemplateWaveforms:

Menjelaskan fungsi bentuk gelombang yang telah dibuat sebelumnya yang tersedia di perangkat serta argumen serta tipe terkait. Secara default, Braket Pulse menawarkan rutinitas bentuk gelombang pra-bangun pada semua perangkat, yaitu:

Konstan

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

`τ`adalah panjang bentuk gelombang dan `iq` merupakan bilangan kompleks.

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

Gaussian

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

`τ`adalah panjang bentuk gelombang, σ adalah lebar Gaussian, dan `A` merupakan amplitudo. Jika disetel `ZaE` ke`True`, Gaussian diimbangi dan diskalakan ulang sedemikian rupa sehingga sama dengan nol pada awal dan akhir bentuk gelombang, dan mencapai maksimum. `A`

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

DRAG Gaussian

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

`τ`adalah panjang bentuk gelombang, σ adalah lebar gaussian, adalah parameter bebas, dan β `A` merupakan amplitudo. Jika disetel `ZaE` ke`True`, Penghapusan Derivatif oleh Gerbang Adiabatik (DRAG) Gaussian diimbangi dan diskalakan ulang sedemikian rupa sehingga sama dengan nol pada awal dan akhir bentuk gelombang, dan bagian sebenarnya mencapai maksimum. `A` Untuk informasi lebih lanjut tentang bentuk gelombang DRAG, lihat paper Pulsa Sederhana untuk Penghapusan Kebocoran pada Qubit Nonlinier Lemah.

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

SupportsLocalPulseElements:

Menjelaskan apakah elemen pulsa, seperti port, bingkai, dan bentuk gelombang dapat didefinisikan secara lokal dalam blok. `defcal` Jika nilainya`false`, elemen harus didefinisikan dalam `cal` blok.

SupportsNonNativeGatesWithPulses:

Menjelaskan apakah kita dapat atau tidak dapat menggunakan gerbang non-asli dalam kombinasi dengan program pulsa. Misalnya, kita tidak dapat menggunakan gerbang non-asli seperti H gerbang dalam program tanpa terlebih dahulu mendefinisikan gerbang `defcal` untuk qubit yang digunakan. Anda dapat menemukan daftar `nativeGateSet` kunci gerbang asli di bawah kemampuan perangkat.

ValidationParameters:

Menjelaskan batas validasi elemen pulsa, termasuk:

- Skala Maksimum/Nilai Amplitudo Maksimum untuk bentuk gelombang (arbitrer dan pra-bangun)
- Bandwidth frekuensi maksimum dari frekuensi pusat yang disediakan dalam Hz
- Panjang/durasi pulsa minimum dalam hitungan detik
- Panjang pulsa maksimum/durasi dalam hitungan detik

Operasi, Hasil, dan Jenis Hasil yang Didukung dengan OpenQASM

Untuk mengetahui fitur OpenQASM 3.0 mana yang didukung setiap perangkat, Anda dapat merujuk ke `braket.ir.openqasm.program` kunci di `action` bidang pada output kemampuan perangkat. Misalnya, berikut ini adalah operasi yang didukung dan jenis hasil yang tersedia untuk simulator SV1 Braket State Vector.

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

```
        "cphaseshift",
        "cphaseshift00",
        "cphaseshift01",
        "cphaseshift10",
        "cswap",
        "cy",
        "cz",
        "h",
        "i",
        "iswap",
        "pswap",
        "phaseshift",
        "rx",
        "ry",
        "rz",
        "s",
        "si",
        "swap",
        "t",
        "ti",
        "v",
        "vi",
        "x",
        "xx",
        "xy",
        "y",
        "yy",
        "z",
        "zz"
      ],
      "supportedPragmas": [
        "braket_unitary_matrix"
      ],
      "forbiddenPragmas": [],
      "maximumQubitArrays": 1,
      "maximumClassicalArrays": 1,
      "forbiddenArrayOperations": [
        "concatenation",
        "negativeIndex",
        "range",
        "rangeWithStep",
        "slicing",
        "selection"
      ],
```

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

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

Simulasikan kebisingan dengan OpenQASM 3.0

Untuk mensimulasikan noise dengan OpenQASM3, Anda menggunakan instruksi pragma untuk menambahkan operator noise. Misalnya, untuk mensimulasikan versi bising dari program GHZ yang disediakan sebelumnya, Anda dapat mengirimkan program OpenQASM berikut.

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

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

h q[0];
#pragma braket noise depolarizing(0.75) q[0] cnot q[0], q[1];
#pragma braket noise depolarizing(0.75) q[0]
#pragma braket noise depolarizing(0.75) q[1] cnot q[1], q[2];
#pragma braket noise depolarizing(0.75) q[0]
#pragma braket noise depolarizing(0.75) q[1]

c = measure q;
```

Spesifikasi untuk semua operator kebisingan pragma yang didukung disediakan dalam daftar berikut.

```
#pragma braket noise bit_flip(<float in [0,1/2]>) <qubit>
#pragma braket noise phase_flip(<float in [0,1/2]>) <qubit>
#pragma braket noise pauli_channel(<float>, <float>, <float>)  <qubit>
#pragma braket noise depolarizing(<float in [0,3/4]>) <qubit>
#pragma braket noise two_qubit_depolarizing(<float in [0,15/16]>) <qubit>, <qubit>
#pragma braket noise two_qubit_dephasing(<float in [0,3/4]>) <qubit>, <qubit>
#pragma braket noise amplitude_damping(<float in [0,1]>) <qubit>
#pragma braket noise generalized_amplitude_damping(<float in [0,1]> <float in [0,1]>)
 <qubit>
#pragma braket noise phase_damping(<float in [0,1]>) <qubit>
#pragma braket noise kraus([[<complex m0_00>, ], ...], [[<complex m1_00>, ], ...], ...)
 <qubit>[, <qubit>]     // maximum of 2 qubits and maximum of 4 matrices for 1 qubit,
 16 for 2
```

Operator Kraus

Untuk menghasilkan operator Kraus, Anda dapat mengulangi melalui daftar matriks, mencetak setiap elemen matriks sebagai ekspresi kompleks.

Saat menggunakan operator Kraus, ingat hal berikut:

- Jumlah tidak qubits boleh melebihi 2. Definisi saat ini dalam skema menetapkan batas ini.
- Panjang daftar argumen harus kelipatan 8. Ini berarti harus terdiri hanya dari matriks 2x2.
- Panjang total tidak melebihi 2 matriks $^{num\_qubits\ 2^*}$. Ini berarti 4 matriks untuk 1 qubit dan 16 untuk 2. qubits
- Semua matriks yang disediakan benar-benar pelestarian jejak positif (CPTP).
- Produk operator Kraus dengan konjugat transpos mereka perlu ditambahkan ke matriks identitas.

Qubitrewiring dengan OpenQASM 3.0

AmazonBraket mendukung qubit notasi fisik dalam OpenQASM pada Rigetti perangkat (untuk mempelajari lebih lanjut lihat halaman ini). Saat menggunakan fisik qubits dengan strategi rewiring naif, pastikan qubits terhubung pada perangkat yang dipilih. Atau, jika qubit register digunakan sebagai gantinya, strategi rewiring PARAL diaktifkan secara default pada Rigetti perangkat.

```
// ghz.qasm
// Prepare a GHZ state
```

```
OPENQASM 3;

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

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

Kompilasi Verbatim dengan OpenQASM 3.0

Ketika Anda menjalankan sirkuit kuantum pada komputer kuantum dariRigetti,OQC, danIonQ, Anda dapat mengarahkan kompiler untuk menjalankan sirkuit Anda persis seperti yang ditentukan, tanpa modifikasi apa pun. Fitur ini dikenal sebagai kompilasi verbatim. Dengan perangkat Rigetti, Anda dapat menentukan dengan tepat apa yang akan dipertahankan—baik seluruh rangkaian atau hanya bagian tertentu saja. Untuk melestarikan hanya bagian tertentu dari sirkuit, Anda harus menggunakan gerbang asli di wilayah yang diawetkan. Saat ini, IonQ dan OQC hanya mendukung kompilasi kata demi kata untuk seluruh rangkaian, sehingga setiap instruksi di sirkuit perlu dilampirkan dalam kotak kata demi kata.

Dengan OpenQASM, Anda dapat menentukan pragma kata demi kata di sekitar kotak kode yang tidak tersentuh dan tidak dioptimalkan oleh rutin kompilasi tingkat rendah perangkat keras. Contoh kode berikut menunjukkan bagaimana menggunakan`#pragma braket verbatim`.

```
OPENQASM 3;

bit[2] c;

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

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

Untuk informasi lebih lanjut tentang kompilasi kata demi kata, lihat buku catatan contoh kompilasi Verbatim.

Konsol Braket

Tugas OpenQASM 3.0 tersedia dan dapat dikelola dalam konsol Braket. Amazon Di konsol, Anda memiliki pengalaman yang sama mengirimkan tugas kuantum di OpenQASM 3.0 seperti yang Anda kirimkan tugas kuantum yang ada.

Sumber daya lainnya

OpenQASM tersedia di semua Amazon Wilayah Braket.

Untuk contoh notebook untuk memulai dengan OpenQASM di Amazon Braket, lihat Tutorial Braket. GitHub

Gradien komputasi dengan OpenQASM 3.0

Amazon Braket mendukung gradien komputasi pada simulator on-demand dan lokal dalam mode `shots=0` (tepat) menggunakan metode diferensiasi adjoint. Anda dapat memberikan pragma yang sesuai untuk menentukan gradien yang ingin Anda hitung seperti yang ditunjukkan pada contoh berikut.

```
OPENQASM 3.0;
input float alpha;

bit[2] b;
qubit[2] q;

h q[0];
h q[1];
rx(alpha) q[0];
rx(alpha) q[1];
b[0] = measure q[0];
b[1] = measure q[1];

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

Alih-alih mencantumkan semua parameter satu per satu, Anda juga dapat menentukan `all` dalam pragma. Ini menghitung gradien sehubungan dengan semua `input` parameter yang tercantum. Ini bisa nyaman bila jumlah parameternya sangat besar. Dalam hal ini, pragma akan terlihat seperti contoh berikut.

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

Semua jenis yang dapat diamati didukung, termasuk operator individu, produk tensor, observable Hermitian, dan. `Sum` Operator yang ingin Anda gunakan untuk menghitung gradien harus dibungkus dalam `expectation()` enkapsulator dan qubit yang digunakan setiap suku harus ditentukan.

Mengukur qubit tertentu dengan OpenQASM 3.0

Simulator vektor keadaan lokal dan simulator matriks kepadatan lokal mendukung pengiriman OpenQASM program di mana subset qubit sirkuit dapat diukur. Ini sering disebut pengukuran paral. Misalnya, dalam kode berikut Anda dapat membuat sirkuit dua-qubit dan hanya mengukur qubit pertama.

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

Ada dua qubit, `q[0]` dan kami `q[1]` hanya mengukur qubit 0 di sini:. `b[0] = measure q[0]` Sekarang, jalankan yang berikut ini pada simulator vektor negara lokal.

```
from braket.devices import LocalSimulator

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

Anda dapat memeriksa apakah perangkat mendukung pengukuran sebagian dengan memeriksa `requiresAllQubitsMeasurement` bidang di properti tindakannya; jika ya`False`, maka pengukuran paral didukung.

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

Di sini, `requiresAllQubitsMeasurement` adalah`False`, yang menunjukkan bahwa tidak semua qubit harus diukur.

Kirim program analog menggunakan QuEra Aquila

Halaman ini memberikan dokumentasi komprehensif tentang kemampuan Aquila mesin dariQuEra. Detail yang dibahas di sini adalah sebagai berikut: 1) Hamiltonian berparameter yang disimulasikan olehAquila, 2) parameter program AHS, 3) konten hasil AHS, 4) parameter kemampuan. Aquila Kami menyarankan menggunakan pencarian teks Ctrl+F untuk menemukan parameter yang relevan dengan pertanyaan Anda.

Hamiltonian

AquilaMesin dari QuEra mensimulasikan Hamiltonian berikut (tergantung waktu) secara asli:

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

> **Note**
>
> Akses ke detuning lokal adalah kemampuan Eksperimental dan tersedia berdasarkan permintaan melalui Braket Direct.

di mana

- $H_{\text{drive},k}(t) = (\frac{1}{2}\Omega(t)\, e^{i\varphi(t)}\, S_{-,k} + \frac{1}{2}\Omega(t)\, e^{-i\varphi(t)}\, S_{+,k}) - (\Delta_{\text{global}}(t)\, n_k)$,
  - $\Omega(t)$ adalah amplitudo penggerak global yang bergantung pada waktu (alias frekuensi Rabi), dalam satuan (rad/ s)
  - $\varphi(t)$ adalah fase global yang bergantung pada waktu, diukur dalam radian
  - $S_{-,k}$ dan $S_{+,k}$ adalah operator penurun dan peningkatan putaran atom k (dalam basis $|\downarrow\rangle = |g\rangle$, $|\uparrow\rangle = |r\rangle$, mereka adalah $S_+ = |g\rangle\langle r|$, $S_- = (S_+)^\dagger = |r\rangle\langle g|$)
  - $\Delta_{\text{global}}(t)$ adalah detuning global yang bergantung pada waktu
  - n_k adalah operator proyeksi pada keadaan Rydberg atom k (yaitu $n = |r\rangle\langle r|$)
- $H_{\text{local detuning},k}(t) = -\Delta_{\text{local}}(t)\, h\, n_k$
  - $\Delta_{\text{local}}(t)$ adalah faktor yang bergantung pada waktu dari pergeseran frekuensi lokal, dalam satuan (rad/s)

  - h_k adalah faktor yang bergantung pada lokasi, bilangan tak berdimensi antara 0,0 dan 1,0
- $V_{vdw,k,l} = C_6/(d_{k,l})^6 n_k n_l$,
  - C_6 adalah koefisien van der Waals, dalam satuan (rad/s) * (m) ^6
  - $d_{k,l}$ adalah jarak Euclidean antara atom k dan l, diukur dalam meter.

Pengguna memiliki kontrol atas parameter berikut melalui skema program Braket AHS.

- Susunan atom 2-d ($_{kk}$koordinat x dan y dari setiap atom k, dalam satuan um), yang mengontrol jarak atom berpasangan $d_{k,l}$ dengan k, l=1,2,... N
- Ω (t), frekuensi Rabi global yang bergantung pada waktu, dalam satuan (rad/s)
- φ (t), fase global yang bergantung pada waktu, dalam satuan (rad)
- Δ_{global} (t), detuning global yang bergantung pada waktu, dalam satuan (rad/s)
- Δ_{local} (t), faktor yang bergantung pada waktu (global) dari besarnya detuning lokal, dalam satuan (rad/s)
- h_k, faktor yang bergantung pada lokasi (statis) dari besarnya detuning lokal, bilangan tak berdimensi antara 0,0 dan 1,0

> **Note**
>
> Pengguna tidak dapat mengontrol level mana yang terlibat (yaitu S_-, S_+, n operator tetap) atau kekuatan koefisien interaksi Rydberg-Rydberg (C). $_6$

Skema program Braket AHS

Braket.ir.ahs.program_v1.Program objek (contoh)

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

```
                [Decimal('0'), Decimal('4e-6')],
                [Decimal('4e-6'), Decimal('0')],
            ],
            filling=[1, 1, 1]
        )
    ),
    hamiltonian=Hamiltonian(
        drivingFields=[
            DrivingField(
                amplitude=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('15700000.0'),
Decimal('15700000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
                    ),
                    pattern='uniform'
                ),
                phase=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001')]
                    ),
                    pattern='uniform'
                ),
                detuning=PhysicalField(
                    time_series=TimeSeries(
                        values=[Decimal('-54000000.0'), Decimal('54000000.0')],
                        times=[Decimal('0'), Decimal('0.000001')]
                    ),
                    pattern='uniform'
                )
            )
        ],
        localDetuning=[
            LocalDetuning(
                magnitude=PhysicalField(
                    times_series=TimeSeries(
                        values=[Decimal('0'), Decimal('25000000.0'),
Decimal('25000000.0'), Decimal('0')],
                        times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
                    ),
                    pattern=Pattern([Decimal('0.8'), Decimal('1.0'), Decimal('0.9')])
```

```
                )
            )
        ]
    )
)
```

JSON (contoh)

```
{
    "braketSchemaHeader": {
        "name": "braket.ir.ahs.program",
        "version": "1"
    },
    "setup": {
        "ahs_register": {
            "sites": [
                [0E-7, 0E-7],
                [0E-7, 4E-6],
                [4E-6, 0E-7],
            ],
            "filling": [1, 1, 1]
        }
    },
    "hamiltonian": {
        "drivingFields": [
            {
                "amplitude": {
                    "time_series": {
                        "values": [0.0, 15700000.0, 15700000.0, 0.0],
                        "times": [0E-9, 0.000001000, 0.000002000, 0.000003000]
                    },
                    "pattern": "uniform"
                },
                "phase": {
                    "time_series": {
                        "values": [0E-7, 0E-7],
                        "times": [0E-9, 0.000001000]
                    },
                    "pattern": "uniform"
                },
                "detuning": {
                    "time_series": {
                        "values": [-54000000.0, 54000000.0],
```

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

Bidang utama

Bidang program	jenis	deskripsi
setup.ahs_register.sites	Daftar [Daftar [Desimal]]	Daftar koordinat 2-d di mana pinset menjebak atom
setup.ahs_register.filling	Daftar [int]	Menandai atom yang menempati situs perangkap dengan 1, dan situs kosong dengan 0
Hamiltonian.drivingFields [] .amplitude.time_series.times	Daftar [Desimal]	titik waktu amplitudo mengemudi, Omega (t)

Bidang program	jenis	deskripsi
Hamiltonian.drivingFields [] .amplitude.time_series.values	Daftar [Desimal]	nilai amplitudo mengemudi, Omega (t)
Hamiltonian.drivingFields [] .amplitude.pattern	str	pola spasial amplitudo mengemudi, Omega (t); harus 'seragam'
Hamiltonian.drivingFields [] .phase.time_series.times	Daftar [Desimal]	titik waktu fase mengemudi, phi (t)
Hamiltonian.drivingFields [] .phase.time_series.values	Daftar [Desimal]	nilai fase mengemudi, phi (t)
Hamiltonian.drivingFields [] .phase.pattern	str	pola spasial fase mengemudi, phi (t); harus 'seragam'
Hamiltonian.drivingFields [] .detuning.time_series.times	Daftar [Desimal]	titik waktu detuning mengemudi, Delta_global (t)
Hamiltonian.drivingFields [] .detuning.time_series.values	Daftar [Desimal]	nilai detuning mengemudi, delta_global (t)
Hamiltonian.drivingFields [] .detuning.pattern	str	pola spasial detuning mengemudi, Delta_global (t); harus 'seragam'

Bidang program	jenis	deskripsi
Hamiltonian.localdeTuning [] .magnitude.time_series.times	Daftar [Desimal]	titik waktu dari faktor yang bergantung pada waktu dari magnitudo detuning lokal, delta_local (t)
Hamiltonian.localdeTuning [] .magnitude.time_series.values	Daftar [Desimal]	nilai faktor yang bergantung pada waktu dari besaran detuning lokal, delta_local (t)
Hamiltonian.localdeTuning [] .magnitude.pattern	Daftar [Desimal]	faktor yang bergantung pada situs dari besaran detuning lokal, h_k (nilai sesuai dengan situs di setup.ahs_register .sites)

Kolom metadata

Bidang program	jenis	deskripsi
braket SchemaHeader .name	str	nama skema; harus 'braket.ir.ahs.program'
braket SchemaHeader .version	str	versi skema

Skema hasil tugas Braket AHS

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.
AnalogHamiltonianSimulationQuantumTaskResult(contoh)

```
AnalogHamiltonianSimulationQuantumTaskResult(
    task_metadata=TaskMetadata(
        braketSchemaHeader=BraketSchemaHeader(
            name='braket.task_result.task_metadata',
            version='1'
        ),
        id='arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef',
        shots=2,
        deviceId='arn:aws:braket:us-east-1::device/qpu/quera/Aquila',
        deviceParameters=None,
        createdAt='2022-10-25T20:59:10.788Z',
        endedAt='2022-10-25T21:00:58.218Z',
        status='COMPLETED',
        failureReason=None
    ),
    measurements=[
        ShotResult(
            status=<AnalogHamiltonianSimulationShotStatus.SUCCESS: 'Success'>,

            pre_sequence=array([1, 1, 1, 1]),
            post_sequence=array([0, 1, 1, 1])
        ),

        ShotResult(
            status=<AnalogHamiltonianSimulationShotStatus.SUCCESS: 'Success'>,

            pre_sequence=array([1, 1, 0, 1]),
            post_sequence=array([1, 0, 0, 0])
        )
    ]
)
```

JSON (contoh)

```
{
    "braketSchemaHeader": {
        "name": "braket.task_result.analog_hamiltonian_simulation_task_result",
```

```
        "version": "1"
    },
    "taskMetadata": {
        "braketSchemaHeader": {
            "name": "braket.task_result.task_metadata",
            "version": "1"
        },
        "id": "arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef",
        "shots": 2,
        "deviceId": "arn:aws:braket:us-east-1::device/qpu/quera/Aquila",

        "createdAt": "2022-10-25T20:59:10.788Z",
        "endedAt": "2022-10-25T21:00:58.218Z",
        "status": "COMPLETED"

    },
    "measurements": [
        {
            "shotMetadata": {"shotStatus": "Success"},
            "shotResult": {
                "preSequence": [1, 1, 1, 1],
                "postSequence": [0, 1, 1, 1]
            }
        },
        {
            "shotMetadata": {"shotStatus": "Success"},
            "shotResult": {
                "preSequence": [1, 1, 0, 1],
                "postSequence": [1, 0, 0, 0]
            }
        }
    ],
    "additionalMetadata": {
        "action": {...}
        "queraMetadata": {
            "braketSchemaHeader": {
                "name": "braket.task_result.quera_metadata",
                "version": "1"
            },
            "numSuccessfulShots": 100
        }
    }
```

```
}
```

Bidang utama

Bidang hasil tugas	jenis	deskripsi
pengukuran [] .shotResult.Presequence	Daftar [int]	Bit pengukuran pra-urutan (satu untuk setiap situs atom) untuk setiap bidikan: 0 jika situs kosong, 1 jika situs diisi, diukur sebelum urutan pulsa yang menjalankan evolusi kuantum
pengukuran [] .shotResult.PostSequence	Daftar [int]	Bit pengukuran pasca-urutan untuk setiap bidikan: 0 jika atom dalam keadaan Rydberg atau situs kosong, 1 jika atom dalam keadaan dasar, diukur pada akhir urutan pulsa yang menjalankan evolusi kuantum

Kolom metadata

Bidang hasil tugas	jenis	deskripsi
braket SchemaHeader .name	str	nama skema; harus 'braket.task_result.analog_hamiltonian_simulation_task_result'
braket SchemaHeader .version	str	versi skema
SchemaHeaderTaskMetadata.braket .name	str	nama skema; harus 'braket.t

Bidang hasil tugas	jenis	deskripsi
		ask_resul t.task_me tadata'
SchemaHeaderTaskMetadata.br aket .version	str	versi skema
Taskmetadata.id	str	ID tugas kuantum. Untuk tugas AWS kuantum, ini adalah tugas kuantum ARN.
Taskmetadata.shots	int	Jumlah bidikan untuk tugas kuantum
Taskmetadata.shots.deviceID	str	ID perangkat tempat tugas kuantum dijalankan. Untuk AWS perangkat , ini adalah perangkat ARN.

Bidang hasil tugas	jenis	deskripsi
taskmetadata.shots.createdat	str	Stempel waktu pembuatan ; formatnya harus dalam format string ISO-8601/ RFC3339 YYYY-MM-D DTHH:MM: SS.sssz. Default adalah None.
taskmetadata.shots.endedat	str	Stempel waktu kapan tugas kuantum berakhir; formatnya harus dalam format string ISO-8601/ RFC3339 YYYY-MM-D DTHH:MM: SS.sssz. Default adalah None.

Bidang hasil tugas	jenis	deskripsi
Taskmetadata.shots.status	str	Status tugas kuantum (CREATED, QUEUED, RUNNING, COMPLETED, FAILED). Default adalah None.
taskmetadata.shots.failurereason	str	Alasan kegagalan tugas kuantum. Default adalah None.
TambahanMetadata.Action	Braket.ir.ahs.program_v1.program	(Lihat bagian skema program Braket AHS)
SchemaHeaderAdditionalMetadata.action.braket .querametadata.name	str	nama skema; harus 'braket.task_result.quera_metadata'
SchemaHeaderAdditionalMetadata.action.braket .querametadata.version	str	versi skema

Bidang hasil tugas	jenis	deskripsi
TambahanMetadata.action.num SuccessfulShots	int	jumlah tembakan yang benar-benar berhasil; harus sama dengan jumlah tembakan yang diminta
pengukuran [] .shotmetadata.shotStatus	int	Status tembakan, (Sukses, Sukses sebagian, Kegagalan); harus “Sukses”

QuEra skema properti perangkat

braket.device_schema.quera.quera_device_capabilities_v1. QueraDeviceKemampuan (contoh)

```
QueraDeviceCapabilities(
    service=DeviceServiceProperties(
        braketSchemaHeader=BraketSchemaHeader(
            name='braket.device_schema.device_service_properties',
            version='1'
            ),
            executionWindows=[
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.MONDAY: 'Monday'>,
                    windowStartHour=datetime.time(1, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.TUESDAY: 'Tuesday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
```

```
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.WEDNESDAY: 'Wednesday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.FRIDAY: 'Friday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.SATURDAY: 'Saturday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(23, 59, 59)
                ),
                DeviceExecutionWindow(
                    executionDay=<ExecutionDay.SUNDAY: 'Sunday'>,
                    windowStartHour=datetime.time(0, 0),
                    windowEndHour=datetime.time(12, 0)
                )
            ],
            shotsRange=(1, 1000),
            deviceCost=DeviceCost(
                price=0.01,
                unit='shot'
            ),
            deviceDocumentation=
                DeviceDocumentation(
                    imageUrl='https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png',
                    summary='Analog quantum processor based on neutral atom arrays',
                    externalDocumentationUrl='https://www.quera.com/aquila'
                ),
                deviceLocation='Boston, USA',
                updatedAt=datetime.datetime(2024, 1, 22, 12, 0,
 tzinfo=datetime.timezone.utc),
                getTaskPollIntervalMillis=None
    ),
    action={
        <DeviceActionType.AHS: 'braket.ir.ahs.program'>: DeviceActionProperties(
                version=['1'],
                actionType=<DeviceActionType.AHS: 'braket.ir.ahs.program'>
```

```
            )
    },
    deviceParameters={},
    braketSchemaHeader=BraketSchemaHeader(
        name='braket.device_schema.quera.quera_device_capabilities',
        version='1'
    ),
    paradigm=QueraAhsParadigmProperties(
        ...
        # See https://github.com/amazon-braket/amazon-braket-schemas-python/blob/main/
src/braket/device_schema/quera/quera_ahs_paradigm_properties_v1.py
        ...
    )
)
```

JSON (contoh)

```
{
    "service": {
        "braketSchemaHeader": {
            "name": "braket.device_schema.device_service_properties",
            "version": "1"
        },
        "executionWindows": [
            {
                "executionDay": "Monday",
                "windowStartHour": "01:00:00",
                "windowEndHour": "23:59:59"
            },
            {
                "executionDay": "Tuesday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            },
            {
                "executionDay": "Wednesday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            },
            {
                "executionDay": "Friday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "23:59:59"
```

```
            },
            {
                "executionDay": "Saturday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "23:59:59"
            },
            {
                "executionDay": "Sunday",
                "windowStartHour": "00:00:00",
                "windowEndHour": "12:00:00"
            }
        ],
        "shotsRange": [
            1,
            1000
        ],
        "deviceCost": {
            "price": 0.01,
            "unit": "shot"
        },
        "deviceDocumentation": {
            "imageUrl": "https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png",
            "summary": "Analog quantum processor based on neutral atom arrays",
            "externalDocumentationUrl": "https://www.quera.com/aquila"
        },
        "deviceLocation": "Boston, USA",
        "updatedAt": "2024-01-22T12:00:00+00:00"
    },
    "action": {
        "braket.ir.ahs.program": {
            "version": [
                "1"
            ],
            "actionType": "braket.ir.ahs.program"
        }
    },
    "deviceParameters": {},
    "braketSchemaHeader": {
        "name": "braket.device_schema.quera.quera_device_capabilities",
        "version": "1"
    },
    "paradigm": {
```

```
        ...
        # See Aquila device page > "Calibration" tab > "JSON" page
        ...
    }
}
```

Bidang properti layanan

Bidang properti layanan	jenis	deskripsi
Service.ExecutionWindows [] .ExecutionDay	ExecutionDay	Hari dari jendela eksekusi; harus 'Setiap Hari', 'Hari Kerja', 'Akhir Pekan', 'Senin', 'Selasa', 'Rabu', Kamis', 'Jumat', 'Sabtu' atau 'Minggu'
Service.ExecutionWindows [] .window StartHour	datetime.time	Format 24 jam UTC saat jendela eksekusi dimulai
Service.ExecutionWindows [] .window EndHour	datetime.time	Format 24 jam UTC saat jendela eksekusi berakhir
Service.qpu_capabilities.service.shotsrange	Tupel [int, int]	Jumlah bidikan minimum dan maksimum untuk perangkat
Service.qpu_capabilities.service.devicecost.price	float	Harga perangkat dalam hal dolar AS
Service.qpu_capabilities.service.devicecost.unit	str	unit untuk mengisi harga, misalnya: 'menit', 'jam', 'tembakan', 'tugas'

Kolom metadata

Bidang metadata	jenis	deskripsi
aksi [] .versi	str	versi skema program AHS
tindakan [] .actionType	ActionType	Nama skema program AHS; harus 'braket.ir.ahs.program'
service.braket .name SchemaHeader	str	nama skema; harus 'braket.device_schema.device_service_properties'
service.braket .version SchemaHeader	str	versi skema
Service.deviceDocumentation.ImageUrl	str	URL untuk gambar perangkat
Service.deviceDocumentation.Summary	str	deskripsi singkat tentang perangkat
Service.deviceDocumentation.External DocumentationUrl	str	URL dokumentasi eksternal
Service.deviceLocation	str	lokasi geografis untuk perangkat
Service.updateDat	datetime	waktu ketika properti perangkat terakhir diperbarui

Bekerja dengan Boto3

Boto3 adalah AWS SDK untuk Python. Dengan Boto3, pengembang Python dapat membuat, mengkonfigurasi, dan mengelola Layanan AWS, seperti Braket. Amazon Boto3 menyediakan akses berorientasi objekAPI, serta tingkat rendah ke Braket. Amazon

Ikuti instruksi di Panduan Memulai Cepat Boto3 untuk mempelajari cara menginstal dan mengkonfigurasi Boto3.

Boto3 menyediakan fungsionalitas inti yang bekerja bersama dengan Amazon Braket Python SDK untuk membantu Anda mengonfigurasi dan menjalankan tugas kuantum Anda. Pelanggan Python selalu perlu menginstal Boto3, karena itu adalah implementasi inti. Jika Anda ingin menggunakan metode pembantu tambahan, Anda juga perlu menginstal Amazon Braket SDK.

Misalnya, saat Anda meneleponCreateQuantumTask, Amazon Braket SDK mengirimkan permintaan ke Boto3, yang kemudian memanggil file. AWS API

Di bagian ini:

- Nyalakan klien Amazon Braket Boto3
- Konfigurasikan AWS CLI profil untuk Boto3 dan Amazon Braket SDK

Nyalakan klien Amazon Braket Boto3

Untuk menggunakan Boto3 dengan Amazon Braket, Anda harus mengimpor Boto3 dan kemudian menentukan klien yang Anda gunakan untuk terhubung ke Braket. Amazon API Dalam contoh berikut, klien Boto3 diberi nama. braket

> **Note**
>
> Untuk kompatibilitas mundur dengan versi lama BraketSchemas, informasi OpenQASM dihilangkan dari panggilan. GetDevice API Untuk mendapatkan informasi ini, agen pengguna perlu menyajikan versi terbaru dari BraketSchemas (1.8.0 atau yang lebih baru). Braket SDK secara otomatis melaporkan ini untuk Anda. Jika Anda tidak melihat hasil OpenQASM dalam GetDevice respons saat menggunakan Braket SDK, Anda mungkin perlu mengatur variabel AWS_EXECUTION_ENV lingkungan untuk mengonfigurasi agen pengguna. Lihat contoh kode yang disediakan dalam topik kesalahan hasil OpenQASM GetDevice tidak mengembalikan OpenQASM untuk cara melakukannya untuk AWS CLI, Boto3, dan Go, Java, dan/SDK. JavaScript TypeScript

```
import boto3
import botocore
```

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

Sekarang setelah Anda memiliki `braket` klien yang didirikan, Anda dapat membuat permintaan dan memproses tanggapan dari layanan Amazon Braket. Anda bisa mendapatkan detail lebih lanjut tentang permintaan dan data tanggapan di Referensi API.

Contoh berikut menunjukkan cara bekerja dengan perangkat dan tugas kuantum.

- Mencari perangkat
- Ambil perangkat
- Membuat tugas kuantum
- Mengambil tugas kuantum
- Mencari tugas kuantum
- Batalkan tugas kuantum

Mencari perangkat

- `search_devices(**kwargs)`

Cari perangkat menggunakan filter yang ditentukan.

```
# 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'])
```

Ambil perangkat

- `get_device(deviceArn)`

Ambil perangkat yang tersedia di Amazon Braket.

```
# Pass the device ARN when sending the request and capture the repsonse
response = braket.get_device(deviceArn='arn:aws:braket:::device/quantum-simulator/
amazon/sv1')

print(f"Device {response['deviceName']} is {response['deviceStatus']}")
```

Membuat tugas kuantum

- `create_quantum_task(**kwargs)`

Buat tugas kuantum.

```
# 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")
```

Mengambil tugas kuantum

- `get_quantum_task(quantumTaskArn)`

Ambil tugas kuantum yang ditentukan.

```
# 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'])
```

Mencari tugas kuantum

- `search_quantum_tasks(**kwargs)`

Cari tugas kuantum yang cocok dengan nilai filter yang ditentukan.

```
# Pass search filters and optional parameters when sending the
#  request and capture the response
response = braket.search_quantum_tasks(filters=[{
    'name': 'deviceArn',
    'operator': 'EQUAL',
    'values': ['arn:aws:braket:::device/quantum-simulator/amazon/sv1']
}], maxResults=25)

print(f"Found {len(response['quantumTasks'])} quantum tasks")

for n in range(len(response['quantumTasks'])):
    task = response['quantumTasks'][n]
    print(f"Quantum task {task['quantumTaskArn']} for {task['deviceArn']} is
 {task['status']}")
```

Batalkan tugas kuantum

- `cancel_quantum_task(quantumTaskArn)`

Batalkan tugas kuantum yang ditentukan.

```
# Pass the quantum task ARN when sending the request and capture the response
```

```
response = braket.cancel_quantum_task(quantumTaskArn='arn:aws:braket:us-
west-1:123456789012:quantum-task/ce78c429-cef5-45f2-88da-123456789012')

print(f"Quantum task {response['quantumTaskArn']} is {response['cancellationStatus']}")
```

Konfigurasikan AWS CLI profil untuk Boto3 dan Amazon Braket SDK

AmazonBraket SDK bergantung pada AWS CLI kredensi default, kecuali jika Anda secara eksplisit menentukan sebaliknya. Sebaiknya Anda menyimpan default saat menjalankan notebook Amazon Braket terkelola karena Anda harus memberikan peran IAM yang memiliki izin untuk meluncurkan instance notebook.

Secara opsional, jika Anda menjalankan kode secara lokal (misalnya pada instans Amazon EC2), Anda dapat membuat profil bernama. AWS CLI Anda dapat memberikan setiap profil set izin yang berbeda, dan bukan secara teratur menimpa profil default.

Bagian ini memberikan penjelasan singkat tentang cara mengkonfigurasi CLI tersebut `profile` dan bagaimana memasukkan profil itu ke dalam Amazon Braket sehingga API panggilan dilakukan dengan izin dari profil itu.

Di bagian ini:

- Langkah 1: Konfigurasikan lokal AWS CLIprofile
- Langkah 2: Buat objek sesi Boto3
- Langkah 3: Masukkan sesi Boto3 ke dalam Braket AwsSession

Langkah 1: Konfigurasikan lokal AWS CLI**`profile`**

Ini di luar cakupan dokumen ini untuk menjelaskan cara membuat pengguna dan cara mengkonfigurasi profil non-default. Untuk informasi lebih lanjut tentang topik ini, lihat:

- Memulai
- Mengkonfigurasi yang akan AWS CLI digunakan AWS IAM Identity Center

Untuk menggunakan Amazon Braket, Anda harus memberikan pengguna ini - dan `profile` CLI terkait - dengan izin Braket yang diperlukan. Misalnya, Anda dapat melampirkan AmazonBraketFullAccesskebijakan.

Langkah 2: Buat objek sesi Boto3

Untuk membuat objek sesi Boto3, gunakan contoh kode berikut.

```
from boto3 import Session

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

> **Note**
>
> Jika API panggilan yang diharapkan memiliki batasan berbasis Region yang tidak selaras dengan Region `profile` default, Anda dapat menentukan Region untuk sesi Boto3 seperti yang ditunjukkan pada contoh berikut.

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

Untuk argumen yang ditunjuk sebagai`region`, gantikan nilai yang sesuai dengan salah satu Wilayah AWS di mana Amazon Braket tersedia seperti`us-east-1`,`us-west-1`, dan sebagainya.

Langkah 3: Masukkan sesi Boto3 ke dalam Braket AwsSession

Contoh berikut menunjukkan cara menginisialisasi sesi Boto3 Braket dan membuat instance perangkat dalam sesi itu.

```
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)
```

Setelah penyiapan ini selesai, Anda dapat mengirimkan tugas kuantum ke `AwsDevice` objek yang dipakai (dengan memanggil `device.run(…)` perintah misalnya). Semua API panggilan yang dilakukan oleh perangkat tersebut dapat memanfaatkan kredenal IAM yang terkait dengan profil CLI yang sebelumnya Anda tetapkan sebagai. `profile`

Kontrol pulsa pada Amazon Braket

Bagian ini menjelaskan cara menggunakan kontrol pulsa pada berbagai QPU di Amazon Braket.

Di bagian ini:

Braket Pulse

Pulsa adalah sinyal analog yang mengontrol qubit di komputer kuantum. Dengan perangkat tertentu di Amazon Braket, Anda dapat mengakses fitur kontrol pulsa untuk mengirimkan sirkuit menggunakan pulsa. Anda dapat mengakses kontrol pulsa melalui Braket SDK, menggunakan OpenQASM 3.0, atau langsung melalui Braket API. Pertama, mari kita perkenalkan beberapa konsep kunci untuk kontrol pulsa di Braket.

Bingkai

Bingkai adalah abstraksi perangkat lunak yang bertindak sebagai jam dalam program kuantum dan fase. Waktu jam bertambah pada setiap penggunaan dan sinyal pembawa stateful yang ditentukan oleh frekuensi. Saat mentransmisikan sinyal ke qubit, bingkai menentukan frekuensi pembawa qubit, offset fase, dan waktu di mana amplop bentuk gelombang dipancarkan. Dalam Braket Pulse, membangun frame tergantung pada perangkat, frekuensi, dan fase. Bergantung pada perangkat, Anda dapat memilih bingkai yang telah ditentukan atau membuat instance frame baru dengan menyediakan port.

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

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

Port

Port adalah abstraksi perangkat lunak yang mewakili setiap komponen perangkat keras input/output yang mengendalikan qubit. Ini membantu vendor perangkat keras menyediakan antarmuka yang dengannya pengguna dapat berinteraksi untuk memanipulasi dan mengamati qubit. Port dicirikan oleh string tunggal yang mewakili nama konektor. String ini juga memperlihatkan kenaikan waktu minimum yang menentukan seberapa halus kita dapat mendefinisikan bentuk gelombang.

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

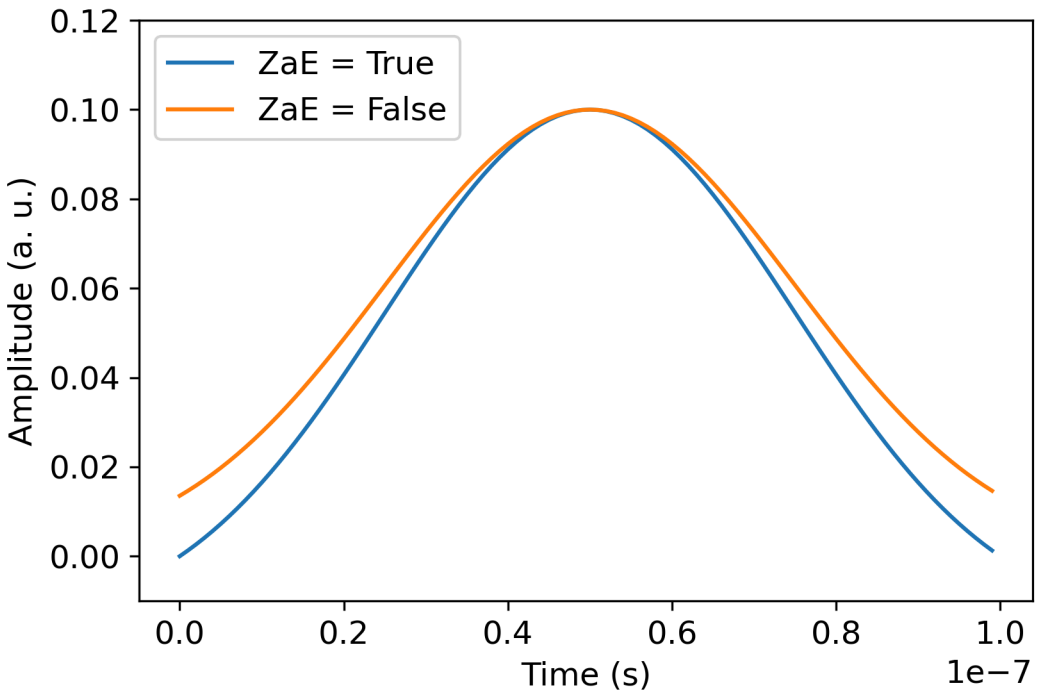
Bentuk gelombang

Bentuk gelombang adalah amplop yang bergantung pada waktu yang dapat kita gunakan untuk memancarkan sinyal pada port output atau menangkap sinyal melalui port input. Anda dapat menentukan bentuk gelombang Anda secara langsung baik melalui daftar bilangan kompleks atau dengan menggunakan templat bentuk gelombang untuk menghasilkan daftar dari penyedia perangkat keras.

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

Braket Pulse menyediakan perpustakaan standar bentuk gelombang, termasuk bentuk gelombang konstan, bentuk gelombang Gaussian, dan bentuk gelombang Penghapusan Derivatif oleh Gerbang Adiabatik (DRAG). Anda dapat mengambil data bentuk gelombang melalui `sample` fungsi untuk menggambar bentuk bentuk gelombang seperti yang ditunjukkan pada contoh berikut.

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



Gambar sebelumnya menggambarkan bentuk gelombang Gaussian yang dibuat dari. `GaussianWaveform` Kami memilih panjang pulsa 100 ns, lebar 25 ns, dan amplitudo 0,1 (unit arbitrer). Bentuk gelombang berpusat di jendela pulsa. `GaussianWaveform`menerima argumen boolean `zero_at_edges` (ZaE dalam legenda). Ketika diatur ke`True`, argumen ini mengimbangi bentuk gelombang Gaussian sedemikian rupa sehingga titik pada t=0 dan t= `length` berada pada nol dan mengubah skala amplitudonya sedemikian rupa sehingga nilai maksimum sesuai dengan argumen. `amplitude`

Sekarang kita telah membahas konsep dasar untuk akses tingkat pulsa, selanjutnya kita akan melihat bagaimana membangun sirkuit menggunakan gerbang dan pulsa.

Peran frame dan port

Bagian ini menjelaskan frame dan port yang telah ditentukan yang tersedia untuk setiap perangkat. Kami juga akan membahas secara singkat mekanisme yang terlibat ketika pulsa dimainkan pada bingkai tertentu.

Rigetti

Bingkai

Rigettiperangkat mendukung frame yang telah ditentukan sebelumnya yang memiliki frekuensi dan fase yang dikalibrasi agar sesuai dengan qubit terkait. Konvensi penamaan adalah `q{i}[_q{j}]_{role}_frame` di mana `{i}` mengacu pada nomor qubit pertama, `{j}` mengacu pada nomor qubit kedua jika bingkai berfungsi untuk mengaktifkan interaksi dua-qubit, dan `{role}` mengacu pada peran bingkai. Perannya adalah sebagai berikut:

- `rf`adalah bingkai untuk mendorong transisi 0-1 dari qubit. Pulsa ditransmisikan sebagai sinyal transien gelombang mikro frekuensi dan fase yang sebelumnya disediakan melalui dan fungsi. `set shift` Amplitudo sinyal yang bergantung pada waktu diberikan oleh bentuk gelombang yang dimainkan pada bingkai. Bingkai menghubungkan interaksi single-qubit, off-diagonal. Untuk informasi lebih lanjut, lihat Krantz et al. dan Rahamim dkk. .
- `rf_f12`mirip dengan `rf` dan parameternya menargetkan transisi 1-2.
- `ro_rx`digunakan untuk mencapai pembacaan dispersif qubit melalui pandu gelombang koplanar yang digabungkan. Frekuensi, fase, dan set lengkap parameter untuk bentuk gelombang pembacaan telah dikalibrasi sebelumnya. Saat ini digunakan melalui`capture_v0`, yang tidak memerlukan argumen apa pun selain pengidentifikasi bingkai.
- `ro_tx`adalah untuk mentransmisikan sinyal dari resonator. Saat ini tidak digunakan.
- `cz`adalah bingkai yang dikalibrasi untuk mengaktifkan gerbang dua-qubit`cz`. Seperti semua frame yang terkait dengan `ff` port, ia mengaktifkan interaksi terjerat melalui garis fluks dengan memodulasi qubit yang dapat disetel dari pasangan pada resonansi dengan tetangganya. Untuk informasi lebih lanjut tentang mekanisme terjerat, lihat Reagor et al. , Caldwell dkk. , dan Didier dkk. .
- `cphase`adalah bingkai yang dikalibrasi untuk mengaktifkan `cphaseshift` gerbang dua-qubit dan dihubungkan ke port. `ff` Untuk informasi lebih lanjut tentang mekanisme terjerat, lihat deskripsi untuk bingkai. `cz`
- `xy`adalah bingkai yang dikalibrasi untuk mengaktifkan gerbang XY (θ) dua-qubit dan dihubungkan ke port. `ff` Untuk informasi lebih lanjut tentang mekanisme belitan dan cara mencapai gerbang XY, lihat deskripsi untuk `cz` bingkai dan Abrams et al. .

Karena frame berdasarkan `ff` port menggeser frekuensi qubit yang dapat disetel, semua frame penggerak lain yang terkait dengan qubit akan di-dephased dengan jumlah yang terkait dengan amplitudo dan durasi pergeseran frekuensi. Akibatnya, Anda harus mengkompensasi efek ini dengan menambahkan pergeseran fase yang sesuai ke bingkai qubit tetangga.

Pelabuhan

RigettiPerangkat menyediakan daftar port yang dapat Anda periksa melalui kemampuan perangkat. Nama port mengikuti konvensi `q{i}_{type}` di mana `{i}` mengacu pada nomor qubit dan `{type}` mengacu pada jenis port. Perhatikan bahwa tidak semua qubit memiliki satu set port yang lengkap. Jenis-jenis port adalah sebagai berikut:

- `rf`merupakan antarmuka utama untuk mendorong transisi qubit tunggal. Hal ini terkait dengan `rf` dan `rf_f12` frame. Ini secara kapasitif digabungkan ke qubit, memungkinkan mengemudi microwave dalam kisaran gigahertz.
- `ro_tx`berfungsi untuk mengirimkan sinyal ke resonator pembacaan yang digabungkan secara kapasitif ke qubit. Pengiriman sinyal pembacaan dimultipleks delapan kali lipat dengan segi delapan.
- `ro_rx`berfungsi untuk menerima sinyal dari resonator pembacaan yang digabungkan ke qubit.
- `ff`mewakili garis fluks cepat yang digabungkan secara induktif ke qubit. Kita dapat menggunakan ini untuk menyetel frekuensi transmon. Hanya qubit yang dirancang agar sangat dapat disetel yang memiliki `ff` port. Port ini berfungsi untuk mengaktifkan interaksi qubit-qubit karena ada kopling kapasitif statis antara setiap pasangan transmon tetangga.

Untuk informasi lebih lanjut tentang arsitektur, lihat Valery et al. .

OQC

Bingkai

OQCperangkat mendukung frame yang telah ditentukan sebelumnya yang memiliki frekuensi dan fase yang dikalibrasi agar sesuai dengan qubit terkait. Konvensi penamaan untuk frame ini adalah sebagai berikut:

- frame mengemudi: `q{i}[_q{j}]_{role}` di mana `{i}` mengacu pada nomor qubit pertama, `{j}` mengacu pada nomor qubit kedua jika frame berfungsi untuk mengaktifkan interaksi dua-qubit, dan `{role}` mengacu pada peran frame seperti yang dijelaskan di bawah ini.
- qubit readout frame: `r{i}_{role}` di mana `{i}` mengacu pada nomor qubit dan `{role}` mengacu pada peran frame seperti yang dijelaskan di bawah ini.

Kami merekomendasikan penggunaan setiap frame untuk peran yang dirancang sebagai berikut:

- `drive`digunakan sebagai kerangka utama untuk mendorong transisi 0-1 dari qubit. Pulsa ditransmisikan sebagai sinyal transien gelombang mikro frekuensi dan fase yang sebelumnya

disediakan melalui dan fungsi. `set shift` Amplitudo sinyal yang bergantung pada waktu diberikan oleh bentuk gelombang yang dimainkan pada bingkai. Bingkai menghubungkan interaksi single-qubit, off-diagonal. Untuk informasi lebih lanjut, lihat Krantz et al. dan Rahamim dkk. .

- `second_state`setara dengan `drive` frame tetapi frekuensinya disetel pada resonansi dengan transisi 1-2.
- `measure`adalah untuk pembacaan. Frekuensi, fase, dan set lengkap parameter untuk bentuk gelombang pembacaan telah dikalibrasi sebelumnya. Saat ini digunakan melalui`capture_v0`, yang tidak memerlukan argumen apa pun selain pengidentifikasi bingkai.
- `acquire`adalah untuk menangkap sinyal dari resonator. Saat ini tidak digunakan.
- `cross_resonance`mengaktifkan interaksi resonansi silang antara qubit `i` dan `j` dengan menggerakkan qubit kontrol `i` pada frekuensi transisi qubit target. `j` Akibatnya, frekuensi frame diatur menggunakan frekuensi qubit target. Interaksi terjadi dengan laju yang sebanding dengan amplitudo penggerak resonansi silang ini. Berbagai jenis crosstalks menginduksi efek yang tidak diinginkan yang memerlukan koreksi. Lihat Patterson dkk. untuk informasi lebih lanjut tentang interaksi resonansi silang dengan qubit transmon berbentuk koaksil ('coaxmons').
- `cross_resonance_cancellation`membantu Anda menambahkan koreksi untuk menekan efek merusak yang disebabkan oleh crosstalks ketika interaksi lintas-resonansi diaktifkan. Frekuensi frame awal diatur ke frekuensi transisi qubit `i` kontrol. Untuk informasi lebih lanjut tentang metode pembatalan, lihat Patterson et al. .

Pelabuhan

OQCPerangkat menyediakan daftar port yang dapat Anda periksa melalui kemampuan perangkat. Frame yang dijelaskan sebelumnya dikaitkan dengan port yang diidentifikasi oleh id mereka `channel_{N}` di `{N}` mana bilangan bulat. Port adalah antarmuka untuk mengontrol garis (arah`tx`) dan resonator pembacaan (arah`rx`) yang terhubung ke koakson. Setiap qubit dikaitkan dengan satu garis kontrol dan satu resonator pembacaan. Port transmisi adalah antarmuka untuk manipulasi single-qubit dan dua-qubit. Port penerimaan berfungsi untuk pembacaan qubit.

Halo Pulse

Di sini, Anda akan belajar cara membuat pasangan Bell sederhana secara langsung dengan pulsa, dan menjalankan program pulsa ini pada Rigetti perangkat. Pasangan Bell adalah sirkuit dua-qubit yang terdiri dari gerbang Hadamard pada qubit pertama diikuti oleh `cnot` gerbang antara qubit pertama dan kedua. Membuat status terjerat dengan pulsa membutuhkan mekanisme khusus yang bergantung pada jenis perangkat keras dan arsitektur perangkat. Kami tidak akan menggunakan

mekanisme asli untuk membuat `cnot` gerbang. Sebagai gantinya, kita akan menggunakan bentuk gelombang dan bingkai tertentu yang mengaktifkan `cz` gerbang secara asli. Dalam contoh ini, kita akan membuat gerbang Hadamard menggunakan gerbang asli qubit tunggal `rx` dan `rz` dan mengekspresikan gerbang menggunakan pulsa`cz`.

Pertama, mari kita impor perpustakaan yang diperlukan. Selain `Circuit` kelas, Anda sekarang juga perlu mengimpor `PulseSequence` kelas.

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

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

Selanjutnya, buat instance perangkat Braket baru menggunakan Amazon Resource Name (ARN) perangkat. Rigetti Aspen-M-3 Lihat halaman Perangkat di konsol Amazon Braket untuk melihat tata letak perangkat. Rigetti Aspen-M-3

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

Karena gerbang Hadamard bukan gerbang asli Rigetti perangkat, itu tidak dapat digunakan dalam kombinasi dengan pulsa. Karena itu Anda perlu menguraikannya menjadi urutan gerbang `rz` asli `rx` dan asli.

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

Untuk `cz` gerbang, kita akan menggunakan bentuk gelombang arbitrer dengan parameter (amplitudo, waktu naik/turun, dan durasi) yang telah ditentukan sebelumnya oleh penyedia perangkat keras

selama tahap kalibrasi. Bentuk gelombang ini akan diterapkan pada. `q10_q113_cz_frame` Untuk versi yang lebih baru dari bentuk gelombang arbitrer yang digunakan di sini, lihat QCS, di situs web. Rigetti Anda mungkin diminta untuk membuat akun QCS.

```
a_b_cz_wfm = ArbitraryWaveform([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00017888439538396808, 0.00046751103636033026, 0.0011372942989106456,
 0.00257705961192697, 0.00544394194463236 6, 0.010731922770068104, 0.01976701723583167,
 0.03406712171899736, 0.05503285980691202, 0.08350670755829034, 0.11932853352131022,
 0.16107456696238298, 0.20614055551722368, 0.2512065440720643, 0.292952577513137,
 0.328774403476157, 0.3572482512275353, 0.3782139893154499, 0.3925140937986156,
 0.40154918826437913, 0.4068371690898149, 0.4097040514225177, 0.41114381673553674,
 0.411813599998087, 0.4121022266390633, 0.4122174383870584, 0.41226003881132406,
 0.4122746298554775, 0.4122792591252675, 0.4122806196003006, 0.41228098995582513,
 0.41228108334474756, 0.4122811051578895, 0.4122811098772742, 0.4122811108230642,
 0.4122811109986316, 0.41228111102881937, 0.41228111103362725, 0.4122811110343365,
 0.41228111103443343, 0.4122811110344457, 0.4122811110344471, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.4122811110344471, 0.4122811110344457, 0.41228111103443343, 0.4122811110343365,
 0.41228111103362725, 0.41228111102881937, 0.4122811109986316, 0.4122811108230642,
 0.4122811098772742, 0.4122811051578895, 0.41228108334474756, 0.41228098995582513,
 0.4122806196003006, 0.4122792591252675, 0.4122746298554775, 0.41226003881132406,
 0.4122174383870584, 0.4121022266390633, 0.411813599998087, 0.41114381673553674,
 0.4097040514225176, 0.4068371690898149, 0.40154918826437913, 0.3925140937986155,
 0.37821398931544986, 0.3572482512275351, 0.32877440347615655, 0.2929525775131368,
 0.2512065440720641, 0.20614055551722307, 0.16107456696238268, 0.11932853352131002,
 0.08350670755829034, 0.05503285980691184, 0.03406712171899729, 0.01976701723583167,
 0.010731922770068058, 0.005443941944632366, 0.002577059611929697,
 0.0011372942989106229, 0.00046751103636033026, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.0])
a_b_cz_frame = device.frames[f'q{a}_q{b}_cz_frame']

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

Ini harus kembali:

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

Sekarang kita dapat membangun `cz` gerbang menggunakan bentuk gelombang yang baru saja kita definisikan. Ingat bahwa `cz` gerbang terdiri dari flip fase qubit target jika qubit kontrol dalam keadaan. `|1>`

```
phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

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

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

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

`a_b_cz_wfm`Bentuk gelombang dimainkan pada bingkai yang terkait dengan port fluks cepat. Perannya adalah menggeser frekuensi qubit untuk mengaktifkan interaksi qubit-qubit. Untuk informasi selengkapnya, lihat Peran frame dan port. Karena frekuensinya bervariasi, frame qubit berputar pada kecepatan yang berbeda dari `rf` frame qubit tunggal yang tetap tidak tersentuh: yang terakhir semakin di-dephased. Pergeseran fase ini dikalibrasi melalui Ramsey urutan sebelumnya dan disediakan di sini sebagai informasi hardcode melalui dan (Periode penuh). `phase_shift_a` `phase_shift_b` Kami memperbaiki dephasing ini dengan menggunakan `shift_phase` instruksi pada `rf` bingkai. Perhatikan bahwa urutan ini hanya akan berfungsi dalam program di mana tidak ada XY bingkai yang terkait dengan qubit a dan b digunakan karena kami tidak mengkompensasi pergeseran fase yang terjadi pada frame ini. Ini adalah kasus untuk program pasangan Bell tunggal ini, yang hanya menggunakan `rf` dan `cz` bingkai. Untuk informasi lebih lanjut, lihat Caldwell et al. .

Sekarang kita siap untuk membuat pasangan Bell dengan pulsa.

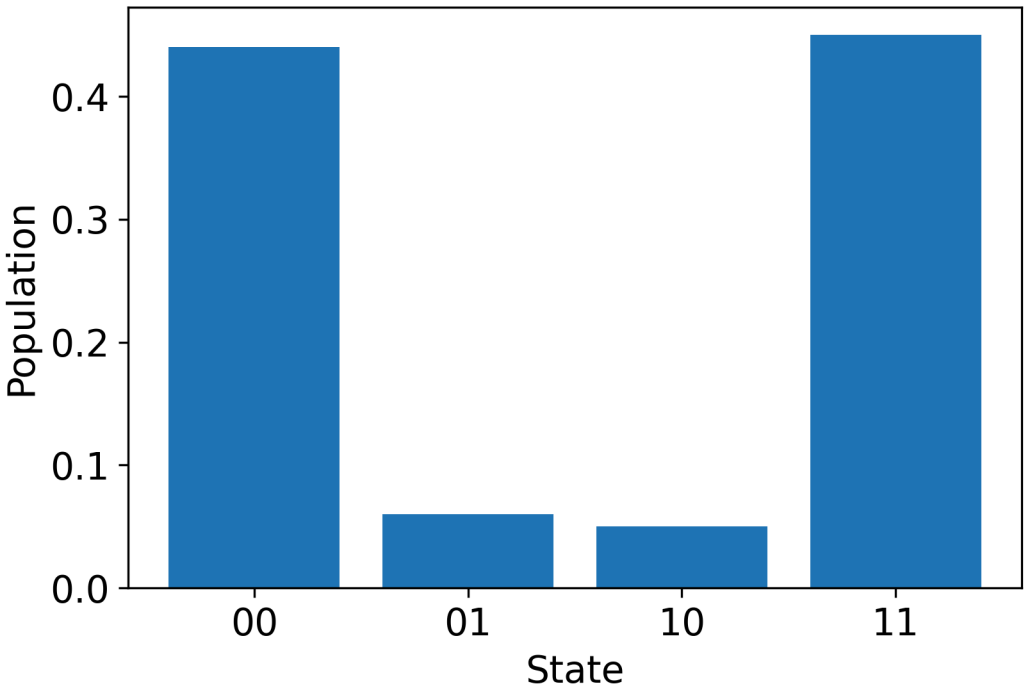
```
bell_circuit_pulse = (
    Circuit()
    .rigetti_native_h(a)
    .rigetti_native_h(b)
```

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

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

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

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

Mari kita jalankan pasangan Bell ini di Rigetti perangkat. Perhatikan bahwa menjalankan blok kode ini akan dikenakan biaya. Untuk informasi selengkapnya tentang biaya ini, lihat halaman Harga Amazon Braket. Kami menyarankan Anda menguji sirkuit Anda menggunakan sejumlah kecil tembakan untuk memastikan bahwa itu dapat berjalan pada perangkat sebelum meningkatkan jumlah tembakan.

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

counts = task.result().measurement_counts

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



Halo Pulse menggunakan OpenPulse

OpenPulseadalah bahasa untuk menentukan kontrol tingkat pulsa dari perangkat kuantum umum dan merupakan bagian dari spesifikasi OpenQASM 3.0. Amazon Braket mendukung OpenPulse pulsa pemrograman langsung menggunakan representasi OpenQASM 3.0.

Braket digunakan OpenPulse sebagai representasi perantara yang mendasari untuk mengekspresikan pulsa dalam instruksi asli. OpenPulsemendukung penambahan kalibrasi instruksi dalam bentuk deklarasi `defcal` (singkatan dari “define calibration”). Dengan deklarasi ini, Anda dapat menentukan implementasi instruksi gerbang dalam tata bahasa kontrol tingkat rendah.

Dalam contoh ini, kita akan membangun sirkuit Bell menggunakan OpenQASM 3.0 dan OpenPulse pada perangkat yang menggunakan transmon yang dapat disetel frekuensi. Ingat bahwa sirkuit Bell adalah sirkuit dua-qubit yang terdiri dari gerbang Hadamard pada qubit pertama diikuti oleh gerbang antara dua qubit. `cnot` Karena `cnot` gerbang berbeda dari `cz` gerbang hanya melalui transformasi dasar, di sini kita akan mendefinisikan pasangan Bell menggunakan Hadamard dan `cz` gerbang sebagai gantinya karena perangkat menyediakan cara yang lebih sederhana untuk membuat `cz` gerbang untuk demonstrasi ini.

Mari kita mulai dengan mendefinisikan gerbang Hadamard menggunakan gerbang asli perangkat.

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

Untuk gerbang CZ, kita akan menggunakan bentuk gelombang arbitrer dengan parameter (amplitudo, waktu naik/turun, dan durasi) yang telah ditentukan sebelumnya. Bentuk gelombang ini akan diterapkan pada. `q10_q113_cz_frame`

```
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.461213110959686859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.484432196472809 6,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

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

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

Durasi `q10_q113_cz_wfm` bentuk gelombang adalah 124 sampel, yang sesuai dengan 124 ns karena kenaikan `dt` waktu minimum adalah 1 ns.

`q10_q113_cz_wfm`Bentuk gelombang dimainkan pada bingkai yang terikat ke port fluks cepat. Perannya adalah menggeser frekuensi qubit untuk mengaktifkan interaksi qubit-qubit. Untuk informasi selengkapnya, lihat Peran frame dan port. Karena frekuensinya bervariasi, frame qubit berputar pada kecepatan yang berbeda dibandingkan dengan `rf` frame qubit tunggal yang tetap tidak tersentuh: yang terakhir semakin di-dephased. Pencabutan ini dapat diukur dengan Ramsey urutan selama tahap kalibrasi dan dikompensasi dengan `shift_phase` instruksi pada dan bingkai. `rf` `xy` Untuk informasi lebih lanjut, lihat Caldwell et al. .

Kita sekarang dapat mengeksekusi sirkuit pasangan Bell di mana kita menguraikan `cnot` gerbang menggunakan beberapa Hadamard dan gerbang. `cz`

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

Representasi OpenQASM 3.0 penuh untuk sirkuit Bell yang dibangun menggunakan kombinasi gerbang dan pulsa asli adalah sebagai berikut.

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

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

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

Anda sekarang dapat menggunakan Braket SDK untuk menjalankan program OpenQASM 3.0 ini pada Rigetti perangkat menggunakan kode berikut.

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

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

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

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

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

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

```
)
```

Mengakses gerbang asli menggunakan pulsa

Para peneliti sering perlu tahu persis bagaimana gerbang asli yang didukung oleh QPU tertentu diimplementasikan sebagai pulsa. Urutan pulsa dikalibrasi dengan hati-hati oleh penyedia perangkat keras, tetapi mengaksesnya memberi peneliti kesempatan untuk merancang gerbang yang lebih baik atau mengeksplorasi protokol untuk mitigasi kesalahan seperti ekstrapolasi nol kebisingan dengan meregangkan pulsa gerbang tertentu.

Amazon Braket mendukung akses terprogram ke gerbang asli dari Rigetti.

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

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

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

> **Note**
>
> Penyedia perangkat keras secara berkala mengkalibrasi QPU, seringkali lebih dari sekali sehari. Braket SDK memungkinkan Anda mendapatkan kalibrasi gerbang terbaru.

```
device.refresh_gate_calibrations()
```

Untuk mengambil gerbang asli yang diberikan, seperti gerbang RX atau XY, Anda harus melewati `Gate` objek dan qubit yang diinginkan. Misalnya, Anda dapat memeriksa implementasi pulsa RX (π/2) yang diterapkan pada 0. qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

Anda dapat membuat serangkaian kalibrasi yang difilter menggunakan `filter` fungsi tersebut. Anda melewati daftar gerbang atau daftar`QubitSet`. Kode berikut membuat dua set yang berisi semua kalibrasi untuk RX (π/2) dan untuk 0. qubit

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

Sekarang Anda dapat memberikan atau memodifikasi aksi gerbang asli dengan melampirkan set kalibrasi khusus. Misalnya, perhatikan rangkaian berikut.

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

Anda dapat menjalankannya dengan kalibrasi gerbang khusus untuk `rx` gerbang `qubit 0` dengan meneruskan kamus `PulseSequence` objek ke argumen `gate_definitions` kata kunci. Anda dapat membuat kamus `pulse_sequences` dari atribut `GateCalibrations` objek. Semua gerbang yang tidak ditentukan diganti dengan kalibrasi pulsa penyedia perangkat keras kuantum.

```
nb_shots = 50
custom_calibration = GateCalibrations({rx_pi_2_q0: pulse_sequence_rx_pi_2_q0})
task=device.run(bell_circuit, gate_definitions=custom_calibration.pulse_sequences,
 shots=nb_shots)
```

Panduan Pengguna Pekerjaan Hibrida Amazon Braket

Bagian ini memberikan petunjuk tentang cara mengatur dan mengelola pekerjaan hybrid di Amazon Braket.

Anda dapat mengakses pekerjaan hybrid di Braket menggunakan:

- SDK Python Amazon Braket.
- Konsol Amazon Braket.
- AmazonBraketAPI.

Di bagian ini:

- Berinteraksi dengan pekerjaan hybrid secara langsung menggunakan API

Apa itu Hybrid Job?

Amazon Braket Hybrid Jobs menawarkan cara bagi Anda untuk menjalankan algoritma kuantum klasik hibrida yang membutuhkan AWS sumber daya klasik dan unit pemrosesan kuantum (QPU). Hybrid Jobs dirancang untuk memutar sumber daya klasik yang diminta, menjalankan algoritme Anda, dan merilis instance setelah selesai sehingga Anda hanya membayar untuk apa yang Anda gunakan.

Hybrid Jobs sangat ideal untuk algoritma iteratif jangka panjang yang melibatkan sumber daya klasik dan kuantum. Anda mengirimkan algoritme untuk dijalankan, Braket menjalankannya di lingkungan kontainer yang dapat diskalakan, dan Anda mengambil hasilnya saat algoritme selesai.

Selain itu, tugas kuantum yang dibuat dari pekerjaan hibrida mendapat manfaat dari antrian prioritas yang lebih tinggi ke QPU target. Ini memastikan tugas kuantum Anda diproses dan berjalan di depan orang lain dalam antrian. Ini sangat bermanfaat untuk algoritme hibrida berulang di mana tugas selanjutnya bergantung pada hasil tugas kuantum sebelumnya. Contoh algoritme tersebut termasuk Algoritma Pengoptimalan Perkiraan Kuantum (QAOA), pemecah eigen kuantum variasional, atau pembelajaran mesin kuantum. Anda juga dapat memantau kemajuan algoritme Anda dalam waktu nyaris nyata, memungkinkan Anda melacak biaya, anggaran, atau metrik khusus seperti kehilangan pelatihan atau nilai ekspektasi.

Kapan menggunakan Amazon Braket Hybrid Jobs

Amazon Braket Hybrid Jobs memungkinkan Anda untuk menjalankan algoritma kuantum klasik hybrid, seperti Variational Quantum Eigensolver (VQE) dan Quantum Perkiraan Optimasi Algoritma (QAOA), yang menggabungkan sumber daya komputasi klasik dengan perangkat komputasi kuantum untuk mengoptimalkan kinerja sistem kuantum saat ini. Amazon Braket Hybrid Jobs memberikan tiga manfaat utama:

1. Kinerja: Amazon Braket Hybrid Jobs memberikan kinerja yang lebih baik daripada menjalankan algoritma hybrid dari lingkungan Anda sendiri. Saat pekerjaan Anda berjalan, ia memiliki akses prioritas ke QPU target yang dipilih. Tugas dari pekerjaan Anda berjalan di depan tugas lain yang antri di perangkat. Ini menghasilkan runtime yang lebih pendek dan lebih dapat diprediksi untuk algoritma hybrid. Amazon Braket Hybrid Jobs juga mendukung kompilasi parametrik. Anda dapat mengirimkan sirkuit menggunakan parameter gratis dan Braket mengkompilasi sirkuit sekali,

tanpa perlu mengkompilasi ulang untuk pembaruan parameter berikutnya ke sirkuit yang sama, menghasilkan runtime yang lebih cepat.

2. Kenyamanan: Amazon Braket Hybrid Jobs menyederhanakan pengaturan dan pengelolaan lingkungan komputasi Anda dan menjaganya tetap berjalan saat algoritma hybrid Anda berjalan. Anda cukup memberikan skrip algoritme Anda dan memilih perangkat kuantum (baik unit pemrosesan kuantum atau simulator) untuk dijalankan. Amazon Braket menunggu perangkat target tersedia, memutar sumber daya klasik, menjalankan beban kerja di lingkungan kontainer yang sudah dibuat sebelumnya, mengembalikan hasilnya ke Amazon Simple Storage Service (Amazon S3), dan merilis sumber daya komputasi.
3. Metrik: Amazon Braket Hybrid Jobs on-the-fly memberikan wawasan tentang algoritme yang sedang berjalan dan memberikan metrik algoritme yang dapat disesuaikan dalam waktu dekat ke Amazon dan CloudWatch konsol Amazon Braket sehingga Anda dapat melacak kemajuan algoritme Anda.

Jalankan kode lokal Anda sebagai pekerjaan hibrida

Amazon Braket Hybrid Jobs menyediakan orkestrasi algoritma klasik kuantum hibrida yang dikelola sepenuhnya, menggabungkan sumber daya komputasi Amazon EC2 dengan akses Unit Pemrosesan Kuantum Amazon Braket (QPU). Tugas kuantum yang dibuat dalam pekerjaan hibrida memiliki antrian prioritas di atas tugas kuantum individu sehingga algoritme Anda tidak akan terganggu oleh fluktuasi dalam antrian tugas kuantum. Setiap QPU mempertahankan antrian pekerjaan hibrida yang terpisah, memastikan bahwa hanya satu pekerjaan hibrida yang dapat berjalan pada waktu tertentu.

Di bagian ini:

- Buat pekerjaan hybrid dari kode Python lokal
- Instal paket Python tambahan dan kode sumber
- Menyimpan dan memuat data ke dalam instance pekerjaan hibrida
- Praktik terbaik untuk dekorator pekerjaan hibrida

Buat pekerjaan hybrid dari kode Python lokal

Anda dapat menjalankan kode Python lokal Anda sebagai Amazon Braket Hybrid Job. Anda dapat melakukan ini dengan membuat anotasi kode Anda dengan `@hybrid_job` dekorator, seperti yang ditunjukkan pada contoh kode berikut. Untuk lingkungan khusus, Anda dapat memilih untuk menggunakan wadah khusus dari Amazon Elastic Container Registry (ECR).

> **Note**
>
> Hanya Python 3.10 yang didukung secara default.

Anda dapat menggunakan `@hybrid_job` dekorator untuk membubuhi keterangan suatu fungsi. Braket mengubah kode di dalam dekorator menjadi skrip algoritma pekerjaan hibrida Braket. Pekerjaan hybrid kemudian memanggil fungsi di dalam dekorator pada instans Amazon EC2. Anda dapat memantau kemajuan pekerjaan dengan `job.state()` atau dengan konsol Braket. Contoh kode berikut menunjukkan bagaimana menjalankan urutan lima status padaState Vector Simulator (SV1) device.

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

device_arn = Devices.Amazon.SV1


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

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

    theta = 0.0  # initial parameter

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

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

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

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

Anda membuat pekerjaan hybrid dengan menjalankan fungsi seperti yang Anda lakukan pada fungsi Python normal. Namun, fungsi dekorator mengembalikan pegangan pekerjaan hibrida daripada hasil fungsi. Untuk mengambil hasil setelah selesai, gunakan`job.result()`.

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

Argumen perangkat di `@hybrid_job` dekorator menentukan perangkat yang memiliki akses prioritas pekerjaan hybrid - dalam hal ini, simulator. SV1 Untuk mendapatkan prioritas QPU, Anda harus memastikan bahwa perangkat ARN yang digunakan dalam fungsi cocok dengan yang ditentukan dalam dekorator. Untuk kenyamanan, Anda dapat menggunakan fungsi pembantu `get_job_device_arn()` untuk menangkap perangkat yang dideklarasikan ARN. `@hybrid_job`

> **Note**
>
> Setiap pekerjaan hybrid memiliki setidaknya satu menit waktu startup karena menciptakan lingkungan kontainer di Amazon EC2. Jadi untuk beban kerja yang sangat singkat, seperti rangkaian tunggal atau sekumpulan sirkuit, mungkin cukup bagi Anda untuk menggunakan tugas kuantum.

Hiperparameter

`run_hybrid_job()`Fungsi mengambil argumen `num_tasks` untuk mengontrol jumlah tugas kuantum yang dibuat. Pekerjaan hybrid secara otomatis menangkap ini sebagai hyperparameter.

> **Note**
>
> Hyperparameters ditampilkan di konsol Braket sebagai string, yang dibatasi hingga 2500 karakter.

Metrik dan pencatatan

Dalam `run_hybrid_job()` fungsi tersebut, metrik dari algoritma iteratif direkam dengan. `log_metrics` Metrik secara otomatis diplot di halaman konsol Braket di bawah tab pekerjaan hybrid. Anda dapat menggunakan metrik untuk melacak biaya tugas kuantum secara mendekati waktu nyata

selama pekerjaan hibrida dijalankan dengan pelacak biaya Braket. Contoh di atas menggunakan nama metrik “probabilitas” yang mencatat probabilitas pertama dari jenis hasil.

Mengambil hasil

Setelah pekerjaan hybrid selesai, Anda gunakan `job.result()` untuk mengambil hasil pekerjaan hibrida. Setiap objek dalam pernyataan pengembalian secara otomatis ditangkap oleh Braket. Perhatikan bahwa objek yang dikembalikan oleh fungsi harus berupa tupel dengan setiap elemen menjadi serializable. Misalnya, kode berikut menunjukkan contoh yang berfungsi, dan gagal.

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

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

Nama Job

Secara default, nama untuk pekerjaan hybrid ini disimpulkan dari nama fungsi. Anda juga dapat menentukan nama kustom hingga 50 karakter. Misalnya, dalam kode berikut nama pekerjaan adalah “my-job-name”.

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

Modus lokal

Pekerjaan lokal dibuat dengan menambahkan argumen `local=True` ke dekorator. Ini menjalankan pekerjaan hybrid di lingkungan kontainer di lingkungan komputasi lokal Anda, seperti laptop Anda. Pekerjaan lokal tidak memiliki antrian prioritas untuk tugas-tugas kuantum. Untuk kasus lanjutan seperti multi-node atau MPI, pekerjaan lokal mungkin memiliki akses ke variabel lingkungan Braket yang diperlukan. Kode berikut membuat pekerjaan hybrid lokal dengan perangkat sebagai simulator SV1.

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

Semua opsi pekerjaan hybrid lainnya didukung. Untuk daftar opsi, lihat modul braket.jobs.quantum_job_creation.

Instal paket Python tambahan dan kode sumber

Anda dapat menyesuaikan lingkungan runtime Anda untuk menggunakan paket Python pilihan Anda. Anda dapat menggunakan `requirements.txt` file, daftar nama paket, atau membawa wadah Anda sendiri (BYOC). Untuk menyesuaikan lingkungan runtime menggunakan `requirements.txt` file, lihat contoh kode berikut.

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

Misalnya, `requirements.txt` file tersebut mungkin menyertakan paket lain untuk diinstal.

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

Atau, Anda dapat memberikan nama paket sebagai daftar Python sebagai berikut.

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

Kode sumber tambahan dapat ditentukan baik sebagai daftar modul, atau modul tunggal seperti pada contoh kode berikut.

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

Menyimpan dan memuat data ke dalam instance pekerjaan hibrida

Menentukan data pelatihan input

Saat membuat pekerjaan hybrid, Anda dapat memberikan kumpulan data pelatihan input dengan menentukan bucket Amazon Simple Storage Service (Amazon S3). Anda juga dapat

menentukan jalur lokal, lalu Braket secara otomatis mengunggah data ke Amazon S3 di. `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>` Jika Anda menentukan jalur lokal, nama saluran default ke “input”. Kode berikut menunjukkan file numpy dari jalur `data/file.npy` lokal.

```
@hybrid_job(device=Devices.Amazon.SV1, input_data="data/file.npy")
def run_hybrid_job(num_tasks = 1):
    data = np.load("data/file.npy")
    return ...
```

Untuk S3, Anda harus menggunakan fungsi `get_input_data_dir()` pembantu.

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

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

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

Anda dapat menentukan beberapa sumber data input dengan menyediakan kamus nilai saluran dan URI S3 atau jalur lokal.

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


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

> **Note**
>
> Ketika data input besar (> 1GB), ada waktu tunggu yang lama sebelum pekerjaan dibuat. Ini karena data input lokal saat pertama kali diunggah ke bucket S3, kemudian jalur S3

ditambahkan ke permintaan pekerjaan. Akhirnya, permintaan pekerjaan diajukan ke layanan Braket.

Menyimpan hasil ke S3

Untuk menyimpan hasil yang tidak termasuk dalam pernyataan pengembalian fungsi yang didekorasi, Anda harus menambahkan direktori yang benar ke semua operasi penulisan file. Contoh berikut, menunjukkan menyimpan array numpy dan angka matplotlib.

```
@hybrid_job(device=Devices.Amazon.SV1)
def run_hybrid_job(num_tasks = 1):
    result = np.random.rand(5)

    # save a numpy array
    np.save("result.npy", result)

    # save a matplotlib figure
    plt.plot(result)
    plt.savefig("fig.png")
    return ...
```

Semua hasil dikompresi menjadi file bernama`model.tar.gz`. Anda dapat mengunduh hasilnya dengan fungsi Python`job.result()`, atau dengan menavigasi ke folder hasil dari halaman pekerjaan hybrid di konsol manajemen Braket.

Menyimpan dan melanjutkan dari pos pemeriksaan

Untuk pekerjaan hybrid yang berjalan lama, disarankan untuk secara berkala menyimpan keadaan perantara algoritma. Anda dapat menggunakan fungsi `save_job_checkpoint()` pembantu bawaan, atau menyimpan file ke `AMZN_BRAKET_JOB_RESULTS_DIR` jalur. Nanti tersedia dengan fungsi `get_job_results_dir()` pembantu.

Berikut ini adalah contoh kerja minimal untuk menyimpan dan memuat pos pemeriksaan dengan dekorator pekerjaan hybrid:

```
from braket.jobs import save_job_checkpoint, load_job_checkpoint, hybrid_job

@hybrid_job(device=None, wait_until_complete=True)
def function():
    save_job_checkpoint({"a": 1})
```

```
job = function()
job_name = job.name
job_arn = job.arn

@hybrid_job(device=None, wait_until_complete=True, copy_checkpoints_from_job=job_arn)
def continued_function():
    load_job_checkpoint(job_name)

continued_job = continued_function()
```

Dalam pekerjaan hybrid pertama, `save_job_checkpoint()` disebut dengan kamus yang berisi data yang ingin kita simpan. Secara default, setiap nilai harus dapat diserialkan sebagai teks. Untuk memeriksa objek Python yang lebih kompleks, seperti array numpy, Anda dapat mengatur. `data_format = PersistedJobDataFormat.PICKLED_V4` Kode ini membuat dan menimpa file pos pemeriksaan dengan nama default di artefak pekerjaan hibrida Anda `<jobname>.json` di bawah subfolder yang disebut “pos pemeriksaan”.

Untuk membuat pekerjaan hybrid baru untuk melanjutkan dari pos pemeriksaan, kita harus lulus `copy_checkpoints_from_job=job_arn` di `job_arn` mana ARN pekerjaan hybrid dari pekerjaan sebelumnya. Kemudian kita gunakan `load_job_checkpoint(job_name)` untuk memuat dari pos pemeriksaan.

Praktik terbaik untuk dekorator pekerjaan hibrida

Merangkul asinkron

Pekerjaan hibrida yang dibuat dengan anotasi dekorator tidak sinkron - mereka berjalan setelah sumber daya klasik dan kuantum tersedia. Anda memantau kemajuan algoritma menggunakan Braket Management Console atau Amazon CloudWatch. Saat Anda mengirimkan algoritme untuk dijalankan, Braket menjalankan algoritme Anda di lingkungan kontainer yang dapat diskalakan dan hasilnya diambil saat algoritme selesai.

Jalankan algoritma variasi berulang

Pekerjaan hybrid memberi Anda alat untuk menjalankan algoritme klasik kuantum berulang. Untuk masalah kuantum murni, gunakan tugas kuantum atau sekumpulan tugas kuantum. Akses prioritas ke QPU tertentu paling bermanfaat untuk algoritme variasional yang berjalan lama yang membutuhkan beberapa panggilan berulang ke QPU dengan pemrosesan klasik di antaranya.

Debug menggunakan mode lokal

Sebelum Anda menjalankan pekerjaan hybrid pada QPU, disarankan untuk pertama kali menjalankan simulator SV1 untuk mengonfirmasi bahwa itu berjalan seperti yang diharapkan. Untuk pengujian skala kecil, Anda dapat menjalankan dengan mode lokal untuk iterasi cepat dan debugging.

Tingkatkan reproduktifitas dengan Bring your own container (BYOC)

Buat eksperimen yang dapat direproduksi dengan mengenkapsulasi perangkat lunak Anda dan dependensinya dalam lingkungan kontainerisasi. Dengan mengemas semua kode, dependensi, dan pengaturan Anda dalam wadah, Anda mencegah potensi konflik dan masalah pembuatan versi.

Simulator terdistribusi multi-instance

Untuk menjalankan sejumlah besar sirkuit, pertimbangkan untuk menggunakan dukungan MPI bawaan untuk menjalankan simulator lokal pada beberapa instance dalam satu pekerjaan hybrid. Untuk informasi selengkapnya, lihat simulator tertanam.

Gunakan sirkuit parametrik

Sirkuit parametrik yang Anda kirimkan dari pekerjaan hybrid secara otomatis dikompilasi pada QPU tertentu menggunakan kompilasi parametrik untuk meningkatkan runtime algoritme Anda.

Pos pemeriksaan secara berkala

Untuk pekerjaan hybrid yang berjalan lama, disarankan untuk secara berkala menyimpan keadaan perantara algoritma.

Untuk contoh lebih lanjut, kasus penggunaan, dan praktik terbaik, lihat contoh Amazon GitHub Braket.

Jalankan pekerjaan hybrid dengan Amazon Braket Hybrid Jobs

Untuk menjalankan pekerjaan hybrid dengan Amazon Braket Hybrid Jobs, Anda harus terlebih dahulu menentukan algoritme Anda. Anda dapat mendefinisikannya dengan menulis skrip algoritma dan, secara opsional, file ketergantungan lainnya menggunakan Amazon Braket Python SDK atau. PennyLane Jika Anda ingin menggunakan pustaka lain (open source atau proprietary), Anda dapat menentukan gambar kontainer kustom Anda sendiri menggunakan Docker, yang menyertakan pustaka ini. Untuk informasi selengkapnya, lihat Membawa wadah Anda sendiri (BYOC).

Dalam kedua kasus tersebut, selanjutnya Anda membuat pekerjaan hybrid menggunakan Amazon BraketAPI, tempat Anda menyediakan skrip atau wadah algoritme Anda, pilih perangkat kuantum

target yang akan digunakan oleh hybrid, lalu pilih dari berbagai pengaturan opsional. Nilai default yang disediakan untuk pengaturan opsional ini berfungsi untuk sebagian besar kasus penggunaan. Agar perangkat target dapat menjalankan Hybrid Job Anda, Anda memiliki pilihan antara QPU, simulator sesuai permintaan (sepertiSV1, DM1 atauTN1), atau instance pekerjaan hybrid klasik itu sendiri. Dengan simulator sesuai permintaan atau QPU, wadah pekerjaan hybrid Anda membuat panggilan API ke perangkat jarak jauh. Dengan simulator tertanam, simulator tertanam dalam wadah yang sama dengan skrip algoritme Anda. Simulator petir dari PennyLane disematkan dengan wadah pekerjaan hibrida bawaan bawaan untuk Anda gunakan. Jika Anda menjalankan kode Anda menggunakan PennyLane simulator tertanam atau simulator khusus, Anda dapat menentukan jenis instance serta berapa banyak instance yang ingin Anda gunakan. Lihat halaman Harga Amazon Braket untuk biaya yang terkait dengan setiap pilihan.

Jika perangkat target Anda adalah simulator sesuai permintaan atau tertanam, Amazon Braket mulai menjalankan pekerjaan hybrid segera. Ini memutar instance pekerjaan hibrida (Anda dapat menyesuaikan jenis instans dalam API panggilan), menjalankan algoritme Anda, menulis hasilnya ke Amazon S3, dan melepaskan sumber daya Anda. Rilis sumber daya ini memastikan bahwa Anda hanya membayar untuk apa yang Anda gunakan.

Jumlah total pekerjaan hibrida bersamaan per unit pemrosesan kuantum (QPU) dibatasi. Saat ini, hanya satu pekerjaan hybrid yang dapat berjalan pada QPU pada waktu tertentu. Antrian digunakan untuk mengontrol jumlah pekerjaan hibrida yang diizinkan untuk dijalankan agar tidak melebihi batas yang diizinkan. Jika perangkat target Anda adalah QPU, pekerjaan hybrid Anda terlebih dahulu memasuki antrian pekerjaan QPU yang dipilih. Amazon Braket memutar instance pekerjaan hybrid yang diperlukan dan menjalankan pekerjaan hybrid Anda di perangkat. Selama algoritme Anda,

pekerjaan hibrida Anda memiliki akses prioritas, yang berarti bahwa tugas kuantum dari pekerjaan hibrida Anda berjalan di depan tugas kuantum Braket lainnya yang antri di perangkat, asalkan tugas kuantum pekerjaan diserahkan ke QPU setiap beberapa menit sekali. Setelah pekerjaan hybrid Anda selesai, sumber daya dilepaskan, artinya Anda hanya membayar untuk apa yang Anda gunakan.

> **Note**
>
> Perangkat bersifat regional dan pekerjaan hybrid Anda berjalan Wilayah AWS sama dengan perangkat utama Anda.

Dalam skenario target simulator dan QPU, Anda memiliki opsi untuk menentukan metrik algoritma khusus, seperti energi Hamiltonian Anda, sebagai bagian dari algoritme Anda. Metrik ini secara otomatis dilaporkan ke Amazon CloudWatch dan dari sana, mereka ditampilkan hampir real-time di konsol Amazon Braket.

> **Note**
>
> Jika Anda ingin menggunakan instance berbasis GPU, pastikan untuk menggunakan salah satu simulator berbasis GPU yang tersedia dengan simulator tertanam di Braket (misalnya,). `lightning.gpu` Jika Anda memilih salah satu simulator tertanam berbasis CPU (misalnya,, ataubraket:default-simulator)`lightning.qubit`, GPU tidak akan digunakan dan Anda mungkin dikenakan biaya yang tidak perlu.

Buat Job Hybrid pertama Anda

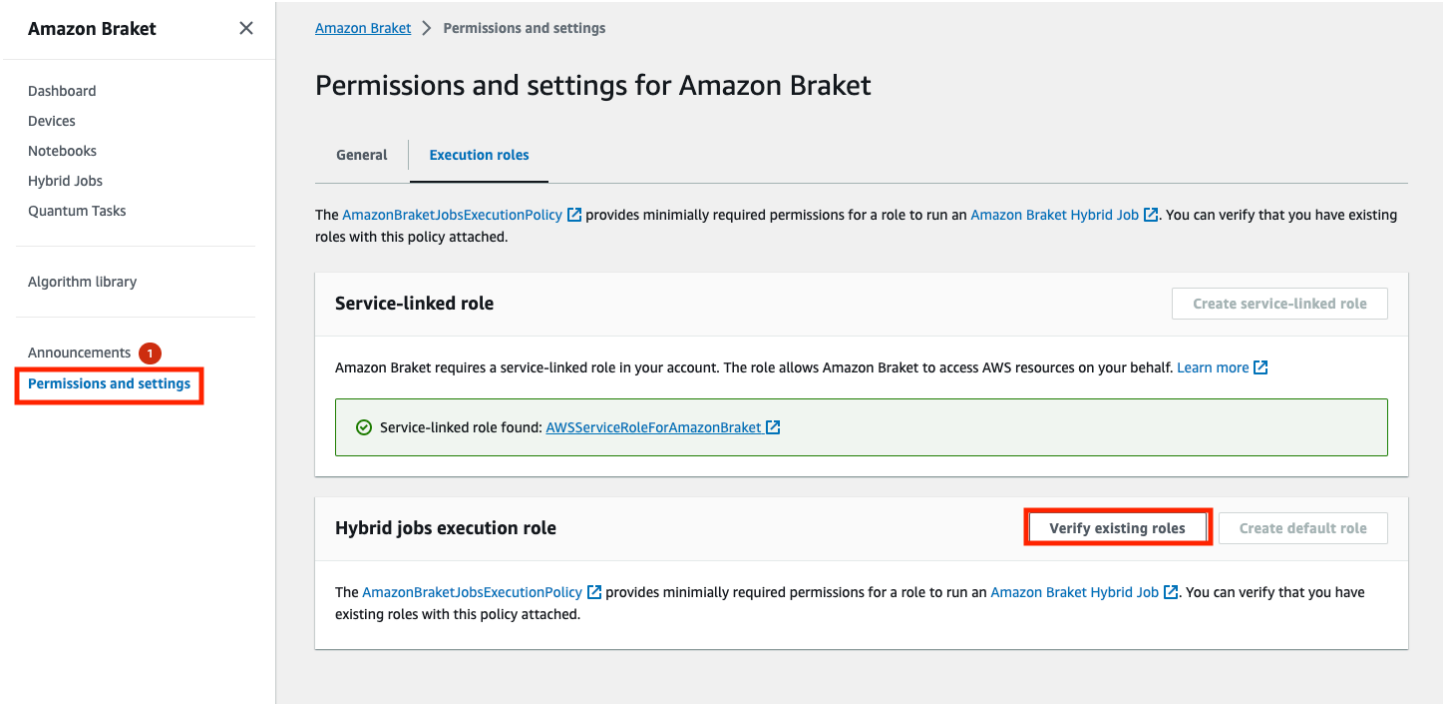
Bagian ini menunjukkan cara membuat Job Hybrid menggunakan skrip Python. Atau, untuk membuat pekerjaan hybrid dari kode Python lokal, seperti lingkungan pengembangan terintegrasi (IDE) pilihan Anda atau notebook Braket, lihat. Jalankan kode lokal Anda sebagai pekerjaan hibrida

Di bagian ini:

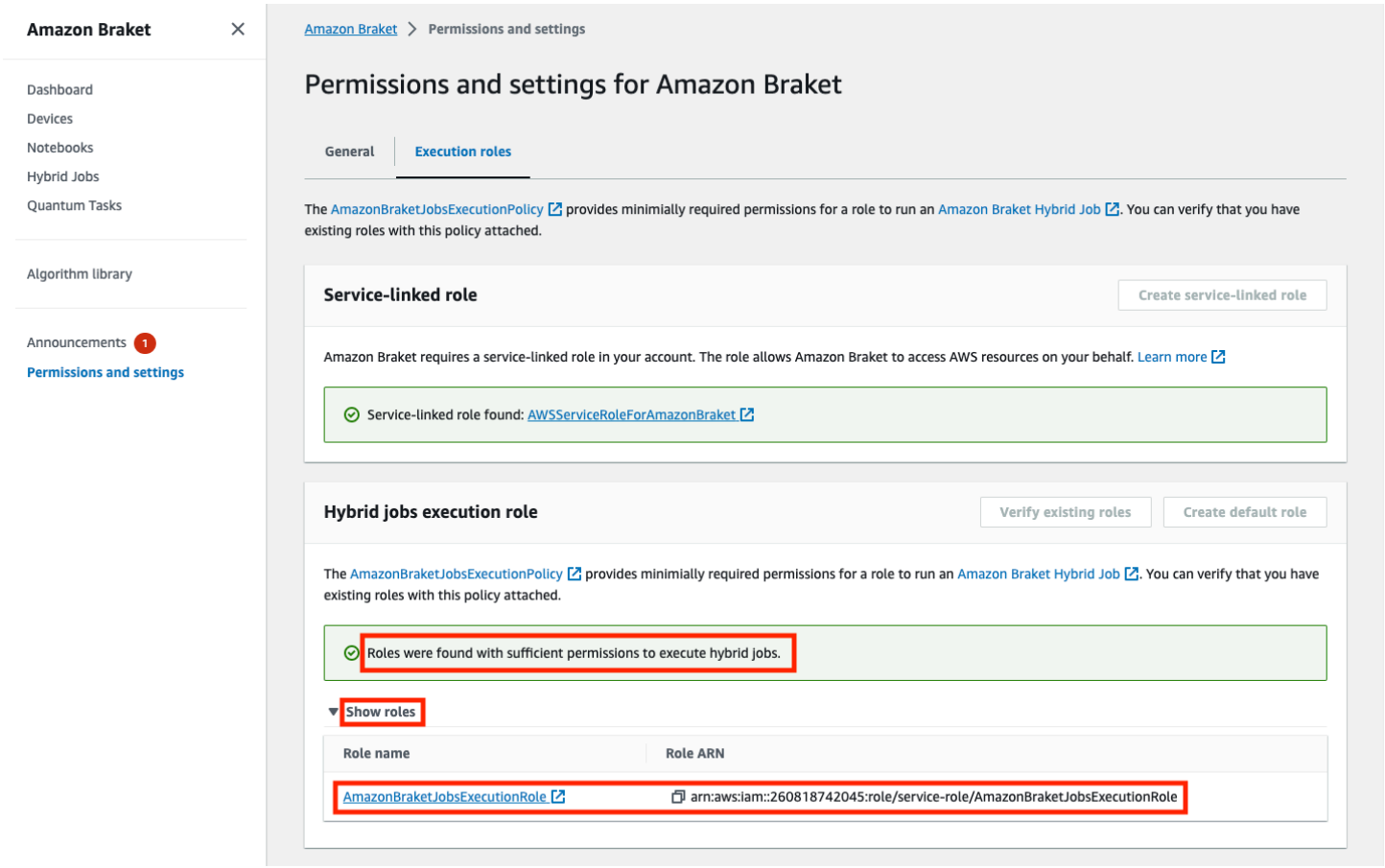
- Tetapkan izin
- Buat dan jalankan
- Memantau hasil

Tetapkan izin

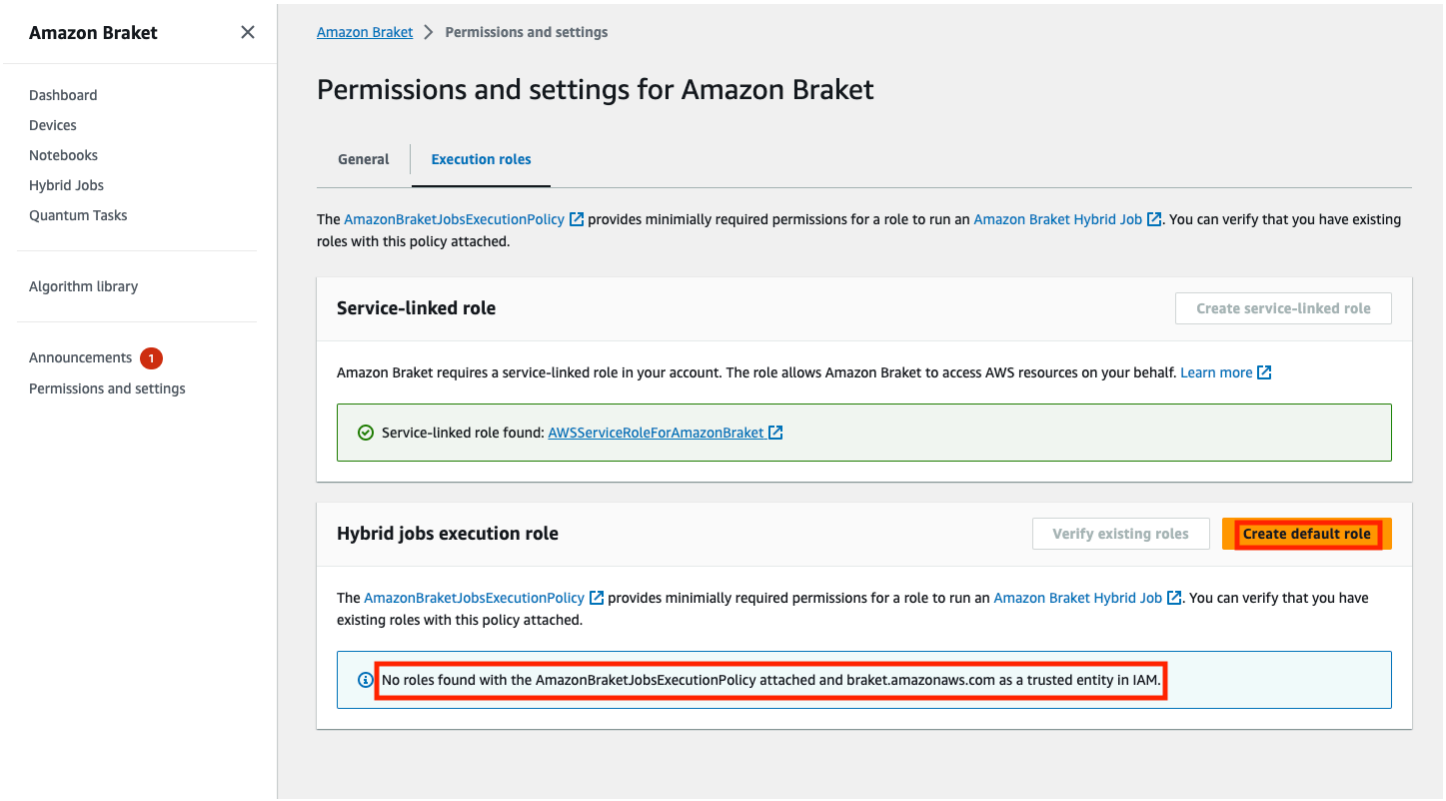
Sebelum Anda menjalankan pekerjaan hybrid pertama Anda, Anda harus memastikan bahwa Anda memiliki izin yang cukup untuk melanjutkan tugas ini. Untuk menentukan bahwa Anda memiliki izin yang benar, pilih Izin dari menu di sisi kiri Konsol Braket. Halaman Manajemen Izin untuk Amazon Braket membantu Anda memverifikasi apakah salah satu peran yang ada memiliki izin yang cukup untuk menjalankan pekerjaan hibrida atau memandu Anda melalui pembuatan peran default yang dapat digunakan untuk menjalankan pekerjaan hibrida jika Anda belum memiliki peran seperti itu.



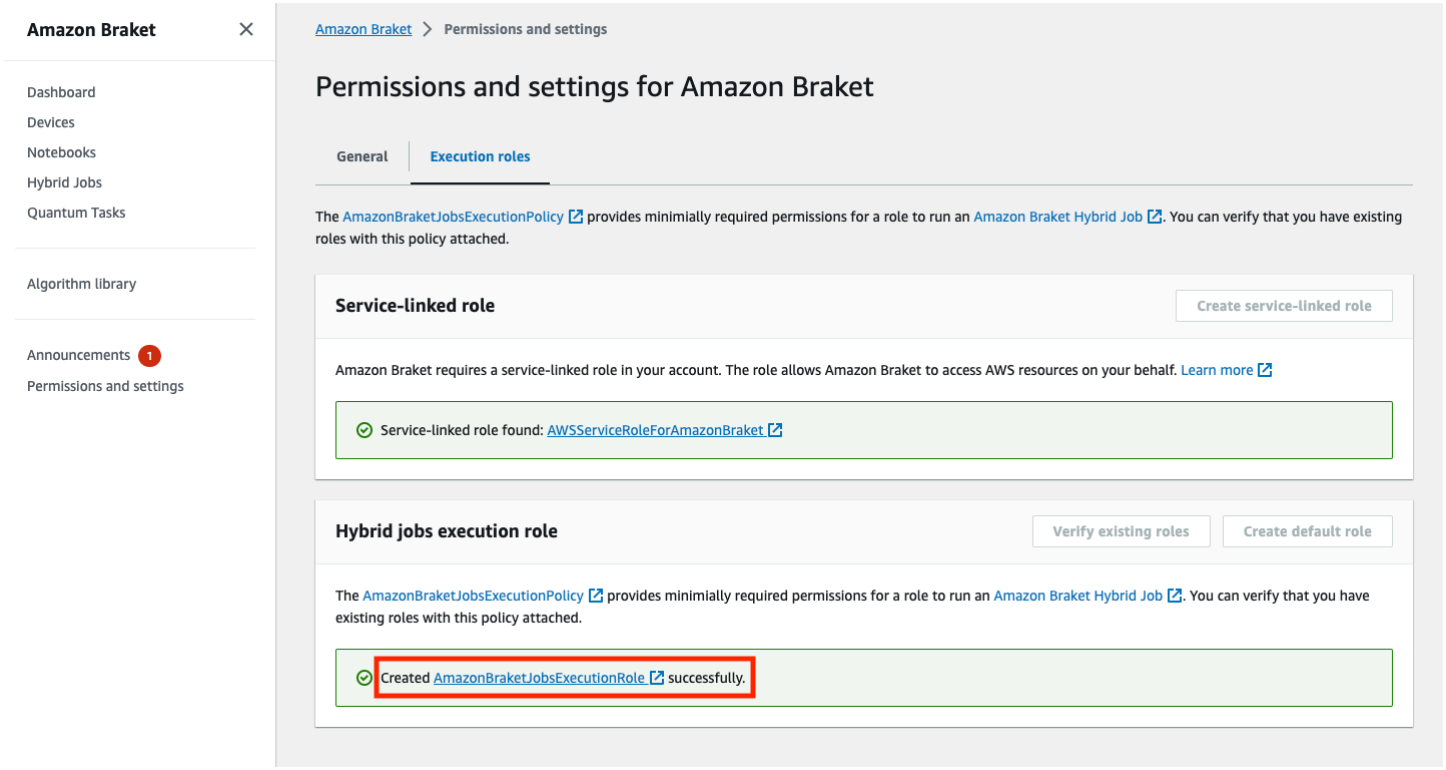
Untuk memverifikasi bahwa Anda memiliki peran dengan izin yang cukup untuk menjalankan pekerjaan hibrida, pilih tombol Verifikasi peran yang ada. Jika ya, Anda mendapat pesan bahwa peran itu ditemukan. Untuk melihat nama peran dan ARN peran mereka, pilih tombol Tampilkan peran.



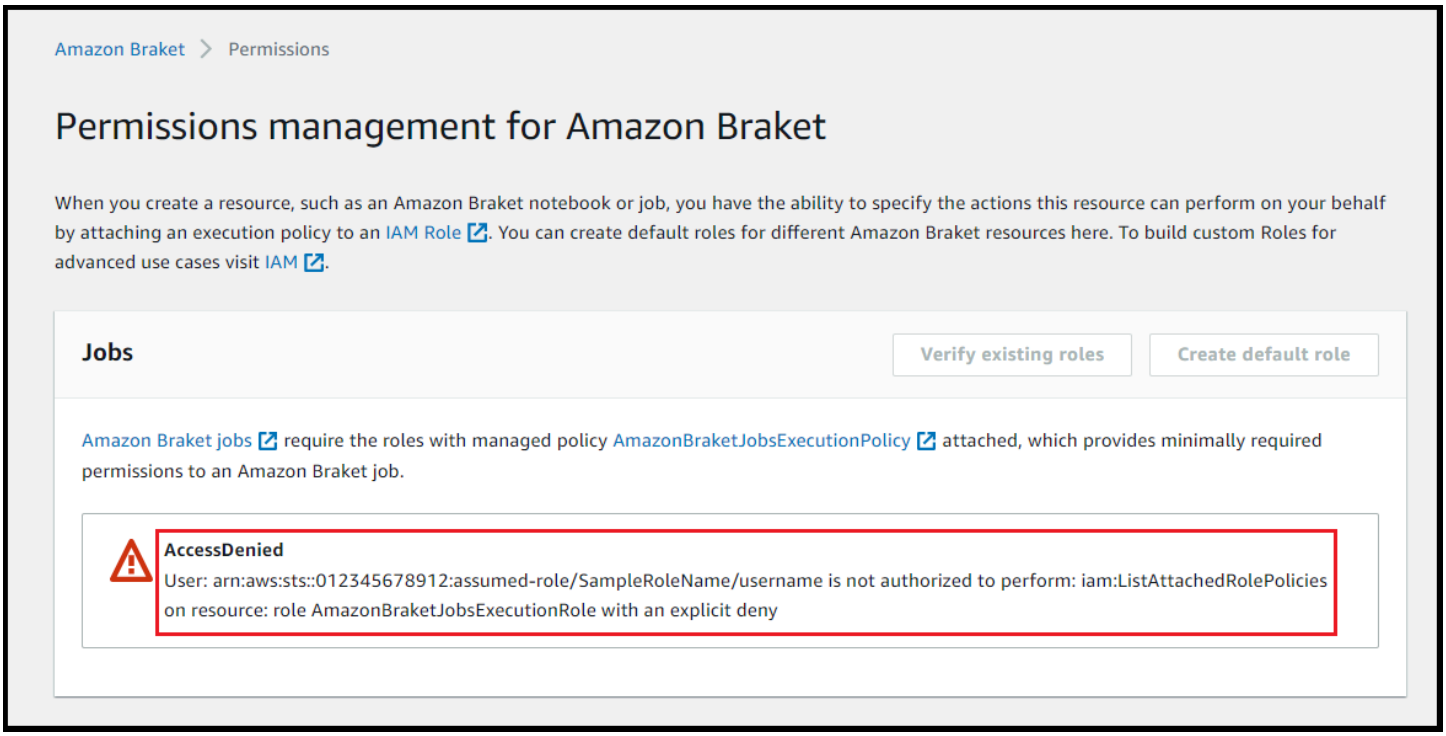
Jika Anda tidak memiliki peran dengan izin yang cukup untuk menjalankan pekerjaan hibrida, Anda mendapatkan pesan bahwa peran tersebut tidak ditemukan. Pilih tombol Buat peran default untuk mendapatkan peran dengan izin yang memadai.



Jika peran berhasil dibuat, Anda mendapatkan pesan yang mengonfirmasi hal ini.

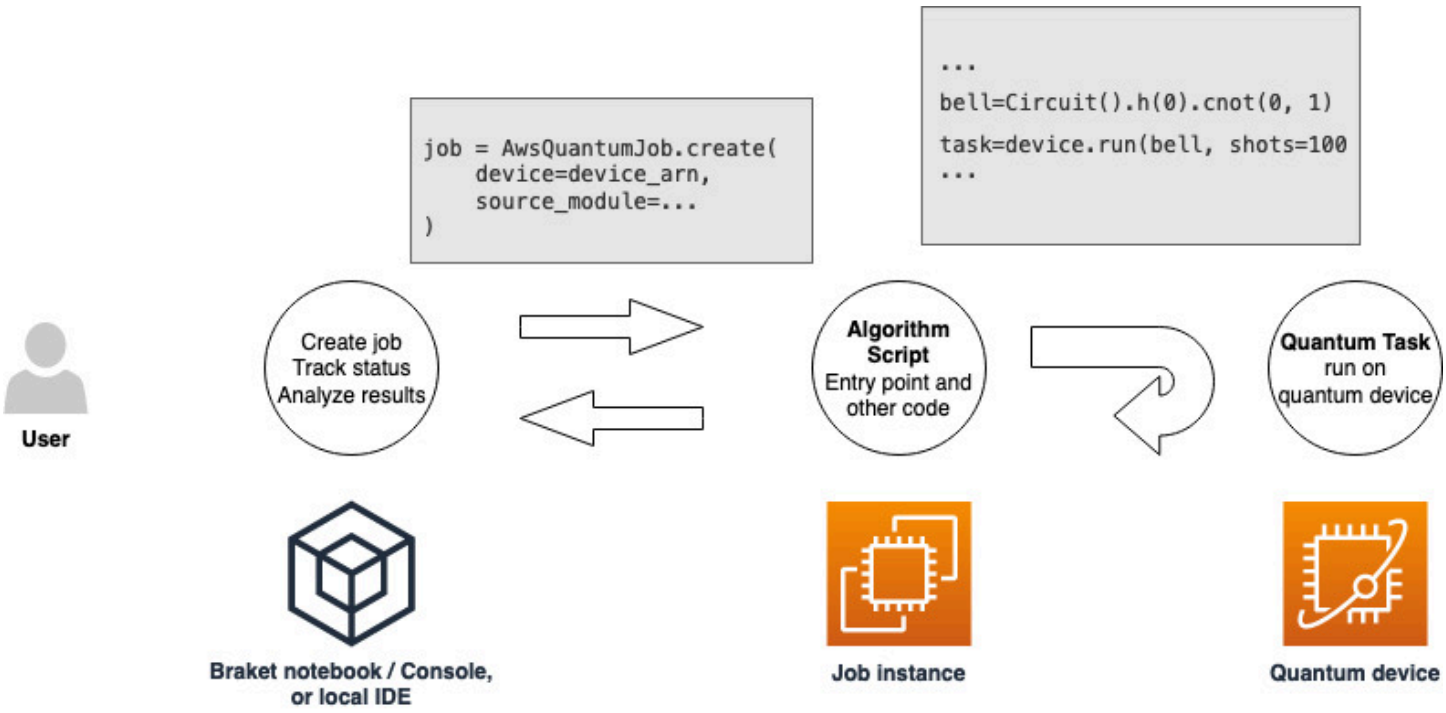


Jika Anda tidak memiliki izin untuk mengajukan pertanyaan ini, Anda akan ditolak aksesnya. Dalam hal ini, hubungi AWS administrator internal Anda.



Buat dan jalankan

Setelah Anda memiliki peran dengan izin untuk menjalankan pekerjaan hibrida, Anda siap untuk melanjutkan. Bagian kunci dari pekerjaan hybrid Braket pertama Anda adalah skrip algoritma. Ini mendefinisikan algoritma yang ingin Anda jalankan dan berisi logika klasik dan tugas kuantum yang merupakan bagian dari algoritma Anda. Selain skrip algoritme Anda, Anda dapat menyediakan file ketergantungan lainnya. Skrip algoritma bersama dengan dependensinya disebut modul sumber. Titik masuk mendefinisikan file atau fungsi pertama yang dijalankan di modul sumber Anda saat pekerjaan hybrid dimulai.



Pertama, pertimbangkan contoh dasar berikut dari skrip algoritme yang menciptakan lima status lonceng dan mencetak hasil pengukuran yang sesuai.

```
import os

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

def start_here():

    print("Test job started!")

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

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

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

Simpan file ini dengan nama algorithm_script.py di direktori kerja Anda saat ini di notebook Braket atau lingkungan lokal Anda. File algorithm_script.py memiliki `start_here()` titik masuk yang direncanakan.

Selanjutnya, buat file Python atau notebook Python di direktori yang sama dengan file algorithm_script.py. Skrip ini memulai pekerjaan hybrid dan menangani pemrosesan asinkron apa pun, seperti mencetak status atau hasil utama yang kami minati. Minimal, skrip ini perlu menentukan skrip pekerjaan hybrid dan perangkat utama Anda.

> **Note**
>
> Untuk informasi selengkapnya tentang cara membuat buku catatan Braket atau mengunggah file, seperti file algorithm_script.py, di direktori yang sama dengan notebook, lihat Menjalankan sirkuit pertama Anda menggunakan Amazon Braket Python SDK

Untuk kasus pertama dasar ini, Anda menargetkan simulator. Jenis perangkat kuantum apa pun yang Anda targetkan, simulator atau unit pemrosesan kuantum aktual (QPU), perangkat yang Anda tentukan `device` dalam skrip berikut digunakan untuk menjadwalkan pekerjaan hibrida dan tersedia untuk skrip algoritme sebagai variabel lingkungan. `AMZN_BRAKET_DEVICE_ARN`

> **Note**
>
> Anda hanya dapat menggunakan perangkat yang tersedia di Wilayah AWS pekerjaan hybrid Anda. Amazon Braket SDK auto memilih ini. Wilayah AWS Misalnya, pekerjaan hybrid di us-east-1 dapat IonQ menggunakanSV1,,, TN1 dan perangkatDM1, tetapi bukan perangkat. Rigetti

Jika Anda memilih komputer kuantum alih-alih simulator, Braket menjadwalkan pekerjaan hibrida Anda untuk menjalankan semua tugas kuantum mereka dengan akses prioritas.

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

job = AwsQuantumJob.create(
    Devices.Amazon.SV1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
```

```
    wait_until_complete=True
)
```

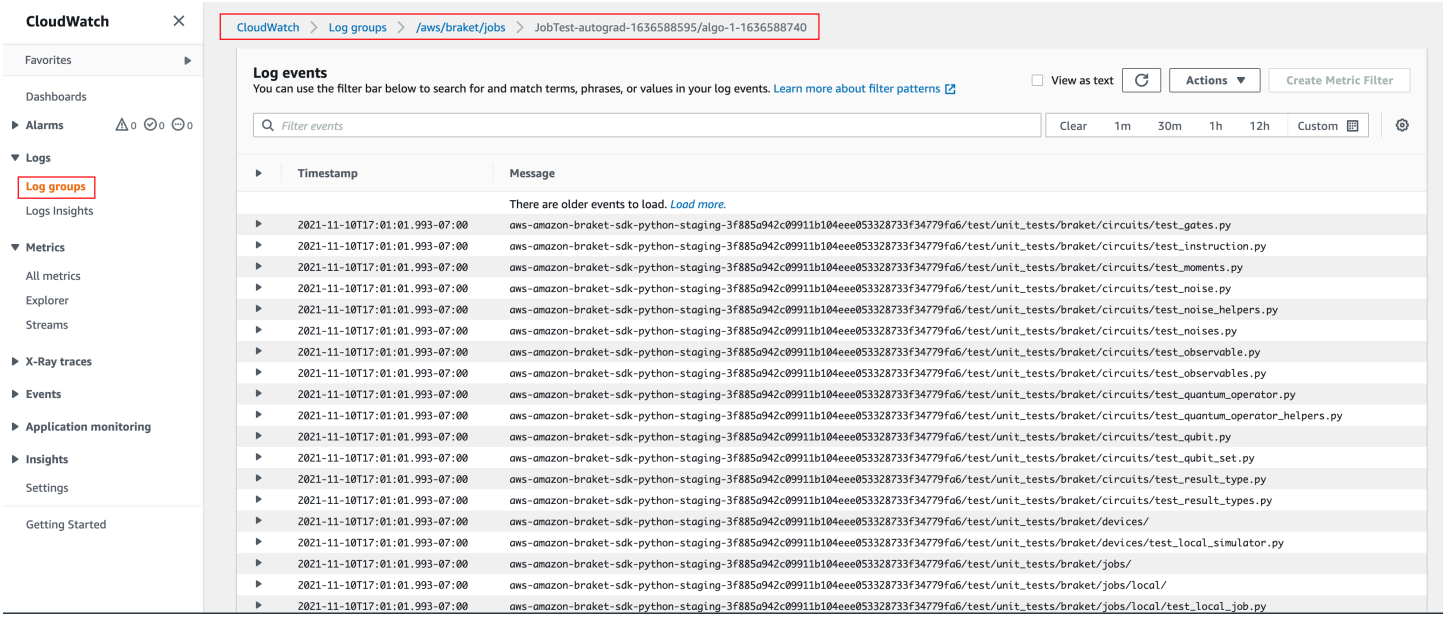
Parameter `wait_until_complete=True` menetapkan mode verbose sehingga pekerjaan Anda mencetak output dari pekerjaan yang sebenarnya saat sedang berjalan. Anda akan melihat output yang mirip dengan contoh berikut.

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

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

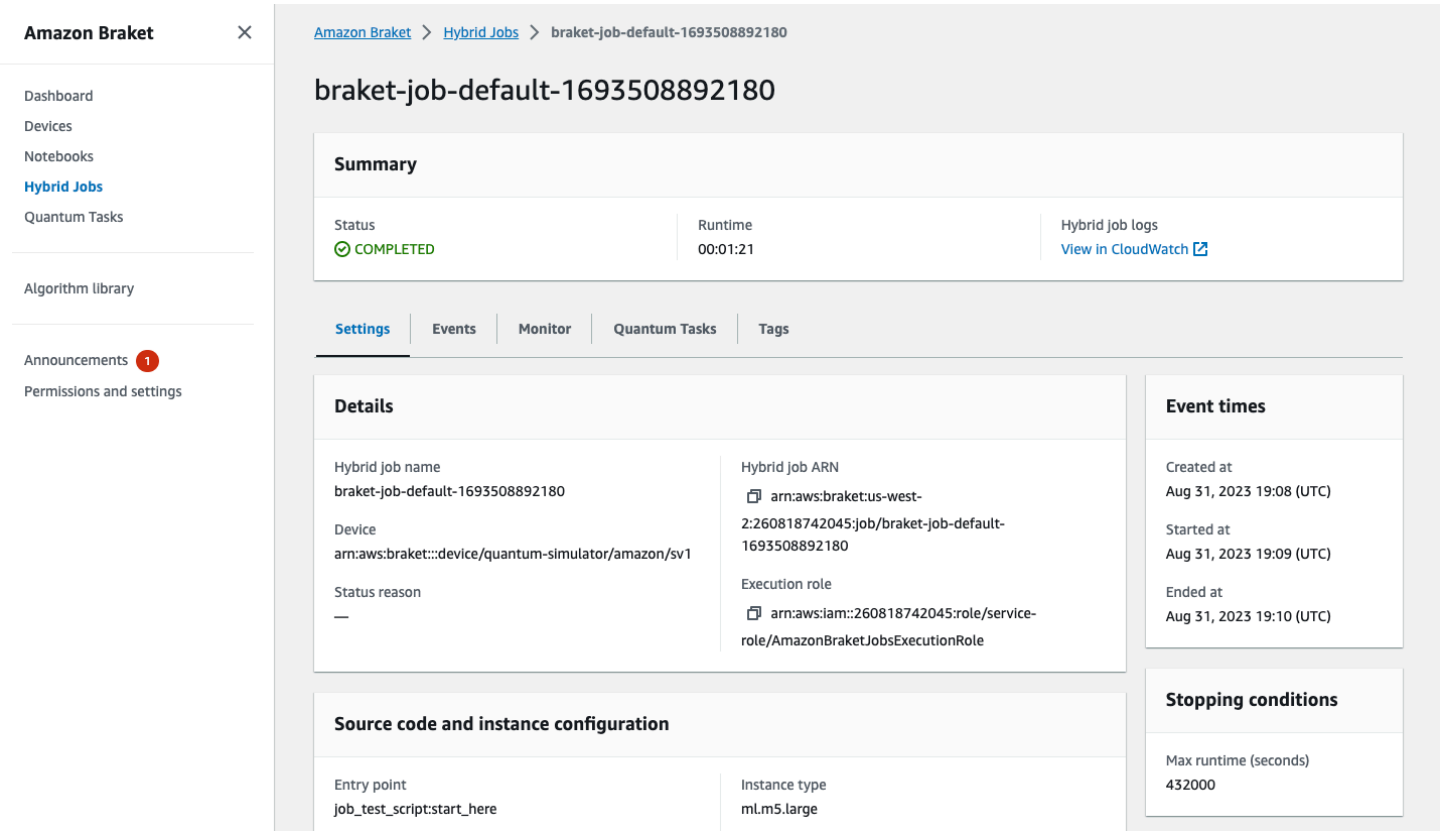
> **Note**
>
> Anda juga dapat menggunakan modul yang dibuat khusus dengan metode AwsQuantumJob.create dengan meneruskan lokasinya (baik jalur ke direktori atau file lokal, atau URI S3 dari file tar.gz). Untuk contoh kerja, lihat Parallelize_TRAINING_FOR_QML.ipynb file di folder pekerjaan hybrid di contoh Amazon Braket repo Github.

Memantau hasil

Atau, Anda dapat mengakses output log dari Amazon CloudWatch. Untuk melakukan ini, buka tab Grup log di menu kiri halaman detail pekerjaan, pilih grup log`aws/braket/jobs`, lalu pilih aliran log yang berisi nama pekerjaan. Dalam contoh di atas, ini adalah`braket-job-default-1631915042705/algo-1-1631915190`.



Anda juga dapat melihat status pekerjaan hybrid di konsol dengan memilih halaman Pekerjaan Hybrid dan kemudian memilih Pengaturan.



Pekerjaan hybrid Anda menghasilkan beberapa artefak di Amazon S3 saat berjalan. Nama bucket S3 default adalah `amazon-braket-<region>-<accountid>` dan isinya ada di `jobs/<jobname>/<timestamp>` direktori. Anda dapat mengonfigurasi lokasi S3 tempat artefak ini disimpan dengan menentukan yang berbeda `code_location` saat pekerjaan hybrid dibuat dengan Braket Python SDK.

> **Note**
>
> Bucket S3 ini harus terletak Wilayah AWS sama dengan skrip pekerjaan Anda.

`jobs/<jobname>/<timestamp>`Direktori berisi subfolder dengan output dari skrip titik masuk dalam `model.tar.gz` file. Ada juga direktori bernama `script` yang berisi artefak skrip algoritme Anda dalam sebuah `source.tar.gz` file. Hasil dari tugas kuantum Anda yang sebenarnya ada di direktori bernama`jobs/<jobname>/tasks`.

Input, output, variabel lingkungan, dan fungsi pembantu

Selain file atau file yang membentuk skrip algoritme lengkap Anda, pekerjaan hybrid Anda dapat memiliki input dan output tambahan. Saat pekerjaan hybrid Anda dimulai, Amazon Braket menyalin input yang disediakan sebagai bagian dari pembuatan pekerjaan hybrid ke dalam wadah yang menjalankan skrip algoritme. Saat pekerjaan hybrid selesai, semua output yang ditentukan selama algoritme akan disalin ke lokasi Amazon S3 yang ditentukan.

> **Note**
>
> Metrik algoritma dilaporkan secara real time dan tidak mengikuti prosedur keluaran ini.

AmazonBraket juga menyediakan beberapa variabel lingkungan dan fungsi pembantu untuk menyederhanakan interaksi dengan input dan output kontainer.

Bagian ini menjelaskan konsep kunci dari `AwsQuantumJob.create` fungsi yang disediakan oleh Amazon Braket Python SDK dan pemetaannya ke struktur file kontainer.

Di bagian ini:

Masukan

Data input: Data input dapat diberikan ke algoritma hybrid dengan menentukan file data input, yang diatur sebagai kamus, dengan `input_data` argumen. Pengguna mendefinisikan `input_data` argumen dalam `AwsQuantumJob.create` fungsi di SDK. Ini menyalin data input ke sistem file kontainer di lokasi yang diberikan oleh variabel lingkungan`"AMZN_BRAKET_INPUT_DIR"`. Untuk beberapa contoh bagaimana data input digunakan dalam algoritma hybrid, lihat QAOA dengan Amazon Braket Hybrid Jobs PennyLane dan dan Quantum machine learning di notebook Amazon Braket Hybrid Jobs Jupyter.

> **Note**
>
> Ketika data input besar (> 1GB), akan ada waktu tunggu yang lama sebelum pekerjaan hybrid diajukan. Hal ini disebabkan oleh fakta bahwa data input lokal pertama-tama akan diunggah ke bucket S3, kemudian jalur S3 akan ditambahkan ke permintaan pekerjaan hybrid, dan, akhirnya, permintaan pekerjaan hybrid diajukan ke layanan Braket.

Hyperparameters: Jika Anda masukhyperparameters, mereka tersedia di bawah variabel "AMZN_BRAKET_HP_FILE" lingkungan.

> **Note**
>
> Untuk informasi selengkapnya tentang cara membuat hyperparameters dan data input dan kemudian meneruskan informasi ini ke skrip pekerjaan hybrid, lihat bagian Use hyperparameters dan halaman github ini.

Checkpoints: Untuk menentukan pos pemeriksaan job-arn yang ingin Anda gunakan dalam pekerjaan hybrid baru, gunakan perintah. copy_checkpoints_from_job Perintah ini menyalin data pos pemeriksaan ke checkpoint_configs3Uri pekerjaan hybrid baru, membuatnya tersedia di jalur yang diberikan oleh variabel lingkungan AMZN_BRAKET_CHECKPOINT_DIR saat pekerjaan berjalan. Defaultnya adalahNone, artinya data pos pemeriksaan dari pekerjaan hybrid lain tidak akan digunakan dalam pekerjaan hybrid baru.

Output

Quantum Tasks: Hasil tugas kuantum disimpan di lokasi s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks S3.

Hasil pekerjaan: Semua yang disimpan skrip algoritme Anda ke direktori yang diberikan oleh variabel lingkungan "AMZN_BRAKET_JOB_RESULTS_DIR" disalin ke lokasi S3 yang ditentukan. output_data_config Jika Anda tidak menentukan nilai ini, defaultnya. s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data Kami menyediakan fungsi pembantu SDK **save_job_result**, yang dapat Anda gunakan untuk menyimpan hasil dengan nyaman dalam bentuk kamus saat dipanggil dari skrip algoritme Anda.

Checkpoints: Jika Anda ingin menggunakan pos pemeriksaan, Anda dapat menyimpannya di direktori yang diberikan oleh variabel lingkungan. `"AMZN_BRAKET_CHECKPOINT_DIR"` Anda juga dapat menggunakan fungsi `save_job_checkpoint` pembantu SDK sebagai gantinya.

Metrik algoritma: Anda dapat menentukan metrik algoritme sebagai bagian dari skrip algoritme yang dipancarkan ke CloudWatch Amazon dan ditampilkan secara real time di Amazon konsol Braket saat pekerjaan hybrid Anda berjalan. Untuk contoh cara menggunakan metrik algoritme, lihat Menggunakan Pekerjaan Hibrid Amazon Braket untuk menjalankan algoritma QAOA.

Variabel lingkungan

AmazonBraket menyediakan beberapa variabel lingkungan untuk menyederhanakan interaksi dengan input dan output kontainer. Kode berikut mencantumkan variabel lingkungan yang digunakan Braket.

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

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

Fungsi pembantu

AmazonBraket menyediakan beberapa fungsi pembantu untuk menyederhanakan interaksi dengan input dan output kontainer. Fungsi pembantu ini akan dipanggil dari dalam skrip algoritma yang digunakan untuk menjalankan Job Hybrid Anda. Contoh berikut menunjukkan cara menggunakannya.

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

Simpan hasil pekerjaan

Anda dapat menyimpan hasil yang dihasilkan oleh skrip algoritme sehingga tersedia dari objek pekerjaan hibrida dalam skrip pekerjaan hibrida serta dari folder keluaran di Amazon S3 (dalam file tar-zip bernama model.tar.gz).

Output harus disimpan dalam file menggunakan format JavaScript Object Notation (JSON). Jika data tidak dapat dengan mudah diserialisasikan ke teks, seperti dalam kasus array numpy, Anda dapat meneruskan opsi untuk membuat serial menggunakan format data acar. Lihat modul braket.jobs.data_persistence untuk detail selengkapnya.

Untuk menyimpan hasil pekerjaan hybrid, Anda menambahkan baris berikut yang dikomentari dengan #ADD ke skrip algoritme.

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

def start_here():
```

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

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

    results = []  #ADD

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

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

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

Anda kemudian dapat menampilkan hasil pekerjaan dari skrip pekerjaan Anda dengan menambahkan baris yang **`print(job.result())`**dikomentari dengan #ADD.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    device_arn="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
)

print(job.arn)
while job.state() not in AwsQuantumJob.TERMINAL_STATES:
    print(job.state())
    time.sleep(10)

print(job.state())
print(job.result())   #ADD
```

Dalam contoh ini, kami telah menghapus `wait_until_complete=True` untuk menekan output verbose. Anda dapat menambahkannya kembali untuk debugging. Ketika Anda menjalankan pekerjaan hibrida ini, ia mengeluarkan pengenal dan`job-arn`, diikuti oleh status pekerjaan hibrida setiap 10 detik hingga pekerjaan hibrida`COMPLETED`, setelah itu menunjukkan kepada Anda hasil sirkuit bel. Lihat contoh berikut ini.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-1234567890123
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47},..., {'00': 51, '11': 49}]}
```

Simpan dan mulai ulang pekerjaan hibrida menggunakan pos pemeriksaan

Anda dapat menyimpan iterasi perantara dari pekerjaan hybrid Anda menggunakan pos pemeriksaan. Dalam contoh skrip algoritma dari bagian sebelumnya, Anda akan menambahkan baris berikut yang dikomentari dengan #ADD untuk membuat file pos pemeriksaan.

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

def start_here():

    print("Test job starts!!!!!")

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

    #ADD the following code
    job_name = os.environ["AMZN_BRAKET_JOB_NAME"]
     save_job_checkpoint(
     checkpoint_data={"data": f"data for checkpoint from {job_name}"},
     checkpoint_file_suffix="checkpoint-1",
```

```
     ) #End of ADD

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

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

Saat Anda menjalankan pekerjaan hybrid, itu membuat file -checkpoint-1.json <jobname>di artefak pekerjaan hybrid Anda di direktori pos pemeriksaan dengan jalur default. `/opt/jobs/checkpoints` Skrip pekerjaan hybrid tetap tidak berubah kecuali Anda ingin mengubah jalur default ini.

Jika Anda ingin memuat pekerjaan hybrid dari pos pemeriksaan yang dihasilkan oleh pekerjaan hybrid sebelumnya, skrip algoritme menggunakan`from braket.jobs import load_job_checkpoint`. Logika untuk memuat dalam skrip algoritma Anda adalah sebagai berikut.

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

Setelah memuat pos pemeriksaan ini, Anda dapat melanjutkan logika berdasarkan konten yang dimuat`checkpoint-1`.

> **Note**
>
> Checkpoint_file_suffix harus cocok dengan akhiran yang ditentukan sebelumnya saat membuat pos pemeriksaan.

Skrip orkestrasi Anda perlu menentukan `job-arn` dari pekerjaan hybrid sebelumnya dengan baris yang dikomentari dengan #ADD.

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

Tentukan lingkungan untuk skrip algoritme Anda

AmazonBraket mendukung tiga lingkungan yang ditentukan oleh kontainer untuk skrip algoritme Anda:

- Sebuah wadah dasar (default, jika tidak `image_uri` ditentukan)
- Wadah dengan Tensorflow dan PennyLane
- Sebuah wadah dengan PyTorch dan PennyLane

Tabel berikut memberikan rincian tentang wadah dan pustaka yang disertakan.

Wadah Amazon Braket

Tipe	PennyLane dengan TensorFlow	PennyLane dengan PyTorch	Pennylane
Basis	292282985366.dkr.ecr.us-east-1.amazonaws.com /amazon-braket-tensorflow-jobs:latest	292282985366.dkr.ecr.us-west-2.amazonaws.com /amazon-braket-pytorch-jobs:latest	292282985366.dkr. ecr.us-west-2.amazonaws.com / amazon-braket-base-jobs:latest
Perpustakaan Warisan	• awscli • numpy • panda • scipy	• awscli • numpy • panda • scipy	
Perpustakaan Tambahan	• simulator default-rem amazon- • amazon-braket-pennylane-plugin • skema rem amazon • amazon-rem-sdk • ipykernel • keras • matplotlib	• simulator default-rem amazon- • amazon-braket-pennylane-plugin • skema rem amazon • amazon-rem-sdk • ipykernel • keras • matplotlib	• simulator default-rem amazon- • amazon-braket-pennylane-plugin • skema rem amazon • amazon-rem-sdk • awscli • boto3 • ipykernel

Tipe	PennyLane dengan TensorFlow	PennyLane dengan PyTorch	Pennylane
	• jaringanx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU petir • CuQuantum	• jaringanx • openbabel • PennyLane • protobuf • psi4 • rsa • PennyLane-GPU petir • CuQuantum	• matplotlib • jaringanx • numpy • openbabel • panda • PennyLane • protobuf • psi4 • rsa • scipy

Anda dapat melihat dan mengakses definisi wadah open source di aws/amazon-braket-containers. Pilih wadah yang paling cocok dengan kasus penggunaan Anda. Wadah harus berada di Wilayah AWS tempat Anda menjalankan pekerjaan hibrida Anda. Anda menentukan gambar kontainer saat membuat pekerjaan hybrid dengan menambahkan salah satu dari tiga argumen berikut ke `create(…)` panggilan Anda dalam skrip pekerjaan hybrid. Anda dapat menginstal dependensi tambahan ke dalam wadah yang Anda pilih saat runtime (dengan biaya startup atau runtime) karena kontainer Amazon Braket memiliki konektivitas internet. Contoh berikut adalah untuk Wilayah us-west-2.

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

Ini juga `image-uris` dapat diambil menggunakan `retrieve_image()` fungsi di Amazon Braket SDK. Contoh berikut menunjukkan cara mengambilnya dari Wilayah AWS us-west-2.

```
from braket.jobs.image_uris import retrieve_image, Framework
```

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

Gunakan hyperparameters

Anda dapat menentukan hiperparameter yang dibutuhkan oleh algoritme Anda, seperti tingkat pembelajaran atau ukuran langkah, saat Anda membuat pekerjaan hibrida. Nilai hyperparameter biasanya digunakan untuk mengontrol berbagai aspek algoritma, dan sering dapat disetel untuk mengoptimalkan kinerja algoritma. Untuk menggunakan hyperparameters dalam pekerjaan hybrid Braket, Anda perlu menentukan nama dan nilainya secara eksplisit sebagai kamus. Perhatikan bahwa nilai harus dari tipe data string. Anda menentukan nilai hyperparameter yang ingin Anda uji saat mencari set nilai optimal. Langkah pertama untuk menggunakan hyperparameters adalah mengatur dan mendefinisikan hyperparameters sebagai kamus, yang dapat dilihat pada kode berikut:

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

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

Anda kemudian akan meneruskan hyperparameters yang ditentukan dalam cuplikan kode yang diberikan di atas untuk digunakan dalam algoritme pilihan Anda dengan sesuatu yang terlihat seperti berikut:

```
import time
from braket.aws import AwsQuantumJob

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

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

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

> **Note**
>
> Untuk mempelajari lebih lanjut tentang data input, lihat bagian Input.

Hyperparameters kemudian akan dimuat ke dalam skrip pekerjaan hybrid menggunakan kode berikut:

```
import json
import os

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

> **Note**
>
> Untuk informasi selengkapnya tentang cara meneruskan informasi seperti data input dan perangkat arn ke skrip pekerjaan hybrid, lihat halaman github ini.

Beberapa panduan yang sangat berguna untuk belajar tentang cara menggunakan hyperparameters diberikan oleh QAOA dengan Amazon Braket Hybrid Jobs dan PennyLane dan Quantum machine learning dalam tutorial Amazon Braket Hybrid Jobs.

Konfigurasikan instance pekerjaan hybrid untuk menjalankan skrip algoritme Anda

Tergantung pada algoritma Anda, Anda mungkin memiliki persyaratan yang berbeda. Secara default, Amazon Braket menjalankan skrip algoritme Anda pada sebuah `ml.m5.large` instance. Namun, Anda dapat menyesuaikan jenis instance ini saat membuat pekerjaan hibrida menggunakan argumen impor dan konfigurasi berikut.

```
from braket.jobs.config import InstanceConfig

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

Jika Anda menjalankan simulasi tertanam dan telah menetapkan perangkat lokal dalam konfigurasi perangkat, Anda juga dapat meminta lebih dari satu instance InstanceConfig dengan menentukan InstanceCount dan menyetelnya menjadi lebih besar dari satu. Batas atas adalah 5. Misalnya, Anda dapat memilih 3 contoh sebagai berikut.

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

Saat Anda menggunakan beberapa instance, pertimbangkan untuk mendistribusikan pekerjaan hybrid Anda menggunakan fitur data parallel. Lihat contoh buku catatan berikut untuk detail selengkapnya tentang cara melihat contoh Braket ini.

Tiga tabel berikut mencantumkan jenis dan spesifikasi instans yang tersedia untuk instans komputasi standar, komputasi yang dioptimalkan, dan dipercepat.

> **Note**
>
> Untuk melihat kuota instans komputasi klasik default untuk Pekerjaan Hybrid, lihat halaman ini.

Contoh Standar	vCPU	Memori
ml.m5.large (default)	2	8 GiB
db.m5.xlarge	4	16 GiB
ml.m5.2xlarge	8	32 GiB
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB
db.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

Hitung Instans yang Dioptimalkan	vCPU	Memori
ml.c4.xlarge	4	7,5 GiB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB

Hitung Instans yang Dioptimalkan	vCPU	Memori
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.xlarge	4	10,5 GiB
ml.c5n.2xbesar	8	21 GiB
ml.c5n.4xbesar	16	42 GiB
ml.c5n.9xlarge	36	96 GiB
ml.c5n.18xlarge	72	192 GiB

Instans Komputasi yang Dipercepat	vCPU	Memori
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB
ml.g4dn.xlarge	4	16 GiB

Instans Komputasi yang Dipercepat	vCPU	Memori
ml.g4dn.2xbesar	8	32 GiB
ml.g4dn.4xbesar	16	64 GiB
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

> **Note**
>
> instans p3 tidak tersedia di us-west-1. Jika pekerjaan hybrid Anda tidak dapat menyediakan kapasitas komputasi ML yang diminta, gunakan Wilayah lain.

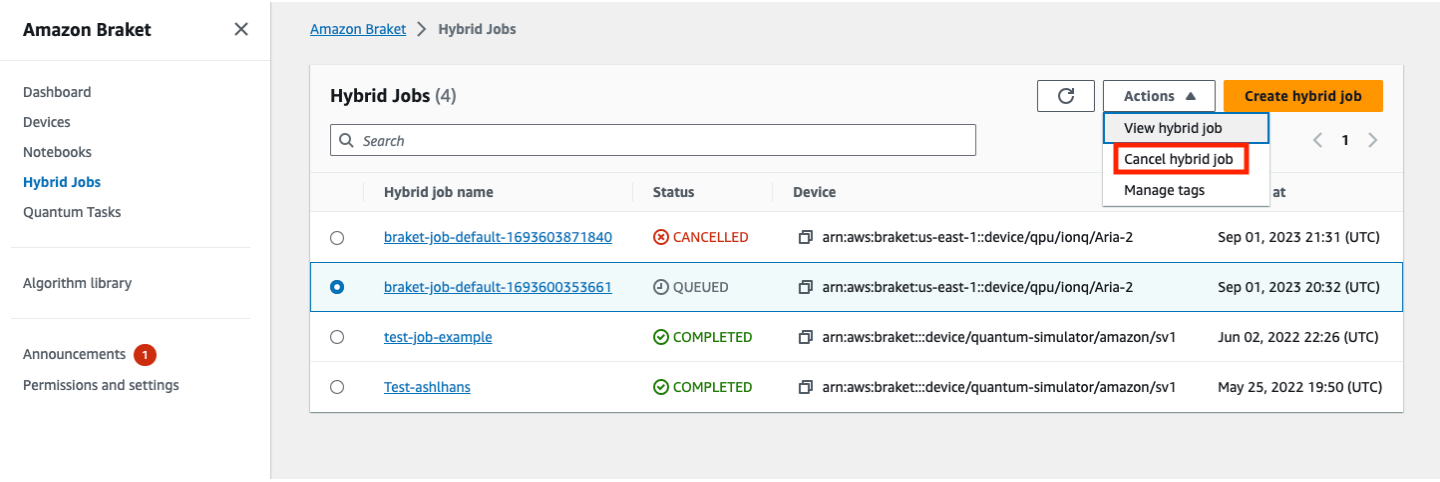
Setiap instans menggunakan konfigurasi default penyimpanan data (SSD) sebesar 30 GB. Tetapi Anda dapat menyesuaikan penyimpanan dengan cara yang sama seperti Anda mengkonfigurasi`instanceType`. Contoh berikut menunjukkan cara meningkatkan total penyimpanan menjadi 50 GB.

```
from braket.jobs.config import InstanceConfig

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

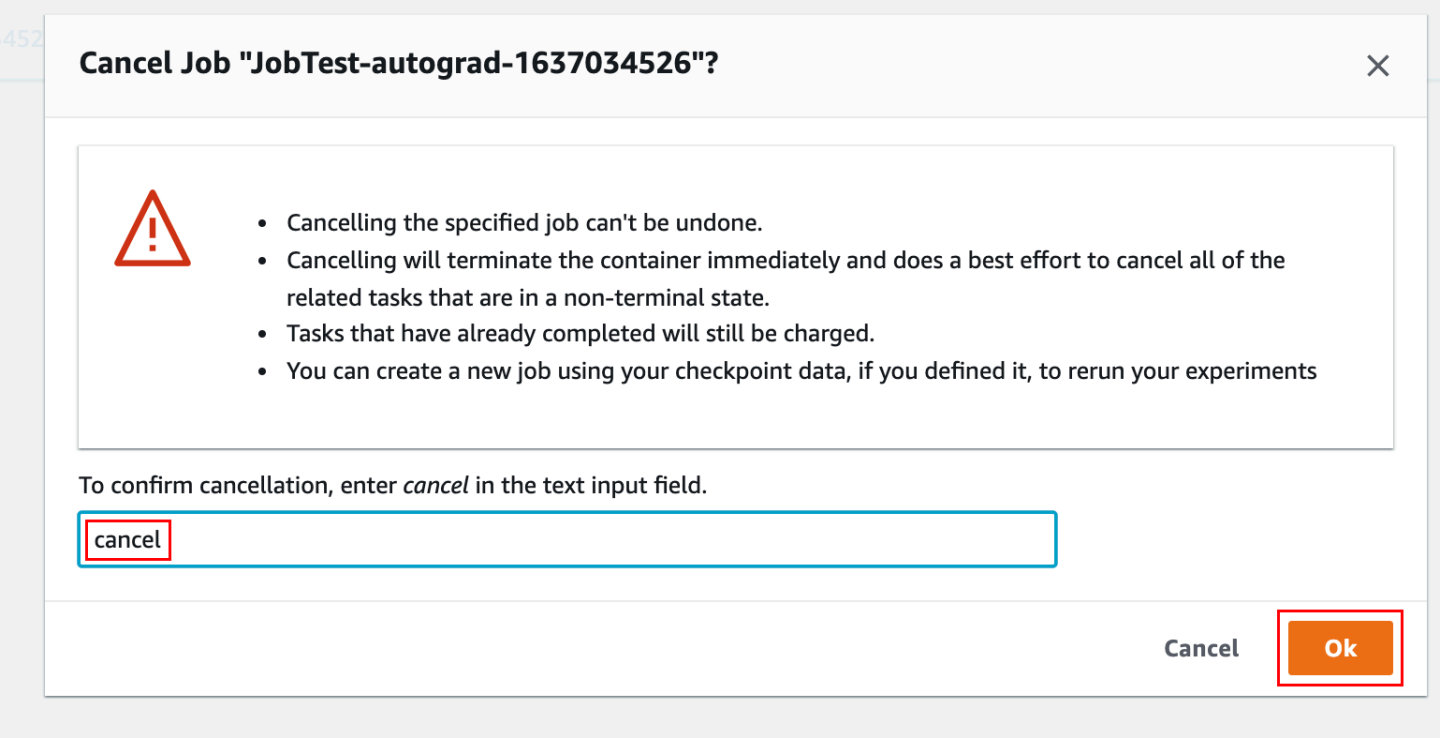
Membatalkan Job Hybrid

Anda mungkin perlu membatalkan pekerjaan hibrida dalam keadaan non-terminal. Ini dapat dilakukan di konsol atau dengan kode.

Untuk membatalkan pekerjaan hibrida Anda di konsol, pilih pekerjaan hibrida yang akan dibatalkan dari halaman Pekerjaan Hybrid, lalu pilih Batalkan pekerjaan hibrida dari menu tarik-turun Tindakan.



Untuk mengonfirmasi pembatalan, masukkan batal ke bidang input saat diminta dan kemudian pilih OK.



Untuk membatalkan pekerjaan hybrid Anda menggunakan kode dari SDK Python Braket, gunakan `job_arn` untuk mengidentifikasi pekerjaan hybrid dan kemudian panggil perintah di atasnya seperti `cancel` yang ditunjukkan pada kode berikut.

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

```
job.cancel()
```

`cancel`Perintah segera menghentikan wadah pekerjaan hibrida klasik dan melakukan upaya terbaik untuk membatalkan semua tugas kuantum terkait yang masih dalam keadaan non-terminal.

Menggunakan kompilasi parametrik untuk mempercepat Pekerjaan Hybrid

Amazon Braket mendukung kompilasi parametrik pada QPU tertentu. Ini memungkinkan Anda untuk mengurangi overhead yang terkait dengan langkah kompilasi yang mahal secara komputasi dengan mengkompilasi sirkuit hanya sekali dan tidak untuk setiap iterasi dalam algoritme hybrid Anda. Ini dapat meningkatkan runtime secara dramatis untuk Hybrid Jobs, karena Anda menghindari kebutuhan untuk mengkompilasi ulang sirkuit Anda di setiap langkah. Cukup kirimkan sirkuit parametris ke salah satu QPU kami yang didukung sebagai Braket Hybrid Job. Untuk pekerjaan hybrid yang berjalan lama, Braket secara otomatis menggunakan data kalibrasi yang diperbarui dari penyedia perangkat keras saat menyusun sirkuit Anda untuk memastikan hasil dengan kualitas terbaik.

Untuk membuat rangkaian parametrik, pertama-tama Anda harus memberikan parameter sebagai input dalam skrip algoritme Anda. Dalam contoh ini, kami menggunakan sirkuit parametrik kecil dan mengabaikan pemrosesan klasik antara setiap iterasi. Untuk beban kerja tipikal, Anda akan mengirimkan banyak sirkuit dalam batch dan melakukan pemrosesan klasik seperti memperbarui parameter di setiap iterasi.

```
import os

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

def start_here():

    print("Test job started.")

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

    circuit = Circuit().rx(0, FreeParameter("theta"))
    parameter_list = [0.1, 0.2, 0.3]

    for parameter in parameter_list:
```

```
        result = device.run(circuit, shots=1000, inputs={"theta": parameter})

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

Anda dapat mengirimkan skrip algoritma untuk dijalankan sebagai Job Hybrid dengan skrip pekerjaan berikut. Saat menjalankan Hybrid Job pada QPU yang mendukung kompilasi parametrik, sirkuit dikompilasi hanya pada proses pertama. Dalam proses berikutnya, sirkuit yang dikompilasi digunakan kembali, meningkatkan kinerja runtime dari Hybrid Job tanpa baris kode tambahan.

```
from braket.aws import AwsQuantumJob

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

> **Note**
>
> Kompilasi parametrik didukung pada semua QPU superkonduktor berbasis gerbang dari Rigetti Computing dan Oxford Quantum Circuits dengan pengecualian program tingkat pulsa.

Gunakan PennyLane dengan Amazon Braket

Algoritma hibrida adalah algoritma yang berisi instruksi klasik dan kuantum. Instruksi klasik dijalankan pada perangkat keras klasik (instans EC2 atau laptop Anda), dan instruksi kuantum dijalankan baik pada simulator atau pada komputer kuantum. Kami menyarankan Anda menjalankan algoritma hybrid menggunakan fitur Hybrid Jobs. Untuk informasi selengkapnya, lihat Kapan menggunakan Pekerjaan Amazon Braket.

AmazonBraket memungkinkan Anda untuk mengatur dan menjalankan algoritma kuantum hybrid dengan bantuan PennyLane plugin Braket, atau dengan Amazon Braket Amazon Python SDK dan contoh repositori notebook. Amazon Braket contoh notebook, berdasarkan SDK, memungkinkan Anda untuk mengatur dan menjalankan algoritma hybrid tertentu tanpa plugin. PennyLane Namun, kami merekomendasikan PennyLane karena memberikan pengalaman yang lebih kaya.

Tentang algoritma kuantum hibrida

Algoritma kuantum hibrida penting bagi industri saat ini karena perangkat komputasi kuantum kontemporer umumnya menghasilkan noise, dan karenanya, kesalahan. Setiap gerbang kuantum

yang ditambahkan ke komputasi meningkatkan kemungkinan menambahkan noise; oleh karena itu, algoritma yang berjalan lama dapat kewalahan oleh noise, yang menghasilkan komputasi yang salah.

Algoritma kuantum murni seperti Shor (contoh Estimasi Fase Kuantum) atau Grover (contoh Grover) membutuhkan ribuan, atau jutaan, operasi. Karena alasan ini, mereka tidak praktis untuk perangkat kuantum yang ada, yang umumnya disebut perangkat (NISQ) kuantum berisikskala menengah.

Dalam algoritme kuantum hibrid, unit pemrosesan kuantum (QPU) bekerja sebagai co-prosesor untuk CPU klasik, khususnya untuk mempercepat penghitungan tertentu dalam algoritme klasik. Eksekusi sirkuit menjadi jauh lebih pendek, dalam jangkauan kemampuan perangkat saat ini.

Amazon Braket dengan PennyLane

AmazonBraket menyediakan dukungan untuk PennyLane, kerangka kerja perangkat lunak open-source yang dibangun di sekitar konsep pemrograman kuantum yang dapat dibedakan. Anda dapat menggunakan kerangka kerja ini untuk melatih sirkuit kuantum dengan cara yang sama seperti Anda melatih jaringan saraf untuk menemukan solusi untuk masalah komputasi dalam kimia kuantum, pembelajaran mesin kuantum, dan optimasi.

PennyLane Perpustakaan menyediakan antarmuka ke alat pembelajaran mesin yang sudah dikenal, termasuk PyTorch dan TensorFlow, untuk membuat sirkuit kuantum pelatihan cepat dan intuitif.

- PennyLane Perpustakaan -- PennyLane sudah diinstal sebelumnya di notebook Amazon Braket. Untuk akses ke perangkat Amazon Braket dari PennyLane, buka buku catatan dan impor PennyLane perpustakaan dengan perintah berikut.

```
import pennylane as qml
```

Notebook tutorial membantu Anda memulai dengan cepat. Atau, Anda dapat menggunakan PennyLane Amazon Braket dari IDE pilihan Anda.

- PennyLane Plugin Amazon Braket — Untuk menggunakan IDE Anda sendiri, Anda dapat menginstal PennyLane plugin Amazon Braket secara manual. Plugin terhubung PennyLane dengan Amazon Braket Python SDK, sehingga Anda dapat menjalankan sirkuit di perangkat Braket. PennyLane Amazon Untuk menginstal PennyLane plugin, gunakan perintah berikut.

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

Contoh berikut menunjukkan cara mengatur akses ke perangkat Amazon Braket di: PennyLane

```
# to use SV1
import pennylane as qml
sv1 = qml.device("braket.aws.qubit", device_arn="arn:aws:braket:::device/quantum-
simulator/amazon/sv1", wires=2)

# to run a circuit:
@qml.qnode(sv1)
def circuit(x):
    qml.RZ(x, wires=0)
    qml.CNOT(wires=[0,1])
    qml.RY(x, wires=1)
    return qml.expval(qml.PauliZ(1))

result = circuit(0.543)


#To use the local sim:
local = qml.device("braket.local.qubit", wires=2)
```

Untuk contoh tutorial dan informasi selengkapnya PennyLane, lihat repositori contoh Amazon Braket.

PennyLane Plugin Amazon Braket memungkinkan Anda untuk beralih antara Amazon Braket QPU dan perangkat simulator tertanam PennyLane dengan satu baris kode. Ini menawarkan dua perangkat kuantum Amazon Braket untuk bekerja dengan PennyLane:

- `braket.aws.qubit`untuk berjalan dengan perangkat kuantum layanan Amazon Braket, termasuk QPU dan simulator
- `braket.local.qubit`untuk berjalan dengan simulator lokal Amazon Braket SDK

PennyLane Plugin Amazon Braket adalah open source. Anda dapat menginstalnya dari GitHub repositori PennyLane Plugin.

Untuk informasi lebih lanjut tentang PennyLane, lihat dokumentasi di PennyLane situs web.

Algoritme hibrid di notebook contoh Amazon Braket

AmazonBraket memang menyediakan berbagai contoh notebook yang tidak bergantung pada PennyLane plugin untuk menjalankan algoritma hybrid. Anda dapat memulai dengan salah satu dari

notebook contoh hibrid Amazon Braket yang menggambarkan metode variasional, seperti Algoritme Pengoptimuman Perkiraan Kuantum (QAOA) atau Variational Quantum Eigensolver (VQE).

Notebook contoh Amazon Braket mengandalkan SDK Python Amazon Braket. SDK menyediakan kerangka kerja untuk berinteraksi dengan perangkat keras komputasi kuantum melalui Amazon Braket. Ini adalah pustaka open source yang dirancang untuk membantu Anda dengan bagian kuantum dari alur kerja hibrida Anda.

Anda dapat menjelajahi Amazon Braket lebih lanjut dengan contoh notebook kami.

Algoritma hybrid dengan simulator tertanam PennyLane

AmazonBraket Hybrid Jobs sekarang hadir dengan simulator tertanam berbasis CPU dan GPU berkinerja tinggi dari. PennyLane Keluarga simulator tertanam ini dapat disematkan langsung di dalam wadah pekerjaan hibrida Anda dan mencakup `lightning.qubit` simulator vektor status cepat, simulator yang dipercepat menggunakan perpustakaan CuQuantum NVIDIA, dan lainnya. `lightning.gpu` Simulator tertanam ini sangat cocok untuk algoritma variasional seperti pembelajaran mesin kuantum yang dapat mengambil manfaat dari metode canggih seperti metode diferensiasi adjoint. Anda dapat menjalankan simulator tertanam ini pada satu atau beberapa instance CPU atau GPU.

Dengan Hybrid Jobs, Anda sekarang dapat menjalankan kode algoritma variasional Anda menggunakan kombinasi co-prosesor klasik dan QPU, simulator sesuai permintaan Amazon Braket sepertiSV1, atau langsung menggunakan simulator tertanam dari. PennyLane

Simulator tertanam sudah tersedia dengan wadah Hybrid Jobs, Anda hanya perlu menghias fungsi Python utama Anda dengan dekorator. `@hybrid_job` Untuk menggunakan PennyLane `lightning.gpu` simulator, Anda juga perlu menentukan instance GPU `InstanceConfig` seperti yang ditunjukkan pada cuplikan kode berikut:

```
import pennylane as qml
from braket.jobs import hybird_job
from braket.jobs.config import InstanceConfig


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

Lihat contoh notebook untuk memulai menggunakan simulator PennyLane tertanam dengan Hybrid Jobs.

Gradien bersebelahan menyala PennyLane dengan simulator Amazon Braket

Dengan PennyLane plugin untuk Amazon Braket, Anda dapat menghitung gradien menggunakan metode diferensiasi adjoint saat berjalan di simulator vektor status lokal atau SV1.

Catatan: Untuk menggunakan metode diferensiasi adjoint, Anda harus menentukan **`diff_method='device'`** dalam metode Anda**`qnode`**, dan bukan. `diff_method='adjoint'` Lihat contoh berikut ini.

```
device_arn = "arn:aws:braket:::device/quantum-simulator/amazon/sv1"
dev = qml.device("braket.aws.qubit", wires=wires, shots=0, device_arn=device_arn)

@qml.qnode(dev, diff_method="device")
def cost_function(params):
    circuit(params)
    return qml.expval(cost_h)

gradient = qml.grad(circuit)
initial_gradient = gradient(params0)
```

> **Note**
>
> Saat ini, PennyLane akan menghitung indeks pengelompokan untuk QAOA Hamiltonians dan menggunakannya untuk membagi Hamiltonian menjadi beberapa nilai ekspektasi. Jika Anda ingin menggunakan kemampuan diferensiasi adjoint SV1 saat menjalankan QAOA dariPennyLane, Anda perlu merekonstruksi biaya Hamiltonian dengan menghapus indeks pengelompokan, seperti: `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False) cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Gunakan Amazon Braket Hybrid Jobs dan PennyLane untuk menjalankan algoritma QAOA

Di bagian ini, Anda akan menggunakan apa yang telah Anda pelajari untuk menulis program hybrid yang sebenarnya menggunakan PennyLane dengan kompilasi parametrik. Anda menggunakan skrip algoritme untuk mengatasi masalah Algoritma Pengoptimalan Perkiraan Kuantum (QAOA). Program ini menciptakan fungsi biaya yang sesuai dengan masalah optimasi Max Cut klasik, menentukan sirkuit kuantum parametris, dan menggunakan metode penurunan gradien sederhana untuk mengoptimalkan parameter sehingga fungsi biaya diminimalkan. Dalam contoh ini, kami menghasilkan grafik masalah dalam skrip algoritme untuk kesederhanaan, tetapi untuk kasus penggunaan yang lebih umum, praktik terbaik adalah memberikan spesifikasi masalah melalui saluran khusus dalam konfigurasi data input. Bendera `parametrize_differentiable` default `True` sehingga Anda secara otomatis mendapatkan manfaat dari peningkatan kinerja runtime dari kompilasi parametrik pada QPU yang didukung.

```
import os
import json
import time

from braket.jobs import save_job_result
from braket.jobs.metrics import log_metric

import networkx as nx
import pennylane as qml
from pennylane import numpy as np
from matplotlib import pyplot as plt

def init_pl_device(device_arn, num_nodes, shots, max_parallel):
    return qml.device(
        "braket.aws.qubit",
        device_arn=device_arn,
        wires=num_nodes,
        shots=shots,
        # Set s3_destination_folder=None to output task results to a default folder
        s3_destination_folder=None,
        parallel=True,
        max_parallel=max_parallel,
        parametrize_differentiable=True, # This flag is True by default.
    )
```

```
def start_here():
    input_dir = os.environ["AMZN_BRAKET_INPUT_DIR"]
    output_dir = os.environ["AMZN_BRAKET_JOB_RESULTS_DIR"]
    job_name = os.environ["AMZN_BRAKET_JOB_NAME"]
    checkpoint_dir = os.environ["AMZN_BRAKET_CHECKPOINT_DIR"]
    hp_file = os.environ["AMZN_BRAKET_HP_FILE"]
    device_arn = os.environ["AMZN_BRAKET_DEVICE_ARN"]

    # Read the hyperparameters
    with open(hp_file, "r") as f:
        hyperparams = json.load(f)

    p = int(hyperparams["p"])
    seed = int(hyperparams["seed"])
    max_parallel = int(hyperparams["max_parallel"])
    num_iterations = int(hyperparams["num_iterations"])
    stepsize = float(hyperparams["stepsize"])
    shots = int(hyperparams["shots"])

    # Generate random graph
    num_nodes = 6
    num_edges = 8
    graph_seed = 1967
    g = nx.gnm_random_graph(num_nodes, num_edges, seed=graph_seed)

    # Output figure to file
    positions = nx.spring_layout(g, seed=seed)
    nx.draw(g, with_labels=True, pos=positions, node_size=600)
    plt.savefig(f"{output_dir}/graph.png")

    # Set up the QAOA problem
    cost_h, mixer_h = qml.qaoa.maxcut(g)

    def qaoa_layer(gamma, alpha):
        qml.qaoa.cost_layer(gamma, cost_h)
        qml.qaoa.mixer_layer(alpha, mixer_h)

    def circuit(params, **kwargs):
        for i in range(num_nodes):
            qml.Hadamard(wires=i)
        qml.layer(qaoa_layer, p, params[0], params[1])

    dev = init_pl_device(device_arn, num_nodes, shots, max_parallel)
```

```
    np.random.seed(seed)
    cost_function = qml.ExpvalCost(circuit, cost_h, dev, optimize=True)
    params = 0.01 * np.random.uniform(size=[2, p])

    optimizer = qml.GradientDescentOptimizer(stepsize=stepsize)
    print("Optimization start")

    for iteration in range(num_iterations):
        t0 = time.time()

        # Evaluates the cost, then does a gradient step to new params
        params, cost_before = optimizer.step_and_cost(cost_function, params)
        # Convert cost_before to a float so it's easier to handle
        cost_before = float(cost_before)

        t1 = time.time()

        if iteration == 0:
            print("Initial cost:", cost_before)
        else:
            print(f"Cost at step {iteration}:", cost_before)

        # Log the current loss as a metric
        log_metric(
            metric_name="Cost",
            value=cost_before,
            iteration_number=iteration,
        )

        print(f"Completed iteration {iteration + 1}")
        print(f"Time to complete iteration: {t1 - t0} seconds")

    final_cost = float(cost_function(params))
    log_metric(
        metric_name="Cost",
        value=final_cost,
        iteration_number=num_iterations,
    )

    # We're done with the hybrid job, so save the result.
    # This will be returned in job.result()
    save_job_result({"params": params.numpy().tolist(), "cost": final_cost})
```

> **Note**
>
> Kompilasi parametrik didukung pada semua QPU superkonduktor berbasis gerbang dari Rigetti Computing dan Oxford Quantum Circuits dengan pengecualian program tingkat pulsa.

Percepat beban kerja hybrid Anda dengan simulator tertanam dari PennyLane

Mari kita lihat bagaimana Anda dapat menggunakan simulator tertanam dari PennyLane di Amazon Braket Hybrid Jobs untuk menjalankan beban kerja hybrid. Simulator tertanam berbasis GPU Pennylane,`lightning.gpu`, menggunakan perpustakaan Nvidia CuQuantum untuk mempercepat simulasi sirkuit. Simulator GPU tertanam sudah dikonfigurasi sebelumnya di semua wadah pekerjaan Braket yang dapat digunakan pengguna di luar kotak. Di halaman ini, kami menunjukkan cara menggunakan `lightning.gpu` untuk mempercepat beban kerja hybrid Anda.

Menggunakan **`lightning.gpu`** untuk beban kerja Algoritma Pengoptimalan Perkiraan Kuantum

Pertimbangkan contoh Quantum Perkiraan Optimasi Algoritma (QAOA) dari buku catatan ini. Untuk memilih simulator tertanam, Anda menentukan `device` argumen untuk menjadi string dari formulir:`"local:<provider>/<simulator_name>"`. Misalnya, Anda akan mengatur `"local:pennylane/lightning.gpu"` untuk`lightning.gpu`. String perangkat yang Anda berikan ke Job Hybrid saat diluncurkan diteruskan ke job sebagai variabel lingkungan`"AMZN_BRAKET_DEVICE_ARN"`.

```
device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
prefix, device_name = device_string.split("/")
device = qml.device(simulator_name, wires=n_wires)
```

Di halaman ini, mari kita bandingkan dua simulator vektor PennyLane status tertanam `lightning.qubit` (yang berbasis CPU) dan `lightning.gpu` (yang berbasis GPU). Anda harus menyediakan simulator dengan beberapa dekomposisi gerbang khusus untuk menghitung berbagai gradien.

Sekarang Anda siap menyiapkan skrip peluncuran pekerjaan hibrida. Anda akan menjalankan algoritma QAOA menggunakan dua jenis instance: `m5.2xlarge` dan. `p3.2xlarge` Jenis

`m5.2xlarge` instans sebanding dengan laptop pengembang standar. `p3.2xlarge`Ini adalah instance komputasi yang dipercepat yang memiliki GPU NVIDIA Volta tunggal dengan memori 16GB.

`hyperparameters`Untuk semua pekerjaan hybrid Anda akan sama. Yang perlu Anda lakukan untuk mencoba berbagai contoh dan simulator adalah mengubah dua baris sebagai berikut.

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

atau:

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

> **Note**
>
> Jika Anda menentukan **`instance_config`** sebagai menggunakan instance berbasis GPU, tetapi memilih **`device`** untuk menjadi simulator berbasis CPU tertanam (**`lightning.qubit`**), GPU tidak akan digunakan. Pastikan untuk menggunakan simulator berbasis GPU tertanam jika Anda ingin menargetkan GPU!

Pertama, Anda dapat membuat dua pekerjaan hibrida dan menyelesaikan Max-Cut dengan QAOA pada grafik dengan 18 simpul. Ini berarti sirkuit 18-qubit — relatif kecil dan layak untuk dijalankan dengan cepat di laptop Anda atau instans. `m5.2xlarge`

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

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

# And similarly for the p3 job
m5_job = AwsQuantumJob.create(
    device=device,
    source_module="qaoa_source",
```

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

Waktu iterasi rata-rata untuk `m5.2xlarge` instance adalah sekitar 25 detik, sedangkan untuk `p3.2xlarge` contoh itu sekitar 12 detik. Untuk alur kerja 18-qubit ini, instance GPU memberi kita percepatan 2x. Jika Anda melihat halaman harga Amazon Braket Hybrid Jobs, Anda dapat melihat bahwa biaya per menit untuk sebuah `m5.2xlarge` instans adalah $0,00768, sedangkan untuk contoh itu $0,06375`p3.2xlarge`. Untuk menjalankan 5 iterasi total, seperti yang Anda lakukan di sini, akan dikenakan biaya $0,016 menggunakan instance CPU atau $0,06375 menggunakan instance GPU — keduanya cukup murah!

Sekarang mari kita membuat masalah lebih sulit, dan mencoba memecahkan masalah Max-Cut pada grafik 24-vertex, yang akan diterjemahkan menjadi 24 qubit. Jalankan pekerjaan hybrid lagi pada dua contoh yang sama dan bandingkan biayanya.

> **Note**
>
> Anda akan melihat bahwa waktu untuk menjalankan pekerjaan hybrid ini pada instance CPU mungkin sekitar lima jam!

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

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

# And similarly for the p3 job
m5_big_job = AwsQuantumJob.create(
    device=device,
    source_module="qaoa_source",
    job_name="qaoa-m5-big-" + str(int(time.time())),
```

```
    image_uri=image_uri,
    # Relative to the source_module
    entry_point="qaoa_source.qaoa_algorithm_script",
    copy_checkpoints_from_job=None,
    instance_config=instance_config,
    # general parameters
    hyperparameters=hyperparameters,
    input_data={"input-graph": input_file_path},
    wait_until_complete=True,
)
```

Waktu iterasi rata-rata untuk `m5.2xlarge` instance kira-kira satu jam, sedangkan untuk `p3.2xlarge` instance kira-kira dua menit. Untuk masalah yang lebih besar ini, instance GPU lebih cepat! Yang harus Anda lakukan untuk mendapatkan keuntungan dari percepatan ini adalah mengubah dua baris kode, menukar jenis instance dan simulator lokal yang digunakan. Untuk menjalankan 5 iterasi total, seperti yang dilakukan di sini, akan menelan biaya sekitar $2.27072 menggunakan instance CPU atau sekitar $0,775625 menggunakan instance GPU. Penggunaan CPU tidak hanya lebih mahal, tetapi juga membutuhkan lebih banyak waktu untuk dijalankan. Mempercepat alur kerja ini dengan instance GPU yang tersedia AWS, menggunakan PennyLane simulator tertanam yang didukung oleh NVIDIA CuQuantum, memungkinkan Anda menjalankan alur kerja dengan jumlah qubit menengah (antara 20 dan 30) dengan biaya total yang lebih sedikit dan dalam waktu yang lebih singkat. Ini berarti Anda dapat bereksperimen dengan komputasi kuantum bahkan untuk masalah yang terlalu besar untuk berjalan cepat di laptop Anda atau instance berukuran serupa.

Pembelajaran mesin kuantum dan paralelisme data

Jika jenis beban kerja Anda adalah pembelajaran mesin kuantum (QML/Quantum Machine Learning) yang melatih pada kumpulan data, Anda dapat lebih mempercepat beban kerja Anda menggunakan paralelisme data. Dalam QML, model berisi satu atau lebih sirkuit kuantum. Model mungkin atau mungkin juga tidak mengandung jaring saraf klasik. Saat melatih model dengan kumpulan data, parameter dalam model diperbarui untuk meminimalkan fungsi kerugian. Fungsi kerugian biasanya didefinisikan untuk satu titik data, dan total kerugian untuk kerugian rata-rata atas seluruh kumpulan data. Dalam QML, kerugian biasanya dihitung secara serial sebelum rata-rata kerugian total untuk perhitungan gradien. Prosedur ini memakan waktu, terutama ketika ada ratusan titik data.

Karena kerugian dari satu titik data tidak tergantung pada titik data lain, kerugian dapat dievaluasi secara paralel! Kerugian dan gradien yang terkait dengan titik data yang berbeda dapat dievaluasi secara bersamaan. Ini dikenal sebagai paralelisme data. Dengan SageMaker perpustakaan paralel

data terdistribusi, Amazon Braket Hybrid Jobs memudahkan Anda memanfaatkan paralelisme data untuk mempercepat pelatihan Anda.

Pertimbangkan beban kerja QML berikut untuk paralelisme data yang menggunakan dataset data Sonar dari repositori UCI yang terkenal sebagai contoh untuk klasifikasi biner. Dataset Sonar memiliki 208 titik data masing-masing dengan 60 fitur yang dikumpulkan dari sinyal sonar yang memantul dari material. Setiap titik data diberi label sebagai “M” untuk tambang atau “R” untuk batu. Model QML kami terdiri dari lapisan input, sirkuit kuantum sebagai lapisan tersembunyi, dan lapisan keluaran. Lapisan input dan output adalah jaring saraf klasik yang diimplementasikan di PyTorch. Sirkuit kuantum terintegrasi dengan jaring PyTorch saraf menggunakan modul PennyLane qml.qnn. Lihat contoh notebook kami untuk detail lebih lanjut tentang beban kerja. Seperti contoh QAOA di atas, Anda dapat memanfaatkan kekuatan GPU dengan menggunakan simulator berbasis GPU tertanam seperti ini `lightning.gpu` untuk meningkatkan kinerja dibandingkan simulator berbasis PennyLane CPU tertanam.

Untuk membuat pekerjaan hybrid, Anda dapat memanggil `AwsQuantumJob.create` dan menentukan skrip algoritma, perangkat, dan konfigurasi lainnya melalui argumen kata kuncinya.

```
instance_config = InstanceConfig(instanceType='ml.p3.2xlarge')

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

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

Untuk menggunakan paralelisme data, Anda perlu memodifikasi beberapa baris kode dalam skrip algoritme agar pustaka SageMaker terdistribusi dapat memparalelkan pelatihan dengan benar. Pertama, Anda mengimpor `smdistributed` paket yang melakukan sebagian besar peningkatan berat untuk mendistribusikan beban kerja Anda di beberapa GPU dan beberapa instance. Paket ini sudah dikonfigurasi sebelumnya di Braket PyTorch dan TensorFlow kontainer. `dist`Modul ini memberi tahu skrip algoritme kami berapa jumlah total GPU untuk pelatihan (`world_size`) serta inti

`rank` dan `local_rank` inti GPU. `rank`adalah indeks absolut GPU di semua instance, sedangkan `local_rank` indeks GPU dalam sebuah instance. Misalnya, jika ada empat instance masing-masing dengan delapan GPU yang dialokasikan untuk pelatihan, `rank` rentang dari 0 hingga 31 dan `local_rank` rentang dari 0 hingga 7.

```
import smdistributed.dataparallel.torch.distributed as dist

dp_info = {
    "world_size": dist.get_world_size(),
    "rank": dist.get_rank(),
    "local_rank": dist.get_local_rank(),
}
batch_size //= dp_info["world_size"] // 8
batch_size = max(batch_size, 1)
```

Selanjutnya, Anda mendefinisikan `DistributedSampler` sesuai dengan `world_size` `rank` dan kemudian meneruskannya ke pemuat data. Sampler ini menghindari GPU mengakses potongan dataset yang sama.

```
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset,
    num_replicas=dp_info["world_size"],
    rank=dp_info["rank"]
)
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=batch_size,
    shuffle=False,
    num_workers=0,
    pin_memory=True,
    sampler=train_sampler,
)
```

Selanjutnya, Anda menggunakan `DistributedDataParallel` kelas untuk mengaktifkan paralelisme data.

```
from smdistributed.dataparallel.torch.parallel.distributed import
 DistributedDataParallel as DDP

model = DressedQNN(qc_dev).to(device)
model = DDP(model)
```

```
torch.cuda.set_device(dp_info["local_rank"])
model.cuda(dp_info["local_rank"])
```

Di atas adalah perubahan yang Anda butuhkan untuk menggunakan paralelisme data. Di QHTML, Anda sering ingin menyimpan hasil dan mencetak kemajuan pelatihan. Jika setiap GPU menjalankan perintah penyimpanan dan pencetakan, log akan dibanjiri dengan informasi berulang dan hasilnya akan saling menimpa. Untuk menghindari hal ini, Anda hanya dapat menyimpan dan mencetak dari GPU yang memiliki `rank` 0.

```
if dp_info["rank"]==0:
    print('elapsed time: ', elapsed)
    torch.save(model.state_dict(), f"{output_dir}/test_local.pt")
    save_job_result({"last loss": loss_before})
```

Amazon Braket Hybrid Jobs mendukung jenis `ml.p3.16xlarge` instans untuk library paralel SageMaker data terdistribusi. Anda mengonfigurasi tipe instance melalui `InstanceConfig` argumen di Hybrid Jobs. Agar pustaka paralel data SageMaker terdistribusi mengetahui bahwa paralelisme data diaktifkan, Anda perlu menambahkan dua hiperparameter tambahan, `"sagemaker_distributed_dataparallel_enabled"` pengaturan ke `"true"` dan `"sagemaker_instance_type"` pengaturan ke jenis instance yang Anda gunakan. Kedua hyperparameters ini digunakan oleh `smdistributed` paket. Skrip algoritme Anda tidak perlu menggunakannya secara eksplisit. Di Amazon Braket SDK, ini menyediakan argumen kata kunci yang nyaman. `distribution` `distribution="data_parallel"`Dengan penciptaan lapangan kerja hybrid, Amazon Braket SDK secara otomatis menyisipkan dua hyperparameter untuk Anda. Jika Anda menggunakan Amazon Braket API, Anda harus menyertakan dua hyperparameters ini.

Dengan paralelisme instance dan data yang dikonfigurasi, Anda sekarang dapat mengirimkan pekerjaan hybrid Anda. Ada 8 GPU dalam satu `ml.p3.16xlarge` contoh. Saat Anda mengatur`instanceCount=1`, beban kerja didistribusikan di 8 GPU dalam instance. Bila Anda menyetel `instanceCount` lebih dari satu, beban kerja didistribusikan ke seluruh GPU yang tersedia di semua instance. Saat menggunakan beberapa instans, setiap instans dikenakan biaya berdasarkan berapa banyak waktu yang Anda gunakan. Misalnya, saat Anda menggunakan empat instance, waktu yang dapat ditagih adalah empat kali waktu proses per instance karena ada empat instance yang menjalankan beban kerja Anda secara bersamaan.

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

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

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

> **Note**
>
> Dalam penciptaan pekerjaan hibrida di atas, `train_dp.py` adalah skrip algoritma yang dimodifikasi untuk menggunakan paralelisme data. Perlu diingat bahwa paralelisme data hanya berfungsi dengan benar ketika Anda memodifikasi skrip algoritme Anda sesuai dengan bagian di atas. Jika opsi paralelisme data diaktifkan tanpa skrip algoritme yang dimodifikasi dengan benar, pekerjaan hibrida dapat menimbulkan kesalahan, atau setiap GPU dapat berulang kali memproses irisan data yang sama, yang tidak efisien.

Mari kita bandingkan waktu berjalan dan biaya dalam contoh di mana ketika melatih model dengan sirkuit kuantum 26-qubit untuk masalah klasifikasi biner yang disebutkan di atas. `ml.p3.16xlarge`Contoh yang digunakan dalam contoh ini berharga $0,4692 per menit. Tanpa paralelisme data, simulator membutuhkan waktu sekitar 45 menit untuk melatih model selama 1 zaman (yaitu, lebih dari 208 titik data) dan harganya sekitar $20. Dengan paralelisme data di 1 instance dan 4 instance, masing-masing hanya membutuhkan waktu 6 menit dan 1,5 menit, yang berarti sekitar $2,8 untuk keduanya. Dengan menggunakan paralelisme data di 4 instance, Anda tidak hanya meningkatkan waktu berjalan sebesar 30x, tetapi juga mengurangi biaya dengan urutan besarnya!

Buat dan debug pekerjaan hybrid dengan mode lokal

Jika Anda sedang membangun algoritma hybrid baru, mode lokal membantu Anda men-debug dan menguji skrip algoritme Anda. Mode lokal adalah fitur yang memungkinkan Anda menjalankan

kode yang Anda rencanakan untuk digunakan di Amazon Braket Hybrid Jobs, tetapi tanpa perlu Braket untuk mengelola infrastruktur untuk menjalankan pekerjaan hybrid. Sebagai gantinya, Anda menjalankan pekerjaan hybrid secara lokal di instance Braket Notebook Anda atau pada klien pilihan seperti laptop atau komputer desktop. Dalam mode lokal, Anda masih dapat mengirim tugas kuantum ke perangkat yang sebenarnya, tetapi Anda tidak mendapatkan manfaat kinerja saat menjalankan QPU aktual saat dalam mode lokal.

Untuk menggunakan mode lokal, ubah `AwsQuantumJob` ke `LocalQuantumJob` mana pun itu terjadi. Misalnya, untuk menjalankan contoh dari Buat pekerjaan hybrid pertama Anda, edit skrip pekerjaan hybrid sebagai berikut.

```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

> **Note**
>
> Docker, yang sudah diinstal sebelumnya di notebook Amazon Braket, perlu diinstal di lingkungan lokal Anda untuk menggunakan fitur ini. Petunjuk untuk menginstal Docker dapat ditemukan di sini. Selain itu, tidak semua parameter didukung dalam mode lokal.

Bawa wadah Anda sendiri (BYOC)

Amazon Braket Hybrid Jobs menyediakan tiga kontainer pra-bangun untuk menjalankan kode di lingkungan yang berbeda. Jika salah satu wadah ini mendukung kasus penggunaan Anda, Anda hanya perlu menyediakan skrip algoritme saat membuat pekerjaan hibrida. Dependensi kecil yang hilang dapat ditambahkan dari skrip algoritme Anda atau dari `requirements.txt` file yang menggunakan. `pip`

Jika tidak ada kontainer ini yang mendukung kasus penggunaan Anda, atau jika Anda ingin mengembangkannya, Braket Hybrid Jobs mendukung menjalankan pekerjaan hibrida dengan gambar Docker penampung kustom Anda sendiri, atau membawa wadah Anda sendiri (BYOC). Tapi sebelum kita menyelami, mari kita pastikan itu benar-benar fitur yang tepat untuk kasus penggunaan Anda.

Kapan membawa wadah saya sendiri keputusan yang tepat?

Membawa wadah Anda sendiri (BYOC) ke Braket Hybrid Jobs menawarkan fleksibilitas untuk menggunakan perangkat lunak Anda sendiri dengan menginstalnya di lingkungan yang dikemas. Bergantung pada kebutuhan spesifik Anda, mungkin ada cara untuk mencapai fleksibilitas yang sama tanpa harus melalui BYOC Docker build penuh - unggahan Amazon ECR - siklus URI gambar khusus.

> **Note**
>
> BYOC mungkin bukan pilihan yang tepat jika Anda ingin menambahkan sejumlah kecil paket Python tambahan (umumnya kurang dari 10) yang tersedia untuk umum. Misalnya, jika Anda menggunakan PyPi.

Dalam hal ini, Anda dapat menggunakan salah satu gambar Braket yang sudah dibuat sebelumnya, dan kemudian menyertakan `requirements.txt` file di direktori sumber Anda pada pengiriman pekerjaan. File secara otomatis dibaca, dan `pip` akan menginstal paket dengan versi yang ditentukan seperti biasa. Jika Anda menginstal sejumlah besar paket, runtime pekerjaan Anda mungkin meningkat secara substansional. Periksa Python dan, jika ada, versi CUDA dari wadah bawaan yang ingin Anda gunakan untuk menguji apakah perangkat lunak Anda akan berfungsi.

BYOC diperlukan ketika Anda ingin menggunakan bahasa non-Python (seperti C ++ atau Rust) untuk skrip pekerjaan Anda, atau jika Anda ingin menggunakan versi Python yang tidak tersedia melalui wadah pra-bangun Braket. Ini juga merupakan pilihan yang baik jika:

- Anda menggunakan perangkat lunak dengan kunci lisensi, dan Anda perlu mengautentikasi kunci itu terhadap server lisensi untuk menjalankan perangkat lunak. Dengan BYOC, Anda dapat menyematkan kunci lisensi dalam Docker gambar Anda dan menyertakan kode untuk mengotentikasi itu.
- Anda menggunakan perangkat lunak yang tidak tersedia untuk umum. Misalnya, perangkat lunak di-host di pribadi GitLab atau GitHub repositori yang Anda perlukan kunci SSH tertentu untuk diakses.
- Anda perlu menginstal rangkaian besar perangkat lunak yang tidak dikemas dalam wadah yang disediakan Braket. BYOC akan memungkinkan Anda untuk menghilangkan waktu startup yang lama untuk wadah pekerjaan hybrid Anda karena instalasi perangkat lunak.

BYOC juga memungkinkan Anda membuat SDK atau algoritme khusus Anda tersedia bagi pelanggan dengan membangun Docker wadah dengan perangkat lunak Anda dan membuatnya tersedia bagi pengguna Anda. Anda dapat melakukan ini dengan menetapkan izin yang sesuai di Amazon ECR.

> **Note**
>
> Anda harus mematuhi semua lisensi perangkat lunak yang berlaku.

Resep untuk membawa wadah Anda sendiri

Di bagian ini, kami menyediakan step-by-step panduan tentang apa yang Anda perlukan bring your own container (BYOC) untuk Braket Hybrid Jobs — skrip, file, dan langkah-langkah untuk menggabungkannya agar dapat bangkit dan berjalan dengan gambar kustom Docker Anda. Kami menyediakan resep untuk dua kasus umum:

1. Instal perangkat lunak tambahan dalam Docker gambar dan gunakan hanya skrip algoritma Python dalam pekerjaan Anda.
2. Gunakan skrip algoritma yang ditulis dalam bahasa non-Python dengan Hybrid Jobs, atau arsitektur CPU selain x86.

Mendefinisikan skrip entri kontainer lebih kompleks untuk kasus 2.

Saat Braket menjalankan Job Hybrid Anda, Braket akan meluncurkan nomor dan jenis instans Amazon EC2 yang diminta, lalu menjalankan gambar Docker yang ditentukan oleh input URI gambar ke pembuatan pekerjaan pada mereka. Saat menggunakan fitur BYOC, Anda menentukan URI gambar yang dihosting di repositori ECR Amazon pribadi yang memiliki akses Baca. Braket Hybrid Jobs menggunakan gambar kustom itu untuk menjalankan pekerjaan.

Komponen spesifik yang Anda butuhkan untuk membangun Docker gambar yang dapat digunakan dengan Hybrid Jobs. Jika Anda tidak terbiasa dengan menulis dan membangun`Dockerfiles`, kami sarankan Anda merujuk ke dokumentasi Dockerfile dan Amazon ECR CLIdokumentasi yang diperlukan saat Anda membaca petunjuk ini.

Berikut ikhtisar tentang apa yang Anda butuhkan:

- Gambar dasar untuk Dockerfile Anda
- (Opsional) Skrip titik masuk kontainer yang dimodifikasi

- A Dockerfile yang menginstal perangkat lunak apa pun yang diperlukan dan menyertakan skrip kontainer

Gambar dasar untuk Dockerfile Anda

Jika Anda menggunakan Python dan ingin menginstal perangkat lunak di atas apa yang disediakan dalam wadah yang disediakan Braket, opsi untuk gambar dasar adalah salah satu gambar wadah Braket, yang dihosting di GitHub repo kami dan di Amazon ECR. Anda perlu mengautentikasi ke Amazon ECR untuk menarik gambar dan membangun di atasnya. Misalnya, baris pertama Docker file BYOC Anda dapat berupa: `FROM [IMAGE_URI_HERE]`

Selanjutnya, isi sisa Dockerfile untuk menginstal dan mengatur perangkat lunak yang ingin Anda tambahkan ke wadah. Gambar Braket yang sudah dibuat sebelumnya sudah berisi skrip titik masuk kontainer yang sesuai, jadi Anda tidak perlu khawatir untuk memasukkannya.

Jika Anda ingin menggunakan bahasa non-Python, seperti C ++, Rust, atau Julia, atau jika Anda ingin membuat gambar untuk arsitektur CPU non-x86, seperti ARM, Anda mungkin perlu membangun di atas gambar publik barebone. Anda dapat menemukan banyak gambar seperti itu di Galeri Publik Amazon Elastic Container Registry. Pastikan Anda memilih salah satu yang sesuai untuk arsitektur CPU, dan jika perlu, GPU yang ingin Anda gunakan.

(Opsional) Skrip titik masuk kontainer yang dimodifikasi

> **Note**
>
> Jika Anda hanya menambahkan perangkat lunak tambahan ke gambar Braket yang sudah dibuat sebelumnya, Anda dapat melewati bagian ini.

Untuk menjalankan kode non-Python sebagai bagian dari pekerjaan hybrid Anda, Anda harus memodifikasi skrip Python yang mendefinisikan titik masuk kontainer. Misalnya, skrip `braket_container.py` python di Amazon Braket Github. Ini adalah skrip gambar yang dibuat sebelumnya oleh Braket digunakan untuk meluncurkan skrip algoritme Anda dan mengatur variabel lingkungan yang sesuai. Skrip titik masuk kontainer itu sendiri harus menggunakan Python, tetapi dapat meluncurkan skrip non-Python. Dalam contoh pra-bangun, Anda dapat melihat bahwa skrip algoritma Python diluncurkan baik sebagai subproses Python atau sebagai proses yang sepenuhnya baru. Dengan memodifikasi logika ini, Anda dapat mengaktifkan skrip titik masuk untuk meluncurkan skrip algoritma non-Python. Misalnya, Anda dapat memodifikasi

thekick_off_customer_script()fungsi untuk meluncurkan proses Rust tergantung pada akhir ekstensi file.

Anda juga dapat memilih untuk menulis yang benar-benar barubraket_container.py. Ini harus menyalin data input, arsip sumber, dan file lain yang diperlukan dari Amazon S3 ke dalam wadah, dan menentukan variabel lingkungan yang sesuai.

A **Dockerfile** yang menginstal perangkat lunak apa pun yang diperlukan dan menyertakan skrip kontainer

> **Note**
>
> Jika Anda menggunakan gambar Braket yang sudah dibuat sebelumnya sebagai gambar Docker dasar Anda, skrip kontainer sudah ada.

Jika Anda membuat skrip kontainer yang dimodifikasi pada langkah sebelumnya, Anda harus menyalinnya ke dalam wadah dan menentukan variabel SAGEMAKER_PROGRAM lingkunganbraket_container.py, atau apa yang telah Anda beri nama skrip titik masuk kontainer baru Anda.

Berikut ini adalah contoh Dockerfile yang memungkinkan Anda untuk menggunakan Julia pada instance Jobs yang dipercepat GPU:

```
FROM nvidia/cuda:12.2.0-devel-ubuntu22.04


 ARG DEBIAN_FRONTEND=noninteractive
 ARG JULIA_RELEASE=1.8
 ARG JULIA_VERSION=1.8.3


 ARG PYTHON=python3.11
 ARG PYTHON_PIP=python3-pip
 ARG PIP=pip


 ARG JULIA_URL = https://julialang-s3.julialang.org/bin/linux/x64/${JULIA_RELEASE}/
 ARG TAR_NAME = julia-${JULIA_VERSION}-linux-x86_64.tar.gz
```

```
ARG PYTHON_PKGS = # list your Python packages and versions here


RUN curl -s -L ${JULIA_URL}/${TAR_NAME} | tar -C /usr/local -x -z --strip-components=1
-f -


RUN apt-get update \

   && apt-get install -y --no-install-recommends \

   build-essential \

   tzdata \

   openssh-client \

   openssh-server \

   ca-certificates \

   curl \

   git \

   libtemplate-perl \

   libssl1.1 \

   openssl \

   unzip \

   wget \

   zlib1g-dev \

   ${PYTHON_PIP} \

   ${PYTHON}-dev \
```

```
 RUN ${PIP} install --no-cache --upgrade ${PYTHON_PKGS}


 RUN ${PIP} install --no-cache --upgrade sagemaker-training==4.1.3


 # Add EFA and SMDDP to LD library path
 ENV LD_LIBRARY_PATH="/opt/conda/lib/python${PYTHON_SHORT_VERSION}/site-packages/
smdistributed/dataparallel/lib:$LD_LIBRARY_PATH"
 ENV LD_LIBRARY_PATH=/opt/amazon/efa/lib/:$LD_LIBRARY_PATH


 # Julia specific installation instructions
 COPY Project.toml /usr/local/share/julia/environments/v${JULIA_RELEASE}/
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using Pkg; Pkg.instantiate(); Pkg.API.precompile()'
 # generate the device runtime library for all known and supported devices
 RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using CUDA; CUDA.precompile_runtime()'


 # Open source compliance scripts
 RUN HOME_DIR=/root \

 && curl -o ${HOME_DIR}/oss_compliance.zip https://aws-dlinfra-
utilities.s3.amazonaws.com/oss_compliance.zip \

 && unzip ${HOME_DIR}/oss_compliance.zip -d ${HOME_DIR}/ \

 && cp ${HOME_DIR}/oss_compliance/test/testOSSCompliance /usr/local/bin/
testOSSCompliance \

 && chmod +x /usr/local/bin/testOSSCompliance \

 && chmod +x ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh \

 && ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh ${HOME_DIR} ${PYTHON} \

 && rm -rf ${HOME_DIR}/oss_compliance*


 # Copying the container entry point script
```

```
 COPY braket_container.py /opt/ml/code/braket_container.py
 ENV SAGEMAKER_PROGRAM braket_container.py
```

Contoh ini, mengunduh dan menjalankan skrip yang disediakan oleh AWS untuk memastikan kepatuhan terhadap semua lisensi Sumber Terbuka yang relevan. Misalnya, dengan menghubungkan kode yang diinstal dengan benar yang diatur oleh file. MIT license

Jika Anda perlu menyertakan kode non-publik, misalnya kode yang di-host di privat GitHub atau GitLab repositori, jangan sematkan kunci SSH pada gambar untuk mengaksesnyaDocker. Sebagai gantinya, gunakan Docker Compose saat Anda membangun Docker untuk memungkinkan mengakses SSH pada mesin host tempat ia dibangun. Untuk informasi selengkapnya, lihat Securely using SSH keys in Docker to access private Github repositories guide.

Membangun dan mengunggah gambar Anda Docker

Dengan didefinisikan dengan benar`Dockerfile`, Anda sekarang siap untuk mengikuti langkah-langkah untuk membuat repositori Amazon ECR pribadi, jika belum ada. Anda juga dapat membuat, menandai, dan mengunggah gambar kontainer Anda ke repositori.

Anda siap untuk membangun, menandai, dan mendorong gambar. Lihat dokumentasi build Docker untuk penjelasan lengkap tentang opsi `docker build` dan beberapa contoh.

Untuk file contoh yang ditentukan di atas, Anda dapat menjalankan:

```
aws ecr get-login-password --region ${your_region} | docker login --username AWS --
password-stdin ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com
 docker build -t braket-julia .
 docker tag braket-julia:latest ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com/
braket-julia:latest
 docker push ${aws_account_id}.dkr.ecr.${your_region}.amazonaws.com/braket-julia:latest
```

Menetapkan izin ECR Amazon yang sesuai

Braket Hybrid Jobs Dockergambar harus di-host di repositori Amazon ECR pribadi. Secara default, repo Amazon ECR pribadi tidak menyediakan akses baca ke Braket Hybrid Jobs IAM role atau ke pengguna lain yang ingin menggunakan gambar Anda, seperti kolaborator atau siswa. Anda harus menetapkan kebijakan repositori untuk memberikan izin yang sesuai. Secara umum, hanya berikan izin kepada pengguna dan IAM peran tertentu yang ingin Anda akses ke gambar Anda, daripada mengizinkan siapa pun image URI yang menariknya.

Menjalankan pekerjaan hybrid Braket di wadah Anda sendiri

Untuk membuat pekerjaan hybrid dengan container Anda sendiri, panggil `AwsQuantumJob.create()` dengan argumen yang `image_uri` ditentukan. Anda dapat menggunakan QPU, simulator sesuai permintaan, atau menjalankan kode Anda secara lokal pada prosesor klasik yang tersedia dengan Braket Hybrid Jobs. Kami merekomendasikan pengujian kode Anda pada simulator seperti SV1, DM1, atau TN1 sebelum berjalan pada QPU nyata.

Untuk menjalankan kode Anda pada prosesor klasik, tentukan `instanceType` dan yang `instanceCount` Anda gunakan dengan memperbarui file`InstanceConfig`. Perhatikan bahwa jika Anda menentukan `instance_count` > 1, Anda perlu memastikan bahwa kode Anda dapat berjalan di beberapa host. Batas atas untuk jumlah instance yang dapat Anda pilih adalah 5. Sebagai contoh:

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

> **Note**
>
> Gunakan perangkat ARN untuk melacak simulator yang Anda gunakan sebagai metadata pekerjaan hibrida. Nilai yang dapat diterima harus mengikuti format`device = "local:<provider>/<simulator_name>"`. Ingat bahwa `<provider>` dan `<simulator_name>` harus hanya terdiri dari huruf, angka,`_`,`-`, dan`.`. String dibatasi hingga 256 karakter.
>
> Jika Anda berencana untuk menggunakan BYOC dan Anda tidak menggunakan Braket SDK untuk membuat tugas kuantum, Anda harus meneruskan nilai variabel lingkungan `AMZN_BRAKET_JOB_TOKEN` ke `jobToken` parameter dalam permintaan. `CreateQuantumTask` Jika tidak, tugas kuantum tidak mendapatkan prioritas dan ditagih sebagai tugas kuantum mandiri biasa.

Konfigurasikan bucket default di **`AwsSession`**

Menyediakan sendiri `AwsSession` memberi Anda fleksibilitas yang lebih besar, misalnya, di lokasi bucket default Anda. Secara default, an `AwsSession` memiliki lokasi bucket default`f"amazon-braket-{id}-{region}"`. Tetapi Anda dapat mengganti default itu saat membuat file`AwsSession`. Pengguna secara opsional dapat meneruskan `AwsSession` objek ke dalam `AwsQuantumJob.create` dengan nama parameter `aws_session` seperti yang ditunjukkan pada contoh kode berikut.

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

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

Berinteraksi dengan pekerjaan hybrid secara langsung menggunakan API

Anda dapat mengakses dan berinteraksi dengan Amazon Braket Hybrid Jobs secara langsung menggunakan. API Namun, metode default dan kenyamanan tidak tersedia saat menggunakan secara langsung. API

> **Note**
>
> Kami sangat menyarankan agar Anda berinteraksi dengan Amazon Braket Hybrid Jobs menggunakan Amazon Braket Python SDK. Ini menawarkan default dan perlindungan yang nyaman yang membantu pekerjaan hibrida Anda berjalan dengan sukses.

Topik ini mencakup dasar-dasar penggunaanAPI. Jika Anda memilih untuk menggunakan API, perlu diingat bahwa pendekatan ini bisa lebih kompleks dan bersiaplah untuk beberapa iterasi agar pekerjaan hybrid Anda berjalan.

Untuk menggunakan API, akun Anda harus memiliki peran dengan kebijakan `AmazonBraketFullAccess` terkelola.

> **Note**
>
> Untuk informasi selengkapnya tentang cara mendapatkan peran dengan kebijakan `AmazonBraketFullAccess` terkelola, lihat halaman Aktifkan Amazon Braket.

Selain itu, Anda memerlukan peran eksekusi. Peran ini akan diteruskan ke layanan. Anda dapat membuat peran menggunakan konsol Amazon Braket. Gunakan tab Peran eksekusi pada halaman Izin dan pengaturan untuk membuat peran default untuk pekerjaan hibrida.

Ini `CreateJob` API mengharuskan Anda menentukan semua parameter yang diperlukan untuk pekerjaan hybrid. Untuk menggunakan Python, kompres file skrip algoritme Anda ke bundel tar, seperti file input.tar.gz, dan jalankan skrip berikut. Perbarui bagian kode dalam kurung miring (<>) agar sesuai dengan informasi akun Anda dan titik masuk yang menentukan jalur, file, dan metode tempat pekerjaan hibrida Anda dimulai.

```
from braket.aws import AwsDevice, AwsSession
import boto3
from datetime import datetime

s3_client = boto3.client("s3")
client = boto3.client("braket")

project_name = "job-test"
job_name = project_name + "-" + datetime.strftime(datetime.now(), "%Y%m%d%H%M%S")
bucket = "amazon-braket-<your_bucket>"
s3_prefix = job_name

job_script = "input.tar.gz"
job_object = f"{s3_prefix}/script/{job_script}"
s3_client.upload_file(job_script, bucket, job_object)

input_data = "inputdata.csv"
input_object = f"{s3_prefix}/input/{input_data}"
s3_client.upload_file(input_data, bucket, input_object)

job = client.create_job(
    jobName=job_name,
    roleArn="arn:aws:iam::<your_account>:role/service-role/
AmazonBraketJobsExecutionRole",  # https://docs.aws.amazon.com/braket/latest/
developerguide/braket-manage-access.html#about-amazonbraketjobsexecution
```

```
    algorithmSpecification={
        "scriptModeConfig": {
            "entryPoint": "<your_execution_module>:<your_execution_method>",
            "containerImage": {"uri": "292282985366.dkr.ecr.us-west-1.amazonaws.com/
amazon-braket-base-jobs:1.0-cpu-py37-ubuntu18.04"}   # Change to the specific region
 you are using
            "s3Uri": f"s3://{bucket}/{job_object}",
            "compressionType": "GZIP"
        }
    },
    inputDataConfig=[
        {
            "channelName": "hellothere",
            "compressionType": "NONE",
            "dataSource": {
                "s3DataSource": {
                    "s3Uri": f"s3://{bucket}/{s3_prefix}/input",
                    "s3DataType": "S3_PREFIX"
                }
            }
        }
    ],
    outputDataConfig={
        "s3Path": f"s3://{bucket}/{s3_prefix}/output"
    },
    instanceConfig={
        "instanceType": "ml.m5.large",
        "instanceCount": 1,
        "volumeSizeInGb": 1
    },
    checkpointConfig={
        "s3Uri":  f"s3://{bucket}/{s3_prefix}/checkpoints",
        "localPath": "/opt/omega/checkpoints"
    },
    deviceConfig={
        "priorityAccess": {
            "devices": [
                "arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3"
            ]
        }
    },
    hyperParameters={
        "hyperparameter key you wish to pass": "<hyperparameter value you wish to
 pass>",
```

```
    },
    stoppingCondition={
        "maxRuntimeInSeconds": 1200,
        "maximumTaskLimit": 10
    },
)
```

Setelah Anda membuat pekerjaan hybrid, Anda dapat mengakses detail pekerjaan hybrid melalui `GetJob` API atau konsol. Untuk mendapatkan rincian pekerjaan hybrid dari sesi Python di mana Anda menjalankan `createJob` kode seperti pada contoh sebelumnya, gunakan perintah Python berikut.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

Untuk membatalkan pekerjaan hibrida, hubungi `CancelJob` API dengan Amazon Resource Name pekerjaan ('JobArn').

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

Anda dapat menentukan pos pemeriksaan sebagai bagian dari `createJob` API penggunaan `checkpointConfig` parameter.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> LocalPath `checkpointConfig` tidak dapat memulai dengan salah satu jalur cadangan berikut:`/opt/ml`,, `/opt/braket/tmp`, atau. `/usr/local/nvidia`

Mitigasi kesalahan

Mitigasi kesalahan kuantum adalah seperangkat teknik yang bertujuan mengurangi efek kesalahan dalam komputer kuantum.

Perangkat kuantum tunduk pada kebisingan lingkungan yang menurunkan kualitas perhitungan yang dilakukan. Sementara komputasi kuantum toleran kesalahan menjanjikan solusi untuk masalah ini, perangkat kuantum saat ini dibatasi oleh jumlah qubit dan tingkat kesalahan yang relatif tinggi. Untuk mengatasi hal ini dalam waktu dekat, para peneliti sedang menyelidiki metode untuk meningkatkan akurasi komputasi kuantum yang bising. Pendekatan ini, yang dikenal sebagai mitigasi kesalahan kuantum, melibatkan penggunaan berbagai teknik untuk mengekstrak sinyal terbaik dari data pengukuran yang bising.

Mitigasi kesalahan pada IonQ Aria

Mitigasi kesalahan melibatkan menjalankan beberapa sirkuit fisik dan menggabungkan pengukuran mereka untuk memberikan hasil yang lebih baik. IonQ AriaPerangkat ini memiliki metode mitigasi kesalahan yang disebut debiasing.

Debiasing memetakan rangkaian menjadi beberapa varian yang bekerja pada permutasi qubit yang berbeda atau dengan dekomposisi gerbang yang berbeda. Ini mengurangi efek kesalahan sistematis seperti rotasi berlebih gerbang atau qubit tunggal yang salah dengan menggunakan implementasi sirkuit yang berbeda yang dapat membiaskan hasil pengukuran. Ini datang dengan mengorbankan overhead ekstra untuk mengkalibrasi beberapa qubit dan gerbang.

Untuk informasi lebih lanjut tentang debiasing, lihat Meningkatkan kinerja komputer kuantum melalui simetrisasi.

> **Note**
>
> Menggunakan debiasing membutuhkan minimal 2500 tembakan.

Anda dapat menjalankan tugas kuantum dengan debiasing pada IoQ Aria perangkat menggunakan kode berikut:

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

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

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

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

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

Ketika tugas kuantum selesai, Anda dapat melihat probabilitas pengukuran dan jenis hasil apa pun dari tugas kuantum. Probabilitas pengukuran dan hitungan dari semua varian digabungkan menjadi satu distribusi. Setiap jenis hasil yang ditentukan dalam rangkaian, seperti nilai ekspektasi, dihitung menggunakan jumlah pengukuran agregat.

Mengasah

Anda juga dapat mengakses probabilitas pengukuran yang dihitung dengan strategi pasca-pemrosesan yang berbeda yang disebut penajaman. Penajaman membandingkan hasil setiap varian dan membuang bidikan yang tidak konsisten, mendukung hasil pengukuran yang paling mungkin di seluruh varian. Untuk informasi lebih lanjut, lihat Meningkatkan kinerja komputer kuantum melalui simetrisasi.

Yang penting, penajaman mengasumsikan bentuk distribusi keluaran menjadi jarang dengan sedikit keadaan probabilitas tinggi dan banyak keadaan probabilitas nol. Ini dapat mendistorsi distribusi probabilitas jika asumsi ini tidak valid.

Anda dapat mengakses probabilitas dari distribusi yang dipertajam di `additional_metadata` bidang pada SDK Python `GateModelTaskResult` Braket. Perhatikan bahwa penajaman tidak mengembalikan jumlah pengukuran, melainkan mengembalikan distribusi probabilitas yang dinormalisasi ulang. Cuplikan kode berikut menunjukkan cara mengakses distribusi setelah mengasah.

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

Braket Langsung

Dengan Braket Direct, Anda dapat memesan akses khusus ke berbagai perangkat kuantum pilihan Anda, terhubung dengan spesialis komputasi kuantum untuk menerima panduan untuk beban kerja Anda, dan mendapatkan akses awal ke kemampuan generasi berikutnya, seperti perangkat kuantum baru dengan ketersediaan terbatas.

Di bagian ini:

Reservasi

Reservasi memberi Anda akses eksklusif ke perangkat kuantum pilihan Anda. Anda dapat menjadwalkan reservasi sesuai keinginan Anda, sehingga Anda tahu persis kapan beban kerja Anda dimulai dan berakhir eksekusi. Reservasi tersedia dalam kenaikan 1 jam dan dapat dibatalkan hingga 48 jam sebelumnya, tanpa biaya tambahan. Anda dapat memilih untuk mengantri tugas kuantum dan pekerjaan hibrida untuk reservasi yang akan datang sebelumnya, atau mengirimkan beban kerja selama reservasi Anda.

Biaya akses perangkat khusus didasarkan pada durasi reservasi Anda, terlepas dari berapa banyak tugas kuantum dan pekerjaan hibrida yang Anda jalankan di Quantum Processing Unit (QPU).

Komputer kuantum berikut tersedia untuk pemesanan:

- Aria IonQ
- Garnet IQM
- QuEraAquila
- Aspen-M-3 Rigetti

Kapan menggunakan reservasi

Memanfaatkan akses perangkat khusus dengan reservasi memberi Anda kenyamanan dan prediktabilitas untuk mengetahui dengan tepat kapan beban kerja kuantum Anda dimulai dan berakhir eksekusi. Dibandingkan dengan mengirimkan tugas dan pekerjaan hibrida sesuai permintaan, Anda

tidak perlu menunggu dalam antrian dengan tugas pelanggan lainnya. Karena Anda memiliki akses eksklusif ke perangkat selama reservasi, hanya beban kerja Anda yang berjalan di perangkat untuk keseluruhan reservasi.

Sebaiknya gunakan akses sesuai permintaan untuk fase desain dan pembuatan prototipe penelitian Anda, memungkinkan iterasi algoritme Anda yang cepat dan hemat biaya. Setelah Anda siap untuk menghasilkan hasil percobaan akhir, pertimbangkan untuk menjadwalkan reservasi perangkat sesuai keinginan Anda untuk memastikan bahwa Anda dapat memenuhi tenggat waktu proyek atau publikasi. Kami juga merekomendasikan penggunaan reservasi saat Anda menginginkan eksekusi tugas selama waktu tertentu, seperti saat Anda menjalankan demo langsung atau lokakarya di komputer kuantum.

Di bagian ini:

- Buat reservasi
- Jalankan beban kerja Anda dengan reservasi
- Membatalkan atau menjadwal ulang reservasi yang sudah ada

Buat reservasi

Untuk membuat reservasi, hubungi tim Braket dengan mengikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.
2. Pilih Braket Direct di panel kiri, lalu di bagian Reservasi, pilih Perangkat cadangan.
3. Pilih Perangkat yang ingin Anda pesan.
4. Berikan informasi kontak Anda termasuk Nama dan Email. Pastikan untuk memberikan alamat email yang valid yang Anda periksa secara teratur.
5. Di bawah Beri tahu kami tentang beban kerja Anda, berikan detail apa pun tentang beban kerja yang harus dijalankan menggunakan reservasi Anda. Misalnya, panjang reservasi yang diinginkan, batasan yang relevan, atau jadwal yang diinginkan.
6. Jika Anda tertarik untuk terhubung dengan ahli Braket untuk sesi persiapan reservasi setelah reservasi Anda dikonfirmasi, secara opsional pilih Saya tertarik dengan sesi persiapan.

Anda juga dapat menghubungi kami untuk membuat reservasi dengan mengikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.

2. Pilih Perangkat di panel kiri dan pilih perangkat yang ingin Anda pesan.
3. Di bagian Ringkasan, pilih Perangkat cadangan.
4. Ikuti langkah 4-6 dalam prosedur sebelumnya.

Setelah Anda mengirimkan formulir, Anda menerima email dari tim Braket dengan langkah selanjutnya untuk membuat reservasi Anda. Setelah reservasi Anda dikonfirmasi, Anda menerima ARN reservasi melalui email.

> **Note**
>
> Reservasi Anda hanya dikonfirmasi setelah Anda menerima ARN reservasi.

Reservasi tersedia dalam kenaikan minimal 1 jam dan perangkat tertentu mungkin memiliki batasan lama reservasi tambahan (termasuk durasi reservasi minimum dan maksimum). Tim Braket membagikan informasi yang relevan dengan Anda sebelum mengonfirmasi reservasi.

Jika Anda menunjukkan minat pada sesi persiapan reservasi, tim Braket menghubungi Anda melalui email untuk mengatur sesi 30 menit dengan ahli Braket.

Jalankan beban kerja Anda dengan reservasi

Selama reservasi, hanya beban kerja Anda yang berjalan di perangkat. Untuk menunjuk tugas kuantum dan pekerjaan hibrida untuk dijalankan selama reservasi perangkat, Anda harus menggunakan ARN reservasi yang valid.

> **Note**
>
> Reservasi bersifat khusus untuk AWS akun dan perangkat. Hanya AWS akun yang membuat reservasi yang dapat menggunakan ARN reservasi Anda. Selain itu, ARN reservasi hanya berlaku pada perangkat yang dipesan pada waktu mulai dan berakhir yang dipilih.

Untuk memaksimalkan waktu yang Anda pesan, Anda dapat memilih untuk mengantri tugas dan pekerjaan sebelum reservasi Anda. Beban kerja ini tetap dalam `QUEUED` status sampai reservasi dimulai. Ketika reservasi dimulai, setiap beban kerja antrian berjalan dalam urutan yang dikirimkan. Tugas Job diprioritaskan sebelum tugas kuantum mandiri.

> **Note**
>
> Karena hanya beban kerja Anda yang berjalan selama reservasi Anda, tidak ada visibilitas antrian untuk tugas dan pekerjaan yang dikirimkan dengan ARN reservasi.

Contoh kode untuk membuat tugas kuantum untuk reservasi:

1. Tentukan sirkuit untuk menyiapkan keadaan GHZ di fomat OpenQASM.

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

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

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

c = measure q;
```

2. Buat tugas kuantum menggunakan sirkuit Anda dan ARN reservasi.

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

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

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

program = Program(source=ghz_qasm_string)

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

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

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

Contoh kode untuk membuat pekerjaan hybrid untuk reservasi Braket Direct:

1. Tentukan skrip algoritme Anda.

```
//algorithm_script.py

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

def start_here():

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

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

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

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

2. Buat pekerjaan hybrid menggunakan skrip algoritma Anda dan ARN reservasi.

```
from braket.aws import AwsQuantumJob

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

3. Buat pekerjaan hybrid menggunakan dekorator jarak jauh..

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

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

Apa yang terjadi di akhir reservasi Anda

Setelah reservasi berakhir, Anda tidak lagi memiliki akses khusus ke perangkat. Beban kerja yang tersisa yang diantri dengan reservasi ini akan dibatalkan secara otomatis.

> **Note**
>
> Pekerjaan apa pun yang `RUNNING` berstatus saat reservasi berakhir dibatalkan. Kami merekomendasikan menggunakan pos pemeriksaan untuk menyimpan dan memulai kembali pekerjaan sesuai keinginan Anda.

Reservasi yang sedang berlangsung, seperti setelah reservasi dimulai dan sebelum reservasi berakhir, tidak dapat diperpanjang karena setiap reservasi mewakili akses perangkat khusus mandiri. Misalnya, dua back-to-back reservasi dianggap terpisah dan tugas yang tertunda dari reservasi pertama dibatalkan secara otomatis. Mereka tidak melanjutkan di reservasi kedua.

> **Note**
>
> Reservasi mewakili akses perangkat khusus untuk AWS akun Anda. Bahkan jika perangkat tetap menganggur, tidak ada pelanggan lain yang dapat menggunakannya. Oleh karena itu, Anda dikenakan biaya untuk jangka waktu yang dipesan, terlepas dari waktu yang digunakan.

Membatalkan atau menjadwal ulang reservasi yang sudah ada

Anda dapat membatalkan reservasi Anda tidak kurang dari 48 jam sebelum waktu mulai reservasi yang dijadwalkan. Untuk membatalkan, tanggapi email konfirmasi reservasi yang Anda terima dengan permintaan pembatalan Anda.

Untuk menjadwal ulang, Anda harus membatalkan reservasi yang ada, dan kemudian membuat yang baru.

Saran ahli

Terhubung dengan pakar komputasi kuantum langsung di konsol manajemen Braket untuk mendapatkan panduan tambahan seputar beban kerja Anda.

Untuk menjelajahi opsi saran ahli melalui Braket Direct, buka konsol Braket, pilih Braket Direct di panel kiri, dan arahkan ke bagian Saran ahli. Opsi saran ahli berikut tersedia:

- Jam kantor Braket: Jam kantor Braket adalah sesi 1:1, pertama datang pertama dilayani, dan berlangsung setiap bulan. Setiap slot jam kantor yang tersedia adalah 30 menit dan gratis. Berbicara dengan pakar Braket dapat membantu Anda beralih dari ide ke eksekusi lebih cepat

dengan menjelajahi use-case-to-device kecocokan, mengidentifikasi opsi untuk memanfaatkan Braket terbaik untuk algoritme Anda, dan mendapatkan rekomendasi tentang cara menggunakan fitur Braket tertentu seperti Pekerjaan Hibrid Amazon Braket, Pulsa Braket, atau Simulasi Hamiltonian Analog.

- Untuk mendaftar jam kantor Braket, pilih Daftar dan isi informasi kontak, detail beban kerja, dan topik diskusi yang Anda inginkan.
- Anda akan menerima undangan kalender ke slot berikutnya yang tersedia melalui email.

> **Note**
>
> Untuk masalah yang muncul atau pertanyaan pemecahan masalah cepat, kami sarankan untuk menghubungi Support.AWS Untuk pertanyaan yang tidak mendesak, Anda juga dapat menggunakan forum AWS Re:Post atau Quantum Computing Stack Exchange, tempat Anda dapat menelusuri pertanyaan yang dijawab sebelumnya dan mengajukan pertanyaan baru.

- Penawaran penyedia perangkat keras kuantum: IonQ,,, Oxford Quantum Circuits QuEra, dan Rigetti masing-masing menyediakan penawaran layanan profesional melalui. AWS Marketplace
  - Untuk menjelajahi penawaran mereka, pilih Connect dan telusuri daftar mereka.
  - Untuk mempelajari lebih lanjut tentang penawaran layanan profesional di AWS Marketplace, lihat Produk layanan profesional.
- AmazonQuantum Solutions Lab (QSL): QSL adalah tim penelitian kolaboratif dan layanan profesional yang dikelola oleh para ahli komputasi kuantum yang dapat membantu Anda menjelajahi komputasi kuantum secara efektif dan menilai kinerja teknologi saat ini.
  - Untuk menghubungi QSL, pilih Connect, dan isi informasi kontak dan detail kasus penggunaan.
  - Tim QSL akan menghubungi Anda melalui email dengan langkah selanjutnya.

Kemampuan eksperimental

Untuk memajukan beban kerja penelitian Anda, penting untuk mendapatkan akses ke kemampuan inovatif baru dengan cepat. Dengan Braket Direct, Anda dapat meminta akses ke kemampuan eksperimental yang tersedia, seperti perangkat kuantum baru dengan ketersediaan terbatas, langsung di konsol Braket.

Beberapa kemampuan eksperimental beroperasi di luar spesifikasi perangkat standar, dan memerlukan panduan langsung yang disesuaikan dengan kasus penggunaan Anda. Untuk

memastikan beban kerja Anda diatur untuk sukses, akses tersedia berdasarkan permintaan melalui Braket Direct.

Akses khusus reservasi ke iONQ Forte

Dengan Braket Direct, Anda mendapatkan akses reservasi saja ke IonQ Forte QPU. Karena ketersediaannya yang terbatas, perangkat ini hanya tersedia melalui Braket Direct.

Untuk mempelajari lebih lanjut dan meminta akses ke IonQ Forte, ikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.
2. Pilih Braket Direct di menu kiri, dan kemudian dalam kemampuan Eksperimental navigasikan ke ionQ Forte. Pilih Lihat perangkat.
3. Pada halaman detail perangkat Forte, di Ringkasan pilih Perangkat cadangan.
4. Berikan informasi kontak Anda, termasuk Nama dan Email. Berikan alamat email yang valid yang Anda periksa secara teratur.
5. Di bawah Beri tahu kami tentang beban kerja Anda, berikan detail tentang beban kerja yang harus dijalankan menggunakan reservasi Anda, seperti lama reservasi yang diinginkan, batasan yang relevan, atau jadwal yang diinginkan.
6. (Opsional) Jika Anda tertarik untuk terhubung dengan ahli Braket untuk sesi persiapan reservasi setelah reservasi Anda dikonfirmasi, pilih Saya tertarik dengan sesi persiapan.

Setelah formulir dikirimkan, tim Braket akan menghubungi Anda dengan langkah selanjutnya.

> **Note**
>
> Karena ketersediaan perangkat yang terbatas, akses ke Forte terbatas. Hubungi kami untuk mempelajari lebih lanjut.

Akses ke detuning lokal di Aquila QuEra

Dengan Braket Direct, Anda dapat meminta akses untuk mengontrol detuning lokal saat memprogram pada QPU. QuEra Aquila Dengan kemampuan ini, Anda dapat menyesuaikan seberapa besar bidang mengemudi memengaruhi setiap qubit tertentu.

Untuk mempelajari lebih lanjut dan meminta akses ke kemampuan ini, ikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.
2. Pilih Braket Direct di menu kiri, dan kemudian di kemampuan Eksperimental navigasikan ke QuEra Aquila - detuning lokal. Pilih Dapatkan Akses.
3. Berikan informasi kontak Anda, termasuk Nama dan Email. Berikan alamat email yang valid yang Anda periksa secara teratur.
4. Di bawah Beri tahu kami tentang beban kerja Anda, berikan detail tentang beban kerja dan di mana Anda berencana untuk menggunakan kemampuan ini..

Akses ke geometri tinggi di Aquila QuEra

Dengan Braket Direct, Anda dapat meminta akses ke geometri yang diperluas saat memprogram pada QPU. QuEra Aquila Dengan kemampuan ini, Anda dapat bereksperimen di luar kemampuan perangkat standar dan menentukan geometri dengan peningkatan ketinggian kisi.

Untuk mempelajari lebih lanjut dan meminta akses ke kemampuan ini, ikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.
2. Pilih Braket Direct di menu kiri, dan kemudian dalam kemampuan Eksperimental navigasikan ke QuEra Aquila - geometri tinggi. Pilih Dapatkan akses.
3. Berikan informasi kontak Anda, termasuk Nama dan Email. Berikan alamat email yang valid yang Anda periksa secara teratur.
4. Di bawah Beri tahu kami tentang beban kerja Anda, berikan detail tentang beban kerja dan di mana Anda berencana untuk menggunakan kemampuan ini..

Akses ke geometri ketat di Aquila QuEra

Dengan Braket Direct, Anda dapat meminta akses ke geometri yang diperluas saat memprogram pada QPU. QuEra Aquila Dengan kemampuan ini, Anda dapat bereksperimen di luar kemampuan perangkat standar dan mengatur baris kisi dengan jarak vertikal yang lebih ketat.

Untuk mempelajari lebih lanjut dan meminta akses ke kemampuan ini, ikuti langkah-langkah berikut:

1. Buka konsol Amazon Braket.
2. Pilih Braket Direct di menu kiri, dan kemudian dalam kemampuan Eksperimental navigasikan ke QuEra Aquila - geometri tinggi. Pilih Dapatkan akses.

3. Berikan informasi kontak Anda, termasuk Nama dan Email. Berikan alamat email yang valid yang Anda periksa secara teratur.
4. Di bawah Beri tahu kami tentang beban kerja Anda, berikan detail tentang beban kerja dan di mana Anda berencana untuk menggunakan kemampuan ini..

Pembuatan Log dan Pemantauan

Setelah Anda mengirimkan tugas kuantum, Anda dapat melacak statusnya melalui Amazon Braket SDK dan konsol. Saat tugas kuantum selesai, Braket menyimpan hasilnya di lokasi Amazon S3 yang Anda tentukan. Penyelesaian mungkin memakan waktu lama, terutama untuk perangkat QPU, tergantung pada panjang antrian. Jenis status meliputi:

- `CREATED`— Amazon Braket menerima tugas kuantum Anda.
- `QUEUED`— Amazon Braket memproses tugas kuantum Anda dan sekarang menunggu untuk berjalan di perangkat.
- `RUNNING`— Tugas kuantum Anda berjalan pada QPU atau simulator sesuai permintaan.
- `COMPLETED`— Tugas kuantum Anda selesai berjalan di QPU atau simulator sesuai permintaan.
- `FAILED`— Tugas kuantum Anda berusaha untuk berjalan dan gagal. Bergantung pada alasan tugas kuantum Anda gagal, coba kirimkan tugas kuantum Anda lagi.
- `CANCELLED`— Anda membatalkan tugas kuantum. Tugas kuantum tidak berjalan.

Di bagian ini:

Melacak tugas kuantum dari Amazon Braket SDK

Perintah `device.run(…)` mendefinisikan tugas kuantum dengan ID tugas kuantum yang unik. Anda dapat membuat kueri dan melacak status dengan `task.state()` seperti yang ditunjukkan dalam contoh berikut.

Catatan: `task = device.run()` adalah operasi asinkron, yang berarti Anda dapat terus bekerja saat sistem memproses tugas kuantum Anda di latar belakang.

Ambil hasilnya

Saat Anda menelepon`task.result()`, SDK mulai melakukan polling Amazon Braket untuk melihat apakah tugas kuantum selesai. SDK menggunakan parameter jajak pendapat yang Anda tetapkan di `.run()`. Setelah tugas kuantum selesai, SDK mengambil hasil dari bucket S3 dan mengembalikannya sebagai objek. `QuantumTaskResult`

```
# create a circuit, specify the device and run the circuit
circ = Circuit().rx(0, 0.15).ry(1, 0.2).cnot(0,2)
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")
task = device.run(circ, s3_location, shots=1000)

# get ID and status of submitted task
task_id = task.id
status = task.state()
print('ID of task:', task_id)
print('Status of task:', status)
# wait for job to complete
while status != 'COMPLETED':
    status = task.state()
    print('Status:', status)
```

```
ID of task:
arn:aws:braket:us-west-2:123412341234:quantum-task/b68ae94b-1547-4d1d-aa92-1500b82c300d
Status of task: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: QUEUED
Status: RUNNING
Status: RUNNING
Status: COMPLETED
```

Batalkan tugas kuantum

Untuk membatalkan tugas kuantum, panggil `cancel()` metode, seperti yang ditunjukkan pada contoh berikut.

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

```
Status of task: CANCELLING
```

Periksa metadata

Anda dapat memeriksa metadata tugas kuantum yang sudah selesai, seperti yang ditunjukkan pada contoh berikut.

```
# get the metadata of the quantum task
metadata = task.metadata()
# example of metadata
shots = metadata['shots']
date = metadata['ResponseMetadata']['HTTPHeaders']['date']
# print example metadata
print("{} shots taken on {}.".format(shots, date))

# print name of the s3 bucket where the result is saved
results_bucket = metadata['outputS3Bucket']
print('Bucket where results are stored:', results_bucket)
# print the s3 object key (folder name)
results_object_key = metadata['outputS3Directory']
print('S3 object key:', results_object_key)

# the entire look-up string of the saved result data
look_up = 's3://'+results_bucket+'/'+results_object_key
print('S3 URI:', look_up)
```

```
1000 shots taken on Wed, 05 Aug 2020 14:44:22 GMT.
Bucket where results are stored: amazon-braket-123412341234
S3 object key: simulation-output/b68ae94b-1547-4d1d-aa92-1500b82c300d
S3 URI: s3://amazon-braket-123412341234/simulation-output/b68ae94b-1547-4d1d-
aa92-1500b82c300d
```

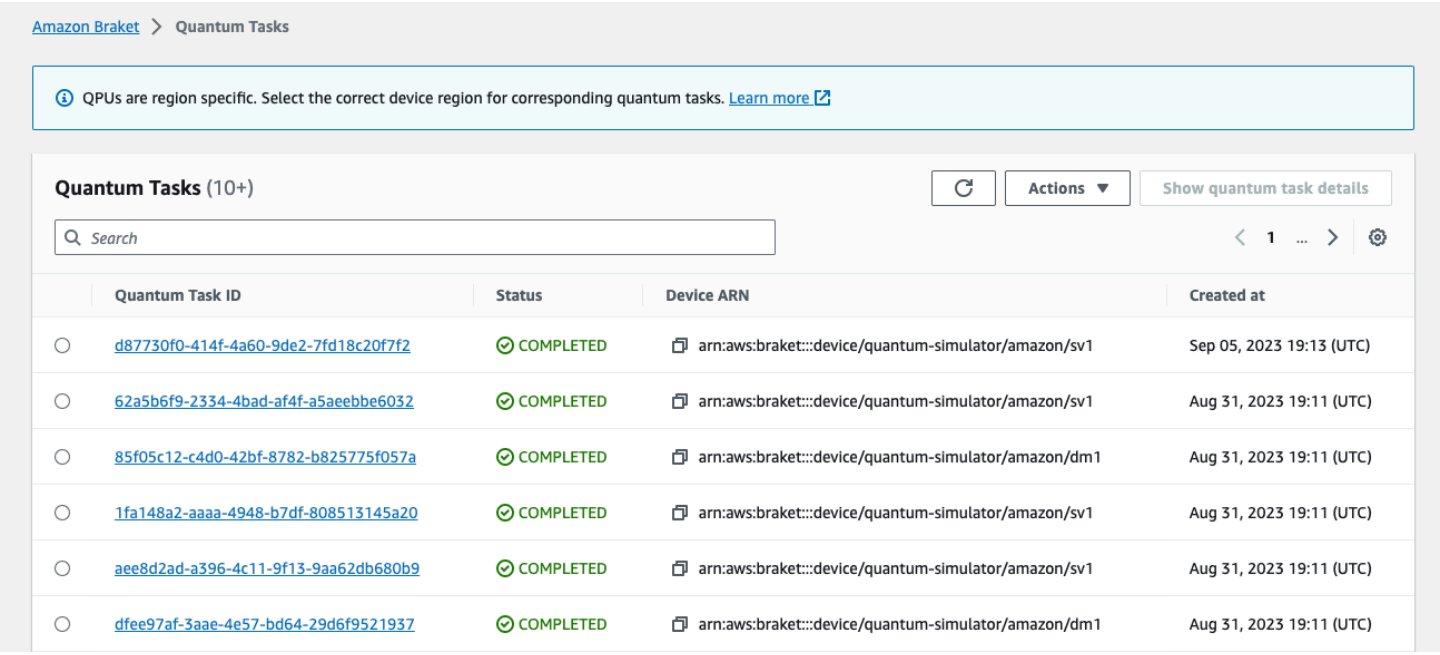
Mengambil tugas atau hasil kuantum

Jika kernel Anda mati setelah Anda mengirimkan tugas kuantum atau jika Anda menutup notebook atau komputer Anda, Anda dapat merekonstruksi `task` objek dengan ARN (Quantum Task ID) yang unik. Kemudian Anda dapat memanggil `task.result()` untuk mendapatkan hasil dari bucket S3 tempat penyimpanannya.

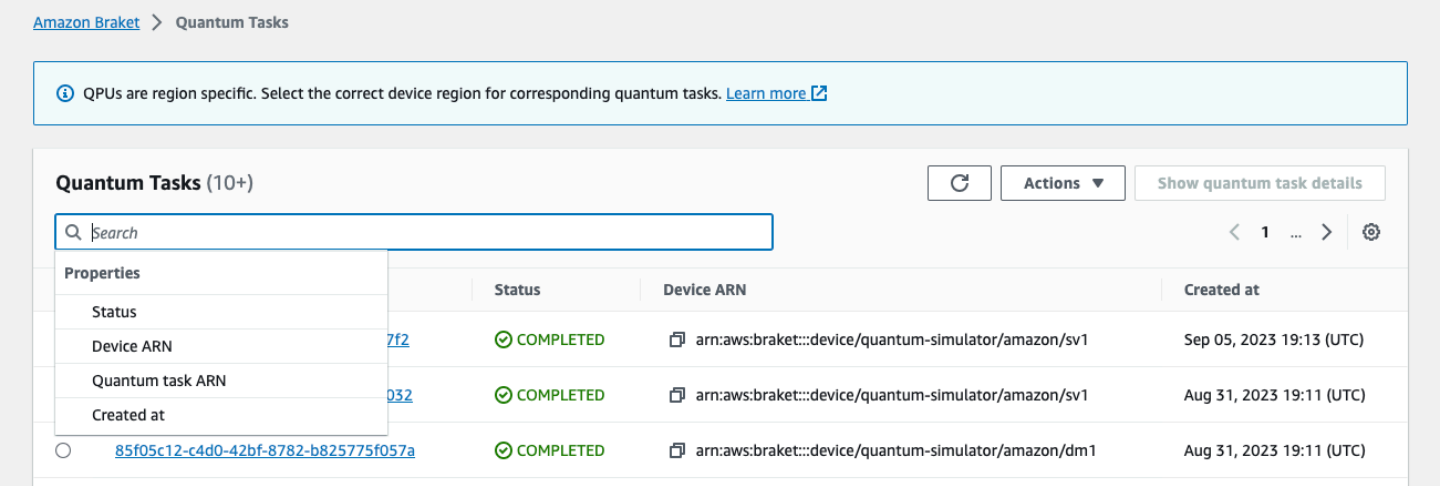
```
from braket.aws import AwsSession, AwsQuantumTask

# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Memantau tugas kuantum melalui konsol Amazon Braket

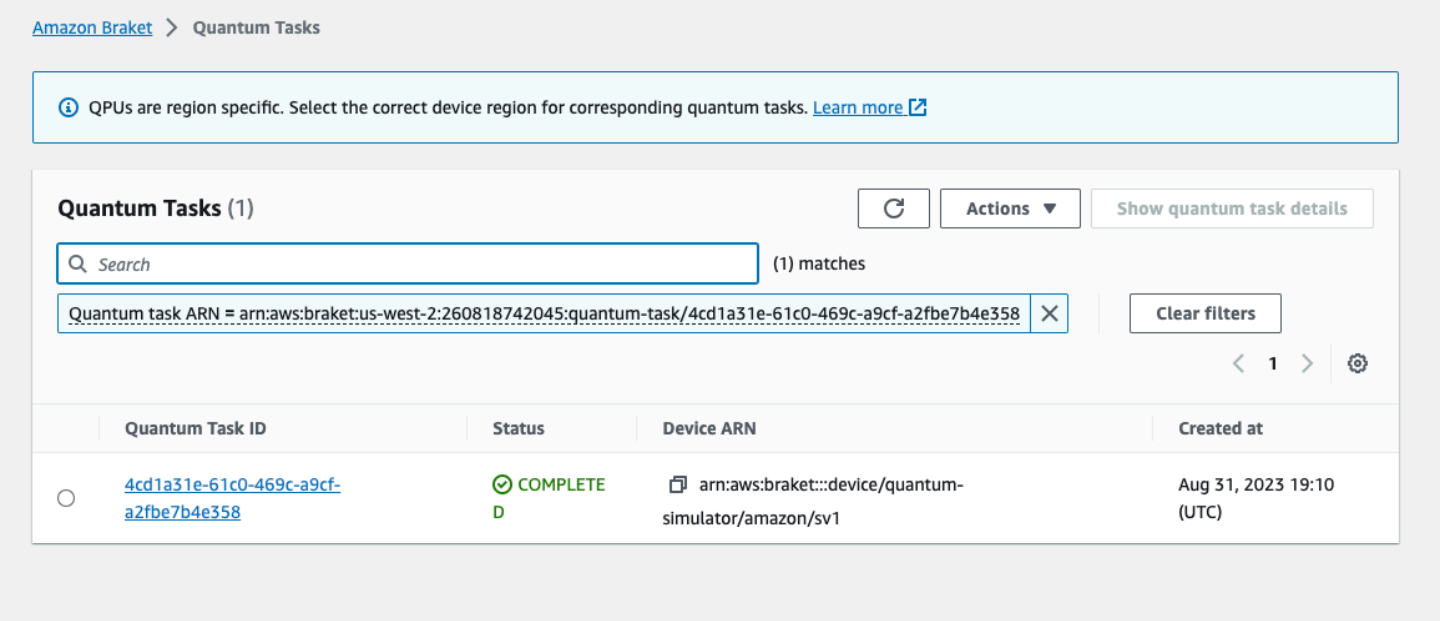
AmazonBraket menawarkan cara mudah untuk memantau tugas kuantum melalui konsol Amazon Braket. Semua tugas kuantum yang dikirimkan tercantum di bidang Quantum Tasks seperti yang ditunjukkan pada gambar berikut. Layanan ini khusus Wilayah, yang berarti Anda hanya dapat melihat tugas-tugas kuantum yang dibuat secara spesifik. Wilayah AWS



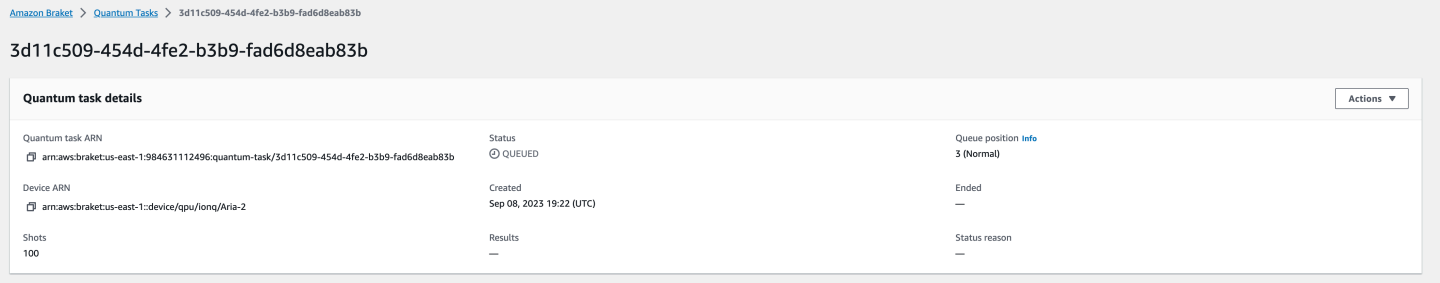
Anda dapat mencari tugas kuantum tertentu melalui bilah navigasi. Pencarian dapat didasarkan pada Quantum Task ARN (ID), status, perangkat, dan waktu pembuatan. Opsi muncul secara otomatis ketika Anda memilih bilah navigasi, seperti yang ditunjukkan pada contoh berikut.



Gambar berikut menunjukkan contoh pencarian tugas kuantum berdasarkan ID tugas kuantum uniknya, yang dapat diperoleh dengan menelepon`task.id`.



Selain itu, terlihat pada gambar di bawah ini, status tugas kuantum dapat dipantau saat berada dalam QUEUED keadaan. Mengklik pada ID tugas kuantum menunjukkan halaman detail. Halaman ini menampilkan posisi antrian dinamis untuk tugas kuantum Anda relatif terhadap perangkat yang akan diprosesnya.



Tugas kuantum yang diajukan sebagai bagian dari pekerjaan hibrida akan diprioritaskan saat dalam antrian. Tugas kuantum yang diajukan di luar pekerjaan hibrida akan memiliki prioritas antrian normal.

Pelanggan yang ingin menanyakan SDK Braket, dapat memperoleh tugas kuantum dan posisi antrian pekerjaan hibrida mereka secara terprogram. Untuk informasi lebih lanjut, lihat halaman Kapan tugas saya akan dijalankan.

Penandaan sumber daya Amazon Braket

Tag adalah label atribut kustom yang Anda tetapkan atau yang ditetapkan ke AWS sumber daya. AWS Tag adalah metadata yang menceritakan lebih lanjut tentang sumber daya Anda. Setiap tanda terdiri atas kunci dan nilai. Bersama-sama ini dikenal sebagai pasangan nilai kunci. Untuk tag yang Anda tetapkan, Anda menentukan kunci dan nilai.

Di konsol Amazon Braket, Anda dapat menavigasi ke tugas kuantum atau buku catatan dan melihat daftar tag yang terkait dengannya. Anda dapat menambahkan tag, menghapus tag, atau memodifikasi tag. Anda dapat menandai tugas kuantum atau buku catatan saat pembuatan, dan kemudian mengelola tag terkait melalui konsol, AWS CLI, atauAPI.

Menggunakan tag

Tag dapat mengatur sumber daya Anda ke dalam kategori yang berguna bagi Anda. Misalnya, Anda dapat menetapkan tag "Departemen" untuk menentukan departemen yang memiliki sumber daya ini.

Setiap tag memiliki dua bagian:

- Kunci tag (misalnya, CostCenter, Lingkungan, atau Proyek). Kunci tag peka huruf besar dan kecil.
- Bidang opsional yang dikenal sebagai nilai tag (misalnya, 111122223333 atau Produksi). Mengabaikan nilai tag sama dengan menggunakan string kosong. Seperti kunci tag, nilai tag peka huruf besar dan kecil.

Tag membantu Anda melakukan hal-hal berikut:

- Identifikasi dan atur AWS sumber daya Anda. Banyak penandaan Layanan AWS dukungan, sehingga Anda dapat menetapkan tag yang sama ke sumber daya dari layanan yang berbeda untuk menunjukkan bahwa sumber daya terkait.
- Lacak AWS biaya Anda. Anda mengaktifkan tag ini di AWS Billing and Cost Management dasbor. AWS menggunakan tag untuk mengkategorikan biaya Anda dan mengirimkan laporan alokasi biaya bulanan kepada Anda. Untuk informasi selengkapnya, lihat Menggunakan tag alokasi biaya di Panduan AWS Billing and Cost Management Pengguna.
- Kontrol akses ke AWS sumber daya Anda. Untuk informasi selengkapnya, lihat Mengontrol akses menggunakan tag.

Lebih lanjut tentang AWS dan tag

- Untuk informasi umum tentang penandaan, termasuk konvensi penamaan dan penggunaan, lihat Menandai AWS Sumber Daya di Referensi Umum.AWS
- Untuk informasi tentang pembatasan penandaan, lihat Batas dan persyaratan penamaan tag di Referensi AWS Umum.
- Untuk praktik terbaik dan strategi penandaan, lihat Menandai praktik terbaik dan Strategi AWS Penandaan.
- Untuk daftar layanan yang men-support penggunaan tag, lihat Referensi API Penandaan Resource Groups.

Bagian berikut memberikan informasi yang lebih spesifik tentang tag untuk Amazon Braket.

Sumber Daya yang didukung di Amazon Braket

Jenis sumber daya berikut di Amazon Braket mendukung penandaan:

- Sumber daya `quantum-task`
- Nama sumber daya: `AWS::Service::Braket`
- ARN Regex: `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

Catatan: Anda dapat menerapkan dan mengelola tag untuk notebook Amazon Braket di konsol Amazon Braket, dengan menggunakan konsol untuk menavigasi ke sumber daya notebook, meskipun notebook sebenarnya adalah sumber daya Amazon. SageMaker Untuk informasi selengkapnya, lihat Metadata Instance Notebook di dokumentasi. SageMaker

Batasan tag

Pembatasan dasar berikut berlaku untuk tag pada sumber daya Amazon Braket:

- Jumlah tanda maksimum tag yang dapat Anda tetapkan ke sumber daya: 50
- Panjang kunci maksimum: 128 karakter Unicode
- Panjang nilai maksimum: 256 karakter Unicode
- Karakter yang valid untuk kunci dan nilai: `a-z, A-Z, 0-9, space`, dan karakter-karakter ini: `_ . : / = + -` dan `@`
- Kunci dan nilai peka huruf besar dan kecil.
- Jangan gunakan `aws` sebagai awalan untuk kunci; itu dicadangkan untuk AWS digunakan.

Mengelola tag di Amazon Braket

Anda mengatur tag sebagai Properti di sumber daya. Anda dapat melihat, menambah, memodifikasi, daftar, dan menghapus tag melalui konsol Amazon Braket, Amazon BraketAPI, atau. AWS CLI Untuk informasi lebih lanjut, lihat Referensi API Amazon Braket.

Tambahkan tag

Anda dapat menambahkan sumber daya yang dapat ditag pada waktu-waktu berikut:

- Saat Anda membuat sumber daya: Gunakan konsol, atau sertakan `Tags` parameter dengan `Create` operasi di AWS API.
- Setelah Anda membuat sumber daya: Gunakan konsol untuk menavigasi ke tugas kuantum atau sumber daya notebook, atau panggil `TagResource` operasi di AWS API.

Untuk menambahkan tag ke sumber daya saat Anda membuatnya, Anda juga memerlukan izin untuk membuat sumber daya dari jenis yang ditentukan.

Lihat tag

Anda dapat melihat tag di salah satu sumber daya yang dapat diberi tag di Amazon Braket dengan menggunakan konsol untuk menavigasi ke tugas atau sumber daya buku catatan, atau dengan memanggil operasi. AWS `ListTagsForResource` API

Anda dapat menggunakan AWS API perintah berikut untuk melihat tag pada sumber daya:

- AWS API: `ListTagsForResource`

Mengedit tag

Anda dapat mengedit tag menggunakan konsol untuk menavigasi ke tugas kuantum atau sumber daya buku catatan atau Anda dapat menggunakan perintah berikut untuk mengubah nilai tag yang dilampirkan ke sumber daya yang dapat diberi tag. Ketika Anda menentukan kunci tag yang sudah ada, nilai untuk kunci tersebut ditimpa:

- AWS API: `TagResource`

Hapus tag

Anda dapat menghapus tag dari sumber daya dengan menentukan kunci yang akan dihapus, dengan menggunakan konsol untuk menavigasi ke tugas kuantum atau sumber daya notebook, atau saat memanggil `UntagResource` operasi.

- AWS API: `UntagResource`

Contoh penandaan CLI di Amazon Braket

Jika Anda bekerja dengan AWS CLI, berikut adalah contoh perintah yang menunjukkan cara membuat tag yang berlaku untuk tugas kuantum yang Anda buat SV1 dengan pengaturan parameter QPU. Rigetti Perhatikan bahwa tag ditentukan pada akhir perintah contoh. Dalam kasus ini, Kunci diberi nilai `state` dan Nilai diberi nilai `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\"}
```

Menandai dengan Amazon Braket API

- Jika Anda menggunakan Amazon Braket API untuk menyiapkan tag pada sumber daya, hubungi. TagResourceAPI

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

- Untuk menghapus tag dari sumber daya, panggil file UntagResourceAPI.

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

- Untuk mencantumkan semua tag yang dilampirkan ke sumber daya tertentu, hubungi file ListTagsForResourceAPI.

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

Acara dan tindakan otomatis untuk Amazon Braket dengan Amazon EventBridge

Amazon EventBridge memantau peristiwa perubahan status dalam tugas kuantum Amazon Braket. Acara dari Amazon Braket dikirim ke EventBridge, hampir secara real time. Anda dapat

menulis aturan sederhana untuk menunjukkan kejadian mana yang sesuai kepentingan Anda, dan memasukkan tindakan otomatis apa yang diambil ketika suatu peristiwa sesuai dengan suatu aturan. Tindakan otomatis yang dapat dipicu meliputi berikut ini:

- Memanggil fungsi AWS Lambda
- Mengaktifkan mesin AWS Step Functions negara
- Memberi tahu topik Amazon SNS

EventBridge memantau peristiwa perubahan status Amazon Braket ini:

- Keadaan tugas qauntum berubah

AmazonBraket menjamin pengiriman peristiwa perubahan status tugas kuantum. Peristiwa ini disampaikan setidaknya sekali, tetapi mungkin rusak.

Untuk informasi selengkapnya, lihat Peristiwa dan Pola Peristiwa di EventBridge.

Di bagian ini:

- Pantau status tugas kuantum dengan EventBridge
- Contoh acara Amazon Braket EventBridge

Pantau status tugas kuantum dengan EventBridge

Dengan EventBridge, Anda dapat membuat aturan yang menentukan tindakan yang harus diambil saat Amazon Braket mengirimkan pemberitahuan perubahan status terkait tugas kuantum Braket. Misalnya, Anda dapat membuat aturan yang mengirimi Anda pesan email setiap kali status tugas kuantum berubah.

1. Masuk untuk AWS menggunakan akun yang memiliki izin untuk menggunakan EventBridge dan Amazon Braket.
2. Buka EventBridge konsol Amazon di https://console.aws.amazon.com/events/.
3. Dengan menggunakan nilai-nilai berikut, buat EventBridge aturan:
   - Untuk Tipe aturan, pilih Aturan dengan pola peristiwa.
   - Untuk sumber acara, pilih Lainnya.
   - Di bagian Pola acara, pilih Pola kustom (editor JSON), lalu tempelkan pola acara berikut ke area teks:

```
{
  "source": [
    "aws.braket"
  ],
  "detail-type": [
    "Braket Task State Change"
  ]
}
```

Untuk menangkap semua peristiwa dari Amazon Braket, kecualikan `detail-type` bagian seperti yang ditunjukkan pada kode berikut:

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

- Untuk jenis Target Layanan AWS, pilih, dan untuk Pilih target, pilih target seperti topik atau AWS Lambda fungsi Amazon SNS. Target dipicu ketika peristiwa perubahan status tugas kuantum diterima dari Amazon Braket.

  Misalnya, Anda dapat menggunakan topik Amazon Simple Notification Service (SNS) untuk mengirim email atau pesan teks ketika peristiwa terjadi. Caranya, Anda harus terlebih dahulu membuat topik Amazon SNS menggunakan konsol Amazon SNS. Untuk mempelajari lebih lanjut, lihat Menggunakan Amazon SNS untuk pemberitahuan pengguna.

Untuk detail tentang membuat aturan, lihat Membuat EventBridge aturan Amazon yang bereaksi terhadap peristiwa.

Contoh acara Amazon Braket EventBridge

Untuk informasi tentang bidang untuk peristiwa Perubahan Status Tugas Quantum Amazon Braket, lihat Peristiwa dan Pola Peristiwa di EventBridge.

Atribut berikut muncul di bidang “detail” JSON.

- **`quantumTaskArn`**(str): Tugas kuantum tempat peristiwa ini dihasilkan.
- **`status`**(Opsional [str]): Status transisi tugas kuantum.

- **`deviceArn`**(str): Perangkat yang ditentukan oleh pengguna tempat tugas kuantum ini dibuat.
- shots(int): Jumlah yang shots diminta oleh pengguna.
- **`outputS3Bucket`**(str): Bucket keluaran yang ditentukan oleh pengguna.
- **`outputS3Directory`**(str): Output key prefix ditentukan oleh pengguna.
- **`createdAt`**(str): Waktu pembuatan tugas kuantum sebagai string ISO-8601.
- **`endedAt`**(Opsional [str]): Waktu di mana tugas kuantum mencapai status terminal. Bidang ini hadir hanya ketika tugas kuantum telah dialihkan ke status terminal.

Kode JSON berikut menunjukkan contoh peristiwa Perubahan Status Tugas Quantum Amazon Braket.

```
{
    "version":"0",
    "id":"6101452d-8caf-062b-6dbc-ceb5421334c5",
    "detail-type":"Braket Task State Change",
    "source":"aws.braket",
    "account":"012345678901",
    "time":"2021-10-28T01:17:45Z",
    "region":"us-east-1",
    "resources":[
        "arn:aws:braket:us-east-1:012345678901:quantum-task/834b21ed-77a7-4b36-a90c-
c776afc9a71e"
    ],
    "detail":{
        "quantumTaskArn":"arn:aws:braket:us-east-1:012345678901:quantum-
task/834b21ed-77a7-4b36-a90c-c776afc9a71e",
        "status":"COMPLETED",
        "deviceArn":"arn:aws:braket:::device/quantum-simulator/amazon/sv1",
        "shots":"100",
        "outputS3Bucket":"amazon-braket-0260a8bc871e",
        "outputS3Directory":"sns-testing/834b21ed-77a7-4b36-a90c-c776afc9a71e",
        "createdAt":"2021-10-28T01:17:42.898Z",
        "eventName":"MODIFY",
        "endedAt":"2021-10-28T01:17:44.735Z"
      }
}
```

Memantau Amazon Braket dengan Amazon CloudWatch

Anda dapat memantau Amazon Braket menggunakan Amazon CloudWatch, yang mengumpulkan data mentah dan memprosesnya menjadi metrik yang dapat dibaca, mendekati real-time. Anda melihat informasi historis yang dihasilkan hingga 15 bulan yang lalu atau metrik pencarian yang telah diperbarui dalam 2 minggu terakhir di CloudWatch konsol Amazon untuk mendapatkan perspektif yang lebih baik tentang kinerja Amazon Braket. Untuk mempelajari lebih lanjut, lihat Menggunakan CloudWatch metrik.

Metrik dan Dimensi Amazon Braket

Metrik adalah konsep dasar dalam CloudWatch. Metrik mewakili kumpulan titik data yang diurutkan waktu yang dipublikasikan ke CloudWatch. Setiap metrik dicirikan oleh serangkaian dimensi. Untuk mempelajari lebih lanjut tentang dimensi metrik CloudWatch, lihat CloudWatch dimensi.

Amazon Braket mengirimkan data metrik berikut, khusus untuk Amazon Braket, ke dalam metrik Amazon: CloudWatch

Metrik Tugas Kuantum

Metrik tersedia jika ada tugas kuantum. Mereka ditampilkan di AWS bawah/Braket/By Device di konsol. CloudWatch

Metrik	Deskripsi
Hitung	Jumlah tugas kuantum.
Latensi	Metrik ini dipancarkan ketika tugas kuantum telah selesai. Ini mewakili total waktu dari inisialisasi tugas kuantum hingga penyelesaian.

Dimensi untuk Metrik Tugas Kuantum

Metrik tugas kuantum diterbitkan dengan dimensi berdasarkan `deviceArn` parameter, yang memiliki bentuk arn:aws:braket: ::device/xxx.

Perangkat yang Didukung

Untuk daftar perangkat yang didukung dan perangkat ARN, lihat Perangkat Braket.

> **Note**
>
> Anda dapat melihat aliran CloudWatch log untuk notebook Amazon Braket dengan menavigasi ke halaman detail Notebook di konsol Amazon. SageMaker Pengaturan notebook Amazon Braket tambahan tersedia melalui konsol. SageMaker

Pencatatan API Amazon Braket dengan CloudTrail

AmazonBraket terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau Layanan AWS di Amazon Braket. CloudTrail menangkap semua API panggilan untuk Amazon Braket sebagai acara. Panggilan yang diambil termasuk panggilan dari konsol Amazon Braket dan panggilan kode ke operasi Amazon Braket Braket. Jika Anda membuat jejak, Anda dapat mengaktifkan pengiriman CloudTrail acara secara terus menerus ke bucket Amazon S3, termasuk acara untuk Amazon Braket. Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di CloudTrail konsol di Riwayat acara. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Amazon Braket, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

Informasi Amazon Braket di CloudTrail

CloudTrail diaktifkan pada Akun AWS saat Anda membuat akun. Ketika aktivitas terjadi di Amazon Braket, aktivitas tersebut dicatat dalam suatu CloudTrail peristiwa bersama dengan Layanan AWS peristiwa lain dalam sejarah Peristiwa. Anda dapat melihat, mencari, dan mengunduh acara terbaru di situs Anda Akun AWS. Untuk informasi selengkapnya, lihat Melihat Acara dengan Riwayat CloudTrail Acara.

Untuk catatan acara yang sedang berlangsung di Anda Akun AWS, termasuk acara untuk Amazon Braket, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi lainnya Layanan AWS untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat berikut:

- Gambaran Umum untuk Membuat Jejak
- CloudTrail Layanan dan Integrasi yang Didukung
- Mengonfigurasi Notifikasi Amazon SNS untuk CloudTrail
- Menerima File CloudTrail Log dari Beberapa Wilayah dan Menerima File CloudTrail Log dari Beberapa Akun

Semua tindakan Amazon Braket dicatat oleh CloudTrail. Misalnya, panggilan ke `GetQuantumTask` atau `GetDevice` tindakan menghasilkan entri dalam file CloudTrail log.

Setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut ini:

- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna terfederasi.
- Apakah permintaan tersebut dibuat oleh Layanan AWS lain.

Untuk informasi selengkapnya, lihat Elemen CloudTrail UserIdentity.

Memahami Entri Berkas Log Amazon Braket

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari API panggilan publik, sehingga tidak muncul dalam urutan tertentu.

Contoh berikut adalah entri log untuk tindakan `GetQuantumTask`, yang mendapat detail tugas kuantum.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
```

```
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
        "userName": "foobar"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:56:57Z"
      }
    }
  },
  "eventTime": "2020-08-07T01:00:08Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetQuantumTask",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "aws-cli/1.18.110 Python/3.6.10
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.17.33",
  "requestParameters": {
    "quantumTaskArn": "foobar"
  },
  "responseElements": null,
  "requestID": "20e8000c-29b8-4137-9cbc-af77d1dd12f7",
  "eventID": "4a2fdb22-a73d-414a-b30f-c0797c088f7c",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```

Berikut ini menunjukkan entri log untuk tindakan `GetDevice`, yang mengembalikan detail kejadian perangkat.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
```

```
    "accessKeyId": "foobar",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
        "userName": "foobar"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:46:29Z"
      }
    }
  },
  "eventTime": "2020-08-07T00:46:32Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetDevice",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "Boto3/1.14.33 Python/3.7.6 Linux/4.14.158-129.185.amzn2.x86_64 exec-
env/AWS_ECS_FARGATE Botocore/1.17.33",
  "errorCode": "404",
  "requestParameters": {
    "deviceArn": "foobar"
  },
  "responseElements": null,
  "requestID": "c614858b-4dcf-43bd-83c9-bcf9f17f522e",
  "eventID": "9642512a-478b-4e7b-9f34-75ba5a3408eb",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```


Buat instance notebook Amazon Braket menggunakan AWS CloudFormation

Anda dapat menggunakan AWS CloudFormation untuk mengelola instans notebook Amazon Braket Anda. Instans notebook Braket dibangun di Amazon. SageMaker Dengan CloudFormation, Anda dapat menyediakan instance notebook dengan file template yang menjelaskan konfigurasi

yang dimaksud. File template ditulis dalam format JSON atau YAMAL. Anda dapat membuat, memperbarui, dan menghapus instance secara teratur dan berulang. Anda mungkin menemukan ini berguna ketika Anda mengelola beberapa instance notebook Braket dalam diri Anda. Akun AWS

Setelah Anda membuat CloudFormation template untuk notebook Braket, Anda gunakan AWS CloudFormation untuk menyebarkan sumber daya. Untuk informasi selengkapnya, lihat Membuat tumpukan di AWS CloudFormation konsol di panduan AWS CloudFormation pengguna.

Untuk membuat instance notebook Braket menggunakan CloudFormation, Anda melakukan tiga langkah berikut:

1. Buat skrip konfigurasi SageMaker siklus hidup Amazon.
2. Buat peran AWS Identity and Access Management (IAM) yang akan diasumsikan oleh SageMaker.
3. Buat instance SageMaker notebook dengan awalan **`amazon-braket-`**

Anda dapat menggunakan kembali konfigurasi siklus hidup untuk semua notebook Braket yang Anda buat. Anda juga dapat menggunakan kembali peran IAM untuk notebook Braket yang Anda tetapkan izin eksekusi yang sama.

Langkah 1: Buat skrip konfigurasi SageMaker siklus hidup Amazon

Gunakan template berikut untuk membuat skrip konfigurasi SageMaker siklus hidup. Skrip menyesuaikan instance SageMaker notebook untuk Braket. Untuk opsi konfigurasi CloudFormation sumber daya siklus hidup, lihat AWS::SageMaker::NotebookInstanceLifecycleConfigdi panduan AWS CloudFormation pengguna.

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

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

```
            ./install.sh
            EOS

            exit 0
```

Langkah 2: Buat peran IAM yang diasumsikan oleh Amazon SageMaker

Saat Anda menggunakan instance notebook Braket, SageMaker lakukan operasi atas nama Anda. Misalnya, Anda menjalankan notebook Braket menggunakan sirkuit pada perangkat yang didukung. Dalam instance notebook, SageMaker jalankan operasi pada Braket untuk Anda. Peran eksekusi notebook mendefinisikan operasi yang tepat yang SageMaker diizinkan untuk dijalankan atas nama Anda. Untuk informasi selengkapnya, lihat SageMaker peran dalam panduan SageMaker pengembang Amazon.

Gunakan contoh berikut untuk membuat peran eksekusi notebook Braket dengan izin yang diperlukan. Anda dapat memodifikasi kebijakan sesuai dengan kebutuhan Anda.

> **Note**
>
> Pastikan bahwa peran tersebut memiliki izin untuk `s3:ListBucket` dan `s3:GetObject` operasi di bucket Amazon S3 yang diawali dengan awalan. `braketnotebookcdk-"` Skrip konfigurasi siklus hidup memerlukan izin ini untuk menyalin skrip instalasi notebook Braket.

```
ExecutionRole:
    Type: "AWS::IAM::Role"
    Properties:
      RoleName: !Sub AmazonBraketNotebookRole-${AWS::StackName}
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "sagemaker.amazonaws.com"
          Action:
          - "sts:AssumeRole"
      Path: "/service-role/"
      ManagedPolicyArns:
```

```
        - arn:aws:iam::aws:policy/AmazonBraketFullAccess
      Policies:
        -
          PolicyName: "AmazonBraketNotebookPolicy"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: Allow
                Action:
                  - s3:GetObject
                  - s3:PutObject
                  - s3:ListBucket
                Resource:
                  - arn:aws:s3:::amazon-braket-*
                  - arn:aws:s3:::braketnotebookcdk-*
              - Effect: "Allow"
                Action:
                  - "logs:CreateLogStream"
                  - "logs:PutLogEvents"
                  - "logs:CreateLogGroup"
                  - "logs:DescribeLogStreams"
                Resource:
                  - !Sub "arn:aws:logs:*:${AWS::AccountId}:log-group:/aws/sagemaker/*"
              - Effect: "Allow"
                Action:
                  - braket:*
                Resource: "*"
```

Langkah 3: Buat instance SageMaker notebook Amazon dengan awalan **`amazon-braket-`**

Gunakan skrip SageMaker siklus hidup dan peran IAM yang dibuat pada langkah 1 dan langkah 2 untuk membuat SageMaker instance notebook. Instans notebook disesuaikan untuk Braket dan dapat diakses dengan konsol Amazon Braket. Untuk informasi selengkapnya tentang opsi konfigurasi untuk CloudFormation sumber daya ini, lihat AWS::SageMaker::NotebookInstancedi panduan AWS CloudFormation pengguna.

```
BraketNotebook:
    Type: AWS::SageMaker::NotebookInstance
    Properties:
      InstanceType: ml.t3.medium
      NotebookInstanceName: !Sub amazon-braket-notebook-${AWS::StackName}
```

```
      RoleArn: !GetAtt ExecutionRole.Arn
      VolumeSizeInGB: 30
      LifecycleConfigName: !GetAtt
 BraketNotebookInstanceLifecycleConfig.NotebookInstanceLifecycleConfigName
```

Pencatatan lanjutan

Anda dapat merekam seluruh proses pengolahan tugas menggunakan logger. Teknik pencatatan lanjutan ini mengizinkan Anda melihat jajak pendapat latar belakang dan membuat catatan untuk debugging nanti.

Untuk menggunakan logger, kami sarankan untuk mengubah `poll_interval_seconds` parameter `poll_timeout_seconds` dan, sehingga tugas kuantum dapat berjalan lama dan status tugas kuantum dicatat terus menerus, dengan hasil disimpan ke file. Anda dapat mentransfer kode ini ke penulisan Python bukan notebook Jupyter, sehingga penulisan dapat berjalan sebagai proses di latar belakang.

Konfigurasikan logger

Pertama, konfigurasikan logger sehingga semua catatan ditulis ke dalam file teks secara otomatis, seperti yang ditunjukkan pada baris contoh berikut.

```
# 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
```

Buat dan jalankan sirkuit

Sekarang Anda dapat membuat sirkuit, mengirimkannya ke perangkat untuk dijalankan, dan melihat apa yang terjadi seperti yang ditunjukkan dalam contoh ini.

```
# 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
)
```

Periksa file log

Anda dapat memeriksa apa yang tertulis ke dalam file dengan memasukkan perintah berikut.

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

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

Dapatkan ARN dari file log

Dari output file berkas log yang dikembalikan, seperti yang ditunjukkan dalam contoh sebelumnya, Anda dapat memperoleh informasi ARN. Dengan ARN ID, Anda dapat mengambil hasil dari tugas kuantum yang diselesaikan.

```
# 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()
```

Keamanan di Amazon Braket

Bab ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Amazon Braket. Ini menunjukkan kepada Anda cara mengonfigurasi Amazon Braket untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan Layanan AWS yang lain yang membantu Anda memantau dan mengamankan sumber daya Amazon Braket Anda.

Keamanan cloud di AWS merupakan prioritas tertinggi. Sebagai seorang pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan dari organisasi yang paling sensitif terhadap keamanan. Anda bertanggung jawab atas faktor lain termasuk sensitivitas data Anda, persyaratan perusahaan Anda, serta hukum dan peraturan yang berlaku.

Tanggung jawab bersama untuk keamanan

Keamanan adalah tanggung jawab bersama antara AWS dan Anda. Model tanggung jawab bersama menggambarkan ini sebagai keamanan dari cloud dan keamanan di dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang berjalan Layanan AWS di dalamnyaAWS Cloud. AWSjuga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi keefektifan keamanan kami sebagai bagian dari program kepatuhan AWS. Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon Braket, lihat AWSLayanan dalam Lingkup berdasarkan Program Kepatuhan.
- Keamanan di cloud — Anda bertanggung jawab untuk menjaga kontrol atas konten Anda yang di-host di AWS infrastruktur ini. Konten ini meliputi konfigurasi keamanan dan tugas-tugas pengelolaan untuk berbagai layanan Layanan AWS yang Anda gunakan.

Perlindungan data

Model tanggung jawab AWS bersama model berlaku untuk perlindungan data di Amazon Braket. Sebagaimana diuraikan dalam model ini, AWS bertanggung jawab untuk memberikan perlindungan terhadap infrastruktur global yang menjalankan semua AWS Cloud. Anda harus bertanggung jawab untuk memelihara kendali terhadap konten yang di-hosting pada infrastruktur ini. Anda juga

bertanggung jawab atas tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Untuk informasi selengkapnya tentang privasi data, lihat FAQ Privasi Data. Untuk informasi tentang perlindungan data di Eropa, silakan lihat postingan blog Model Tanggung Jawab Bersama AWS dan GDPR di Blog Keamanan AWS.

Untuk tujuan perlindungan data, sebaiknya Anda melindungi kredensial Akun AWS dan menyiapkan AWS IAM Identity Center atau AWS Identity and Access Management (IAM) untuk pengguna individu. Dengan cara seperti itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugas mereka. Kami juga merekomendasikan agar Anda mengamankan data Anda dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk melakukan komunikasi dengan sumber daya AWS. Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Siapkan API dan log aktivitas pengguna dengan AWS CloudTrail.
- Gunakan solusi enkripsi AWS, bersama dengan semua kontrol keamanan default dalam Layanan AWS.
- Gunakan layanan keamanan terkelola lanjutan seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 ketika mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Untuk informasi selengkapnya tentang titik akhir FIPS yang tersedia, silakan lihat Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Sebaiknya Anda tidak memasukkan informasi rahasia atau sensitif, seperti alamat email pelanggan, ke dalam tanda atau bidang teks bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan Amazon Braket atau lainnya Layanan AWS menggunakan konsol, APIAWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam tanda atau bidang teks bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau diagnostik. Saat Anda memberikan URL ke server eksternal, sebaiknya Anda tidak menyertakan informasi kredensial di URL untuk memvalidasi permintaan Anda ke server tersebut.

Retensi data

Setelah 90 hari, Amazon Braket secara otomatis menghapus semua ID tugas kuantum dan metadata lain yang terkait dengan tugas kuantum Anda. Akibat kebijakan penyimpanan data ini, tugas dan

hasil ini tidak lagi dapat diambil dengan mencari dari konsol Amazon Braket, meskipun mereka tetap disimpan dalam bucket S3 Anda.

Jika Anda memerlukan akses ke tugas kuantum historis dan hasil yang disimpan di bucket S3 selama lebih dari 90 hari, Anda harus menyimpan catatan terpisah dari ID tugas Anda dan metadata lain yang terkait dengan data tersebut. Pastikan untuk menyimpan informasi sebelum 90 hari. Anda dapat menggunakan informasi yang disimpan untuk mengambil data historis.

Mengelola akses ke Amazon Braket

Bab ini menjelaskan izin yang diperlukan untuk menjalankan Amazon Braket, atau untuk membatasi akses pengguna dan peran tertentu. Anda dapat memberikan (atau menolak) izin yang diperlukan untuk setiap pengguna atau peran di akun Anda. Untuk melakukannya, lampirkan kebijakan Amazon Braket yang sesuai ke pengguna atau peran tersebut di akun Anda seperti yang dijelaskan di bagian berikut.

Sebagai prasyarat, Anda harus Mengaktifkan Amazon Braket. Untuk mengaktifkan Braket, pastikan untuk masuk sebagai pengguna atau peran yang memiliki (1) izin administrator atau (2) ditetapkan AmazonBraketFullAccesskebijakan dan memiliki izin untuk membuat bucket Amazon Simple Storage Service (Amazon S3).

Di bagian ini:

- Sumber daya Amazon Braket
- Notebook dan peran
- Tentang AmazonBraketFullAccess kebijakan
- Tentang AmazonBraketJobsExecutionPolicy kebijakan
- Batasi akses pengguna ke perangkat tertentu
- Amazon Braket memperbarui kebijakan terkelola AWS
- Batasi akses pengguna ke instance notebook tertentu
- Batasi akses pengguna ke bucket S3 tertentu

Sumber daya Amazon Braket

Braket menciptakan satu jenis sumber daya: sumber daya tugas kuantum. Nama Sumber Daya Amazon (ARN) untuk jenis sumber daya ini adalah sebagai berikut:

- Nama Sumber Daya:: :Service AWS: :Braket
- ARN Regex: arn: $ {Partition} :braket: $ {Wilayah} :$ {Akun} :quantum-task/$ {} RandomId

Notebook dan peran

Anda dapat menggunakan jenis sumber daya noteboook di Braket. Notebook adalah SageMaker sumber daya Amazon yang dapat dibagikan oleh Braket. Untuk menggunakan notebook dengan Braket, Anda harus menentukan peran IAM dengan nama yang dimulai dengan. `AmazonBraketServiceSageMakerNotebook`

Untuk membuat buku catatan, Anda harus menggunakan peran dengan izin admin atau yang memiliki kebijakan sebaris berikut yang dilampirkan padanya.

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

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

Untuk membuat peran, ikuti langkah-langkah yang diberikan di halaman Buat buku catatan atau minta administrator membuatnya untuk Anda. Pastikan bahwa AmazonBraketFullAccesskebijakan tersebut terlampir.

Setelah membuat peran, Anda dapat menggunakan kembali peran tersebut untuk semua notebook yang diluncurkan di masa mendatang.

Tentang AmazonBraketFullAccess kebijakan

AmazonBraketFullAccessKebijakan ini memberikan izin untuk operasi Amazon Braket, termasuk izin untuk tugas ini:

- Unduh kontainer dari Amazon Elastic Container Registry — Untuk membaca dan mengunduh gambar kontainer yang digunakan untuk fitur Amazon Braket Hybrid Jobs. Wadah harus sesuai dengan format “arn:aws:ecr: ::repository/amazon-braket”.
- Simpan AWS CloudTrail log — Untuk semua tindakan deskripsikan, dapatkan, dan daftar selain memulai dan menghentikan kueri, menguji filter metrik, dan memfilter peristiwa log. File AWS CloudTrail log berisi catatan semua API aktivitas Amazon Braket yang terjadi di akun Anda.
- Memanfaatkan peran untuk mengontrol sumber daya — Untuk membuat peran terkait layanan di akun Anda. Peran terkait layanan memiliki akses ke AWS sumber daya atas nama Anda. Ini hanya bisa digunakan oleh layanan Amazon Braket. Juga, untuk meneruskan peran IAM ke Amazon `CreateJob` API Braket dan untuk membuat peran dan melampirkan kebijakan yang dicakup AmazonBraketFullAccess ke peran tersebut.
- Buat grup log, peristiwa log, dan grup log kueri untuk mempertahankan file log penggunaan untuk akun Anda — Untuk membuat, menyimpan, dan melihat informasi pencatatan tentang penggunaan Amazon Braket di akun Anda. Metrik kueri pada grup log pekerjaan hibrida. Mencakup jalur Braket yang tepat dan memungkinkan menempatkan data log. Masukkan data metrik CloudWatch.

- Buat dan simpan data di bucket Amazon S3, dan daftarkan semua bucket — Untuk membuat bucket S3, daftarkan bucket S3 di akun Anda, lalu masukkan objek ke dalam dan dapatkan objek dari bucket apa pun di akun Anda yang namanya dimulai dengan amazon-braket-. Izin ini diperlukan untuk Braket untuk menempatkan file yang berisi hasil dari tugas kuantum yang diproses ke dalam bucket dan mengambilnya dari bucket.
- Lulus peran IAM — Untuk meneruskan peran IAM ke. `CreateJob` API
- Amazon SageMaker Notebook - Untuk membuat dan mengelola instance SageMaker notebook yang dicakup ke sumber daya dari “arn:aws:sagemaker: ::notebook-instance/amazon-braket-”.
- Validasi kuota layanan — Untuk membuat SageMaker notebook dan pekerjaan Amazon Braket Hybrid, jumlah sumber daya Anda tidak dapat melebihi kuota untuk akun Anda.

Isi kebijakan

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

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

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

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

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

Tentang AmazonBraketJobsExecutionPolicy kebijakan

AmazonBraketJobsExecutionPolicyKebijakan ini memberikan izin untuk peran eksekusi yang digunakan dalam Pekerjaan Hibrid Amazon Braket sebagai berikut:

- Unduh wadah dari Amazon Elastic Container Registry - Izin untuk membaca dan mengunduh gambar kontainer yang digunakan untuk fitur Pekerjaan Hibrida Amazon Braket. Wadah harus sesuai dengan format “arn:aws:ecr: *:*:repository/amazon-braket*”.
- Buat grup log dan peristiwa log dan grup log kueri untuk mempertahankan file log penggunaan untuk akun Anda — Buat, simpan, dan lihat informasi pencatatan tentang penggunaan Amazon Braket di akun Anda. Metrik kueri pada grup log pekerjaan hibrida. Mencakup jalur Braket yang tepat dan memungkinkan menempatkan data log. Masukkan data metrik CloudWatch.
- Simpan data di bucket Amazon S3 — Cantumkan bucket S3 di akun Anda, masukkan objek ke dalam dan dapatkan objek dari bucket apa pun di akun Anda yang dimulai dengan amazon-braket - dalam namanya. Izin ini diperlukan untuk Braket untuk menempatkan file yang berisi hasil dari tugas kuantum yang diproses ke dalam bucket, dan untuk mengambilnya dari bucket.
- Lulus peran IAM — Melewati peran IAM ke. CreateJob API Peran harus sesuai dengan format arn:aws:iam: :* *. :role/service-role/AmazonBraketJobsExecutionRole

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

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

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

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

Batasi akses pengguna ke perangkat tertentu

Untuk membatasi akses pengguna tertentu ke perangkat Braket tertentu, Anda dapat menambahkan kebijakan penolakan izin ke peran tertentu. IAM

Tindakan berikut dapat dibatasi dengan izin tersebut:

- `CreateQuantumTask`- untuk menolak pembuatan tugas kuantum pada perangkat tertentu.
- `CreateJob`- untuk menolak penciptaan lapangan kerja hybrid pada perangkat tertentu.
- `GetDevice`- untuk menolak mendapatkan detail perangkat yang ditentukan.

Contoh berikut membatasi akses ke semua QPU untuk. Akun AWS `123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
```

```
  ]
}
```

Untuk mengadaptasi kode ini, gantikan Amazon Resource Number (ARN) dari perangkat terbatas untuk string yang ditunjukkan pada contoh sebelumnya. String ini memberikan nilai Resource. Di Braket, perangkat mewakili QPU atau simulator yang dapat Anda panggil untuk menjalankan tugas kuantum. Perangkat yang tersedia tercantum di halaman Perangkat. Ada dua skema yang digunakan untuk menentukan akses ke perangkat ini:

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

Berikut adalah contoh untuk berbagai jenis akses perangkat

- Untuk memilih semua QPU di semua wilayah: `arn:aws:braket:*:*:device/qpu/*`
- Untuk memilih semua QPU di wilayah us-barat-2 SAJA: `arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- Secara setara, untuk memilih semua QPU di wilayah us-barat-2 SAJA (karena perangkat adalah sumber daya layanan, bukan sumber daya pelanggan): `arn:aws:braket:us-west-2:* :device/qpu/*`
- Untuk membatasi akses ke semua perangkat simulator sesuai permintaan: `arn:aws:braket:* :123456789012:device/quantum-simulator/*`
- Untuk membatasi akses ke IonQ Harmony perangkat di wilayah us-east-1: `arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- Untuk membatasi akses ke perangkat dari penyedia tertentu (misalnya, ke Rigetti QPU perangkat): `arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- Untuk membatasi akses ke TN1 perangkat: `arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

Amazon Braket memperbarui kebijakan terkelola AWS

Tabel berikut memberikan rincian tentang pembaruan kebijakan AWS terkelola untuk Braket sejak layanan ini mulai melacak perubahan ini.

Ubah	Deskripsi	Tanggal
AmazonBraketFullAccess- Kebijakan akses penuh untuk Braket	Menambahkan GetMetric Data tindakan servicequotas: GetServiceQuota dan cloudwatch: untuk disertakan dalam kebijakan. AmazonBraketFullAccess .	24 Maret 2023
AmazonBraketFullAccess- Kebijakan akses penuh untuk Braket	Braket disesuaikan iam: PassRole izin AmazonBraketFullAccess untuk menyertakan jalur. `service-role/`	29 November 2021
AmazonBraketJobsExecutionPolicy- Kebijakan eksekusi pekerjaan Hybrid untuk Amazon Braket Hybrid Jobs	Braket memperbarui peran eksekusi pekerjaan hybrid ARN untuk memasukkan `service-role/` jalur.	29 November 2021
Braket mulai melacak perubahan	Braket mulai melacak perubahan untuk kebijakan AWS terkelola nya.	29 November 2021

Batasi akses pengguna ke instance notebook tertentu

Untuk membatasi akses bagi pengguna tertentu ke instance notebook Braket tertentu, Anda dapat menambahkan kebijakan penolakan izin ke peran, pengguna, atau grup tertentu.

Contoh berikut menggunakan variabel kebijakan untuk secara efisien membatasi izin untuk memulai, menghentikan, dan mengakses instance notebook tertentu di Akun AWS `123456789012`, yang dinamai sesuai dengan pengguna yang seharusnya memiliki akses (misalnya, pengguna `Alice` akan memiliki akses ke instance notebook bernama). `amazon-braket-Alice`

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

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

Batasi akses pengguna ke bucket S3 tertentu

Untuk membatasi akses pengguna tertentu ke bucket Amazon S3 tertentu, Anda dapat menambahkan kebijakan penolakan ke peran, pengguna, atau grup tertentu.

Contoh berikut membatasi izin untuk mengambil dan menempatkan objek ke dalam S3 bucket (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) tertentu dan juga membatasi daftar objek tersebut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "s3:ListBucket"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice"
      ]
    },
    {
      "Effect": "Deny",
      "Action": [
        "s3:GetObject"
      ],
      "NotResource": [
        "arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice/*"
      ]
    }
  ]
}
```

Untuk membatasi akses ke bucket untuk instance notebook tertentu, Anda dapat menambahkan kebijakan sebelumnya ke peran eksekusi notebook.

Peran tertaut layanan Amazon Braket

Saat Anda mengaktifkan Amazon Braket, peran tertaut layanan dibuat di akun Anda.

Peran tertaut layanan adalah tipe IAM role unik yang, dalam hal ini, ditautkan langsung ke Amazon Braket. Peran terkait layanan Amazon Braket telah ditentukan sebelumnya untuk menyertakan semua izin yang diperlukan Braket saat menelepon lainnya atas nama Anda. Layanan AWS

Peran tertaut layanan mempermudah pengaturan Amazon Braket karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Amazon Braket mendefinisikan izin peran

tertaut layanan. Kecuali Anda mengubah definisi ini, hanya Amazon Braket dapat mengasumsikan perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin. Kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Peran terkait layanan yang disiapkan Amazon Braket adalah bagian dari kemampuan peran terkait layanan AWS Identity and Access Management (IAM). Untuk informasi tentang peran lain Layanan AWS yang mendukung peran terkait layanan, lihat AWSLayanan yang Bekerja dengan IAM dan cari layanan yang memiliki Ya di kolom Peran Tertaut Layanan. Pilih Yes (Ya) bersama tautan untuk melihat dokumentasi peran tertaut layanan untuk layanan tersebut.

Izin peran tertaut layanan untuk Amazon Braket

Amazon Braket menggunakan peran `AWSServiceRoleForAmazonBraket` terkait layanan yang mempercayai entitas braket.amazonaws.com untuk mengambil peran tersebut.

Anda harus mengonfigurasi izin untuk mengizinkan entitas IAM (seperti grup atau peran) untuk membuat, mengedit, atau menghapus peran terkait layanan. Untuk selengkapnya, lihat Izin Peran Tertaut Layanan.

Peran terkait layanan di Amazon Braket diberikan izin berikut secara default:

- Amazon S3 — izin untuk mencantumkan bucket di akun Anda, dan memasukkan objek ke dalam dan mendapatkan objek dari bucket apa pun di akun Anda dengan nama yang dimulai dengan amazon-braket-.
- Amazon CloudWatch Logs — izin untuk mendaftar dan membuat grup log, membuat aliran log terkait, dan memasukkan peristiwa ke dalam grup log yang dibuat untuk Amazon Braket.

Kebijakan berikut dilampirkan pada peran **`AWSServiceRoleForAmazonBraket`**terkait layanan:

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

Ketahanan dalam Amazon Braket

Infrastruktur AWS global dibangun di sekitar Wilayah AWS dan Availability Zone.

Setiap Wilayah menyediakan beberapa Availability Zone yang secara fisik terpisah dan terisolasi. Availability Zone (AZ) ini terhubung melalui jaringan latensi rendah, throughput tinggi, dan sangat berlebih. Akibatnya, Availability Zone memiliki ketersediaan yang tinggi, toleran terhadap kesalahan, dan dapat diskalakan jika dibandingkan dengan infrastruktur pusat data tunggal atau ganda tradisional.

Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis mengalami failover antar AZ tanpa gangguan.

Untuk informasi selengkapnya tentang Wilayah AWS dan zona ketersediaan, lihat Infrastruktur AWS Global.

Validasi Kepatuhan untuk Amazon Braket

Auditor pihak ketiga secara teratur menilai keamanan dan kepatuhan Amazon Braket dan integrasi kami dengan penyedia perangkat keras pihak ketiga. Untuk up-to-date daftar informasi kepatuhan untuk Braket, lihat Layanan AWSdalam lingkup oleh program kepatuhan. Untuk informasi umum, lihat AWSkepatuhan.

Anda bisa mengunduh laporan audit pihak ke tiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh laporan di AWS Artifact.

> **Note**
>
> AWSLaporan kepatuhan tidak mencakup QPU dari penyedia perangkat keras pihak ketiga yang dapat memilih untuk menjalani audit independen mereka sendiri.

Tanggung jawab kepatuhan Anda saat menggunakan Amazon Braket ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta hukum dan peraturan yang berlaku. AWSmenyediakan sumber daya berikut untuk membantu dengan kepatuhan:

- Panduan Quick Start Keamanan dan Kepatuhan – Panduan deployment ini membahas pertimbangan arsitektur dan memberikan langkah untuk menerapkan lingkungan dasar yang berfokus pada keamanan dan kepatuhan di AWS.
- Sumber Daya Kepatuhan AWS – Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.

Keamanan Infrastruktur di Amazon Braket

Sebagai layanan terkelola, Amazon Braket dilindungi oleh prosedur keamanan jaringan AWS global yang dijelaskan dalam whitepaper AWS: Gambaran Umum Proses Keamanan.

Untuk akses ke Amazon Braket melalui jaringan, Anda melakukan panggilan ke AWS API yang dipublikasikan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau yang lebih baru. Klien juga harus mendukung suite cipher dengan Perfect Forward Secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Sebagian besar sistem-sistem modern seperti Java 7 dan versi yang lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang dikaitkan dengan prinsipal utama utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

Keamanan Penyedia Perangkat Keras Amazon Braket

QPU di Amazon Braket dihosting oleh penyedia perangkat keras pihak ketiga. Saat Anda menjalankan tugas kuantum di QPU, Amazon Braket menggunakan DevicEarn sebagai pengenal saat mengirim sirkuit ke QPU yang ditentukan untuk diproses.

Jika Anda menggunakan Amazon Braket untuk akses ke perangkat keras komputasi kuantum yang dioperasikan oleh salah satu penyedia perangkat keras pihak ketiga, sirkuit Anda dan data yang terkait diproses oleh penyedia perangkat keras di luar fasilitas yang dioperasikan olehAWS. Informasi tentang lokasi fisik danAWSWilayah di mana setiap QPU tersedia dapat ditemukan diDetail Perangkatbagian dari konsol Amazon Braket.

Konten Anda dianonimkan. Hanya konten yang diperlukan untuk memproses sirkuit dikirim ke pihak ketiga.Akun AWSinformasi tidak dikirimkan ke pihak ketiga.

Semua data dienkripsi saat istirahat dan dalam transit. Data didekripsi untuk pemrosesan saja. Penyedia pihak ketiga Amazon Braket tidak diizinkan untuk menyimpan atau menggunakan konten Anda untuk tujuan selain memproses sirkuit Anda. Setelah rangkaian selesai, hasilnya dikembalikan ke Amazon Braket dan disimpan dalam bucket S3 Anda.

Keamanan penyedia perangkat keras kuantum pihak ketiga Amazon Braket diaudit secara berkala, untuk memastikan bahwa standar keamanan jaringan, kontrol akses, perlindungan data, dan keamanan fisik terpenuhi.

Amazon VPC endpoint untuk Amazon Braket

Anda dapat membuat koneksi pribadi antara VPC dan Amazon Braket Anda dengan membuat antarmuka VPC endpoint. Endpoint antarmuka didukung oleh AWS PrivateLink, teknologi yang memungkinkan akses ke Braket API tanpa gateway internet, perangkat NAT, koneksi VPN, atau koneksi. AWS Direct Connect Instans dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan API.

Setiap titik akhir antarmuka diwakili oleh satu atau beberapa Antarmuka Jaringan Elastis di subnet Anda.

Dengan PrivateLink, lalu lintas antara VPC dan Braket Anda tidak meninggalkan Amazon jaringan, yang meningkatkan keamanan data yang Anda bagikan dengan aplikasi berbasis cloud, karena mengurangi paparan data Anda ke internet publik. Untuk informasi selengkapnya, lihat Titik akhir VPC Antarmuka (AWS PrivateLink) di Panduan Pengguna Amazon VPC.

Pertimbangan untuk VPC endpoint Amazon Braket

Sebelum Anda menyiapkan VPC endpoint antarmuka untuk Braket, pastikan bahwa Anda meninjau Properti dan batasan titik akhir antarmuka di Panduan Pengguna Amazon VPC.

Braket mendukung panggilan ke semua tindakan API miliknya dari VPC Anda.

Secara default, akses penuh ke Braket diizinkan melalui VPC endpoint. Anda dapat mengontrol akses jika Anda menentukan kebijakan VPC endpoint. Untuk informasi lebih lanjut, lihat Mengendalikan akses ke layanan dengan VPC endpoint di Panduan Pengguna Amazon VPC.

Mengatur Braket dan PrivateLink

Untuk menggunakan AWS PrivateLink Amazon Braket, Anda harus membuat titik akhir Amazon Virtual Private Cloud (Amazon VPC) sebagai antarmuka, lalu sambungkan ke titik akhir melalui layanan Braket. Amazon API

Berikut adalah langkah-langkah umum proses ini, yang dijelaskan secara rinci di bagian selanjutnya.

- Konfigurasikan dan luncurkan VPC Amazon untuk meng-host sumber daya AndaAWS. Jika Anda sudah memiliki VPC, Anda dapat melewati langkah ini.
- Buat Amazon VPC endpoint untuk Braket
- Connect dan jalankan tugas kuantum Braket melalui endpoint Anda

Langkah 1: Luncurkan Amazon VPC jika diperlukan

Ingat bahwa Anda dapat melewati langkah ini jika akun Anda sudah memiliki VPC yang beroperasi.

VPC mengendalikan pengaturan jaringan Anda, seperti jangkauan alamat IP, subnet, tabel rute, dan gateway jaringan. Pada dasarnya, Anda meluncurkan AWS sumber daya Anda di jaringan virtual khusus. Untuk informasi lebih lanjut tentang Amazon VPC, lihat Panduan Pengguna Amazon VPC.

Buka Konsol Amazon VPC dan buat VPC baru dengan subnet, grup keamanan, dan gateway jaringan.

Langkah 2: Buat VPC endpoint antarmuka untuk Braket

Anda dapat membuat titik akhir VPC untuk layanan Braket menggunakan konsol VPC Amazon atau (). AWS Command Line Interface AWS CLI Untuk informasi lebih lanjut, lihat Membuat titik akhir antarmuka di Panduan Pengguna Amazon VPC.

Untuk membuat VPC endpoint di konsol, buka Konsol Amazon VPC, buka laman Titik akhir, dan lanjutkan untuk membuat titik akhir baru. Buat catatan dari ID titik akhir untuk referensi di kemudian hari. Ini diperlukan sebagai bagian dari `–endpoint-url` bendera saat Anda melakukan panggilan tertentu ke BraketAPI.

Buat VPC endpoint untuk Braket menggunakan nama layanan berikut:

- `com.amazonaws.substitute_your_region.braket`

Catatan: Jika Anda mengaktifkan DNS pribadi untuk titik akhir, Anda dapat membuat API permintaan ke Braket menggunakan nama DNS default untuk Wilayah, misalnya,. `braket.us-east-1.amazonaws.com`

Untuk informasi lebih lanjut, lihat Mengakses layanan melalui titik akhir antarmuka di Panduan Pengguna Amazon VPC.

Langkah 3: Connect dan jalankan tugas kuantum Braket melalui endpoint Anda

Setelah Anda membuat titik akhir VPC, Anda dapat menjalankan perintah CLI yang menyertakan `endpoint-url` parameter untuk menentukan titik akhir antarmuka ke API atau runtime, seperti contoh berikut:

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

Jika Anda mengaktifkan nama host DNS pribadi untuk VPC endpoint Anda, Anda tidak perlu menentukan titik akhir sebagai URL dalam perintah CLI Anda. Sebagai gantinya, nama host API DNS Amazon Braket, yang digunakan CLI dan Braket SDK secara default, diselesaikan ke titik akhir VPC Anda. Ini memiliki bentuk yang ditunjukkan dalam contoh berikut:

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

Posting blog yang disebut Akses langsung ke SageMaker notebook Amazon dari Amazon VPC dengan menggunakan AWS PrivateLink titik akhir memberikan contoh cara mengatur titik akhir untuk membuat koneksi aman ke notebook, yang SageMaker mirip dengan notebook Braket. Amazon

Jika Anda mengikuti langkah-langkah dalam posting blog, ingatlah untuk mengganti nama AmazonBraket untuk Amazon SageMaker. Untuk Nama Layanan, masukkan `com.amazonaws.us-east-1.braket` atau ganti Wilayah AWS nama Anda yang benar ke dalam string itu, jika Wilayah Anda bukan us-east-1.

Lebih lanjut tentang membuat titik akhir

- Untuk informasi tentang cara membuat VPC dengan subnet pribadi, lihat Buat VPC dengan subnet pribadi
- Untuk informasi tentang membuat dan mengonfigurasi titik akhir menggunakan konsol Amazon VPC atau AWS CLI, lihat Membuat Titik Akhir Antarmuka dalam Panduan Pengguna Amazon VPC.
- Untuk informasi tentang membuat dan mengonfigurasi titik akhir menggunakanAWS CloudFormation, lihat sumber daya AWS: :EC2: :VPCendPoint di Panduan Pengguna. AWS CloudFormation

Mengontrol akses dengan kebijakan Amazon VPC endpoint

Untuk mengontrol akses konektivitas ke Amazon Braket, Anda dapat melampirkan kebijakan endpoint AWS Identity and Access Management (IAM) ke endpoint Amazon VPC Anda. Kebijakan menentukan informasi berikut ini:

- (Pengguna atau peran) utama yang dapat melakukan tindakan.
- Tindakan yang dapat dilakukan.
- Sumber daya yang dapat digunakan untuk mengambil tindakan.

Untuk informasi lebih lanjut, lihat Mengendalikan akses ke layanan dengan VPC endpoint di Panduan Pengguna Amazon VPC.

Contoh: Kebijakan titik akhir VPC untuk tindakan Braket

Berikut adalah contoh kebijakan VPC endpoint untuk Braket. Jika dilampirkan ke titik akhir, kebijakan ini memberikan akses ke tindakan Braket yang terdaftar untuk semua yang utama di semua sumber daya.

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

Anda dapat membuat aturan IAM kompleks dengan melampirkan beberapa kebijakan titik akhir. Untuk informasi lebih lanjut dan contoh, lihat:

- Kebijakan Titik Akhir Amazon Virtual Private Cloud untuk Step Functions
- Membuat Izin IAM Granular untuk Pengguna Non-Admin
- Mengontrol Akses ke Layanan dengan Titik Akhir VPC

Memecahkan Masalah Amazon Braket

Gunakan informasi dan solusi pemecahan masalah di bagian ini untuk membantu menyelesaikan masalah dengan Amazon Braket.

Di bagian ini:

AccessDeniedException

Jika Anda menerima AccessDeniedExceptionsaat mengaktifkan atau menggunakan Braket, Anda mungkin mencoba mengaktifkan atau menggunakan Braket di wilayah di mana peran terbatas Anda tidak memiliki akses.

Dalam kasus seperti itu, Anda harus menghubungi AWS administrator internal Anda untuk memahami kondisi berikut yang berlaku:

- Jika ada pembatasan peran yang mencegah akses ke suatu wilayah.
- Jika peran yang Anda coba gunakan diizinkan untuk menggunakan Braket.

Jika peran Anda tidak memiliki akses ke wilayah tertentu saat menggunakan Braket, maka Anda tidak akan dapat menggunakan perangkat di wilayah tertentu.

Terjadi kesalahan (ValidationException) saat memanggil CreateQuantumTask operasi

Jika Anda menerima kesalahan yang mirip dengan: `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller`

`doesn't have access to amazon-braket-…` Periksa apakah Anda merujuk ke folder s3_yang ada. Braket tidak secara otomatis membuat bucket dan awalan Amazon S3 baru untuk Anda.

Jika Anda mengakses secara API langsung dan menerima kesalahan yang mirip dengan: `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` Periksa apakah Anda tidak termasuk `s3://` dalam jalur bucket Amazon S3.

Fitur SDK tidak bekerja

Versi Python Anda harus 3.9 atau lebih tinggi. Untuk Pekerjaan Hibrida Amazon Braket, kami merekomendasikan Python 3.10.

Verifikasi SDK dan skema Anda. up-to-date Untuk memperbarui SDK dari notebook atau editor python Anda, jalankan perintah berikut:

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Untuk memperbarui skema, jalankan perintah berikut:

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

Jika Anda mengakses Amazon Braket dari klien Anda sendiri, verifikasi Wilayah AWS Anda disetel ke wilayah yang didukung oleh Amazon Braket.

Pekerjaan hybrid gagal karena ServiceQuotaExceededException

Pekerjaan hibrida yang menjalankan tugas kuantum terhadap simulator Amazon Braket dapat gagal dibuat jika Anda melebihi batas tugas kuantum bersamaan untuk perangkat simulator yang Anda targetkan. Untuk informasi selengkapnya tentang batas layanan, lihat topik Kuota.

Jika Anda menjalankan tugas bersamaan terhadap perangkat simulator dalam beberapa pekerjaan hybrid dari akun Anda, Anda dapat mengalami kesalahan ini.

Untuk melihat jumlah tugas kuantum bersamaan terhadap perangkat simulator tertentu, gunakan `search-quantum-tasks`API, seperti yang ditunjukkan pada contoh kode berikut.

```
DEVICE_ARN=arn:aws:braket:::device/quantum-simulator/amazon/sv1
```

```
task_list=""
for status_value in "CREATED" "QUEUED" "RUNNING" "CANCELLING"; do
    tasks=$(aws braket search-quantum-tasks --filters
 name=status,operator=EQUAL,values=${status_value}
 name=deviceArn,operator=EQUAL,values=$DEVICE_ARN --max-results 100 --query
 'quantumTasks[*].quantumTaskArn' --output text)
    task_list="$task_list $tasks"
done;
echo "$task_list" | tr -s ' \t' '[\n*]' | sort | uniq
```

Anda juga dapat melihat tugas kuantum yang dibuat terhadap perangkat menggunakan CloudWatch metrik Amazon: Braket > Berdasarkan Perangkat.

Untuk menghindari kesalahan ini:

1. Minta peningkatan kuota layanan untuk jumlah tugas kuantum bersamaan untuk perangkat simulator. Ini hanya berlaku untuk SV1 perangkat.
2. Tangani `ServiceQuotaExceeded` pengecualian dalam kode Anda dan coba lagi.

Komponen berhenti bekerja di instance notebook

Jika beberapa komponen notebook Anda berhenti bekerja, coba hal berikut:

1. Unduh buku catatan apa pun yang Anda buat atau modifikasi ke drive lokal.
2. Hentikan instans notebook Anda.
3. Hapus instans notebook Anda.
4. Buat instans notebook baru dengan nama yang berbeda.
5. Unggah notebook ke instans baru.

Kuota Amazon Braket

Tabel berikut memuat daftar service quotas untuk Amazon Braket. Kuota layanan, juga disebut sebagai batas, adalah jumlah maksimum sumber daya layanan atau operasi untuk Anda Akun AWS.

Beberapa kuota dapat ditingkatkan. Untuk informasi lebih lanjut, lihat Layanan AWS kuota.

- Kuota tarif ledakan tidak dapat ditingkatkan.

- Kenaikan tarif maksimum untuk kuota yang dapat disesuaikan (kecuali tarif ledakam, yang tidak dapat disesuaikan) adalah 2x batas tarif default yang ditentukan. Misalnya, kuota default 60 dapat disesuaikan dengan maksimum 120.
- Kuota yang dapat disesuaikan untuk tugas kuantum concurrent SV1 (DM1) memungkinkan maksimum 60 per. Wilayah AWS
- Jumlah maksimum instans komputasi yang diizinkan untuk pekerjaan hibrida adalah 5, dan kuota dapat disesuaikan.

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Permintaan tarif API	Jumlah maksimum permintaan per detik yang dapat Anda kirim di akun ini di Wilayah saat ini.	140	Ya
Permintaan tarif lonjakan API	Jumlah maksimum permintaan tambahan per detik (RPS) yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	600	Tidak
Permintaan tarif `CreateQuantumTask`	Jumlah maksimum permintaan `CreateQuantumTask` yang dapat Anda kirim per detik di akun ini per Wilayah.	20	Ya
Permintaan tarif lonjakan `CreateQuantumTask`	Jumlah maksimum tambahan permintaan per detik (RPS) `CreateQua`	40	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
	`ntumTask` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.		
Permintaan tarif `SearchQuantumTasks`	Jumlah maksimum permintaan `SearchQuantumTasks` yang dapat Anda kirim per detik di akun ini per Wilayah.	5	Ya
Permintaan tarif lonjakan `SearchQuantumTasks`	Jumlah maksimum tambahan permintaan per detik (RPS) `SearchQuantumTasks` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	50	Tidak
Permintaan tarif `GetQuantumTask`	Jumlah maksimum permintaan `GetQuantumTask` yang dapat Anda kirim per detik di akun ini per Wilayah.	100	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Permintaan tarif lonjakan `GetQuantumTask`	Jumlah maksimum tambahan permintaan per detik (RPS) `GetQuantumTask` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	500	Tidak
Permintaan tarif `CancelQuantumTask`	Jumlah maksimum permintaan `CancelQuantumTask` yang dapat Anda kirim per detik di akun ini per Wilayah.	2	Ya
Permintaan tarif lonjakan `CancelQuantumTask`	Jumlah maksimum tambahan permintaan per detik (RPS) `CancelQuantumTask` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	20	Tidak
Permintaan tarif `GetDevice`	Jumlah maksimum permintaan `GetDevice` yang dapat Anda kirim per detik di akun ini per Wilayah.	5	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Permintaan tarif lonjakan `GetDevice`	Jumlah maksimum tambahan permintaan per detik (RPS) `GetDevice` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	50	Tidak
Permintaan tarif `SearchDevices`	Jumlah maksimum permintaan `SearchDevices` yang dapat Anda kirim per detik di akun ini per Wilayah.	5	Ya
Permintaan tarif lonjakan `SearchDevices`	Jumlah maksimum tambahan permintaan per detik (RPS) `SearchDevices` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	50	Tidak
Permintaan tarif `CreateJob`	Jumlah maksimum permintaan `CreateJob` yang dapat Anda kirim per detik di akun ini per Wilayah.	1	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Permintaan tarif lonjakan `CreateJob`	Jumlah maksimum tambahan permintaan per detik (RPS) `CreateJob` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	5	Tidak
Permintaan tarif `SearchJob`	Jumlah maksimum permintaan `SearchJob` yang dapat Anda kirim per detik di akun ini per Wilayah.	5	Ya
Permintaan tarif lonjakan `SearchJob`	Jumlah maksimum tambahan permintaan per detik (RPS) `SearchJob` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	50	Tidak
Permintaan tarif `GetJob`	Jumlah maksimum permintaan `GetJob` yang dapat Anda kirim per detik di akun ini per Wilayah.	5	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Permintaan tarif lonjakan `GetJob`	Jumlah maksimum tambahan permintaan per detik (RPS) `GetJob` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	25	Tidak
Permintaan tarif `CancelJob`	Jumlah maksimum permintaan `CancelJob` yang dapat Anda kirim per detik di akun ini per Wilayah.	2	Ya
Permintaan tarif lonjakan `CancelJob`	Jumlah maksimum tambahan permintaan per detik (RPS) `CancelJob` yang dapat Anda kirim dalam satu ledakan di akun ini di Wilayah saat ini.	5	Tidak
Jumlah tugas SV1kuantum bersamaan	Jumlah maksimum tugas kuantum bersamaan yang berjalan pada simulator vektor status (SV1) di Wilayah saat ini.	100 kita-timur-1, 50 kita-barat-1, 100 kami-barat-2, 50 eu-west-2	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan
Jumlah tugas DM1kuantum bersamaan	Jumlah maksimum tugas kuantum bersamaan yang berjalan pada simulator matriks kepadatan (DM1) di Wilayah saat ini.	100 kita-timur-1, 50 kita-barat-1, 100 kami-barat-2, 50 eu-west-2	Tidak
Jumlah tugas TN1kuantum bersamaan	Jumlah maksimum tugas kuantum bersamaan yang berjalan pada simulator jaringan tensor (TN1) di Wilayah saat ini.	10 kita-timur-1, 10 kami-barat-2, 5 eu-barat-2,	Ya
Jumlah pekerjaan hibrida bersamaan	Jumlah maksimum pekerjaan hibrida bersamaan di Wilayah saat ini.	5	Ya
Batas runtime pekerjaan hybrid	Jumlah waktu maksimum dalam beberapa hari yang dapat dijalankan oleh pekerjaan hibrida.	5	Tidak

Berikut ini adalah kuota contoh komputasi klasik default untuk Hybrid Jobs. Untuk menaikkan kuota ini, silakan hubungi AWS Support. Selain itu, wilayah yang tersedia ditentukan untuk setiap instance.

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c4.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c4.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Tidak
Jumlah maksimum instance ml.c4.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c4.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun	5	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
	dan wilayah ini.							
Jumlah maksimum instance ml.c4.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c4.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c4.8xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c4.8xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	1	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5.9xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5.9xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	1	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5.18xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5.18xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5n.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5n.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5n.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5n.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5n.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5n.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5n.9xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5n.9xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.c5n.18xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.c5n.18xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.8xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.8xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.12xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.12xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.g4dn.16xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.g4dn.16xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m4.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m4.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m4.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m4.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m4.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m4.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	2	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m4.10x large untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m4.10x large diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m4.16x large untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m4.16x large diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.large untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.large diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.4xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.4xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	5	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.12x large untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.12x large diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.m5.24x large untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.m5.24x large diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Ya	Ya	Ya	Ya

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.p2.xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p2.xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instans ml.p2.8xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p2.8xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instans ml.p2.16xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p2.16xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.p3.2xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p3.2xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.p4d.24xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p4d.24xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instance ml.p3dn.24xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p3dn.24xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Tidak	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instans ml.p3.8xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p3.8xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Ya	Tidak

Sumber Daya	Deskripsi	Batas	Dapat Disesuaikan	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Jumlah maksimum instans ml.p3.16xlarge untuk pekerjaan hybrid	Jumlah maksimum instance tipe ml.p3.16xlarge diizinkan untuk semua Pekerjaan Hibrida Amazon Braket di akun dan wilayah ini.	0	Ya	Ya	Tidak	Ya	Ya	Tidak

Meminta pembaruan batas

Jika Anda menerima ServiceQuotaExceeded pengecualian untuk jenis instans dan tidak memiliki instans yang cukup tersedia untuknya, Anda dapat meminta peningkatan batas dari halaman Service Quotas di AWS konsol dan mencari Amazon Braket di bawah Layanan. AWS

> **Note**
>
> Jika pekerjaan hybrid Anda tidak dapat menyediakan kapasitas komputasi ML yang diminta, gunakan wilayah lain. Selain itu, jika Anda tidak melihat instance dalam tabel, itu tidak tersedia untuk Hybrid Jobs.

Kuota dan batas tambahan

- Tindakan tugas kuantum Amazon Braket dibatasi hingga 3MB.
- Jumlah maksimum bidikan per tugas yang diizinkan untukSV1,DM1, dan Rigetti perangkat adalah 100.000.
- Jumlah maksimum tembakan per tugas yang diizinkan TN1 adalah 1000.
- Untuk IonQ perangkat Aria-1 dan Aria-2, maksimumnya adalah 5.000 tembakan per tugas. Untuk perangkat IonQ Harmony dan Forte, dan OQC perangkat, maksimum adalah 10.000.
- UntukQuEra, bidikan maksimum yang diizinkan per tugas adalah 1000.
- Untuk TN1 dan QPU perangkat, bidikan per tugas harus> 0.

Memecahkan masalah OpenQASM

Bagian ini menyediakan pointer pemecahan masalah yang mungkin berguna saat menghadapi kesalahan menggunakan OpenQASM 3.0.

Di bagian ini:

- Sertakan kesalahan pernyataan
- Kesalahan tidak bersebelahan qubits
- Mencampur fisik qubits dengan qubits kesalahan virtual
- Meminta jenis hasil dan mengukur qubits dalam kesalahan program yang sama
- Batas klasik dan qubit register melebihi kesalahan
- Kotak tidak didahului oleh kesalahan pragma kata demi kata
- Kotak kata demi kata kehilangan kesalahan gerbang asli
- Kotak kata demi kata kehilangan kesalahan fisik qubits
- Pragma kata demi kata tidak memiliki kesalahan “braket”
- Kesalahan tunggal qubits tidak dapat diindeks
- Fisik qubits di dua qubit gerbang tidak terhubung kesalahan
- GetDevice tidak mengembalikan kesalahan hasil OpenQASM
- Peringatan dukungan simulator lokal

Sertakan kesalahan pernyataan

Braket saat ini tidak memiliki file perpustakaan gerbang standar untuk disertakan dalam program OpenQASM. Misalnya, contoh berikut memunculkan kesalahan parser.

```
OPENQASM 3;
include "standardlib.inc";
```

Kode ini menghasilkan pesan kesalahan: `No terminal matches '"' in the current parser context, at line 2 col 17.`

Kesalahan tidak bersebelahan qubits

Menggunakan perangkat yang tidak qubits bersebelahan yang `requiresContiguousQubitIndices` disetel ke `true` dalam kemampuan perangkat menghasilkan kesalahan.

Saat menjalankan tugas kuantum pada simulator danIonQ, program berikut memicu kesalahan.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

Kode ini menghasilkan pesan kesalahan: `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

Mencampur fisik qubits dengan qubits kesalahan virtual

Pencampuran fisik qubits dengan virtual qubits dalam program yang sama tidak diperbolehkan dan menghasilkan kesalahan. Kode berikut menghasilkan kesalahan.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

Kode ini menghasilkan pesan kesalahan: `[line 4] mixes physical qubits and qubits registers.`

Meminta jenis hasil dan mengukur qubits dalam kesalahan program yang sama

Meminta jenis hasil dan yang diukur qubits secara eksplisit dalam program yang sama menghasilkan kesalahan. Kode berikut menghasilkan kesalahan.

```
OPENQASM 3;

qubit[2] q;

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

#pragma braket result expectation x(q[0]) @ z(q[1])
```

Kode ini menghasilkan pesan kesalahan: `Qubits should not be explicitly measured when result types are requested.`

Batas klasik dan qubit register melebihi kesalahan

Hanya satu register klasik dan satu qubit register yang diizinkan. Kode berikut menghasilkan kesalahan.

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

Kode ini menghasilkan pesan kesalahan: `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

Kotak tidak didahului oleh kesalahan pragma kata demi kata

Semua kotak harus didahului dengan pragma kata demi kata. Kode berikut menghasilkan kesalahan.

```
box{
rx(0.5) $0;
```

```
}
```

Kode ini menghasilkan pesan kesalahan: `In verbatim boxes, native gates are required. x is not a device native gate.`

Kotak kata demi kata kehilangan kesalahan gerbang asli

Kotak kata demi kata harus memiliki gerbang asli dan fisik. qubits Kode berikut menghasilkan kesalahan gerbang asli.

```
#pragma braket verbatim
box{
x $0;
}
```

Kode ini menghasilkan pesan kesalahan: `In verbatim boxes, native gates are required. x is not a device native gate.`

Kotak kata demi kata kehilangan kesalahan fisik qubits

Kotak kata demi kata harus memiliki fisik. qubits Kode berikut menghasilkan qubits kesalahan fisik yang hilang.

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

Kode ini menghasilkan pesan kesalahan: `Physical qubits are required in verbatim box.`

Pragma kata demi kata tidak memiliki kesalahan “braket”

Anda harus memasukkan “braket” dalam pragma kata demi kata. Kode berikut menghasilkan kesalahan.

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

Kode ini menghasilkan pesan kesalahan: `You must include “braket” in the verbatim pragma`

Kesalahan tunggal qubits tidak dapat diindeks

Single qubits tidak dapat diindeks. Kode berikut menghasilkan kesalahan.

```
OPENQASM 3;

qubit q;
h q[0];
```

Kode ini menghasilkan kesalahan: `[line 4] single qubit cannot be indexed.`

Namun, qubit array tunggal dapat diindeks sebagai berikut:

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

Fisik qubits di dua qubit gerbang tidak terhubung kesalahan

Untuk menggunakan fisikqubits, pertama-tama konfirmasikan bahwa perangkat menggunakan fisik qubits dengan memeriksa `device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits` dan kemudian memverifikasi grafik konektivitas dengan memeriksa `device.properties.paradigm.connectivity.connectivityGraph` atau`device.properties.paradigm.connectivity.fullyConnected`.

```
OPENQASM 3;

cnot $0, $14;
```

Kode ini menghasilkan pesan kesalahan: `[line 3] has disconnected qubits 0 and 14`

GetDevice tidak mengembalikan kesalahan hasil OpenQASM

Jika Anda tidak melihat hasil OpenQASM dalam GetDevice respons saat menggunakan Braket SDK, Anda mungkin perlu menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi

agen-pengguna. Lihat contoh kode yang disediakan di bawah ini untuk mengetahui cara melakukannya untuk Go dan Java SDK.

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen-pengguna saat menggunakan: AWS CLI

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

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen-pengguna saat menggunakan Boto3:

```
import boto3
import botocore

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

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen-pengguna saat menggunakan/(SDK v2): JavaScript TypeScript

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

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen-pengguna saat menggunakan/(SDK v3): JavaScript TypeScript

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

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen pengguna saat menggunakan Go SDK:

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

Untuk menyetel variabel lingkungan AWS_EXECUTION_ENV untuk mengonfigurasi agen-pengguna saat menggunakan Java SDK:

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

Peringatan dukungan simulator lokal

Ini `LocalSimulator` mendukung fitur-fitur canggih di OpenQASM yang mungkin tidak tersedia di QPU atau simulator sesuai permintaan. Jika program Anda berisi fitur bahasa khusus hanya untuk`LocalSimulator`, seperti yang terlihat pada contoh berikut, Anda akan menerima peringatan.

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

Kode ini menghasilkan peringatan: `Program ini menggunakan fitur bahasa OpenQASM yang hanya didukung di. LocalSimulator Beberapa fitur ini mungkin tidak didukung pada QPU atau simulator sesuai permintaan.

Untuk informasi lebih lanjut tentang fitur OpenQASM yang didukung, klik di sini.

Panduan Referensi API & SDK untuk Amazon Braket

Amazon Braket menyediakan API, SDK, dan antarmuka baris perintah yang dapat Anda gunakan untuk membuat dan mengelola instans notebook serta melatih dan menerapkan model.

- Amazon Braket Python SDK (Direkomendasikan)
- Referensi API Amazon Braket
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto)
- AWS SDK for Ruby

Anda juga bisa mendapatkan contoh kode dari repositori Amazon Braket TutorialGitHub.

- Tutorial Braket GitHub

Riwayat dokumen

Tabel berikut menjelaskan dokumentasi untuk rilis dari Amazon Braket.

- APIversi: April 28, 2022
- Pembaruan API Referensi Terbaru: 25 September 2023
- Pembaruan dokumentasi terbaru: 22 Mei 2024

Ubah	Deskripsi	Tanggal
Perangkat IQM Garnet dan wilayah baru Europe North 1	Menambahkan dukungan untuk perangkat IQM Garnet. Perangkat 20-qubit dengan topologi kisi persegi. Diperluas Braket mendukung wilayah ke Eropa Utara 1 (Stockholm).	22 Mei 2024
Detuning lokal dirilis	Kemampuan eksperimental sekarang termasuk fitur detuning lokal Aquila QuEra QPU.	April 11, 2024
Manajer ketidakaktifan notebook dirilis	Saat membuat instance notebook, aktifkan manajer tidak aktif dan setel waktu durasi idle untuk secara otomatis mengatur ulang instance notebook Braket.	Maret 27, 2024
Daftar isi pengerjaan ulang	Menata ulang daftar isi Amazon Braket untuk mematuhi AWS persyaratan panduan gaya dan meningkatkan aliran konten untuk pengalaman pelanggan.	Desember 12, 2023

Braket Langsung dirilis	Ditambahkan dukungan untuk fitur langsung Braket, termasuk: • Reservasi • Saran ahli • Kemampuan eksperimental	27 November 2023
Diperbarui Buat instans notebook Amazon Braket	Memperbarui dokumentasi untuk menambahkan informasi guna membuat instance notebook untuk pelanggan Amazon Braket baru dan yang sudah ada.	27 November 2023
Diperbarui Bawa wadah Anda sendiri (BYOC)	Memperbarui dokumenta si untuk menambahkan informasi tentang kapan ke BYOC, resep ke BYOC, dan menjalankan Braket Hybrid Jobs pada wadah.	18 Oktober 2023
Dekorator pekerjaan hibrida dirilis	Jalankan kode lokal Anda sebagai pekerjaan hibridaHalaman ditambahkan. Berisi contoh: • Buat pekerjaan hybrid dari kode Python lokal • Instal paket Python tambahan dan kode sumber • Menyimpan dan memuat data ke dalam instance pekerjaan hibrida • Praktik terbaik untuk dekorator pekerjaan hibrida	16 Oktober 2023

Menambahkan visibilitas Antrian	Memperbarui dokumentasi Panduan Pengembang untuk menyertakan queue depth danqueue position. Memperbarui documnetation API untuk mencerminkan perubahan API baru untuk visibilitas antrian.	25 September 2023
Standarisasi penamaan dalam dokumentasi	Memperbarui dokumentasi untuk mengubah instance “pekerjaan” apa pun menjadi “pekerjaan hibrida” dan “tugas” menjadi “tugas kuantum”	11 September 2023
Perangkat baru IonQ Aria 2	Menambahkan dukungan untuk IonQ Aria 2 perangkat	September 8, 2023
Gerbang Asli yang Diperbarui	Memperbarui dokumentasi untuk menambahkan informasi tentang akses terprogram ke gerbang asli dari Rigetti.	16 Agustus 2023
Xanadukeberangkatan	Diperbarui dokumentasi untuk menghapus semua Xanadu perangkat	Juni 2, 2023
Perangkat baru IonQ Aria	Menambahkan dukungan untuk IonQ Aria perangkat	16 Mei 2023
Perangkat pensiunan Rigetti	Dukungan yang dihentikan untuk Rigetti Aspen-M-2	2 Mei 2023

Informasi AmazonBraketFullAccesskebijakan yang diperbarui	Memperbarui skrip yang mendefinisikan isi AmazonBraketFullAccesskebijakan untuk menyertakan GetMetricData tindakan servicequotas: GetServiceQuota dan cloudwatch: serta informasi tentang batasan sehubungan dengan kuota.	19 April 2023
Peluncuran Perjalanan Terpandu	Mengubah dokumentasi untuk mencerminkan metode yang lebih mutakhir dan disederhanakan untuk orientasi Braket.	5 April 2023
Perangkat baru Rigetti Aspen-M-3	Menambahkan dukungan untuk Rigetti Aspen-M-3 perangkat	Januari 17, 2023
Fitur gradien adjoint baru	Menambahkan informasi tentang fitur gradien adjoint yang ditawarkan oleh SV1	7 Desember 2022
Fitur pustaka algoritma baru	Menambahkan informasi tentang pustaka algoritma Braket, yang menyediakan katalog algoritma kuantum pra-bangun	28 November 2022
D-Wavekeberangkatan	Diperbarui dokumentasi untuk mengakomodasi penghapusan semua D-Wave perangkat	17 November 2022
Perangkat baru QuEra Aquila	Menambahkan dukungan untuk QuEra Aquila perangkat	31 Oktober 2022

Support untuk Braket Pulse	Menambahkan dukungan untuk Braket Pulse, yang memungkinkan kontrol pulsa digunakan pada Rigetti dan perangkat OQC	20 Oktober 2022
Support untuk gerbang asli IonQ	Menambahkan dukungan untuk set gerbang asli yang ditawarkan oleh perangkat ionQ	13 September 2022
Kuota contoh baru	Memperbarui kuota instans komputasi klasik default yang terkait dengan Pekerjaan Hybrid	22 Agustus 2022
Dasbor layanan baru	Tangkapan layar konsol yang diperbarui untuk menyertakan dasbor layanan	17 Agustus 2022
Perangkat baru Rigetti Aspen-M-2	Menambahkan dukungan untuk Rigetti Aspen-M-2 perangkat	12 Agustus 2022
Fitur OpenQASM baru	Menambahkan dukungan fitur OpenQASM untuk simulator lokal (braket_sv dan braket_dm)	Agustus 4, 2022
Prosedur pelacakan biaya baru	Menambahkan cara mendapatkan perkiraan biaya maksimum mendekati waktu nyata untuk simulator dan beban kerja perangkat keras	18 Juli 2022
Xanadu BorealisPerangkat baru	Menambahkan dukungan untuk Xanadu Borealis perangkat	2 Juni 2022

Prosedur penyederhanaan orientasi baru	Menambahkan informasi tentang cara kerja prosedur orientasi yang baru dan disederhanakan	Mei 16, 2022
Perangkat baru D-Wave Advantage_system6.1	Menambahkan dukungan untuk D-Wave Advantage _system6.1 perangkat	12 Mei 2022
Support untuk simulator tertanam	Menambahkan cara menjalankan simulasi tertanam dengan pekerjaan hybrid dan cara menggunakan simulator PennyLane petir	4 Mei, 2022
AmazonBraketFullAccess - Kebijakan akses penuh untuk Amazon Braket	Menambahkan s3: ListAllMy Buckets izin untuk memungkinkan pengguna melihat dan memeriksa bucket yang dibuat dan digunakan untuk Amazon Braket	31 Maret 2022
Support untuk OpenQASM	Menambahkan dukungan OpenQASM 3.0 untuk perangkat kuantum dan simulator berbasis gerbang	7 Maret 2022
Penyedia Perangkat Keras Quantum Baru, Oxford Quantum Circuits dan wilayah baru, eu-west-2	Menambahkan dukungan untuk OQC dan eu-west-2	28 Februari 2022
RigettiPerangkat baru	Menambahkan dukungan untuk Rigetti Aspen M-1	Februari 15, 2022
Batas sumber daya baru	Meningkatkan jumlah maksimum konkuren DM1 dan SV1 tugas dari 55 menjadi 100	5 Januari 2022

RigettiPerangkat baru	Menambahkan dukungan untuk Rigetti Aspen-11	Desember 20, 2021
Perangkat pensiunan Rigetti	Dukungan yang dihentikan untuk perangkat Rigetti Aspen-10	Desember 20, 2021
Jenis hasil baru	Mengurangi jenis hasil matriks densitas yang didukung oleh simulator dan DM1 perangkat matriks kepadatan lokal	Desember 20, 2021
Deskripsi kebijakan yang diperbarui	Amazon Braket memperbarui peran ARN untuk menyertakan jalur. `servicerole/` Untuk informasi tentang pembaruan kebijakan, lihat tabel pembaruan Amazon Braket ke kebijakan AWS terkelola.	29 November 2021
Lowongan Amazon Braket	Panduan pengguna untuk Amazon Braket Hybrid Jobs dan ditambahkan API	29 November 2021
RigettiPerangkat baru	Menambahkan dukungan untuk Rigetti Aspen-10	November 20, 2021
Perangkat pensiunan D-Wave	Dukungan yang dihentikan untuk D-Wave QPU, Advantage_system1	4 November 2021
D-WavePerangkat baru	Menambahkan dukungan untuk D-Wave QPU tambahan, Advantage _system4	5 Oktober 2021

Simulator kebisingan baru	Menambahkan dukungan untuk simulator matriks Densitas (DM1), yang dapat mensimulasikan sirkuit hingga 17 qubits dan simulator kebisingan lokal braket_dm	25 Mei 2021
PennyLane dukungan	Ditambahkan dukungan untuk PennyLane di Amazon Braket	8 Desember 2020
Simulator baru	Menambahkan dukungan untuk Tensor Network Simulator (TN1), yang memungkinkan sirkuit yang lebih besar	8 Desember 2020
Pembuatan batch tugas	Braket mendukung pembuatan batach tugas pelanggan	24 November 2020
Alokasi manual qubit	Braket mendukung qubit alokasi manual pada perangkat Rigetti	24 November 2020
Kuota yang dapat disesuaikan	Braket mendukung kuota yang dapat disesuaikan layanan mandiri untuk sumber daya tugas Anda	30 Oktober 2020
Support untuk PrivateLink	Anda dapat mengatur VPC endpoint pribadi untuk pekerjaan Braket Anda	30 Oktober 2020
Support untuk tag	Braket mendukung tag API berbasis untuk sumber daya tugas kuantum	30 Oktober 2020

D-WavePerangkat baru	Menambahkan dukungan untuk D-Wave QPU tambahan, Advantage _system1	29 September 2020
Rilis awal	Rilis awal dokumentasi Amazon Braket	12 Agustus 2020

AWSGlosarium

Untuk AWS terminologi terbaru, lihat AWSglosarium di Referensi Glosarium. AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.