aws

개발자 가이드

Amazon Braket

Amazon Braket: 개발자 가이드

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

아마존 브라켓이란 무엇입니까?

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

AmazonBraket은 연구자, 과학자 및 개발자가 양자 컴퓨팅을 시작하는 데 도움이 AWS 서비스 되는 완전 관리형 제품입니다. 양자 컴퓨팅은 양자 역학의 법칙을 활용하여 새로운 방식으로 정보를 처리하기 때문에 기존 컴퓨터로는 해결할 수 없는 계산 문제를 해결할 수 있는 잠재력을 가지고 있습니다.

양자 컴퓨팅 하드웨어에 액세스하는 것은 비용이 많이 들고 불편할 수 있습니다. 액세스가 제한되면 알고리즘을 실행하고, 설계를 최적화하고, 기술의 현재 상태를 평가하고, 최대 이익을 위해 리소스를 투자할 시기를 계획하기가 어렵습니다. 브라켓은 이러한 문제를 극복할 수 있도록 도와줍니다.

Braket은 다양한 양자 컴퓨팅 기술에 액세스할 수 있는 단일 지점을 제공합니다. Braket을 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- 양자 및 하이브리드 알고리즘을 탐색하고 설계하세요.
- 다양한 양자 회로 시뮬레이터에서 알고리즘을 테스트해 보세요.
- 다양한 유형의 양자 컴퓨터에서 알고리즘을 실행합니다.
- 개념 증명 애플리케이션을 만드세요.

양자 문제를 정의하고 이를 해결하기 위해 양자 컴퓨터를 프로그래밍하려면 새로운 기술이 필요합니다. 이러한 기술을 익힐 수 있도록 Braket은 양자 알고리즘을 시뮬레이션하고 실행할 수 있는 다양한 환경을 제공합니다. 요구 사항에 가장 적합한 접근 방식을 찾고 노트북이라는 일련의 예제 환경으로 빠르게 시작할 수 있습니다.

브라켓 개발에는 빌드, 테스트, 실행의 세 단계가 있습니다.

빌드 - 브라켓은 쉽게 시작할 수 있는 완전 관리형 Jupyter 노트북 환경을 제공합니다. Braket 노트북에는 Braket SDK를 비롯한 샘플 알고리즘, 리소스 및 개발자 도구가 사전 설치되어 있습니다. Amazon AmazonBraket SDK를 사용하면 양자 알고리즘을 구축한 다음 한 줄의 코드를 변경하여 다양한 양자 컴퓨터 및 시뮬레이터에서 테스트하고 실행할 수 있습니다.

테스트 - Braket은 완전 관리형 고성능 양자 회로 시뮬레이터에 대한 액세스를 제공합니다. 회로를 테스트하고 검증할 수 있습니다. 브라켓은 모든 기본 소프트웨어 구성 요소와 Amazon Elastic Compute

Cloud (Amazon EC2) 클러스터를 처리하여 기존의 고성능 컴퓨팅 (HPC) 인프라에서 양자 회로를 시뮬레이션하는 부담을 없애줍니다.

Run - Braket은 다양한 유형의 양자 컴퓨터에 대한 안전한 온디맨드 액세스를 제공합니다. IonQ,OQC, 에서 게이트 기반 양자 컴퓨터를 이용할 수 있으며 Rigetti 아날로그 해밀턴 시뮬레이터도 이용할 수 있습니다. QuEra 또한 사전 약정이 필요 없으며 개별 공급자를 통해 액세스를 조달할 필요도 없습니다.

양자 컴퓨팅 및 브라켓에 대해

양자 컴퓨팅은 초기 개발 단계에 있습니다. 현재로서는 내결함성을 갖춘 범용 양자 컴퓨터가 존재하지 않는다는 점을 이해하는 것이 중요합니다. 따라서 특정 유형의 양자 하드웨어는 각 사용 사례에 더 적합하며 다양한 컴퓨팅 하드웨어에 액세스하는 것이 중요합니다. Braket은 타사 공급자를 통해 다양한 하드웨어를 제공합니다.

기존 양자 하드웨어는 노이즈로 인해 제한되어 오류가 발생합니다. 업계는 노이즈가 많은 중급 양자 (NISQ) 시대에 살고 있습니다. NISQ 시대에 양자 컴퓨팅 장치는 소음이 너무 심해서 쇼어 알고리즘이나 그로버 알고리즘과 같은 순수 양자 알고리즘을 유지할 수 없습니다. 더 나은 양자 오류 수정이 가능해질 때까지 가장 실용적인 양자 컴퓨팅을 위해서는 기존 (기존) 컴퓨팅 리소스와 양자 컴퓨터를 결합하여 하이브리드 알고리즘을 만들어야 합니다. Braket은 하이브리드 양자 알고리즘을 사용할 수 있도록 도와줍니다.

하이브리드 양자 알고리즘에서는 양자 처리 장치 (QPU) 가 CPU의 보조 프로세서로 사용되므로 기존 알고리즘의 특정 계산 속도가 빨라집니다. 이러한 알고리즘은 기존 컴퓨터와 양자 컴퓨터 간에 계산이 이동하는 반복 처리를 활용합니다. 예를 들어, 현재 양자 컴퓨팅이 화학, 최적화, 기계 학습 분야에서 응용되고 있는 양자 컴퓨팅은 하이브리드 양자 알고리즘의 일종인 변이 양자 알고리즘을 기반으로 합니다. 변형 양자 알고리즘에서 기존의 최적화 루틴은 매개변수화된 양자 회로의 파라미터를 반복적으로 조정합니다. 이는 기계 학습 세트의 오류를 기반으로 신경망의 가중치를 반복적으로 조정하는 것과 거의 같습니다. Braket은 다양한 양자 알고리즘을 지원하는 PennyLane 오픈 소스 소프트웨어 라이브러리에 대한 액세스를 제공합니다.

양자 컴퓨팅은 다음과 같은 네 가지 주요 영역에서 계산 분야에서 주목을 받고 있습니다.

- 인수 분해 및 암호화를 포함한 정수론 (예: Shor의 알고리즘은 정수론 계산을 위한 기본 양자 방법임)
- 최적화 — 제약 조건 충족, 선형 시스템 해결, 기계 학습 포함
- 검색, 숨겨진 부분군, 순서 찾기를 포함한 오라클 컴퓨팅 (예: Grover의 알고리즘은 오라큘러 계산을 위한 기본 양자 방법임)
- 시뮬레이션 — 직접 시뮬레이션, 매듭 불변성, 양자 근사 최적화 알고리즘 (QAOA) 응용 프로그램 포함

몇 가지 예를 들자면 금융 서비스, 생명공학, 제조, 제약 분야에서 이러한 범주의 계산을 적용할 수 있습니다. Braket은 특정 실제 문제 외에도 많은 개념 증명 문제에 이미 적용할 수 있는 기능과 예제 노트북을 제공합니다.

아마존 브라켓 용어 및 개념

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배우세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Braket에서는 다음과 같은 용어와 개념을 사용합니다.

아날로그 해밀턴 시뮬레이션

아날로그 해밀턴 시뮬레이션 (AHS) 은 다물체 시스템의 시간 종속 양자 역학을 직접 시뮬레이션하기 위한 고유한 양자 컴퓨팅 패러다임입니다. AHS에서는 사용자가 시간 종속 해밀턴식을 직접 지정하면 양자 컴퓨터는 이 해밀턴 방식의 연속적인 시간 변화를 직접 에뮬레이션하는 방식으로 조정됩니다. AHS 장치는 일반적으로 특수 목적 장치이며 게이트 기반 장치와 같은 범용 양자 컴퓨터가 아닙니다. 시뮬레이션할 수 있는 해밀턴 클래스로 제한됩니다. 그러나 이러한 Hamiltonian은 기본적으로 장치에 구현되므로 AHS는 알고리즘을 회로로 공식화하고 게이트 연산을 구현하는 데 필요한 오버헤드를 겪지 않습니다.

브라켓

우리는 양자역학의 표준 표기법인 브라켓 표기법을 따서 Braket 서비스의 이름을 지었습니다. 디락 표기법은 1939년 폴 디랙 (Paul Dirac) 이 양자 시스템의 상태를 설명하기 위해 도입했으며, 디락 표기법이라고도 합니다.

브라켓 하이브리드 잡

Amazon브라켓에는 하이브리드 알고리즘의 완전 관리형 실행을 제공하는 Amazon 브라켓 하이브리드 잡스라는 기능이 있습니다. Braket 하이브리드 작업은 세 가지 구성 요소로 구성됩니다.

1. 스크립트, Python 모듈 또는 Docker 컨테이너로 제공할 수 있는 알고리즘의 정의
2. 알고리즘을 실행하는 데 사용할 Amazon EC2 기반 하이브리드 작업 인스턴스입니다. 기본값은 ml.m5.xlarge 인스턴스입니다.
3. 알고리즘의 일부인 양자 작업을 실행하는 데 사용되는 양자 장치. 단일 하이브리드 작업에는 일반적으로 여러 양자 작업이 포함됩니다.

장치

AmazonBraket에서 디바이스는 양자 작업을 실행할 수 있는 백엔드입니다. 기기는 QPU 또는 양자 회로 시뮬레이터일 수 있습니다. 자세히 알아보려면 Amazon Braket 지원 디바이스를 참조하십시오.

게이트 기반 양자 컴퓨팅

회로 기반 QC라고도 하는 게이트 기반 양자 컴퓨팅 (QC) 에서는 계산을 기본 연산 (게이트) 으로 세분화합니다. 특정 게이트 세트는 범용적이므로 모든 계산을 해당 게이트의 유한한 시퀀스로 표현할 수 있습니다. 게이트는 양자 회로의 구성 요소이며 기존 디지털 회로의 로직 게이트와 유사합니다.

해밀턴 사람

물리적 시스템의 양자 역학은 시스템 구성 요소 간의 상호 작용과 외인적 추진력의 영향에 대한 모든 정보를 인코딩하는 해밀턴식 해밀턴에 의해 결정됩니다. N-큐비트 시스템의 해밀턴 행렬은 고전 기계에서 흔히 $2N \times 2^{N}$ 복소수 행렬로 표현됩니다. 양자 장치에서 아날로그 해밀턴식 시뮬레이션을 실행하면 이러한 기하급수적인 리소스 요구 사항을 피할 수 있습니다.

펄스

펄스는 큐비트로 전송되는 일시적인 물리적 신호입니다. 이는 캐리어 신호를 지원하는 프레임에서 재생되는 파형으로 설명되며 하드웨어 채널 또는 포트에 바인딩됩니다. 고객은 고주파 정현파 반송파 신호를 변조하는 아날로그 엔벨로프를 제공하여 자체 펄스를 설계할 수 있습니다. 프레임은 고유한 주파수와 위상으로 설명되는데, 이 주파수와 위상은 큐비트의 |0γ와 |1γ의 에너지 수준 간의 에너지 분리에 따라 공진하도록 선택되는 경우가 많습니다. 따라서 게이트는 미리 정해진 모양과 함께 진폭, 주파수, 지속 시간 등의 파라미터가 조정된 펄스로 실행됩니다. 템플릿 파형이 다루지 않는 사용 사례는 고정된 물리적 사이클 시간으로 구분된 값 목록을 제공하여 단일 샘플 해상도로 지정되는 사용자 지정 파형을 통해 사용할 수 있습니다.

양자 회로

양자 회로는 게이트 기반 양자 컴퓨터의 계산을 정의하는 명령 세트입니다. 양자 회로는 양자 게이트의 시퀀스로, qubit 레지스터에서 측정 명령과 함께 역으로 변환됩니다.

양자 회로 시뮬레이터

양자 회로 시뮬레이터는 기존 컴퓨터에서 실행되며 양자 회로의 측정 결과를 계산하는 컴퓨터 프로그램입니다. 일반 회로의 경우 양자 시뮬레이션의 리소스 요구량은 시뮬레이션할 개수가 많아질수록 기하급수적으로 증가합니다. qubits Braket은 관리형 (Braket을 통해 액세스) 양자 회로 시뮬레이터와 로컬 (Braket API SDK의 일부) 양자 회로 시뮬레이터 모두에 대한 액세스를 제공합니다. Amazon

양자 컴퓨터

양자 컴퓨터는 중첩 및 얽힘과 같은 양자 역학 현상을 사용하여 계산을 수행하는 물리적 장치입니다. 양자 컴퓨팅 (QC) 에는 게이트 기반 QC와 같은 다양한 패러다임이 있습니다.

양자 처리 장치 (QPU)

QPU는 양자 작업을 수행할 수 있는 물리적 양자 컴퓨팅 장치입니다. QPU는 게이트 기반 QC와 같은 다양한 QC 패러다임을 기반으로 할 수 있습니다. 자세히 알아보려면 Amazon Braket 지원 디바이스를 참조하십시오.

QPU 네이티브 게이트

QPU 네이티브 게이트는 QPU 제어 시스템을 통해 제어 펄스에 직접 매핑할 수 있습니다. 네이티브 게이트는 추가 컴파일 없이 QPU 기기에서 실행할 수 있습니다. QPU 지원 게이트의 하위 세트. AmazonBraket 콘솔의 디바이스 페이지와 Braket SDK를 통해 디바이스의 네이티브 게이트를 찾을 수 있습니다.

QPU 지원 게이트

QPU 지원 게이트는 QPU 디바이스에서 허용하는 게이트입니다. 이러한 게이트는 QPU에서 직접 실행되지 않을 수 있습니다. 즉, 네이티브 게이트로 분해해야 할 수도 있습니다. 디바이스의 지원되는 게이트는 Amazon Braket 콘솔의 디바이스 페이지와 Braket SDK를 통해 찾을 수 있습니다. Amazon

퀀텀 태스크

Braket에서 양자 과제는 장치에 대한 원자적 요청입니다. 게이트 기반 QC 장치의 경우 여기에는 양자 회로 (측정 지침 및 개수 포함shots) 및 기타 요청 메타데이터가 포함됩니다. AmazonBraket SDK를 통해 또는 작업을 직접 사용하여 양자 작업을 생성할 수 있습니다. CreateQuantumTask API 양자 태스크를 생성한 후에는 요청된 디바이스를 사용할 수 있을 때까지 큐에 남습니다. AmazonBraket 콘솔의 Quantum Tasks 페이지에서 또는 연산을 사용하여 양자 작업을 볼 수 있습니다. GetQuantumTask SearchQuantumTasks API

Qubit

양자 컴퓨터의 기본 정보 단위를 a qubit (양자 비트) 라고 하는데, 이는 고전 컴퓨팅의 비트와 매우 비슷합니다. qubitA는 초전도 회로 또는 개별 이온 및 원자와 같은 다양한 물리적 구현을 통해 구현할 수 있는 2단계 양자 시스템입니다. 다른 qubit 유형은 광자, 전자 또는 핵 스핀 또는 좀 더 특이한 양자 시스템을 기반으로 합니다.

Queue depth

Queue depth특정 장치가 대기하고 있는 양자 작업 및 하이브리드 작업의 수를 나타냅니다. 장치의 양자 작업 및 하이브리드 작업 대기열 수는 OR를 통해 액세스할 수 있습니다. Braket Software Development Kit (SDK) Amazon Braket Management Console

1. 작업 대기열 깊이는 현재 정상 우선 순위로 실행 대기 중인 양자 작업의 총 수를 나타냅니다.
2. 우선순위 태스크 대기열 깊이는 제출된 양자 태스크가 실행되기를 기다리는 총 개수를 의미합니다Amazon Braket Hybrid Jobs. 하이브리드 작업이 시작되면 이러한 작업이 독립형 작업보다 우선합니다.
3. 하이브리드 작업 대기열 길이는 현재 장치에 대기하고 있는 하이브리드 작업의 총 수를 나타냅니다. Quantum tasks하이브리드 작업의 일부로 제출된 항목은 우선 순위를 가지며 에 집계됩니다. Priority Task Queue

Queue position

Queue position각 장치 대기열 내에서 양자 작업 또는 하이브리드 작업의 현재 위치를 나타냅니다. 양자 작업 또는 하이브리드 작업의 경우 OR를 통해 확인할 수 Amazon Braket Management Console 있습니다. Braket Software Development Kit (SDK)

Shots

양자 컴퓨팅은 본질적으로 확률론적이므로 정확한 결과를 얻으려면 모든 회로를 여러 번 평가해야 합니다. 단일 회로 실행 및 측정을 샷이라고 합니다. 회로의 샷 수 (반복 실행) 는 원하는 결과 정확도에 따라 선택됩니다.

AWS 아마존 브라켓에 대한 용어 및 팁

IAM 정책

IAM 정책은 리소스에 대한 권한을 허용하거나 거부하는 AWS 서비스 문서입니다. IAM 정책을 사용하면 리소스에 대한 사용자의 액세스 수준을 사용자 지정할 수 있습니다. 예를 들어, 사용자가 자신의 AWS 계정모든 Amazon S3 버킷에 액세스하거나 특정 버킷에만 액세스하도록 허용할 수 있습니다.

- 모범 사례: 권한을 부여할 때는 최소 권한의 보안 원칙을 따르십시오. 이 원칙을 따르면 사용자나 역할이 퀀텀 작업을 수행하는 데 필요한 것보다 더 많은 권한을 갖는 것을 방지할 수 있습니다. 예를 들어 직원이 특정 버킷에만 액세스해야 하는 경우 직원에게 모든 버킷에 대한 액세스 권한을 부여하는 대신 IAM 정책에 버킷을 지정하십시오. AWS 계정

IAM 역할

IAM 역할은 권한에 대한 임시 액세스 권한을 부여하는 것으로 간주할 수 있는 ID입니다. 사용자, 애플리케이션 또는 서비스가 IAM 역할을 수임하려면 먼저 해당 역할로 전환할 수 있는 권한을 부여받아야 합니다. IAM 역할을 수임하는 사람은 이전 역할에서 가졌던 이전 권한을 모두 포기하고 새 역할의 권한을 인수합니다.

- 모범 사례: IAM 역할은 서비스 또는 리소스에 대한 액세스 권한을 장기간 부여하지 않고 일시적으로 부여해야 하는 상황에 적합합니다.

아마존 S3 버킷

Amazon Simple Storage Service (Amazon S3) 는 데이터를 AWS 서비스 버킷에 객체로 저장할 수 있는 서비스입니다. Amazon S3 버킷은 무제한 스토리지 공간을 제공합니다. Amazon S3 버킷에 있는 객체의 최대 크기는 5TB입니다. 이미지, 동영상, 텍스트 파일, 백업 파일, 웹 사이트용 미디어 파일, 보관된 문서, Braket 양자 작업 결과 등 모든 유형의 파일 데이터를 Amazon S3 버킷에 업로드할 수 있습니다.

- 모범 사례: S3 버킷에 대한 액세스를 제어할 권한을 설정할 수 있습니다. 자세한 내용은 Amazon S3 설명서의 버킷 정책 및 사용자 정책을 참조하십시오.

아마존 브라켓 가격

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket을 사용하면 사전 약정 없이 온디맨드 양자 컴퓨팅 리소스에 액세스할 수 있습니다. 사용한 만큼만 지불합니다. 요금에 대한 자세한 내용은 요금 페이지를 참조하십시오.

거의 실시간으로 비용을 추적할 수 있습니다.

Braket SDK는 양자 워크로드에 거의 실시간으로 비용 추적을 추가할 수 있는 옵션을 제공합니다. 각 예제 노트북에는 Braket의 QPU (양자 처리 장치) 및 온디맨드 시뮬레이터에 대한 최대 비용 추정치를 제공하는 비용 추적 코드가 포함되어 있습니다. 예상 최대 비용은 USD로 표시되며 크레딧이나 할인은 포함되지 않습니다.

> **Note**
>
> 표시된 요금은 Amazon Braket 시뮬레이터 및 QPU (양자 처리 장치) 작업 사용량을 기준으로 한 추정치입니다. 표시된 예상 요금은 실제 요금과 다를 수 있습니다. 예상 요금에는 할인이나 크레딧이 포함되지 않으며 Amazon Elastic Compute Cloud (Amazon EC2) 와 같은 다른 서비스 사용에 따라 추가 요금이 발생할 수 있습니다.

SV1의 비용 추적

비용 추적 기능을 어떻게 사용할 수 있는지 보여주기 위해 Bell State 회로를 구성하여 SV1 시뮬레이터에서 실행할 예정입니다. 먼저 Braket SDK 모듈을 가져와서 Bell State를 정의한 다음 `Tracker()` 함수를 회로에 추가합니다.

```
#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)
```

노트북을 실행하면 벨 상태 시뮬레이션에 다음과 같은 결과가 출력될 것으로 예상할 수 있습니다. 트래커 기능은 전송된 샷의 수, 완료된 양자 작업, 실행 기간, 청구된 실행 시간, 최대 비용 (미국 달러) 을 보여줍니다. 실행 시간은 각 시뮬레이션마다 다를 수 있습니다.

```
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
```

비용 추적기를 사용하여 최대 비용 설정

비용 추적기를 사용하여 프로그램의 최대 비용을 설정할 수 있습니다. 특정 프로그램에 지출하려는 금액의 최대 한도가 있을 수 있습니다. 이러한 방식으로 비용 추적기를 사용하여 실행 코드에 비용 제어 로직을 구축할 수 있습니다. 다음 예시는 Rigetti QPU에서 동일한 회로를 사용하고 비용을 1 USD로 제한합니다. 코드에서 회로를 한 번 반복하는 데 드는 비용은 0.37 USD입니다. 총 비용이 1 USD를 초과할 때까지 반복을 반복하도록 로직을 설정했습니다. 따라서 다음 반복이 1 USD를 초과할 때까지 코드 스니펫이 세 번 실행됩니다. 일반적으로 프로그램은 원하는 최대 비용에 도달할 때까지 (이 경우 3회 반복) 을 계속합니다.

```
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**
>
> 비용 추적기는 실패한 TN1 양자 작업의 기간을 추적하지 않습니다. TN1시뮬레이션 중에 리허설이 완료되었지만 수축 단계가 실패하면 리허설 요금이 비용 추적기에 표시되지 않습니다.

비용 절감 모범 사례

Amazon Braket을 사용하기 위한 다음 모범 사례를 고려해 보십시오. 시간을 절약하고 비용을 최소화하며 일반적인 오류를 방지하십시오.

시뮬레이터로 검증하십시오.

- QPU에서 회로를 실행하기 전에 시뮬레이터를 사용하여 회로를 검증하면 QPU 사용에 따른 요금 없이 회로를 미세 조정할 수 있습니다.

- 시뮬레이터에서 회로를 실행한 결과가 QPU에서 회로를 실행한 결과와 동일하지 않을 수도 있지만 시뮬레이터를 사용하여 코딩 오류나 구성 문제를 식별할 수 있습니다.

특정 장치에 대한 사용자 액세스를 제한하십시오.

- 권한 없는 사용자가 특정 디바이스에서 양자 작업을 제출하지 못하도록 제한을 설정할 수 있습니다. 액세스를 제한하는 권장 방법은 AWS IAM을 사용하는 것입니다. 이를 수행하는 방법에 대한 자세한 내용은 액세스 제한을 참조하십시오.
- Amazon Braket 디바이스에 대한 사용자 액세스를 제공하거나 제한하는 방법으로 관리자 계정을 사용하지 않는 것이 좋습니다.

청구 경보 설정

- 청구서가 사전 설정된 한도에 도달하면 알림을 받도록 청구 경보를 설정할 수 있습니다. 알람을 설정하는 권장 방법은 다음과 같습니다 AWS Budgets. 사용자 지정 예산을 설정하고 비용이나 사용량이 예산 금액을 초과할 경우 알림을 받을 수 있습니다. 자세한 내용은 에서 확인할 수 있습니다. AWS Budgets

샷 수가 적은 상태에서 TN1 양자 작업을 테스트하십시오.

- 시뮬레이터는 QHP보다 비용이 적게 들지만 샷 수가 많은 양자 작업을 실행하는 경우 특정 시뮬레이터는 비용이 많이 들 수 있습니다. 적은 수로 TN1 작업을 테스트하는 것이 좋습니다. shot Shot개수는 로컬 시뮬레이터 작업 비용 SV1 및 비용에 영향을 주지 않습니다.

모든 지역에서 양자 태스크를 확인하세요.

- 콘솔에는 현재 사용자의 양자 작업만 표시됩니다 AWS 리전. 제출된 청구 가능한 양자 작업을 찾으려면 모든 지역을 확인하세요.
- 지원되는 장치 설명서 페이지에서 장치 및 관련 지역 목록을 볼 수 있습니다.

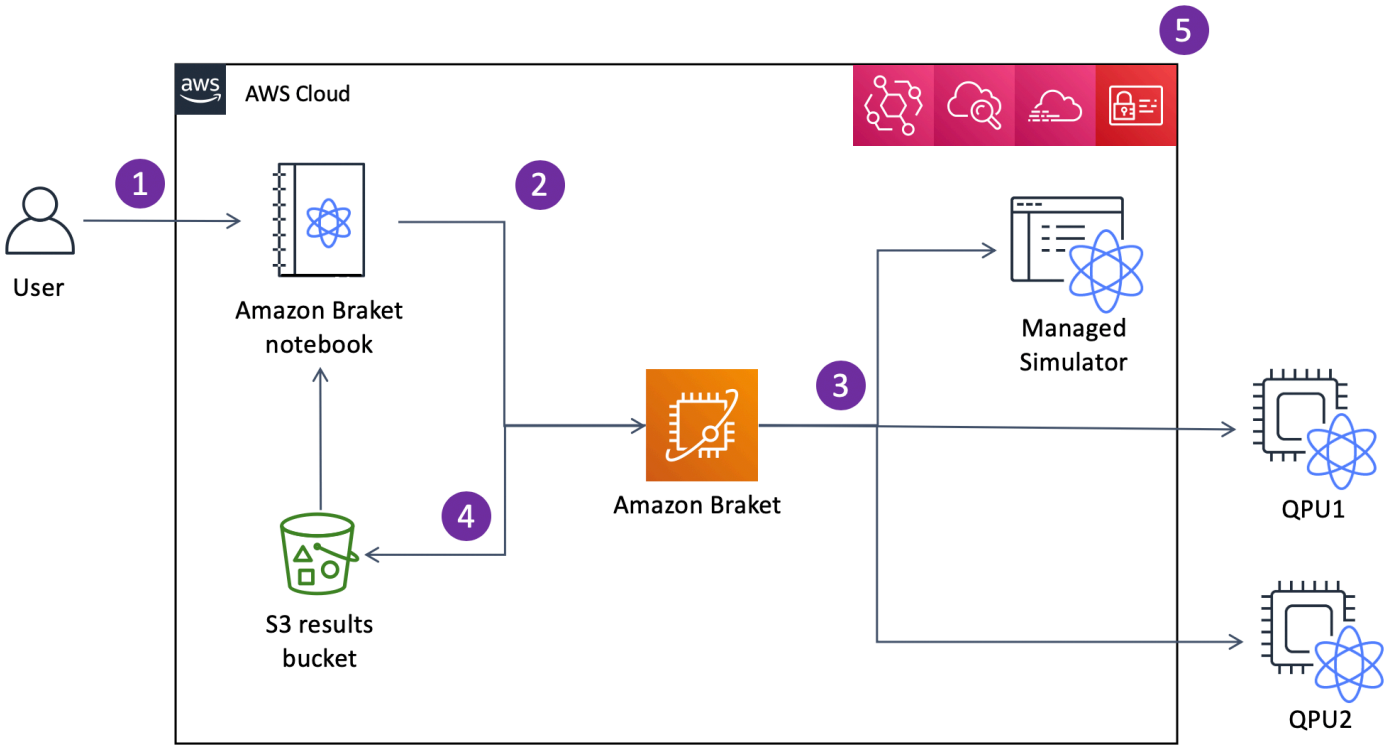
아마존 브라켓 작동 방식

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket은 온디맨드 회로 시뮬레이터 및 다양한 유형의 QPU를 비롯한 양자 컴퓨팅 디바이스에 대한 온디맨드 액세스를 제공합니다. Amazon Braket에서 디바이스에 대한 원자적 요청은 양자 작업입니다. 게이트 기반 QC 디바이스의 경우 이 요청에는 양자 회로 (측정 지침 및 샷 수 포함) 및 기타 요청 메타데이터가 포함됩니다. 아날로그 해밀턴 시뮬레이터의 경우 양자 작업에는 양자 레지스터의 물리적 레이아웃과 조작 필드의 시간 및 공간 종속성이 포함됩니다.

이 섹션에서는 Braket에서 양자 작업을 실행하는 높은 수준의 흐름에 대해 알아보겠습니다. Amazon

아마존 브라켓 퀀텀 태스크 플로우



Jupyter노트북을 사용하면 Amazon Braket 콘솔 또는 Amazon Braket SDK를 사용하여 양자 작업을 편리하게 정의, 제출 및 모니터링할 수 있습니다. SDK에서 직접 양자 회로를 구축할 수 있습니다. 하지만 아날로그 해밀턴 시뮬레이터의 경우 레지스터 레이아웃과 제어 필드를 정의합니다. 양자 작업이 정의되면 이를 실행할 디바이스를 선택하고 Amazon Braket API (2) 에 제출할 수 있습니다. 선택한 디바이스에 따라 Quantum 작업은 디바이스를 사용할 수 있게 될 때까지 대기열에 추가되며, 작업은 구현을 위해 QPU 또는 시뮬레이터로 전송됩니다 (3). Amazon Braket을 사용하면 다양한 유형의 QPU (IonQ,,,)Oxford Quantum Circuits (OQC), QuEra 온디맨드 시뮬레이터 3개 (,,Rigetti), 로컬 시뮬레이터 2개 SV1DM1, TN1 내장 시뮬레이터 1개에 액세스할 수 있습니다. 자세히 알아보려면 Amazon Braket 지원 디바이스를 참조하십시오.

양자 작업을 처리한 후 Amazon Braket은 데이터를 AWS 계정 (4) 에 저장하는 Amazon S3 버킷으로 결과를 반환합니다. 이와 동시에 SDK는 백그라운드에서 결과를 폴링하여 양자 작업 완료 시 Jupyter 노트북에 로드합니다. AmazonBraket 콘솔의 Quantum Tasks 페이지에서 또는 Braket의 작동을 사용하여 양자 작업을 보고 관리할 수도 있습니다. `GetQuantumTask` Amazon API

AmazonBraket은 사용자 액세스 관리 CloudWatch, 모니터링 및 로깅은 물론 이벤트 기반 처리를 EventBridge 위해 AWS Identity and Access Management (IAM), AWS CloudTrail Amazon 및 Amazon과 통합됩니다 (5).

타사 데이터 처리

QPU 장치에 제출된 양자 작업은 타사 공급자가 운영하는 시설에 위치한 양자 컴퓨터에서 처리됩니다. Amazon Braket의 보안 및 타사 처리에 대한 자세한 내용은 Amazon Braket 하드웨어 공급자의 보안을 참조하십시오.

Braket용 코어 리포지토리 및 플러그인

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

코어 리포지토리

다음은 Braket에 사용되는 주요 패키지가 포함된 핵심 리포지토리 목록을 표시합니다.

- 브라켓 Python SDK - 브라켓 Python SDK를 사용하여 Python 프로그래밍 언어로 Jupyter 노트북에 코드를 설정할 수 있습니다. Jupyter노트북을 설정한 후에는 Braket 장치 및 시뮬레이터에서 코드를 실행할 수 있습니다.
- 브라켓 스키마 - 브라켓 SDK와 브라켓 서비스 간의 계약입니다.
- 브라켓 디폴트 시뮬레이터 - 브라켓을 위한 모든 로컬 양자 시뮬레이터 (상태 벡터 및 밀도 매트릭스).

플러그인

그런 다음 다양한 장치 및 프로그래밍 도구와 함께 사용되는 다양한 플러그인이 있습니다. 여기에는 아래와 같이 Braket 지원 플러그인과 타사에서 지원하는 플러그인이 포함됩니다.

아마존 브라켓 지원:

- Amazon Braket 알고리즘 라이브러리 - Python으로 작성된 사전 구축된 양자 알고리즘의 카탈로그입니다. 그대로 실행하거나 출발점으로 삼아 더 복잡한 알고리즘을 구축할 수 있습니다.
- 브라켓 PennyLane 플러그인 - 브라켓에서 QML PennyLane 프레임워크로 사용합니다.

타사 (브라켓 팀 모니터링 및 기여):

- 키스킷-브라켓 제공자 - SDK를 사용하여 브라켓 리소스에 액세스할 수 Qiskit 있습니다.
- 브라켓-줄리아 SDK - (실험용) 브라켓 SDK의 줄리아 네이티브 버전

아마존 브라켓 지원 디바이스

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 알아보십시오! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket에서 디바이스는 양자 작업을 실행하기 위해 호출할 수 있는 QPU 또는 시뮬레이터를 나타냅니다. Amazon Braket은,, 및 온디맨드 시뮬레이터 3개 IonQIQM, 로컬 시뮬레이터 3개 Oxford Quantum Circuits QuEraRigetti, 내장 시뮬레이터 1개에서 QPU 디바이스에 액세스할 수 있도록 합니다. 모든 디바이스에 대해 Amazon Braket 콘솔의 디바이스 탭 또는 API를 통해 디바이스 토폴로지, 보

정 데이터, 네이티브 게이트 세트와 같은 추가 디바이스 속성을 찾을 수 있습니다. GetDevice 시뮬레이터로 회로를 구성할 때 Amazon Braket에서는 현재 연속 큐비트 또는 인덱스를 사용할 것을 요구합니다. Amazon Braket SDK로 작업하는 경우 다음 코드 예제와 같이 디바이스 속성에 액세스할 수 있습니다.

```
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
```

지원되는 양자 하드웨어 제공업체

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

지원되는 시뮬레이터

- 로컬 상태 벡터 시뮬레이터 (braket_sv) ('디폴트 시뮬레이터')
- 로컬 밀도 `braket_dm` 매트릭스 시뮬레이터 ()
- 로컬 AHS 시뮬레이터
- 상태 벡터 시뮬레이터 (SV1)
- 밀도 매트릭스 시뮬레이터 (DM1)
- 텐서 네트워크 시뮬레이터 () TN1
- PennyLane의 라이트닝 시뮬레이터

양자 작업에 가장 적합한 시뮬레이터를 선택하세요.

- 시뮬레이터 비교

> **Note**
>
> 각 기기에서 사용할 수 있는 AWS 리전 제품을 보려면 다음 표를 오른쪽으로 스크롤하세요.

아마존 브라켓 디바이스

공급자	디바이스 이름	패러다임	유형	디바이스 ARN	리전
IonQ	Aria 1	게이트 기반	QPU	arn:aws:braket:us-east-1: :디바이스/QPU/IONQ/ARIA-1	us-east-1
IonQ	Aria 2	게이트 기반	QPU	arn:aws:braket:us-east-1: :디바이스/QPU/IONQ/ARIA-2	us-east-1
IonQ	Forte 1	게이트 기반	QPU (예약 전용)	arn:aws:braket:us-east-1: :디바이스/QPU/IONQ/포르테-1	us-east-1
IonQ	Harmony	게이트 기반	QPU	arn:aws:braket:us-east-1: :디바이스/QPU/IONQ/하모니	us-east-1
IQM	Garnet	게이트 기반	QPU	arn:aws:braket:eu-north-1: :디바이스/QPU/IQM/가넷	eu-north-1
Oxford Quantum Circuits	Lucy	게이트 기반	QPU	arn:aws:braket:eu-west-2: :디바이스/QPU/OQC/루시	eu-west-2
QuEra	Aquila	아날로그 해밀턴 시뮬레이션	QPU	arn:aws:braket:us-east-1: :디바이스/QPU/쿼라/아퀼라	us-east-1
Rigetti	Aspen M-3	게이트 기반	QPU	arn:aws:braket:us-west-1: :디바이스/QPU/리게티/아스펜-M-3	us-west-1
AWS	braket_sv	게이트 기반	로컬 시뮬레이터	N/A (브라켓 SDK의 로컬 시뮬레이터)	N/A

공급자	디바이스 이름	패러다임	유형	디바이스 ARN	리전
AWS	braket_dm	게이트 기반	로컬 시뮬레이터	N/A (브라켓 SDK의 로컬 시뮬레이터)	N/A
AWS	SV1	게이트 기반	온디맨드 시뮬레이터	arn:aws:브래킷: ::디바이스/퀀텀 시뮬레이터/아마존/sv1	브라켓을 사용할 수 있는 모든 지역. Amazon
AWS	DM1	게이트 기반	온디맨드 시뮬레이터	arn:aws:브래킷: ::디바이스/퀀텀 시뮬레이터/아마존/dm1	브라켓을 사용할 수 있는 모든 지역. Amazon
AWS	TN1	게이트 기반	온디맨드 시뮬레이터	arn:aws:브래킷: ::디바이스/퀀텀 시뮬레이터/아마존/tn1	us-west-2, us-east-1 및 eu-west-2

> **Note**
>
> 특정 QPU는 브라켓 다이렉트를 통한 예약을 통해서만 이용할 수 있습니다 (예약 참조).

Braket과 함께 사용할 수 있는 QPU에 대한 추가 세부 정보를 보려면 Amazon Amazon Braket 하드웨어 공급자를 참조하십시오.

IonQ

IonQ이온 트랩 기술을 기반으로 하는 게이트 기반 QPU를 제공합니다. IonQ's트랩형 이온 QPU는 진공 챔버 내에서 미세하게 제작된 표면 전극 트랩을 통해 공간적으로 제한된 171Yb+ 이온 체인 위에 구축됩니다.

IonQ디바이스는 다음과 같은 양자 게이트를 지원합니다.

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

축어 컴파일을 사용하면 IonQ QPU는 다음과 같은 네이티브 게이트를 지원합니다.

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

네이티브 MS 게이트를 사용할 때 위상 파라미터를 두 개만 지정하면 완전히 얽힌 MS 게이트가 실행됩니다. 완전히 얽힌 MS 게이트는 항상 π/2 회전을 수행합니다. 다른 각도를 지정하고 부분적으로 얽힌 MS 게이트를 실행하려면 세 번째 매개변수를 추가하여 원하는 각도를 지정합니다. 자세한 내용은 braket.circuits.gate 모듈을 참조하십시오.

이러한 네이티브 게이트는 축어 컴파일에만 사용할 수 있습니다. 축어 컴파일에 대한 자세한 내용은 축어 컴파일을 참조하십시오.

IQM

IQM양자 프로세서는 초전도 트랜스 몬 큐비트를 기반으로 하는 범용 게이트 모델 장치입니다. 이 IQM Garnet 장치는 정사각형 격자 토폴로지를 가진 20큐비트 장치입니다.

이 IQM 장치는 다음과 같은 양자 게이트를 지원합니다.

```
"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"
```

축어 컴파일을 사용하면 IQM 기기가 다음과 같은 네이티브 게이트를 지원합니다.

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

Rigetti

Rigetti양자 프로세서는 모든 튜닝이 가능한 초전도를 기반으로 하는 범용 게이트 모델 기계입니다. qubits 79큐비트 Aspen-M-3 장치는 독자적인 멀티칩 기술을 활용하며 2개의 40큐비트 프로세서로 조립됩니다.

이 장치는 다음과 같은 양자 게이트를 지원합니다. Rigetti

```
'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'
```

축어 컴파일을 통해 Rigetti 디바이스는 다음과 같은 네이티브 게이트를 지원합니다.

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

Rigetti초전도 양자 프로세서는 ±π/2 또는 ±π의 각도만으로 'rx' 게이트를 실행할 수 있습니다.

디바이스에서 펄스 레벨 제어를 사용할 수 있으며, 이 Rigetti 디바이스는 다음과 같은 유형의 사전 정의된 프레임 세트를 지원합니다.

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

이러한 프레임에 대한 자세한 내용은 프레임 및 포트의 역할을 참조하십시오.

Oxford Quantum Circuits (OQC)

OQC양자 프로세서는 확장 가능한 Coaxmon 기술을 사용하여 제작된 범용 게이트 모델 시스템입니다. 이 OQC Lucy 시스템은 각각 가장 가까운 이웃 두 개에 연결되는 링 모양의 토폴로지를 가진 8-qubit qubit 장치입니다.

이 Lucy 장치는 다음과 같은 양자 게이트를 지원합니다.

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

OQC 기기는 축어 컴파일을 통해 다음과 같은 네이티브 게이트를 지원합니다.

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

디바이스에서 펄스 레벨 제어를 사용할 수 있습니다. OQC OQC장치는 다음 유형의 사전 정의된 프레임 세트를 지원합니다.

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

OQC디바이스는 유효한 포트 식별자를 제공하는 경우 프레임의 동적 선언을 지원합니다. 이러한 프레임 및 포트에 대한 자세한 내용은 프레임 및 포트의 역할을 참조하십시오.

> **Note**
>
> 펄스 제어를 사용하는 경우 프로그램 길이는 최대 90마이크로초를 초과할 수 없습니다. OQC 최대 지속 시간은 단일 큐비트 게이트의 경우 약 50나노초, 2-큐비트 게이트의 경우 1마이크로초입니다. 이 수치는 사용된 큐비트, 장치의 전류 보정 및 회로 컴파일에 따라 달라질 수 있습니다.

QuEra

QuEra아날로그 해밀턴 시뮬레이션 (AHS) 양자 작업을 실행할 수 있는 중성 원자 기반 장치를 제공합니다. 이 특수 목적 장치는 동시에 상호 작용하는 수백 개의 큐비트의 시간에 따른 양자 역학을 충실하게 재현합니다.

큐비트 레지스터의 레이아웃과 조작 필드의 시간적, 공간적 의존성을 규정함으로써 아날로그 해밀턴 시뮬레이션의 패러다임에 따라 이러한 장치를 프로그래밍할 수 있습니다. Amazon Braket은 파이썬 SDK의 AHS 모듈을 통해 이러한 프로그램을 구성하는 유틸리티를 제공합니다. `braket.ahs`

자세한 내용은 아날로그 해밀턴 시뮬레이션 예제 노트북 또는 의 Aquila를 사용하여 아날로그 프로그램 제출 페이지를 참조하십시오. QuEra

로컬 상태 벡터 시뮬레이터 () `braket_sv`

로컬 상태 벡터 시뮬레이터 (`braket_sv`) 는 사용자 환경에서 로컬로 실행되는 Amazon Braket SDK의 일부입니다. Braket 노트북 인스턴스의 하드웨어 사양 또는 로컬 환경에 따라 소형 회로 (최대 25개 qubits) 에서 신속한 프로토타이핑에 적합합니다.

로컬 시뮬레이터는 Amazon Braket SDK의 모든 게이트를 지원하지만 QPU 기기는 더 작은 하위 집합을 지원합니다. 장치 속성에서 장치의 지원되는 게이트를 찾을 수 있습니다.

> **Note**
>
> 로컬 시뮬레이터는 QPU 장치 또는 기타 시뮬레이터에서는 지원되지 않을 수 있는 고급 OpenQASM 기능을 지원합니다. 지원되는 기능에 대한 자세한 내용은 OpenQASM 로컬 시뮬레이터 노트북에 제공된 예제를 참조하십시오.

시뮬레이터 사용 방법에 대한 자세한 내용은 Amazon Braket 예제를 참조하십시오.

로컬 밀도 매트릭스 시뮬레이터 () `braket_dm`

로컬 밀도 매트릭스 시뮬레이터 (`braket_dm`) 는 사용자 환경에서 로컬로 실행되는 Amazon Braket SDK의 일부입니다. Braket 노트북 인스턴스의 하드웨어 사양 또는 로컬 환경에 따라 노이즈 (최대 12개qubits) 가 있는 소형 회로에서 빠른 프로토타이핑을 수행하는 데 적합합니다.

비트 플립 및 디폴라라이징 오류와 같은 게이트 노이즈 연산을 사용하여 노이즈가 발생하는 일반적인 회로를 처음부터 만들 수 있습니다. 잡음 유무에 관계없이 실행되도록 설계된 기존 회로의 특정 게이트 qubits 및 게이트에 잡음 연산을 적용할 수도 있습니다.

`braket_dm`로컬 시뮬레이터는 지정된 개수가 주어지면 다음과 같은 결과를 제공할 수 있습니다shots.

- 밀도 감소 매트릭스: Shots = 0

> **Note**
>
> 로컬 시뮬레이터는 고급 OpenQASM 기능을 지원하지만 QPU 장치 또는 기타 시뮬레이터에서는 지원되지 않을 수 있습니다. 지원되는 기능에 대한 자세한 내용은 OpenQASM 로컬 시뮬레이터 노트북에 제공된 예제를 참조하십시오.

로컬 밀도 매트릭스 시뮬레이터에 대한 자세한 내용은 Braket 입문 노이즈 시뮬레이터 예제를 참조하십시오.

로컬 AHS 시뮬레이터 () `braket_ahs`

로컬 AHS (아날로그 해밀턴 시뮬레이션) 시뮬레이터 (`braket_ahs`) 는 사용자 환경에서 로컬로 실행되는 Amazon Braket SDK의 일부입니다. AHS 프로그램의 결과를 시뮬레이션하는 데 사용할 수 있습니다. Braket 노트북 인스턴스의 하드웨어 사양 또는 로컬 환경에 따라 작은 레지스터 (최대 10-12 원자) 에서 프로토타이핑하는 데 적합합니다.

로컬 시뮬레이터는 하나의 균일한 구동 필드, 하나의 (비균일) 이동 필드 및 임의의 원자 배열을 갖는 AHS 프로그램을 지원합니다. 자세한 내용은 브라켓 AHS 클래스 및 브라켓 AHS 프로그램 스키마를 참조하십시오.

로컬 AHS 시뮬레이터에 대한 자세한 내용은 Hello AHS: 첫 번째 아날로그 해밀턴 시뮬레이션 실행 페이지 및 아날로그 해밀턴 시뮬레이션 예제 노트북을 참조하십시오.

상태 벡터 시뮬레이터 () SV1

SV1온디맨드 고성능 범용 상태 벡터 시뮬레이터입니다. 최대 34개의 회로를 시뮬레이션할 수 있습니다. qubits 고밀도 사각형 회로 (회로 깊이 = 34) 는 사용된 게이트 유형 및 기타 요인에 따라 완료하는 데 약 1~2시간이 걸릴 것으로 예상할 수 있습니다. 34-qubit all-to-all 게이트가 있는 회로가 적합합니다. SV1 전체 상태 벡터 또는 진폭 배열과 같은 형식으로 결과를 반환합니다.

SV1최대 실행 시간은 6시간입니다. 기본적으로 35개의 동시 양자 작업과 최대 100개 (us-west-1 및 eu-west-2의 경우 50개) 의 동시 양자 작업이 있습니다.

SV1결과

SV1지정된 개수가 주어지면 다음과 같은 결과를 제공할 수 있습니다shots.

- 샘플: Shots > 0
- 기대치: Shots >= 0
- 편차: >= 0 Shots
- 확률: > 0 Shots
- 진폭: = 0 Shots
- 인접 그래디언트: = 0 Shots

결과에 대한 자세한 내용은 결과 유형을 참조하십시오.

SV1항상 사용할 수 있으며 필요에 따라 회로를 실행하며 여러 회로를 병렬로 실행할 수 있습니다. 런타임은 작업 수에 따라 선형적으로 확장되고 작업 수에 따라 기하급수적으로 확장됩니다. qubits 개수는 shots 런타임에 미미한 영향을 미칩니다. 자세히 알아보려면 시뮬레이터 비교를 참조하십시오.

시뮬레이터는 Braket SDK의 모든 게이트를 지원하지만 QPU 기기는 더 작은 하위 집합을 지원합니다. 기기 속성에서 기기의 지원되는 게이트를 찾을 수 있습니다.

밀도 매트릭스 시뮬레이터 (DM1)

DM1온디맨드 고성능 밀도 매트릭스 시뮬레이터입니다. 최대 17개의 회로를 시뮬레이션할 수 있습니다. qubits

DM1최대 실행 시간은 6시간, 기본값은 동시 양자 작업 35개, 동시 양자 작업은 최대 50개입니다.

DM1결과

DM1지정된 개수가 주어지면 다음과 같은 결과를 제공할 수 있습니다shots.

- 샘플: Shots > 0
- 기대치: Shots >= 0
- 편차: >= 0 Shots
- 확률: > 0 Shots
- 밀도 감소 행렬: Shots = 0, 최대 8 qubits

결과에 대한 자세한 내용은 결과 유형을 참조하십시오.

DM1항상 사용할 수 있으며 필요에 따라 회로를 실행하며 여러 회로를 병렬로 실행할 수 있습니다. 런타임은 작업 수에 따라 선형적으로 확장되고 작업 수에 따라 기하급수적으로 확장됩니다. qubits 개수는 shots 런타임에 미미한 영향을 미칩니다. 자세히 알아보려면 시뮬레이터 비교를 참조하십시오.

노이즈 게이트 및 제한

```
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]
```

텐서 네트워크 시뮬레이터 () TN1

TN1온디맨드 고성능 텐서 네트워크 시뮬레이터입니다. TN1최대 50개qubits, 회로 깊이가 1,000 이하인 특정 회로 유형을 시뮬레이션할 수 있습니다. TN1희소 회로, 로컬 게이트가 있는 회로, 양자 푸리에

변환 (QFT) 회로와 같은 특수 구조의 기타 회로에 특히 강력합니다. TN1두 단계로 작동합니다. 먼저 리허설 단계에서는 회로의 효율적인 계산 경로를 식별하여 수축 단계라고 하는 다음 단계의 실행 시간을 TN1 추정할 수 있도록 합니다. 예상 수축 시간이 TN1 시뮬레이션 런타임 제한을 초과하는 경우에는 수축을 시도하지 않습니다. TN1

TN1런타임 제한은 6시간입니다. 동시 양자 작업은 최대 10개 (eu-west-2의 경우 5개) 로 제한됩니다.

TN1결과

수축 단계는 일련의 행렬 곱셈으로 구성됩니다. 일련의 곱셈은 결과에 도달하거나 결과에 도달할 수 없다고 판단될 때까지 계속됩니다.

참고: 0보다 Shots 커야 합니다.

결과 유형에는 다음이 포함됩니다.

- Sample
- 기대치
- 변화

결과에 대한 자세한 내용은 결과 유형을 참조하십시오.

TN1항상 사용할 수 있으며 필요에 따라 회로를 실행하며 여러 회로를 병렬로 실행할 수 있습니다. 자세히 알아보려면 시뮬레이터 비교를 참조하십시오.

시뮬레이터는 Braket SDK의 모든 게이트를 지원하지만 QPU 기기는 더 작은 하위 집합을 지원합니다. 기기 속성에서 기기의 지원되는 게이트를 찾을 수 있습니다.

AmazonBraket GitHub 리포지토리를 방문하여 시작하는 데 도움이 되는 TN1 예제 노트북을 찾아보십시오. TN1

작업 모범 사례 TN1

- all-to-all 회로를 피하세요.
- 적은 수의 shots 새 회로나 회로 종류를 테스트하여 회로의 “경도”를 TN1 파악하십시오.
- 대규모 shot 시뮬레이션을 여러 양자 작업으로 나눕니다.

임베디드 시뮬레이터

임베디드 시뮬레이터는 동일한 컨테이너에 알고리즘 코드와 함께 시뮬레이션을 임베드하고 하이브리드 작업 인스턴스에서 직접 시뮬레이션을 실행하는 방식으로 작동합니다. 이는 시뮬레이션이 원격 장치와 통신하는 것과 관련된 병목 현상을 제거하는 데 유용할 수 있습니다. 그 결과 메모리 사용량이 크게 줄어들고, 원하는 결과를 얻기 위한 회로 실행 횟수가 줄어들며, 성능이 10배 이상 향상될 수 있습니다. 임베디드 시뮬레이터에 대한 자세한 내용은 Amazon Braket 하이브리드 작업 페이지를 사용하여 하이브리드 작업 실행 페이지를 참조하십시오.

PennyLane의 라이트닝 시뮬레이터

PennyLane의 라이트닝 시뮬레이터를 Braket의 임베디드 시뮬레이터로 사용할 수 있습니다. PennyLane의 라이트닝 시뮬레이터를 사용하면 인접 미분과 같은 고급 그래디언트 계산 방법을 활용하여 그래디언트를 더 빠르게 평가할 수 있습니다. lightning.qubit 시뮬레이터는 Braket NBI를 통한 장치 및 임베디드 시뮬레이터로 사용할 수 있지만, lightning.gpu 시뮬레이터는 GPU 인스턴스가 있는 임베디드 시뮬레이터로 실행해야 합니다. lightning.gpu를 사용하는 예제는 브라켓 하이브리드 잡스 노트북의 임베디드 시뮬레이터를 참조하십시오.

시뮬레이터 비교

이 섹션에서는 몇 가지 개념, 제한 사항 및 사용 사례를 설명하여 양자 작업에 가장 적합한 Amazon Braket 시뮬레이터를 선택할 수 있도록 도와줍니다.

로컬 시뮬레이터와 온디맨드 시뮬레이터 중 선택 (,,) SV1 TN1 DM1

로컬 시뮬레이터의 성능은 로컬 환경을 호스팅하는 하드웨어 (예: 시뮬레이터 실행에 사용되는 Braket 노트북 인스턴스) 에 따라 달라집니다. 온디맨드 시뮬레이터는 AWS 클라우드에서 실행되며 일반적인 로컬 환경 이상으로 확장할 수 있도록 설계되었습니다. 온디맨드 시뮬레이터는 대형 회로에 최적화되어 있지만 양자 작업 또는 양자 작업 배치당 지연 시간 오버헤드가 약간 추가됩니다. 이는 많은 양자 작업이 수반되는 경우 절충점을 의미할 수 있습니다. 이러한 일반적인 성능 특성을 고려할 때, 다음 지침은 노이즈가 있는 시뮬레이션을 포함한 시뮬레이션 실행 방법을 선택하는 데 도움이 될 수 있습니다.

시뮬레이션의 경우:

- 18명 미만을 고용할 때는 로컬 qubits 시뮬레이터를 사용하십시오.
- 18~24명을 사용하는 경우 qubits 워크로드에 따라 시뮬레이터를 선택하십시오.
- 24명 이상을 고용하는 경우 온디맨드 qubits 시뮬레이터를 사용하십시오.

소음 시뮬레이션의 경우:

- 9명 미만을 사용하는 qubits 경우 로컬 시뮬레이터를 사용하십시오.
- 9~12명을 사용하는 경우 qubits 워크로드에 따라 시뮬레이터를 선택하십시오.
- 12명 이상을 고용할 때는 사용하십시오. qubits DM1

상태 벡터 시뮬레이터란 무엇입니까?

SV1범용 상태 벡터 시뮬레이터입니다. 양자 상태의 전파 함수를 저장하고 상태에 게이트 연산을 순차적으로 적용합니다. 가능성이 극히 낮은 가능성을 포함한 모든 가능성을 저장합니다. 양자 작업을 위한 SV1 시뮬레이터의 실행 시간은 회로의 게이트 수에 따라 선형적으로 증가합니다.

밀도 매트릭스 시뮬레이터란 무엇입니까?

DM1잡음이 있는 양자 회로를 시뮬레이션합니다. 시스템의 전체 밀도 매트릭스를 저장하고 회로의 게이트 및 노이즈 연산을 순차적으로 적용합니다. 최종 밀도 행렬에는 회로 실행 후의 양자 상태에 대한 완전한 정보가 포함됩니다. 런타임은 일반적으로 연산 횟수에 따라 선형적으로 확장되고 연산 횟수에 따라 기하급수적으로 확장됩니다. qubits

텐서 네트워크 시뮬레이터란 무엇인가요?

TN1양자 회로를 구조화된 그래프로 인코딩합니다.

- 그래프의 노드는 양자 게이트, 즉. qubits
- 그래프의 가장자리는 게이트 간의 연결을 나타냅니다.

이 구조의 결과로 비교적 크고 복잡한 양자 회로에 대한 시뮬레이션 솔루션을 찾을 TN1 수 있습니다.

TN1두 단계가 필요합니다.

일반적으로 양자 TN1 계산을 시뮬레이션하기 위한 2단계 접근 방식으로 작동합니다.

- 리허설 단계: 이 단계에서는 그래프를 효율적으로 탐색할 수 있는 방법을 고안합니다. 이 단계에서는 원하는 측정값을 얻을 수 있도록 모든 노드를 방문해야 합니다. TN1 두 단계를 함께 TN1 수행하므로 고객에게는 이 단계가 표시되지 않습니다. 첫 번째 단계를 완료하고 현실적인 제약에 따라 두 번째 단계를 자체적으로 수행할지 여부를 결정합니다. 시뮬레이션이 시작된 후에는 결정을 내릴 때 어떤 의견도 제시할 수 없습니다.
- 축소 단계: 이 단계는 기존 컴퓨터의 계산 실행 단계와 유사합니다. 이 단계는 일련의 행렬 곱셈으로 구성됩니다. 이러한 곱셈의 순서는 계산의 난이도에 큰 영향을 미칩니다. 따라서 그래프에서 가장 효과적인 계산 경로를 찾기 위해 리허설 단계를 먼저 수행합니다. 리허설 단계에서 수축 경로를 찾은 후 회로의 게이트를 함께 TN1 수축시켜 시뮬레이션 결과를 생성합니다.

TN1그래프는 지도와 비슷합니다.

은유적으로 보면 기본 TN1 그래프를 도시의 거리와 비교할 수 있습니다. 계획된 그리드가 있는 도시에서는 지도를 사용하여 목적지까지의 경로를 쉽게 찾을 수 있습니다. 계획되지 않은 도로, 중복된 도로 이름 등이 있는 도시에서는 지도를 보고 목적지까지 가는 경로를 찾기가 어려울 수 있습니다.

리허설 단계를 수행하지 TN1 않았다면 지도를 먼저 보는 대신 도시의 거리를 돌아다니며 목적지를 찾는 것과 같을 것입니다. 지도를 보는 데 더 많은 시간을 할애하면 걸을 수 있다는 측면에서 정말 도움이 될 수 있습니다. 마찬가지로 리허설 단계에서도 중요한 정보를 얻을 수 있습니다.

TN1이 회로가 통과하는 기본 회로의 구조를 어느 정도 “인식”하고 있다고 말할 수 있습니다. 리허설 단계에서 이러한 인식을 얻습니다.

이러한 각 유형의 시뮬레이터에 가장 적합한 문제 유형

SV1주로 특정 개수의 AND 게이트를 사용하는 데 의존하는 모든 종류의 문제에 적합합니다. qubits 일반적으로 필요한 시간은 게이트 수에 따라 선형적으로 증가하지만 게이트 수에 따라 달라지지는 않습니다. shots SV1일반적으로 qubits 28개 TN1 미만의 회로보다 빠릅니다.

SV1qubit숫자가 클수록 모든 가능성, 심지어 극히 희박한 가능성까지도 실제로 시뮬레이션하기 때문에 속도가 더 느릴 수 있습니다. 어떤 결과가 나올지 판단할 방법이 없습니다. 따라서 30-qubit 평가를 위해서는 2^30개의 SV1 구성을 계산해야 합니다. AmazonBraket SV1 시뮬레이터의 한도는 메모리 및 스토리지 제한으로 인한 실질적인 제약입니다. qubits 다음과 같이 생각할 수 있습니다. qubit a를 추가할 때마다 문제가 두 배로 어려워집니다. SV1

여러 종류의 문제에서 그래프의 구조를 활용하기 SV1 때문에 TN1 실제 시간에 훨씬 큰 회로를 평가할 TN1 수 있습니다. 기본적으로 솔루션의 발전 과정을 처음부터 추적하고 효율적인 탐색에 기여하는 구성만 보존합니다. 다시 말해, 구성을 저장하여 행렬 곱셈의 순서를 정하면 평가 프로세스가 더 간단해집니다.

TN1AND 게이트의 qubits 개수도 중요하지만 그래프의 구조가 훨씬 더 중요하기 때문입니다. 예를 들어, TN1 는 게이트가 근거리 (즉, 가장 가까운 이웃에만 게이트로 연결됨) 인 회로 (그래프) 와 연결 (또는 게이트qubits) 의 범위가 비슷한 회로 (그래프) 를 평가하는 데 매우 유용합니다. qubit 일반적인 TN1 범위는 각 사람이 5분 qubits 거리에 있는 다른 사람과만 qubit qubits 통신하는 것입니다. 대부분의 구조를 더 많거나 더 작거나 더 균일한 행렬로 표현할 수 있는 이러한 단순한 관계로 분해할 수 있는 경우 평가를 쉽게 TN1 수행합니다.

의 한계 TN1

TN1그래프의 구조적 복잡성에 SV1 의존하는 것보다 느릴 수 있습니다. 특정 그래프의 경우 다음 두 가지 이유 중 하나로 인해 리허설 단계 후에 시뮬레이션을 TN1 종료하고 상태를 표시합니다. `FAILED`

- 경로를 찾을 수 없음 — 그래프가 너무 복잡하면 적절한 순회 경로를 찾기가 너무 어려워 시뮬레이터가 계산을 포기합니다. TN1축소를 수행할 수 없습니다. 다음과 비슷한 오류 메시지가 표시될 수 있습니다. `No viable contraction path found.`
- 수축 단계가 너무 어려움 — 일부 그래프에서는 순회 경로를 찾을 TN1 수 있지만 평가하는 데 매우 길고 시간이 많이 걸립니다. 이 경우 수축 비용이 너무 많이 들기 때문에 비용이 너무 많이 들기 때문에 대신 리허설 단계 이후에 종료됩니다. TN1 다음과 비슷한 오류 메시지가 표시될 수 있습니다. `Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> 수축을 하지 않고 상태가 표시되는 TN1 경우에도 리허설 단계에 대한 요금이 청구됩니다. `FAILED`

예상 런타임도 개수에 따라 달라집니다. shot 최악의 시나리오에서는 TN1 수축 시간이 카운트에 따라 선형적으로 달라집니다. shot 회로는 더 적은 수로 축소할 수 있습니다. shots 예를 들어, 양자 과제를 shots 100으로 제출하면 계약할 수 없는 것으로 TN1 결정되지만 10개만 가지고 다시 제출하면 축소가 진행됩니다. 이 상황에서는 100개의 샘플을 얻기 위해 동일한 회로에 shots 대해 10개로 구성된 양자 과제 10개를 제출하고 그 결과를 종합하면 됩니다.

가장 좋은 방법은 더 많은 수로 진행하기 전에 항상 회로 또는 회로 클래스를 몇 개 shots (예: 10개) 로 테스트하여 회로의 난이도를 shots 확인하는 것이 좋습니다. TN1

> **Note**
>
> 수축 단계를 형성하는 일련의 곱셈은 작은 NxN 행렬에서 시작됩니다. 예를 들어 2-qubit 게이트에는 4x4 행렬이 필요합니다. 수축 중에 필요한 중간 행렬이 너무 어렵다고 판단되면 크기가 매우 큽니다. 이러한 계산을 완료하려면 며칠이 걸릴 것입니다. 이것이 Amazon 브라켓이 매우 복잡한 수축을 시도하지 않는 이유입니다.

동시성

모든 Braket 시뮬레이터는 여러 회로를 동시에 실행할 수 있는 기능을 제공합니다. 동시성 한도는 시뮬레이터 및 지역에 따라 다릅니다. 동시성 한도에 대한 자세한 내용은 할당량 페이지를 참조하십시오.

아마존 브라켓 리전 및 엔드포인트

Amazon Braket은 다음에서 사용할 수 있습니다. AWS 리전

Amazon Braket의 지역 가용성

리전 이름	지역	브라켓 엔드포인트	QPU
미국 동부(버지니아 북부)	us-east-1	braket.us-east-1.amazonaws.com	아이온큐
미국 동부(버지니아 북부)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
미국 서부(캘리포니아 북부)	us-west-1	braket.us-west-1.amazonaws.com	리게티
EU 북부 1 (스톡홀름)	eu-north-1	braket.eu-north-1.amazonaws.com	IQM
유럽 서부 2 (런던)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

Amazon Braket을 사용할 수 있는 모든 지역에서 실행할 수 있지만 각 QPU는 단일 지역에서만 사용할 수 있습니다. QPU 디바이스에서 실행되는 Quantum 작업은 해당 디바이스 지역의 Amazon Braket 콘솔에서 볼 수 있습니다. Amazon Braket SDK를 사용하는 경우, 작업 중인 지역과 상관없이 모든 QPU 디바이스에 양자 작업을 제출할 수 있습니다. SDK는 지정된 QPU에 대해 지역에 대한 세션을 자동으로 생성합니다.

지역 및 엔드포인트의 AWS 작동 방식에 대한 일반적인 정보는 일반 참조의 AWS 서비스 엔드포인트를 참조하십시오.AWS

퀀텀 태스크는 언제 실행되나요?

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

회로를 제출하면 Amazon Braket이 지정한 디바이스로 회로를 전송합니다. 양자 처리 장치 (QPU) 및 온디맨드 시뮬레이터 양자 작업은 수신된 순서대로 대기하고 처리됩니다. 양자 작업을 제출한 후 처리하는 데 필요한 시간은 다른 Amazon Braket 고객이 제출한 작업의 수와 복잡성, 선택한 QPU의 가용성에 따라 달라집니다.

이메일 또는 SMS를 통한 상태 변경 알림

Amazon Braket은 QPU의 가용성이 변경되거나 양자 작업의 상태가 변경될 때 EventBridge Amazon에 이벤트를 전송합니다. 이메일 또는 SMS 메시지로 디바이스 및 양자 작업 상태 변경 알림을 받으려면 다음 단계를 따르십시오.

1. Amazon SNS 주제를 생성하고 이메일 또는 SMS를 구독하십시오. 이메일 또는 SMS의 사용 가능 여부는 지역에 따라 다릅니다. 자세한 내용은 Amazon SNS 시작하기 및 SMS 메시지 전송을 참조하십시오.
2. SNS 주제에 대한 알림을 EventBridge 트리거하는 규칙을 생성하십시오. 자세한 내용은 Amazon을 통한 Amazon Braket 모니터링을 참조하십시오. EventBridge

퀀텀 작업 완료 알림

Amazon Simple Notification Service (SNS) 를 통해 알림을 설정하여 Amazon Braket 퀀텀 작업이 완료될 때 알림을 받을 수 있습니다. 활성 알림은 대기 시간이 길어질 것으로 예상되는 경우 (예: 대규모 작업을 제출하거나 디바이스의 가용성 기간이 지난 후에 작업을 제출할 때) 유용합니다. 작업이 완료될 때까지 기다리지 않으려면 SNS 알림을 설정할 수 있습니다.

Amazon Braket 노트북은 설정 단계를 안내합니다. 자세한 내용은 알림 설정을 위한 Amazon Braket 예제 노트북을 참조하십시오.

QPU 가용성 기간 및 상태

QPU 가용성은 장치마다 다릅니다.

Amazon Braket 콘솔의 디바이스 페이지에서 현재 및 향후 가용성 창과 디바이스 상태를 확인할 수 있습니다. 또한 각 디바이스 페이지에는 양자 작업 및 하이브리드 작업에 대한 개별 대기열 깊이가 표시됩니다.

고객이 사용할 수 없는 디바이스는 가용성 기간에 관계없이 오프라인 상태로 간주됩니다. 예를 들어 예정된 유지 관리, 업그레이드 또는 운영 문제로 인해 오프라인 상태일 수 있습니다.

대기열 가시성

양자 작업 또는 하이브리드 작업을 제출하기 전에 장치 대기열 깊이를 확인하여 앞에 있는 양자 작업 또는 하이브리드 작업의 수를 확인할 수 있습니다.

대기열 길이

Queue depth특정 디바이스에 대해 대기중인 양자 작업 및 하이브리드 작업 수를 나타냅니다. 장치의 양자 작업 및 하이브리드 작업 대기열 수는 OR를 통해 액세스할 수 있습니다. Braket Software Development Kit (SDK) Amazon Braket Management Console

1. 작업 대기열 깊이는 현재 정상 우선 순위로 실행 대기 중인 양자 작업의 총 수를 나타냅니다.
2. 우선순위 태스크 대기열 깊이는 제출된 양자 태스크가 실행되기를 기다리는 총 개수를 의미합니다 Amazon Braket Hybrid Jobs. 이러한 작업은 독립형 작업보다 먼저 실행됩니다.
3. 하이브리드 작업 대기열 깊이는 현재 장치에 대기하고 있는 하이브리드 작업의 총 수를 나타냅니다. Quantum tasks하이브리드 작업의 일부로 제출된 항목은 우선 순위를 가지며 에 집계됩니다. Priority Task Queue

를 통해 대기열 길이를 확인하려는 고객은 다음 코드 스니펫을 수정하여 퀀텀 태스크 또는 하이브리드 작업의 대기열 위치를 확인할 Braket SDK 수 있습니다.

```
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'
```

양자 작업 또는 하이브리드 작업을 QPU에 제출하면 워크로드가 정상 상태가 될 수 있습니다. QUEUED Amazon Braket은 고객에게 양자 작업 및 하이브리드 작업 대기열 위치에 대한 가시성을 제공합니다.

대기열 위치

Queue position각 디바이스 대기열 내 퀀텀 태스크 또는 하이브리드 작업의 현재 위치를 나타냅니다. 양자 작업 또는 하이브리드 작업의 경우 OR를 통해 확인할 수 Amazon Braket Management Console 있습니다. Braket Software Development Kit (SDK)

를 통해 대기열 위치를 확인하려는 고객은 다음 코드 스니펫을 수정하여 양자 작업 또는 하이브리드 작업의 대기열 위치를 확인할 Braket SDK 수 있습니다.

```
# 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
```

아마존 브라켓으로 시작하세요

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배워보세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket 활성화의 지침을 따른 후 Braket을 Amazon 시작할 수 있습니다.

시작하기 위한 단계는 다음과 같습니다.

- 아마존 브라켓 활성화
- 아마존 브라켓 노트북 인스턴스 생성
- Amazon Braket Python SDK를 사용하여 첫 번째 회로를 실행하십시오
- 첫 번째 양자 알고리즘을 실행해 보세요.

아마존 브라켓 활성화

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배우세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

콘솔을 통해 계정에서 Amazon Braket을 활성화할 수 있습니다.AWS

사전 조건

Amazon Braket을 활성화하고 실행하려면 Amazon Braket 작업을 시작할 권한이 있는 사용자 또는 역할이 있어야 합니다. 이러한 권한은 AmazonBraketFullAccess IAM 정책 (arn:aws:iam: :aws:policy/) 에 포함되어 있습니다. AmazonBraket FullAccess

> **Note**
>
> 관리자인 경우:

다른 사용자에게 Amazon Braket에 대한 액세스 권한을 부여하려면 정책을 연결하거나 직접 생성한 사용자 지정 AmazonBraketFullAccess정책을 첨부하여 사용자에게 권한을 부여하십시오. Amazon Braket을 사용하는 데 필요한 권한에 대해 자세히 알아보려면 Amazon Braket에 대한 액세스 관리를 참조하십시오.

Amazon Braket을 활성화하는 단계

1. 사용자 계정을 사용하여 Amazon Braket 콘솔에 로그인합니다. AWS 계정
2. 아마존 브라켓 콘솔을 엽니다.
3. Braket 랜딩 페이지에서 시작하기를 클릭하면 서비스 대시보드 페이지로 이동합니다. 서비스 대시보드 상단의 알림은 다음 세 단계를 안내합니다.
   a. 서비스 연결 역할 (SLR) 생성
   b. 타사 양자 컴퓨터에 대한 액세스 지원
   c. 새 Jupyter 노트북 인스턴스 생성

타사 양자 장치를 사용하려면 사용자 자신과 해당 장치 간의 데이터 전송과 관련된 특정 조건에 동의해야 합니다. AWS본 계약의 이용 약관은 Amazon Braket 콘솔의 권한 및 설정 페이지의 일반 탭에 나와 있습니다.

> **Note**
>
> Braket 로컬 시뮬레이터 또는 온디맨드 시뮬레이터와 같이 제3자가 관여하지 않는 Quantum 디바이스는 타사 디바이스 활성화 계약에 동의하지 않고도 사용할 수 있습니다.
> 타사 하드웨어에 액세스하는 경우 이 약관에 동의하여 타사 장치를 사용할 수 있도록 하려면 계정당 한 번만 수락하면 됩니다.

아마존 브라켓 노트북 인스턴스 생성

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배우세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket은 시작하는 데 도움이 되는 완전 관리형 Jupyter 노트북을 제공합니다. Amazon Braket 노트북 인스턴스는 아마존 SageMaker 노트북 인스턴스를 기반으로 합니다. 다음 지침은 신규 및 기존 고객을 위한 새 노트북 인스턴스를 생성하는 데 필요한 단계를 간략하게 설명합니다.

아마존 브라켓의 신규 고객

1. Amazon Braket 콘솔을 열고 왼쪽 창의 대시보드 페이지로 이동합니다.
2. 대시보드 페이지 중앙에 있는 Amazon Braket Welcome to Amazon Braket 모달에서 시작하기를 클릭하여 노트북 이름을 입력합니다. 그러면 기본 Jupyter 노트북이 생성됩니다.
3. 노트북을 만드는 데 몇 분 정도 걸릴 수 있습니다. 노트북은 노트북 페이지에 보류 중 상태로 나열됩니다. 노트북 인스턴스를 사용할 준비가 되면 상태가 로 InService변경됩니다. 노트북의 업데이트된 상태를 표시하려면 페이지를 새로 고쳐야 할 수 있습니다.

기존 아마존 브라켓 고객

1. Amazon Braket 콘솔을 열고 왼쪽 창에서 노트북을 선택하고 노트북 인스턴스 생성을 선택합니다. 노트북이 없는 경우 표준 설정을 선택하여 기본 Jupyter 노트북을 생성하고 영숫자와 하이픈만 사용하여 Notebook 인스턴스 이름을 입력하고 원하는 비주얼 모드를 선택합니다. 그런 다음 노트북의 비활성 관리자를 활성화하거나 비활성화합니다.
   a. 활성화된 경우 노트북을 재설정하기 전에 원하는 유휴 시간을 선택하십시오. 노트북이 재설정되면 컴퓨팅 요금은 더 이상 발생하지 않지만 스토리지 요금은 계속 청구됩니다.
   b. 노트북 인스턴스의 남은 유휴 시간을 보려면 명령 표시줄로 이동하여 Braket 탭과 비활성 관리자 탭을 차례로 선택합니다.

   > **Note**
   >
   > 작업이 손실되지 않도록 하려면 SageMaker 노트북 인스턴스를 git 리포지토리와 통합하는 것을 고려해 보세요. 대안으로, `/Braket Algorithms` 및 `/Braket Examples` 폴더 외부로 작업을 옮기면 노트북 인스턴스가 다시 시작되어 파일을 덮어쓰는 것을 방지할 수 있습니다.

2. (선택 사항) 고급 설정을 사용하면 액세스 권한, 추가 구성 및 네트워크 액세스 설정이 포함된 노트북을 만들 수 있습니다.
   a. 노트북 구성에서 인스턴스 유형을 선택합니다. 비용 효율적인 표준 인스턴스 유형인 ml.t3.medium이 기본적으로 선택됩니다. 인스턴스 요금에 대한 자세한 내용은 Amazon SageMaker 요금을 참조하십시오. 퍼블릭 Github 리포지토리를 노트북 인스턴스에 연결하려면

Git 리포지토리 드롭다운을 클릭하고 리포지토리 드롭다운 메뉴에서 url에서 퍼블릭 git 리포지토리 복제를 선택합니다. Git 리포지토리 URL 텍스트 표시줄에 리포지토리의 URL을 입력합니다.

b. 권한에서 선택적 IAM 역할, 루트 액세스 및 암호화 키를 구성합니다.

c. 네트워크에서 Jupyter Notebook 인스턴스에 대한 사용자 지정 네트워크 및 액세스 설정을 구성합니다.

3. 설정을 검토하고 노트북 인스턴스를 식별할 태그를 설정한 다음 Launch를 클릭합니다.

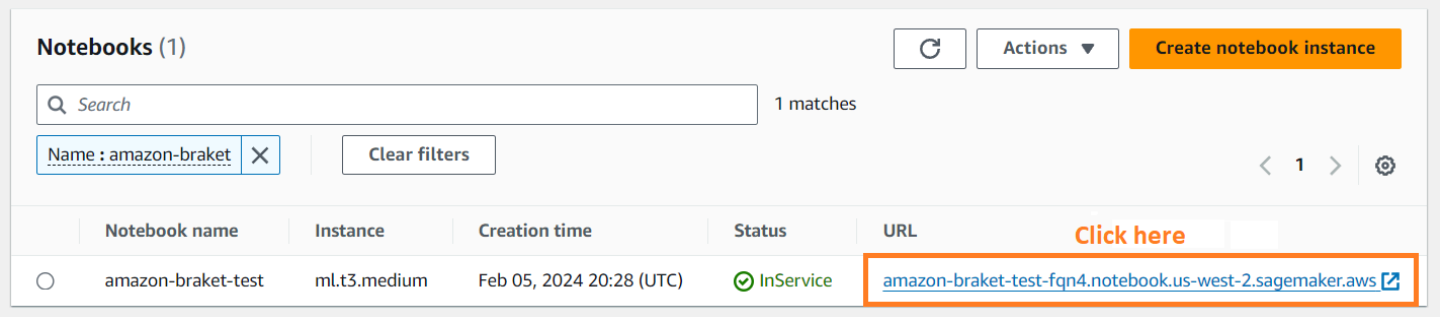
> **Note**
>
> 아마존 브라켓 및 아마존 콘솔에서 아마존 브라켓 노트북 인스턴스를 보고 관리할 수 있습니다. SageMaker 추가 Amazon Braket 노트북 설정은 콘솔을 통해 사용할 수 있습니다. SageMaker

Amazon Braket 콘솔에서 작업하는 경우 Amazon Braket SDK 내에서 AWS 작업하면 생성한 노트북에 플러그인이 미리 로드되어 있습니다. 자체 시스템에서 실행하려는 경우 명령을 실행할 때 `pip install amazon-braket-sdk` 또는 플러그인과 함께 사용하기 위해 명령을 실행할 때 SDK와 플러그인을 설치할 수 있습니다. `pip install amazon-braket-pennylane-plugin` PennyLane

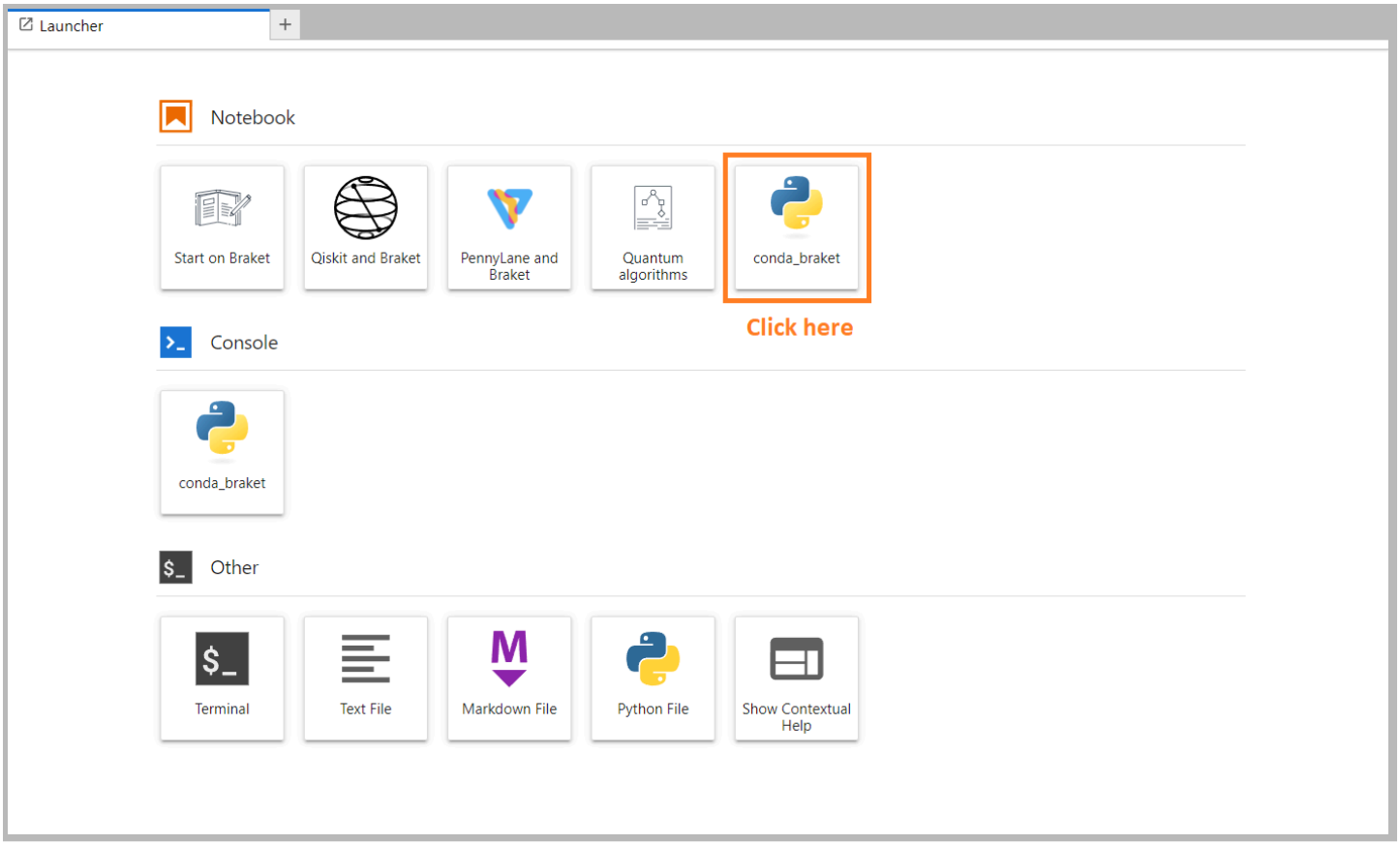
Amazon Braket Python SDK를 사용하여 첫 번째 회로를 실행하십시오

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배우세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

노트북 인스턴스가 시작된 후 방금 만든 노트북을 선택하여 표준 Jupyter 인터페이스로 인스턴스를 엽니다.



AmazonBraket 노트북 인스턴스는 Amazon Braket SDK 및 모든 종속 항목과 함께 사전 설치되어 있습니다. 커널을 사용하여 새 노트북을 만드는 것부터 시작하세요. `conda_braket`



“Hello, world!” 라는 간단한 말로 시작할 수 있습니다. 예시. 먼저 벨 상태를 준비하는 회로를 구성한 다음 이 회로를 여러 장치에서 실행하여 결과를 얻습니다.

먼저 Amazon Braket SDK 모듈을 가져와서 간단한 Bell State 회로를 정의하십시오.

```
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)
```

다음 명령을 사용하여 회로를 시각화할 수 있습니다.

```
print(bell)
```

로컬 시뮬레이터에서 회로를 실행하세요.

다음으로 회로를 실행할 양자 장치를 선택합니다. AmazonBraket SDK에는 신속한 프로토타이핑 및 테스트를 위한 로컬 시뮬레이터가 함께 제공됩니다. 최대 25개 qubits (로컬 하드웨어에 따라 다름) 의 소형 회로에는 로컬 시뮬레이터를 사용하는 것이 좋습니다.

로컬 시뮬레이터를 인스턴스화하는 방법은 다음과 같습니다.

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

그리고 회로를 실행하세요:

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

다음과 같은 결과가 표시될 것입니다.

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

준비한 구체적인 벨 상태는 |00>과 |11>의 동일한 중첩이며, 예상대로 측정 결과와 거의 동일한 (최대 shot 노이즈까지) 분포를 찾을 수 있습니다.

온디맨드 시뮬레이터에서 회로를 실행해 보세요.

Amazon또한 Braket은 대형 회로를 실행하기 위한 온디맨드 고성능 시뮬레이터에 대한 액세스를 제공합니다. SV1 SV1최대 34개의 양자 회로를 시뮬레이션할 수 있는 온디맨드 상태 벡터 시뮬레이터입니다. qubits 지원 장치 섹션 및 SV1 콘솔에서 자세한 내용을 확인할 수 있습니다. AWS SV1(TN1또는 모든 QPU에서) 양자 작업을 실행할 때 양자 작업의 결과는 계정의 S3 버킷에 저장됩니다. 버킷을 지정하

지 않으면 Braket SDK가 기본 버킷을 amazon-braket-{region}-{accountID} 자동으로 생성합니다. 자세한 내용은 Amazon Braket에 대한 액세스 관리를 참조하십시오.

> **Note**
>
> 다음 예시가 버킷 example-bucket 이름으로 표시된 실제 기존 버킷 이름을 입력합니다. AmazonBraket의 버킷 이름은 항상 사용자가 추가하는 다른 식별 문자로 시작하고 amazon-braket- 그 뒤에 추가됩니다. S3 버킷을 설정하는 방법에 대한 정보가 필요한 경우 Amazon S3 시작하기를 참조하십시오.

```
# 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)
```

회로를 실행하려면 이전에 .run() 호출에서 SV1 위치 인수로 선택한 S3 버킷의 위치를 제공해야 합니다.

```
# 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)
```

AmazonBraket 콘솔은 양자 작업에 대한 추가 정보를 제공합니다. 콘솔에서 Quantum Tasks 탭으로 이동하면 양자 태스크가 목록 맨 위에 표시됩니다. 또는 고유한 양자 작업 ID 또는 기타 기준을 사용하여 양자 작업을 검색할 수 있습니다.

> **Note**
>
> 90일이 지나면 Amazon Braket은 양자 작업과 관련된 모든 양자 작업 ID 및 기타 메타데이터를 자동으로 제거합니다. 자세한 내용은 데이터 보존을 참조하십시오.

QPU에서 실행

Amazon Braket을 사용하면 코드 한 줄만 변경하여 물리적 양자 컴퓨터에서 이전 양자 회로 예제를 실행할 수 있습니다. Amazon Braket은IonQ, Oxford Quantum CircuitsQuEra, 및 에서 QPU 디바이스에 대한 액세스를 제공합니다. Rigetti 지원되는 디바이스 섹션과 AWS 콘솔의 디바이스 탭에서 다양한 디바이스 및 가용성 창에 대한 정보를 찾을 수 있습니다. 다음 예제는 장치를 인스턴스화하는 방법을 보여줍니다. Rigetti

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

이 코드가 있는 IonQ 기기를 선택하세요.

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

디바이스를 선택한 후 워크로드를 실행하기 전에 다음 코드를 사용하여 디바이스 대기열 길이를 쿼리하여 양자 작업 또는 하이브리드 작업의 수를 확인할 수 있습니다. 또한 고객은 의 디바이스 페이지에서 디바이스별 대기열 깊이를 볼 수 있습니다. 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
```

작업을 실행하면 Amazon Braket SDK가 결과를 폴링합니다 (기본 제한 시간은 5일). 다음 예와 같이 `.run()` 명령의 `poll_timeout_seconds` 파라미터를 수정하여 이 기본값을 변경할 수 있습니다. 폴링 제한 시간이 너무 짧으면 QPU를 사용할 수 없고 로컬 시간 초과 오류가 반환되는 경우와 같이 폴링 시간 내에 결과가 반환되지 않을 수 있다는 점에 유의하세요. 함수를 호출하여 폴링을 다시 시작할 수 있습니다. `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)
```

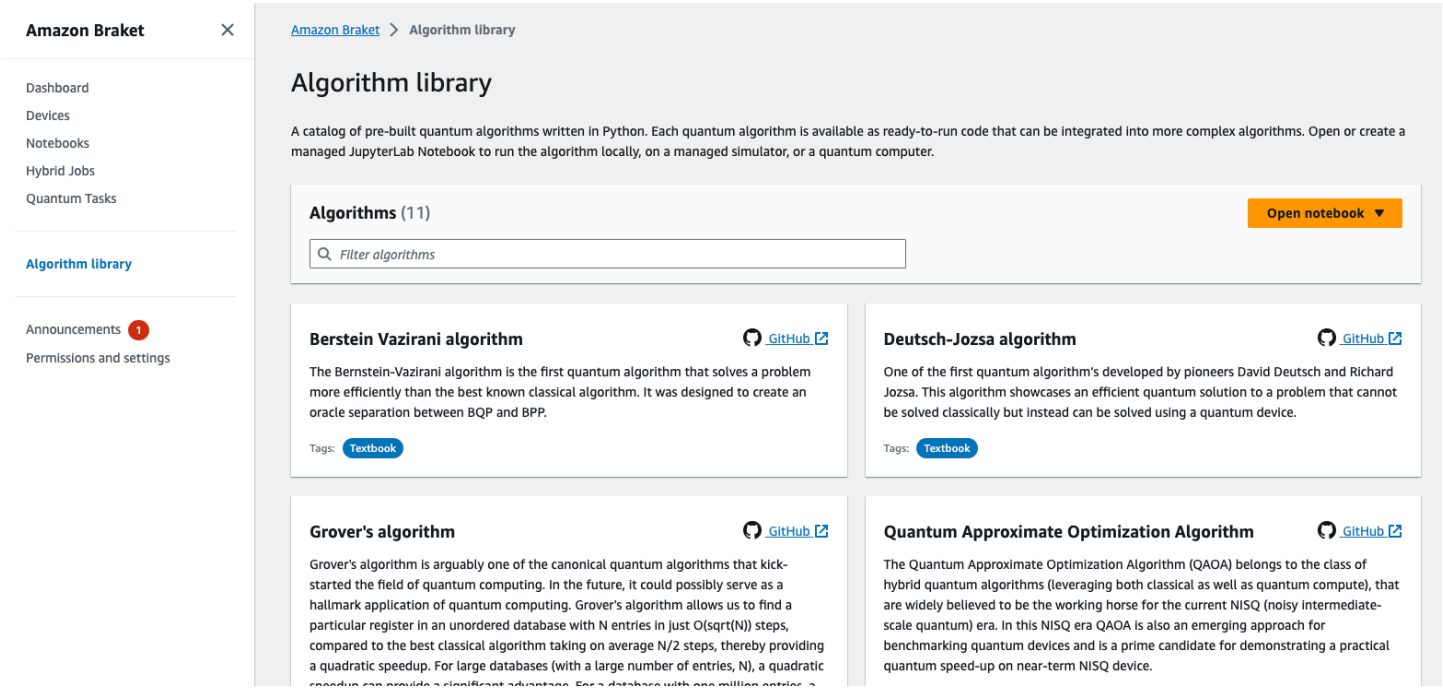
또한 퀀텀 태스크 또는 하이브리드 작업을 제출한 후 `queue_position()` 함수를 호출하여 대기열 위치를 확인할 수 있습니다.

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

첫 번째 양자 알고리즘을 실행해 보세요.

> **Tip**
>
> 를 통해 양자 컴퓨팅의 기초를 배우세요! AWS Amazon Braket 디지털 학습 플랜에 등록하고 일련의 학습 과정과 디지털 평가를 완료한 후 디지털 배지를 획득하십시오.

Amazon Braket 알고리즘 라이브러리는 Python으로 작성된 사전 구축된 양자 알고리즘의 카탈로그입니다. 이러한 알고리즘을 그대로 실행하거나 더 복잡한 알고리즘을 빌드하기 위한 출발점으로 사용할 수 있습니다. Braket 콘솔에서 알고리즘 라이브러리에 액세스할 수 있습니다. 깃허브 (Github) 의 브라켓 알고리즘 라이브러리 (https://github.com/aws-samples/amazon-braket-algorithm-library) 에도 액세스할 수 있습니다.



Braket 콘솔은 알고리즘 라이브러리에서 사용 가능한 각 알고리즘에 대한 설명을 제공합니다. GitHub 링크를 선택하여 각 알고리즘의 세부 정보를 보거나 노트북 열기를 선택하여 사용 가능한 모든 알고리즘이 포함된 노트북을 열거나 만들 수 있습니다. 노트북 옵션을 선택하면 노트북의 루트 폴더에서 Braket 알고리즘 라이브러리를 찾을 수 있습니다.

아마존 브라켓과 함께 작업하세요

이 섹션에서는 Amazon Braket SDK를 사용하여 양자 회로를 설계하고, 이러한 문제를 양자 작업으로 디바이스에 제출하고, 양자 작업을 모니터링하는 방법을 보여줍니다.

Braket의 리소스와 상호 작용하는 주요 방법은 다음과 같습니다. Amazon

- Amazon Braket 콘솔은 리소스 및 양자 작업을 생성, 관리 및 모니터링하는 데 도움이 되는 디바이스 정보와 상태를 제공합니다.
- Amazon Braket Python SDK와 콘솔을 통해 양자 작업을 제출하고 실행할 수 있습니다. SDK는 사전 Amazon 구성된 Braket 노트북을 통해 액세스할 수 있습니다.
- 아마존 브라켓 API는 Amazon 브라켓 파이썬 SDK와 노트북을 통해 액세스할 수 있습니다. 양자 컴퓨팅을 프로그래밍 방식으로 사용하는 애플리케이션을 구축하는 API 경우 를 직접 호출할 수 있습니다.

이 섹션의 예제는 브라켓용 Python SDK (Boto3) 와 함께 AmazonAmazon 브라켓 Python SDK를 사용하여 브라켓을 API 직접 사용하는 방법을 보여줍니다.AWS

Amazon브라켓 파이썬 SDK에 대한 자세한 정보

Amazon브라켓 Python SDK를 사용하려면 먼저 브라켓용 AWS Python SDK (Boto3) 를 설치하여 와 통신할 수 있도록 하십시오. AWS API AmazonBraket Python SDK는 퀀텀 고객을 위한 Boto3에 대한 편리한 래퍼라고 생각할 수 있습니다.

- Boto3에는 활용해야 하는 인터페이스가 포함되어 있습니다. AWS API (참고로 Boto3는 와 통신하는 대형 Python SDK입니다. AWS API 대부분은 AWS 서비스 Boto3 인터페이스를 지원합니다.
- AmazonBraket Python SDK에는 회로, 게이트, 장치, 결과 유형 및 양자 작업의 기타 부분을 위한 소프트웨어 모듈이 포함되어 있습니다. 프로그램을 생성할 때마다 해당 양자 작업에 필요한 모듈을 가져옵니다.
- AmazonBraket Python SDK는 양자 작업을 실행하는 데 필요한 모든 모듈과 종속성이 사전 로드된 노트북을 통해 액세스할 수 있습니다.
- 노트북으로 작업하고 싶지 않은 경우 Amazon Braket Python SDK에서 모든 Python 스크립트로 모듈을 가져올 수 있습니다.

Boto3를 설치한 후 Amazon Braket Python SDK를 통해 양자 작업을 생성하는 단계에 대한 개요는 다음과 같습니다.

1. (선택 사항) 노트북을 엽니다.
2. 회로에 필요한 SDK 모듈을 가져오세요.
3. QPU 또는 시뮬레이터를 지정하십시오.
4. 회로를 인스턴스화합니다.
5. 회로를 실행합니다.
6. 결과를 수집하세요.

이 섹션의 예는 각 단계의 세부 정보를 보여줍니다.

더 많은 예제를 보려면 의 Amazon Braket 예제 리포지토리를 참조하십시오. GitHub

이 섹션:

- 안녕하세요 AHS: 첫 아날로그 해밀턴 시뮬레이션을 실행해 보세요.
- SDK에서 회로를 구성하세요.
- QPU 및 시뮬레이터에 양자 작업 제출
- OpenQASM 3.0으로 회로를 실행하십시오.
- QuEra's Aquila를 사용하여 아날로그 프로그램을 제출하십시오.
- Boto3로 작업하기

안녕하세요 AHS: 첫 아날로그 해밀턴 시뮬레이션을 실행해 보세요.

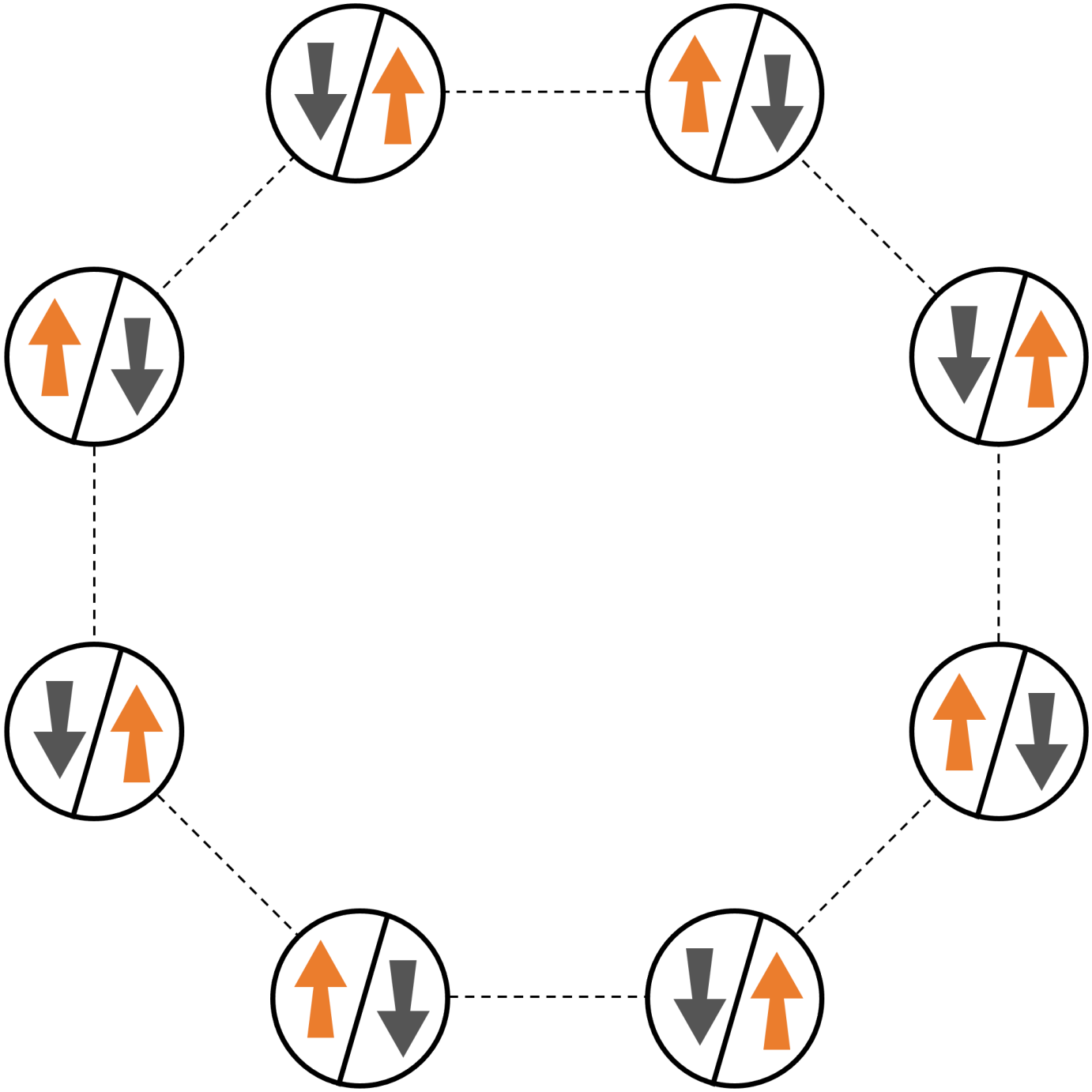
AHS

아날로그 해밀턴 시뮬레이션 (AHS) 은 양자 회로와는 다른 양자 컴퓨팅의 패러다임입니다. 각 게이트가 한 번에 몇 큐비트에서만 작동하는 일련의 게이트 대신 해당 해밀턴의 시간 및 공간 종속 매개변수로 AHS 프로그램을 정의합니다. 시스템의 해밀턴 방식은 시스템의 에너지 준위와 외부 힘의 영향을 암호화하며, 외부 힘은 모두 해당 상태의 시간 변화를 제어합니다. N-큐비트 시스템의 경우 해밀턴은 복소수로 구성된 2N X2 N 정사각형 행렬로 표현할 수 있습니다.

AHS를 수행할 수 있는 양자 장치는 매개 변수 (예: 일관된 구동장의 진폭 및 디튜닝) 를 조정하여 사용자 지정 해밀턴식 양자 시스템의 시간 변화를 근사화합니다. AHS 패러다임은 상호 작용하는 여러 입자의 양자 시스템의 정적 및 동적 특성을 시뮬레이션하는 데 적합합니다. 의 Aquila 디바이스와 같이 특수 목적으로 설계된 QPU는 기존 하드웨어로는 구현할 QuEra 수 없는 크기의 시스템의 시간 변화를 시뮬레이션할 수 있습니다.

상호 작용하는 스핀 체인

상호 작용하는 여러 입자로 구성된 시스템의 표준 예를 들어 8개의 스핀으로 이루어진 고리 (각각 “업” ↑‡와 “down” ↓† 상태일 수 있음) 를 생각해 보겠습니다. 비록 작지만 이 모델 시스템은 이미 자연 발생 자성 물질의 흥미로운 현상을 몇 가지 보여주고 있습니다. 이 예제에서는 연속적인 스핀이 반대 방향을 가리키는 이른바 반강자성 오더를 만드는 방법을 보여줍니다.



배열

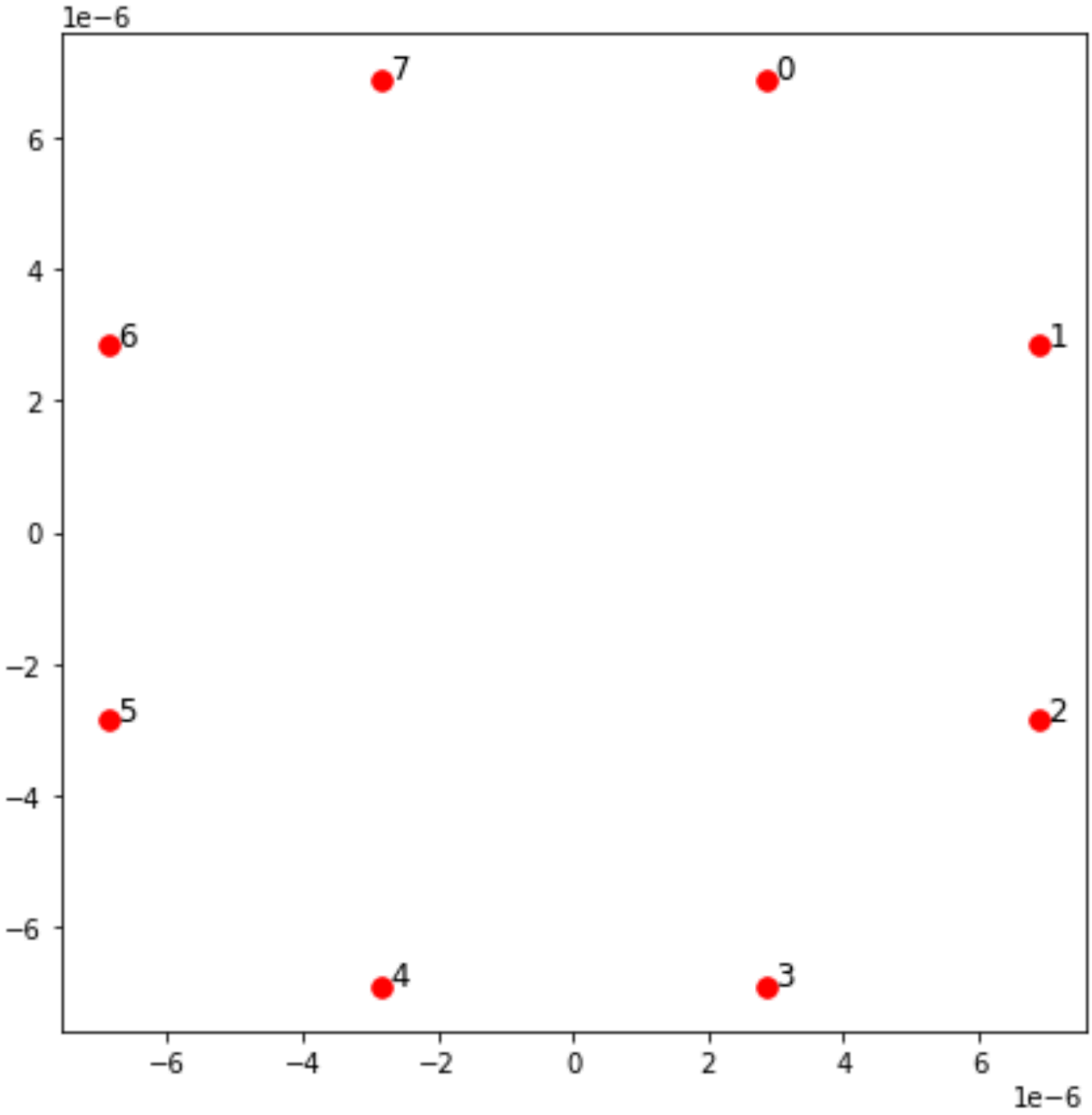
회전할 때마다 하나의 중성 원자를 사용할 것이고, “업” 및 “다운” 스핀 상태는 각각 원자의 들뜬 리드버그 상태와 그라운드 상태로 인코딩될 것입니다. 먼저 2차원 배열을 만듭니다. 다음 코드를 사용하여 위의 스핀 링을 프로그래밍할 수 있습니다.

전제 조건: Braket SDK를 pip 설치해야 합니다. (Braket 호스팅 노트북 인스턴스를 사용하는 경우 이 SDK는 노트북과 함께 사전 설치되어 제공됩니다.) 플롯을 재현하려면 셸 명령을 사용하여 matplotlib를 별도로 설치해야 합니다. `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)
```

다음과 같이 플롯할 수도 있습니다.

```
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
```



상호 작용

반강자성 단계를 준비하려면 인접 스핀 간의 상호 작용을 유도해야 합니다. 이를 위해 반 데르 발스 상호작용을 사용하는데, 이는 기본적으로 중성 원자 장치 (예: 의 장치) 에 의해 구현됩니다. Aquila

QuEra 스핀 표현을 사용하면 이 상호작용에 대한 해밀턴 항을 모든 스핀 쌍 (j, k) 에 대한 합으로 표현할 수 있습니다.

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

여기서 nj=↑ ‡↑ j 는 스핀 j가 “up” 상태인 경우에만 값 1을 취하고, 그렇지 않으면 0을 취하는 연산자입니다. j 강도는 V j,k =C6/(dj,k) 6입니다. 여기서 C는 고정 계수이고 6 d는 스핀 j와 k 사이의 유클리드 j,k 거리입니다. 이 교호작용 항의 즉각적인 효과는 스핀 j와 스핀 k가 모두 “업”인 모든 상태에서는 에너지가 (V만큼) 상승한다는 것입니다. j,k AHS 프로그램의 나머지 부분을 신중하게 설계함으로써 이 상호 작용은 인접한 스핀이 모두 “업” 상태에 있는 것을 방지할 수 있으며, 이러한 효과를 일반적으로 “리드버그 차단”이라고 합니다.

드라이빙 필드

AHS 프로그램이 시작될 때 모든 스핀은 (기본적으로) “다운” 상태에서 시작되며, 소위 강자성 단계에 있습니다. 반강자성 위상을 준비한다는 목표를 염두에 두고 스핀을 이 상태에서 “상승” 상태가 선호되는 다물체 상태로 부드럽게 전환하는 시간 종속 일관된 구동 필드를 지정합니다. 해당하는 해밀턴식은 다음과 같이 쓸 수 있습니다.

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

여기서 Ω (t), (t), Δ (t) 는 모든 스핀에 균일하게 영향을 미치는 구동 필드의 시간 종속 글로벌 진폭 (일명 Rabi 주파수), 위상 및 디튜닝입니다. 여기서 S −,k =↓ k §↑ k 와 S +,k =(S−,k) † =↑ k ‡↓ ↓ k 는 각각 스핀 k의 하강 연산자와 상승 연산자이고 n k =↑ k k ‡↑는 이전과 동일한 연산자입니다. 주행장의 Ω 부분은 모든 스핀의 “다운” 상태와 “업” 상태를 동시에 일관되게 연결하는 반면 Δ 부분은 “업” 상태에 대한 에너지 보상을 제어합니다.

강자성 위상에서 반강자성 위상으로의 원활한 전환을 프로그래밍하기 위해 다음 코드를 사용하여 구동 필드를 지정합니다.

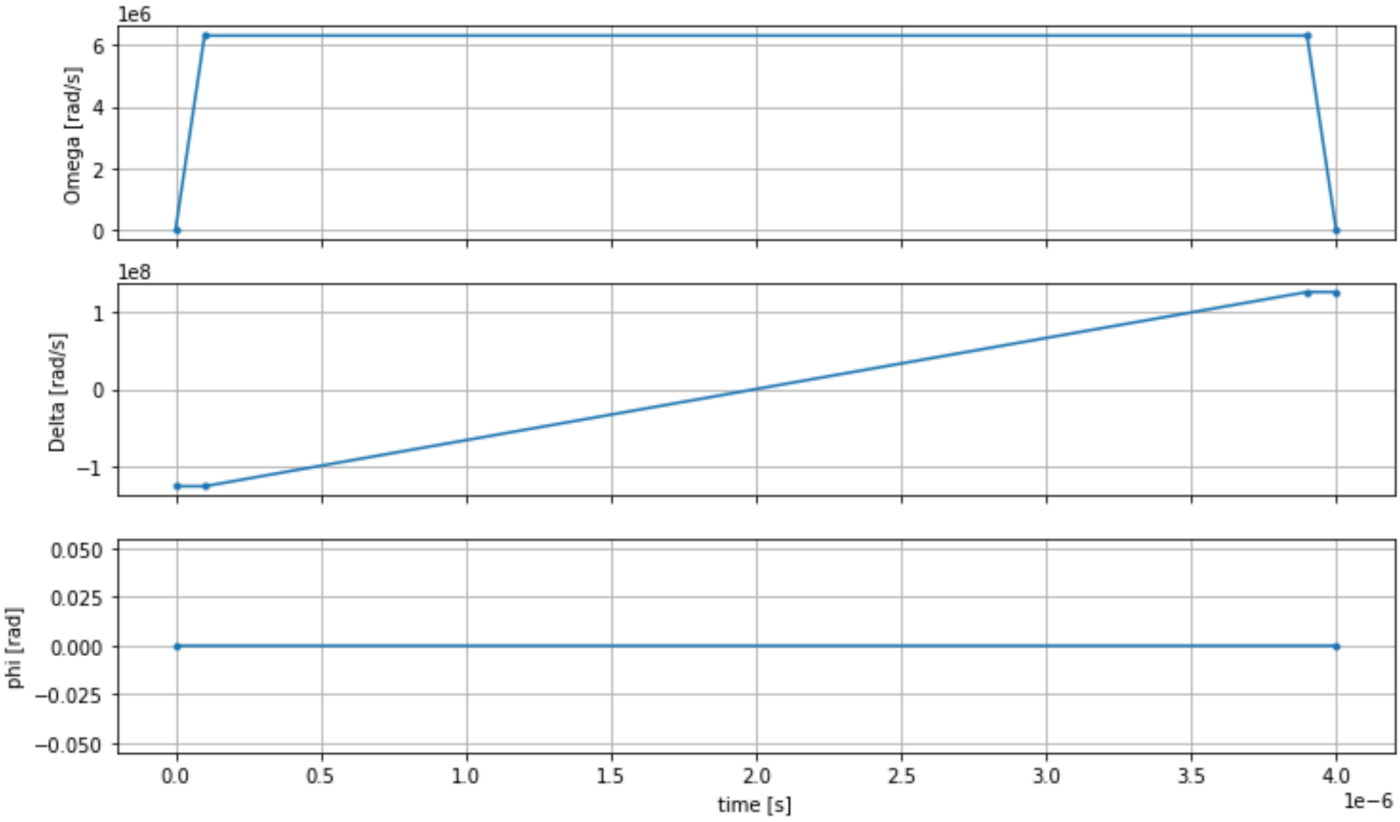
```
from braket.timings.time_series import TimeSeries
from braket.ahs.driving_field import DrivingField

# 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
)
```

다음 스크립트를 사용하여 구동장의 시계열을 시각화할 수 있습니다.

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

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

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

ax = axes[2]
time_series = drive.phase.time_series
# Note: time series of phase is understood as a piecewise constant function
ax.step(time_series.times(), time_series.values(), '.-', where='post');
```

```
ax.set_ylabel('phi [rad]')
ax.grid()
ax.set_xlabel('time [s]')

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



AHS 프로그램

레지스터, 드라이빙 필드 (및 암시적 반 데르 발스 상호작용) 는 아날로그 해밀턴 시뮬레이션 프로그램을 구성합니다. `ahs_program`

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

로컬 시뮬레이터에서 실행

이 예제는 크기가 작기 때문에 (15회전 미만) AHS 호환 QPU에서 실행하기 전에 Braket SDK와 함께 제공되는 로컬 AHS 시뮬레이터에서 실행할 수 있습니다. 로컬 시뮬레이터는 Braket SDK와 함께 무료로 제공되므로 코드를 올바르게 실행할 수 있도록 하는 것이 가장 좋습니다.

여기서는 샷 수를 높은 값 (예: 100만 개) 으로 설정할 수 있습니다. 로컬 시뮬레이터가 양자 상태의 시간 변화를 추적하고 최종 상태에서 샘플을 추출하기 때문에 샷 수는 늘리고 총 런타임은 약간만 늘리기 때문입니다.

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

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

시뮬레이터 결과 분석

다음 함수를 사용하여 각 스핀의 상태 (“다운”의 경우 “d”, “업”의 경우 “u”, 빈 사이트의 경우 “e”일 수 있음) 를 유추하고 전체 샷에서 각 구성이 발생한 횟수를 계산하는 다음 함수를 사용하여 샷 결과를 집계할 수 있습니다.

```
from collections import Counter

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

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

    Args:
       result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

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

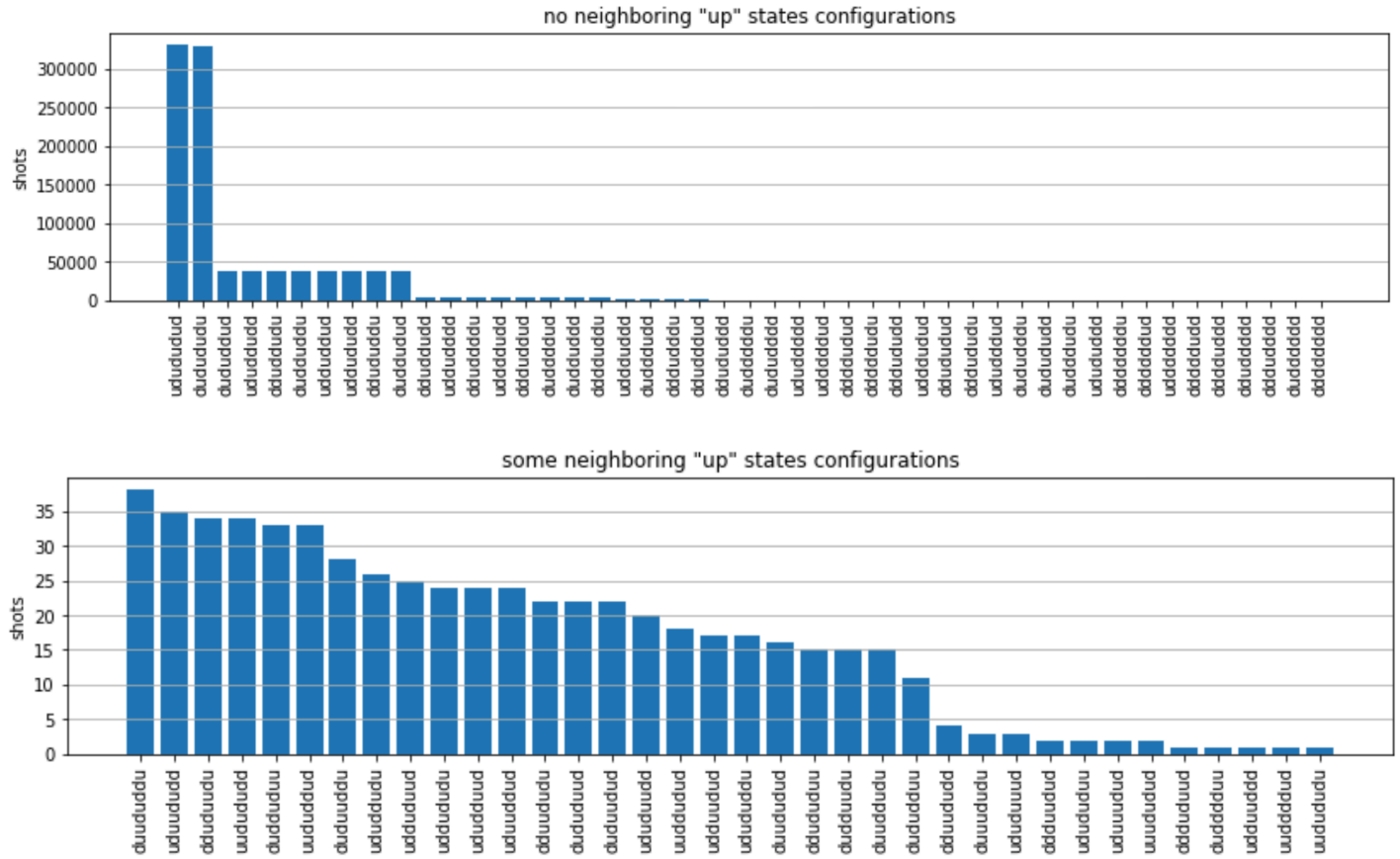
```
    """
    state_counts = Counter()
    states = ['e', 'u', 'd']
    for shot in result.measurements:
        pre = shot.pre_sequence
        post = shot.post_sequence
        state_idx = np.array(pre) * (1 + np.array(post))
        state = "".join(map(lambda s_idx: states[s_idx], state_idx))
 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

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

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

counts다음은 샷 전체에서 각 상태 구성이 관찰된 횟수를 세는 사전입니다. 다음 코드를 사용하여 시각화할 수도 있습니다.

```
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)
```



도표를 통해 반강자성 위상을 성공적으로 준비했는지 확인할 수 있는 다음 관찰 결과를 확인할 수 있습니다.

1. 일반적으로, 적어도 한 쌍의 인접 스핀이 모두 “업” 상태인 상태보다 차단되지 않은 상태 (인접 스핀이 “업” 상태에 있지 않은 상태) 가 더 일반적입니다.
2. 일반적으로 구성이 차단되지 않는 한 “상승” 자극이 더 많은 상태가 선호됩니다.
3. 가장 일반적인 상태는 완벽한 반강자성 상태와 입니다. "dudududu" "udududud"
4. 두 번째로 흔한 상태는 1, 2, 2의 연속적인 분리가 있는 “상승” 자극이 3회밖에 없는 상태입니다. 이는 반 데르 발스 교호작용이 그 다음으로 가장 가까운 이웃 국가에도 (비록 작긴 하지만) 영향을 미친다는 것을 보여줍니다.

러닝 온의 Aquila QPU QuEra

전제 조건: Braket SDK를 pip로 설치하는 것 외에도 Amazon Braket을 처음 사용하는 경우 필요한 시작 단계를 완료했는지 확인하십시오.

> **Note**
>
> Braket 호스팅 노트북 인스턴스를 사용하는 경우 Braket SDK가 인스턴스와 함께 사전 설치되어 제공됩니다.

모든 종속성이 설치되었으면 QPU에 연결할 수 있습니다. Aquila

```
from braket.aws import AwsDevice

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

AHS 프로그램을 QuEra 기계에 적합하게 만들려면 QPU에서 허용하는 정밀도 수준을 준수하도록 모든 값을 반올림해야 합니다Aquila. (이러한 요구 사항은 이름에 “해상도”가 포함된 장치 매개변수에 의해 제어됩니다. `aquila_qpu.properties.dict()`노트북에서 실행하면 확인할 수 있습니다. Aquila의 기능 및 요구 사항에 대한 자세한 내용은 Aquila 소개 노트북을 참조하십시오.) 메서드를 호출하여 이 작업을 수행할 수 있습니다. `discretize`

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

이제 Aquila QPU에서 프로그램을 실행할 수 있습니다 (현재는 100개의 샷만 실행).

> **Note**
>
> Aquila프로세서에서 이 프로그램을 실행하면 비용이 발생합니다. Amazon Braket SDK에는 고객이 비용 한도를 설정하고 거의 실시간으로 비용을 추적할 수 있는 비용 추적기가 포함되어 있습니다.

```
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
```

양자 작업을 실행하는 데 걸리는 시간의 편차가 크기 때문에 (가용성 창과 QPU 사용률에 따라 다름), 나중에 다음 코드 스니펫을 통해 상태를 확인할 수 있도록 양자 작업 ARN을 메모해 두는 것이 좋습니다.

```
# 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
```

상태가 COMPLETED (Amazon Braket 콘솔의 양자 작업 페이지에서도 확인 가능) 되면 다음을 사용하여 결과를 쿼리할 수 있습니다.

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

QPU 결과 분석

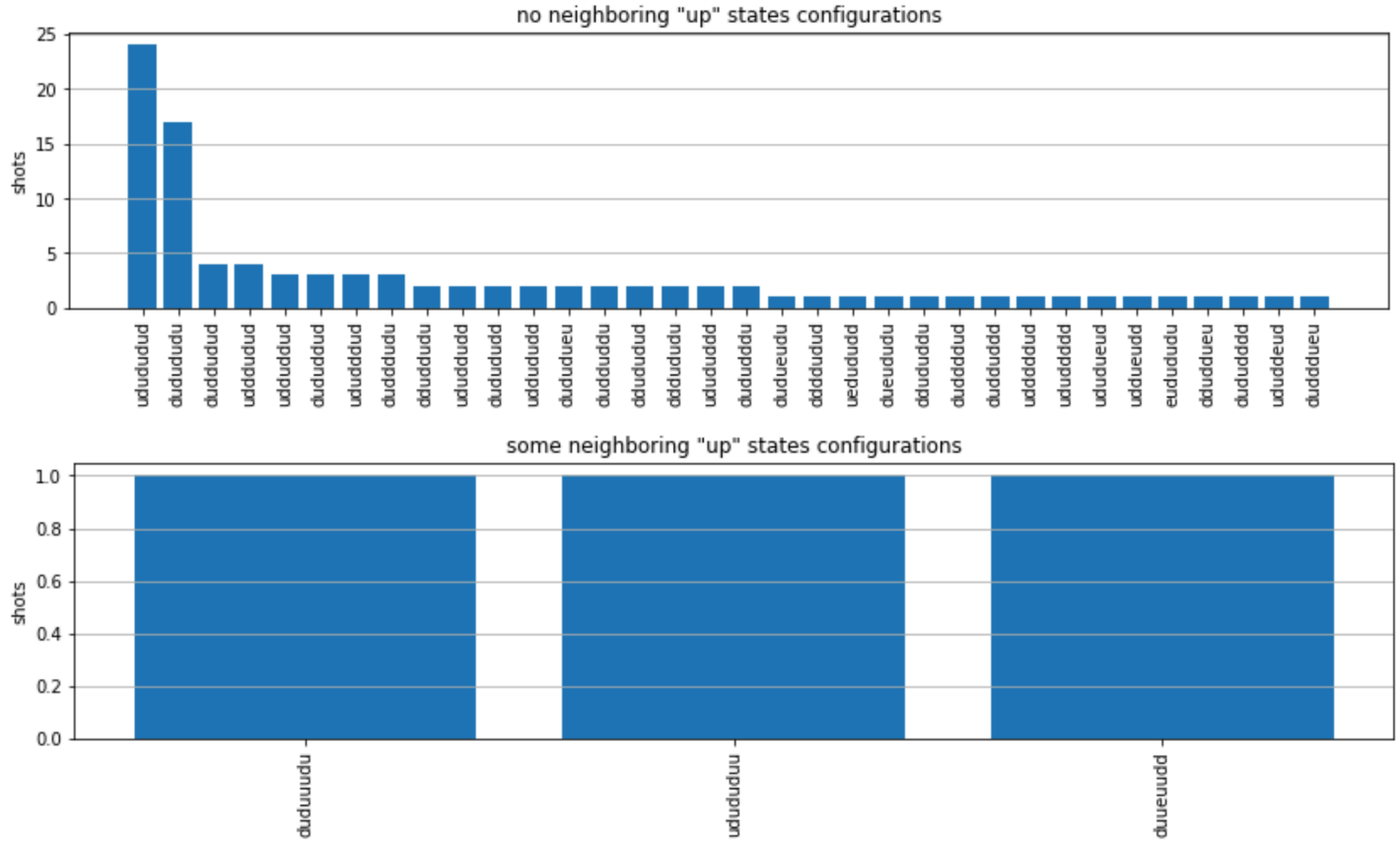
이전과 동일한 `get_counts` 함수를 사용하여 개수를 계산할 수 있습니다.

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

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

그리고 다음과 같이 `plot_counts` 플로팅합니다.

```
plot_counts(counts_aquila)
```



참고로, 일부 샷에는 빈 사이트 (“e”로 표시) 가 있습니다. 이는 QPU의 원자당 준비 불완전성 1~ 2% 때문입니다. Aquila 이 외에도 샷 수가 적기 때문에 예상되는 통계적 변동 범위 내에서 결과가 시뮬레이션과 일치합니다.

Next

축하합니다. 이제 로컬 AHS 시뮬레이터와 Aquila QPU를 사용하여 Amazon Braket에서 첫 번째 AHS 워크로드를 실행했습니다.

Rydberg 물리학, 아날로그 해밀턴 시뮬레이션 및 장치에 대한 자세한 내용은 예제 노트북을 Aquila 참조하십시오.

SDK에서 회로를 구성하세요.

이 섹션에서는 회로 정의, 사용 가능한 게이트 보기, 회로 확장, 각 장치가 지원하는 게이트 보기의 예를 제공합니다. 또한 회로를 수동으로 할당하는 방법qubits, 컴파일러에 회로를 정의된 대로 정확하게 실행하도록 지시하는 방법, 노이즈 시뮬레이터로 잡음이 많은 회로를 구성하는 방법에 대한 지침도 포함되어 있습니다.

또한 Braket에서 특정 QPU를 사용하는 다양한 게이트의 펄스 레벨에서 작업할 수 있습니다. 자세한 내용은 Amazon Braket의 펄스 제어를 참조하십시오.

이 섹션:

- 게이트 및 회로
- 부분 측정
- 수동 할당 qubit
- 축어 컴파일
- 노이즈 시뮬레이션
- 회로 검사
- 결과 유형

게이트 및 회로

양자 게이트 및 회로는 Amazon Braket Python SDK `braket.circuits`클래스에 정의되어 있습니다. SDK에서 를 호출하여 새 회로 객체를 인스턴스화할 수 있습니다. `Circuit()`

예: 회로 정의

이 예제는 표준 단일 큐비트 Hadamard 게이트와 2큐비트 CNOT 게이트로 구성된 네 개의 샘플 회로 qubits (,, 및 라고 `q0` `q3` 표시됨) 를 정의하는 것으로 시작합니다. `q1` `q2` 다음 예제와 같이 함수를 호출하여 이 회로를 시각화할 수 있습니다. `print`

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

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

```
T  : |0| 1 |

q0 : -H-C---
        |
q1 : -H-|-C-
        | |
q2 : -H-X-|-
          |
```

```
q3 : -H---X-

T  : |0| 1 |
```

예: 파라미터화된 회로 정의

이 예제에서는 자유 파라미터에 의존하는 게이트가 있는 회로를 정의합니다. 이러한 파라미터의 값을 지정하여 새 회로를 만들거나 회로를 제출할 때 특정 장치에서 양자 작업으로 실행되도록 할 수 있습니다.

```
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)
```

다음과 같이 각 매개변수의 값을 지정하는 단일 인수 `float` (모든 자유 매개변수가 취하는 값) 또는 키워드 인수를 회로에 제공하여 매개변수화된 회로에서 매개변수화되지 않은 새 회로를 만들 수 있습니다.

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

참고로 `my_circuit` 이는 수정되지 않았으므로 매개변수 값이 고정된 새 회로를 여러 개 인스턴스화하는 데 사용할 수 있습니다.

예: 회로의 게이트 수정

다음 예제에서는 제어 및 전력 수정자를 사용하는 게이트가 있는 회로를 정의합니다. 이러한 수정을 사용하여 제어 Ry 게이트와 같은 새 게이트를 만들 수 있습니다.

```
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)
```

게이트 수정자는 로컬 시뮬레이터에서만 지원됩니다.

예: 사용 가능한 모든 게이트 보기

다음 예제는 Amazon Braket에서 사용 가능한 모든 게이트를 보는 방법을 보여줍니다.

```
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)
```

이 코드의 출력에는 모든 게이트가 나열됩니다.

```
['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']
```

해당 회로 유형에 대한 메서드를 호출하여 이러한 게이트를 회로에 추가할 수 있습니다. 예를 들어, 를 `circ.h(0)` 호출하여 첫 번째 게이트에 Hadamard 게이트를 추가할 수 있습니다. qubit

> **Note**
>
> 게이트가 제자리에 추가되고 다음 예제에서는 이전 예제에 나열된 모든 게이트를 동일한 회로에 추가합니다.

```
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)
```

사전 정의된 게이트 세트 외에도 자체 정의된 단일 게이트를 회로에 적용할 수도 있습니다. 이는 단일 큐비트 게이트 (다음 소스 코드 참조) 이거나 매개변수로 정의된 값에 적용되는 다중 큐비트 게이트일 수 있습니다. qubits `targets`

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

예: 기존 회로 확장

명령을 추가하여 기존 회로를 확장할 수 있습니다. `InstructionAn`은 양자 장치에서 수행해야 하는 양자 작업을 설명하는 양자 지침입니다. `Instruction`연산자에는 유형의 `Gate` 객체만 포함됩니다.

```
# 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})
```

예: 각 장치가 지원하는 게이트 보기

시뮬레이터는 Braket SDK의 모든 게이트를 지원하지만 QPU 기기는 더 작은 하위 집합을 지원합니다. 기기 속성에서 기기의 지원되는 게이트를 찾을 수 있습니다. 다음은 IonQ 디바이스를 사용한 예제입니다.

```
# 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']
```

지원되는 게이트를 퀀텀 하드웨어에서 실행하려면 먼저 네이티브 게이트로 컴파일해야 할 수 있습니다. 회로를 제출하면 Amazon Braket은 이 컴파일을 자동으로 수행합니다.

예: 기기에서 지원하는 네이티브 게이트의 충실도를 프로그래밍 방식으로 검색합니다.

Braket 콘솔의 장치 페이지에서 충실도 정보를 볼 수 있습니다. 동일한 정보에 프로그래밍 방식으로 액세스하는 것이 도움이 되는 경우도 있습니다. 다음 코드는 QPU의 두 qubit 게이트 사이의 두 게이트 충실도를 추출하는 방법을 보여줍니다.

```
# 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"])
```

부분 측정

이전 예제에 따라 양자 회로의 모든 큐비트를 측정했습니다. 그러나 개별 큐비트 또는 큐비트의 하위 집합을 측정하는 것은 가능합니다.

예: 큐비트의 하위 집합 측정

이 예제에서는 대상 큐비트가 포함된 `measure` 명령을 회로 끝에 추가하여 부분 측정을 보여줍니다.

```
# 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)
```

수동 할당 qubit

에서 Rigetti 양자 컴퓨터에서 양자 회로를 실행할 때 선택적으로 수동 qubit 할당을 사용하여 알고리즘에 qubits 사용되는 회로를 제어할 수 있습니다. Amazon Braket 콘솔과 Amazon Braket SDK를 사용하면 선택한 양자 처리 장치 (QPU) 디바이스의 최신 보정 데이터를 검사하여 실험에 가장 적합한 것을 선택할 수 있습니다. qubits

수동 qubit 할당을 통해 회로를 더 정확하게 실행하고 개별 속성을 조사할 수 있습니다. qubit 연구원과 고급 사용자는 최신 장치 보정 데이터를 기반으로 회로 설계를 최적화하고 더 정확한 결과를 얻을 수 있습니다.

다음 예제는 명시적으로 qubits 할당하는 방법을 보여줍니다.

```
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)
```

자세한 내용은 이 노트북의 Amazon Braket 예제 또는 보다 구체적으로 설명하자면 QPU 디바이스에 GitHub 큐비트 할당을 참조하십시오.

> **Note**
>
> OQC컴파일러는 설정을 지원하지 않습니다. `disable_qubit_rewiring=True` 이 플래그를 로 `True` 설정하면 다음 오류가 발생합니다. `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`

축어 컴파일

RigettiIonQ, 또는 Oxford Quantum Circuits (OQC) 에서 양자 컴퓨터의 양자 회로를 실행하면 컴파일러가 회로를 수정하지 않고 정의된 대로 정확하게 실행하도록 지시할 수 있습니다. 축어 컴파일을 사용하

면 전체 회로를 지정된 대로 정확하게 보존 (RigettiIonQ, 및 지원OQC) 하거나 회로의 특정 부분만 보존 (only 지원) 하도록 지정할 수 있습니다. Rigetti 하드웨어 벤치마킹 또는 오류 완화 프로토콜용 알고리즘을 개발할 때는 하드웨어에서 실행 중인 게이트 및 회로 레이아웃을 정확하게 지정할 수 있는 옵션이 필요합니다. 축어 컴파일을 사용하면 특정 최적화 단계를 해제하여 컴파일 프로세스를 직접 제어할 수 있으므로 회로가 설계된 대로 정확하게 실행되도록 할 수 있습니다.

Verbatim 컴파일은 현재Rigetti,IonQ, Oxford Quantum Circuits (OQC) 기기에서 지원되며 네이티브 게이트를 사용해야 합니다. 축어 컴파일을 사용하는 경우 장치의 토폴로지를 확인하여 연결된 상태에서 qubits 게이트가 호출되고 회로가 하드웨어에서 지원되는 네이티브 게이트를 사용하는지 확인하는 것이 좋습니다. 다음 예제는 디바이스에서 지원하는 네이티브 게이트 목록에 프로그래밍 방식으로 액세스하는 방법을 보여줍니다.

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

의 Rigetti 경우 축어 컴파일에 `disableQubitRewiring=True` 사용하도록 설정하여 qubit 재배선을 해제해야 합니다. 컴파일에서 축어 상자를 사용할 때 설정하면 양자 회로가 검증에 실패하고 `disableQubitRewiring=False` 실행되지 않습니다.

회로에 대해 축어 컴파일을 활성화하고 해당 회로를 지원하지 않는 QPU에서 실행하면 지원되지 않는 작업으로 인해 작업이 실패했음을 나타내는 오류가 생성됩니다. 컴파일러 함수를 기본적으로 지원하는 양자 하드웨어가 늘어남에 따라 이 기능도 이러한 장치를 포함하도록 확장될 예정입니다. 축어 컴파일을 지원하는 기기는 다음 코드로 쿼리하면 지원되는 작업으로 포함됩니다.

```
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
```

축어 컴파일 사용과 관련된 추가 비용은 없습니다. Braket QPU 디바이스, 노트북 인스턴스 및 온디맨드 시뮬레이터에서 실행된 양자 작업에 대해서는 Amazon Braket 가격 책정 페이지에 명시된 현재 요금을 기준으로 계속 요금이 부과됩니다. 자세한 내용은 Verbatim 컴파일 예제 노트북을 참조하십시오.

> **Note**
>
> OpenQASM을 사용하여 OQC 및 IonQ 장치의 회로를 작성하고 회로를 물리적 큐비트에 직접 매핑하려면 OpenQASM에서 `disableQubitRewiring` 플래그를 완전히 무시하므로 를 사용해야 합니다. `#pragma braket verbatim`

노이즈 시뮬레이션

로컬 노이즈 시뮬레이터를 인스턴스화하려면 다음과 같이 백엔드를 변경할 수 있습니다.

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

다음과 같은 두 가지 방법으로 잡음이 있는 회로를 만들 수 있습니다.

1. 시끄러운 회로를 아래에서 위로 구성하십시오.
2. 소음이 없는 기존 회로를 가져와 전체에 잡음을 주입하세요.

다음 예제는 디분극 노이즈가 있는 간단한 회로와 사용자 지정 Kraus 채널을 사용하는 방법을 보여줍니다.

```
# 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)
```

다음 두 예에서 볼 수 있듯이 회로를 실행하는 것은 이전과 동일한 사용자 경험을 제공합니다.

예 1

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

Or

예제 2

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

더 많은 예제는 Braket 입문용 노이즈 시뮬레이터 예제를 참조하십시오.

회로 검사

Amazon브라켓의 양자 회로에는 유사시간이라는 개념이 있습니다. `Moments` 각 게이트는 한 개당 한 개씩 qubit 발생할 수 있습니다. `Moment` 의 `Moments` 목적은 회로와 해당 게이트의 주소를 더 쉽게 지정하고 임시 구조를 제공하는 것입니다.

> **Note**
>
> 모멘트는 일반적으로 QPU에서 게이트가 실행되는 실시간 시간과 일치하지 않습니다.

회로의 깊이는 해당 회로의 총 모멘트 수로 계산됩니다. 다음 예와 `circuit.depth` 같이 메서드를 호출하는 회로 깊이를 볼 수 있습니다.

```
# 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
```

위 회로의 전체 회로 깊이는 3입니다 (모멘트 01, 및 로 표시2). 매 순간의 게이트 작동을 확인할 수 있습니다.

Moments키-값 쌍의 딕셔너리 역할을 합니다.

- 키는 유사 시간과 정보를 포함하는 입니다MomentsKey(). qubit
- 값은 의 유형으로 할당됩니다. 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)]))
```

를 통해 회로에 게이트를 추가할 수도 Moments 있습니다.

```
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|
```

결과 유형

AmazonBraket은 를 사용하여 ResultType 회로를 측정할 때 다양한 유형의 결과를 반환할 수 있습니다. 회로는 다음과 같은 유형의 결과를 반환할 수 있습니다.

- AdjointGradient제공된 관찰 가능 항목의 기대값의 기울기 (벡터 도함수) 를 반환합니다. 이 옵저버블은 지정된 매개변수와 관련하여 제공된 대상에 대해 부속 미분 방법을 사용하여 작용합니다. 이 메서드는 shots=0인 경우에만 사용할 수 있습니다.
- Amplitude출력 파동 함수에서 지정된 양자 상태의 진폭을 반환합니다. SV1및 로컬 시뮬레이터에서만 사용할 수 있습니다.
- Expectation주어진 옵저버블의 기대값을 반환하며, 이 값은 이 장의 뒷부분에서 소개할 Observable 클래스를 사용하여 지정할 수 있습니다. 관찰 가능 항목을 측정하는 qubits 데 사용되는 대상을 지정해야 하며, 지정된 대상의 수는 관찰 가능 항목이 작동하는 대상 수와 같아야 합니다. qubits 대상이 지정되지 않은 경우 Observable은 1에서만 작동해야 qubit 하며 병렬로 모두에 qubits 적용됩니다.
- Probability계산 기반 상태를 측정할 확률을 반환합니다. 대상이 지정되지 않은 경우 모든 기준 상태를 측정할 확률을 Probability 반환합니다. 목표값이 지정된 경우 지정된 기저 벡터의 한계 확률만 반환됩니다. qubits
- Reduced density matrix의 시스템에서 지정된 qubits 목표값의 하위 시스템에 대한 밀도 행렬을 반환합니다. qubits 이 결과 유형의 크기를 제한하기 위해 Braket은 대상 qubits 수를 최대 8개로 제한합니다.
- StateVector전체 상태 벡터를 반환합니다. 로컬 시뮬레이터에서 사용할 수 있습니다.
- Sample지정된 대상 qubit 세트 및 관찰 가능한 대상 세트의 측정 횟수를 반환합니다. 대상이 지정되지 않은 경우 Observable은 1에서만 작동해야 qubit 하며 병렬로 모두에 qubits 적용됩니다. 목표가 지정된 경우 지정된 대상의 수는 관찰 가능 대상이 작용하는 대상의 qubits 수와 같아야 합니다.

- `Variance`지정된 대상 qubit 세트와 옵저버블의 variance (`mean([x-mean(x)]`^{2})) 를 요청된 결과 유형으로 반환합니다. 대상이 지정되지 않은 경우 Observable은 1에서만 작동해야 qubit 하며 병렬로 모두에 qubits 적용됩니다. 그렇지 않으면 지정된 대상의 수가 옵저버블을 적용할 수 있는 대상의 qubits 수와 같아야 합니다.

다양한 기기에 지원되는 결과 유형:

	로컬 SIM	SV1	DM1	TN1	Rigetti	IonQ	OQC
인접 그래디언트	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
기대치	Y	Y	Y	Y	Y	Y	Y
Probability	Y	Y	Y	N	Y*	Y	Y
밀도 감소 매트릭스	Y	N	Y	N	N	N	N
상태 벡터	Y	N	N	N	N	N	N
Sample	Y	Y	Y	Y	Y	Y	Y
변화	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> *는 최대 40개의 확률 결과 Rigetti 유형만 지원합니다qubits.

다음 예와 같이 장치 속성을 검사하여 지원되는 결과 유형을 확인할 수 있습니다.

```
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
```

ResultTypea를 호출하려면 다음 예와 같이 회로에 추가하십시오.

```
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**
>
> 일부 장치는 측정값 (예:Rigetti) 을 결과로 제공하고 다른 장치는 확률을 결과로 제공합니다 (예: IonQ 및OQC). SDK는 결과에 대한 측정 속성을 제공하지만 확률을 반환하는 기기의 경우 사후 계산이 이루어집니다. 따라서 에서 제공하는 기기와 같은 기기는 샷당 측정값이 반환되지 않으므로 확률에 따라 측정 결과가 결정됩니다. IonQ OQC 이 파일에 표시된 대로 결과 개체를 보면 결과가 사후 계산되었는지 확인할 수 있습니다. measurements_copied_from_device

관찰 가능 항목

Amazon브라켓에는 측정할 관찰 가능 항목을 지정하는 데 사용할 수 있는 Observable 클래스가 포함되어 있습니다.

각각에 고유한 비동일성 관찰 가능 항목을 최대 하나만 적용할 수 있습니다. qubit 동일하지 않은 옵저버블을 두 개 이상 지정하면 오류가 qubit 발생합니다. 이를 위해 텐서 곱의 각 인자는 개별 관찰 가능

인자로 계산되므로 텐서 곱에 작용하는 인자가 동일하다면 여러 텐서 곱이 동일한 qubit 것에 작용하는 것은 허용됩니다. qubit

옵저버블을 스케일링하고 옵저버블을 추가할 수도 있습니다 (스케일 적용 여부). 이렇게 하면 결과 Sum 유형에 사용할 수 있는 항목이 생성됩니다. AdjointGradient

Observable클래스에는 다음과 같은 옵저버블이 포함됩니다.

```
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)))
```

Parameters

회로에는 “한 번 생성하여 여러 번 실행” 방식으로 사용하고 그래디언트를 계산하는 데 사용할 수 있는 자유 매개변수가 포함될 수 있습니다. 자유 매개변수는 문자열로 인코딩된 이름을 사용하여 해당 값을 지정하거나 자유 매개 변수를 기준으로 미분할지 여부를 결정하는 데 사용할 수 있습니다.

```
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]
```

구별하려는 매개 변수의 경우 이름 (문자열) 을 사용하거나 직접 참조를 사용하여 매개 변수를 지정하십시오. 참고로 `AdjointGradient` 결과 유형을 사용하여 그래디언트를 계산하는 것은 옵저버블의 기대값을 기준으로 수행됩니다.

참고: 자유 매개변수 값을 매개변수화된 회로에 인수로 전달하여 수정한 경우 지정된 결과 유형과 매개변수를 사용하여 `AdjointGradient` 회로를 실행하면 오류가 발생합니다. 이는 차별화하는 데 사용하는 매개변수가 더 이상 존재하지 않기 때문입니다. 다음 예를 참조하세요.

```
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
```

QPU 및 시뮬레이터에 양자 작업 제출

AmazonBraket은 양자 작업을 실행할 수 있는 여러 장치에 대한 액세스를 제공합니다. 양자 태스크를 개별적으로 제출하거나 양자 태스크 배칭을 설정할 수 있습니다.

QPU

언제든지 QPU에 양자 작업을 제출할 수 있지만 작업은 Amazon Braket 콘솔의 장치 페이지에 표시되는 특정 가용성 창 내에서 실행됩니다. 다음 섹션에서 소개하는 양자 작업 ID를 사용하여 양자 작업의 결과를 검색할 수 있습니다.

- 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(예약 전용): `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`

시뮬레이터

- 밀도 매트릭스 시뮬레이터, DM1 `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- 상태 벡터 시뮬레이터SV1, `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- 텐서 네트워크 시뮬레이터, TN1 `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- 로컬 시뮬레이터: `LocalSimulator()`

> **Note**
>
> QPU 및 온디맨드 시뮬레이터의 `CREATED` 상태에서 양자 작업을 취소할 수 있습니다. 온디맨드 시뮬레이터 및 QPU의 경우 최선을 다해 해당 `QUEUED` 주의 양자 작업을 취소할 수 있습니다. 참고로 QPU `QUEUED` 양자 작업은 QPU 가용성 기간 중에 성공적으로 취소될 가능성은 거의 없습니다.

이 섹션:

Amazon Braket의 양자 작업 예시

이 섹션에서는 디바이스 선택부터 결과 보기에 이르기까지 예제 양자 작업을 실행하는 단계를 안내합니다. AmazonBraket의 모범 사례로는 다음과 같은 SV1 시뮬레이터에서 회로를 실행하는 것으로 시작하는 것이 좋습니다.

이 섹션:

- 디바이스를 지정하세요.
- 예제 양자 태스크 제출하기
- 파라미터화된 작업 제출
- shots를 지정합니다.
- 설문 조사 결과
- 예제 결과 보기

디바이스를 지정하세요.

먼저 양자 작업에 사용할 디바이스를 선택하고 지정합니다. 이 예제에서는 시뮬레이터를 선택하는 방법을 보여줍니다SV1.

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

다음과 같이 이 장치의 일부 속성을 볼 수 있습니다.

```
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)])
```

예제 양자 태스크 제출하기

온디맨드 시뮬레이터에서 실행할 예제 양자 작업을 제출하세요.

```
# 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()
```

이 `device.run()` 명령은 `CreateQuantumTask` API를 통해 양자 작업을 생성합니다. 초기화 시간이 짧으면 디바이스에서 양자 작업을 실행할 수 있는 용량이 확보될 때까지 양자 작업이 대기됩니다. 이 경우 기기는 입니다. SV1 디바이스가 계산을 완료한 후 Amazon Braket은 호출에 지정된 Amazon S3 위치에 결과를 기록합니다. 위치 `s3_location` 인수는 로컬 시뮬레이터를 제외한 모든 디바이스에 필요합니다.

> **Note**
>
> Braket 퀀텀 태스크 액션의 크기는 3MB로 제한됩니다.

파라미터화된 작업 제출

Amazon Braket 온디맨드 및 로컬 시뮬레이터와 QPU는 작업 제출 시 사용 가능한 파라미터 값 지정도 지원합니다. 다음 예와 같이 to `inputs` `device.run()` 인수를 사용하여 이 작업을 수행할 수 있습니다. 는 문자열-부동 쌍의 `inputs` 딕셔너리여야 합니다. 여기서 키는 파라미터 이름이어야 합니다.

파라메트릭 컴파일은 특정 QPU에서 파라메트릭 회로를 실행하는 성능을 향상시킬 수 있습니다. 파라메트릭 회로를 지원되는 QPU에 양자 작업으로 제출하면 Braket은 회로를 한 번 컴파일하고 결과를 캐시합니다. 동일한 회로에 대한 후속 파라미터 업데이트 시 다시 컴파일할 필요가 없으므로 동일한 회로를 사용하는 작업의 런타임이 더 빨라집니다. Braket은 회로를 컴파일할 때 하드웨어 제공업체의 업데이트된 보정 데이터를 자동으로 사용하여 최고 품질의 결과를 보장합니다.

> **Note**
>
> 파라메트릭 컴파일은 펄스 레벨 프로그램을 제외한 모든 초전도, 게이트 기반 QPU에서 Rigetti Computing 지원됩니다. Oxford Quantum Circuits

```
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})
```

shots를 지정합니다.

shots인수는 원하는 측정 횟수를 나타냅니다. shots 와 같은 시뮬레이터는 두 가지 시뮬레이션 모드를 SV1 지원합니다.

- shots= 0인 경우 시뮬레이터는 정확한 시뮬레이션을 수행하여 모든 결과 유형에 대해 실제 값을 반환합니다. (에서는 TN1 사용할 수 없음)

- 0이 아닌 값의 경우 시뮬레이터는 출력 분포를 샘플링하여 실제 QPU의 shot 노이즈를 에뮬레이션합니다. shots QPU 기기는 0보다 큰 값만 허용합니다. shots

양자 작업당 최대 촬영 횟수에 대한 자세한 내용은 브라켓 할당량을 참조하십시오.

설문 조사 결과

실행 `my_task.result()` 시 SDK는 퀀텀 태스크 생성 시 정의한 파라미터를 사용하여 결과를 폴링하기 시작합니다.

- `poll_timeout_seconds`온디맨드 시뮬레이터 및/또는 QPU 디바이스에서 양자 작업을 실행할 때 시간이 초과되기 전에 양자 작업을 폴링하는 데 걸리는 시간 (초) 입니다. 기본값은 432,000초이며, 이는 5일입니다.
- 참고: Rigetti 및 와 IonQ 같은 QPU 장치의 경우 며칠을 허용하는 것이 좋습니다. 폴링 제한 시간이 너무 짧으면 폴링 시간 내에 결과가 반환되지 않을 수 있습니다. 예를 들어, QPU를 사용할 수 없는 경우 로컬 타임아웃 오류가 반환됩니다.
- `poll_interval_seconds`양자 작업이 폴링되는 빈도입니다. 온디맨드 시뮬레이터와 QPU 장치에서 양자 작업이 실행될 때 상태를 확인하기 API 위해 Braket을 호출하는 빈도를 지정합니다. 기본값은 1초입니다.

이 비동기 실행은 항상 사용할 수 없는 QPU 장치와의 상호 작용을 용이하게 합니다. 예를 들어 정기 유지 관리 기간 중에는 기기를 사용하지 못할 수 있습니다.

반환된 결과에는 양자 작업과 관련된 다양한 메타데이터가 포함됩니다. 다음 명령을 사용하여 측정 결과를 확인할 수 있습니다.

```
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]]
```

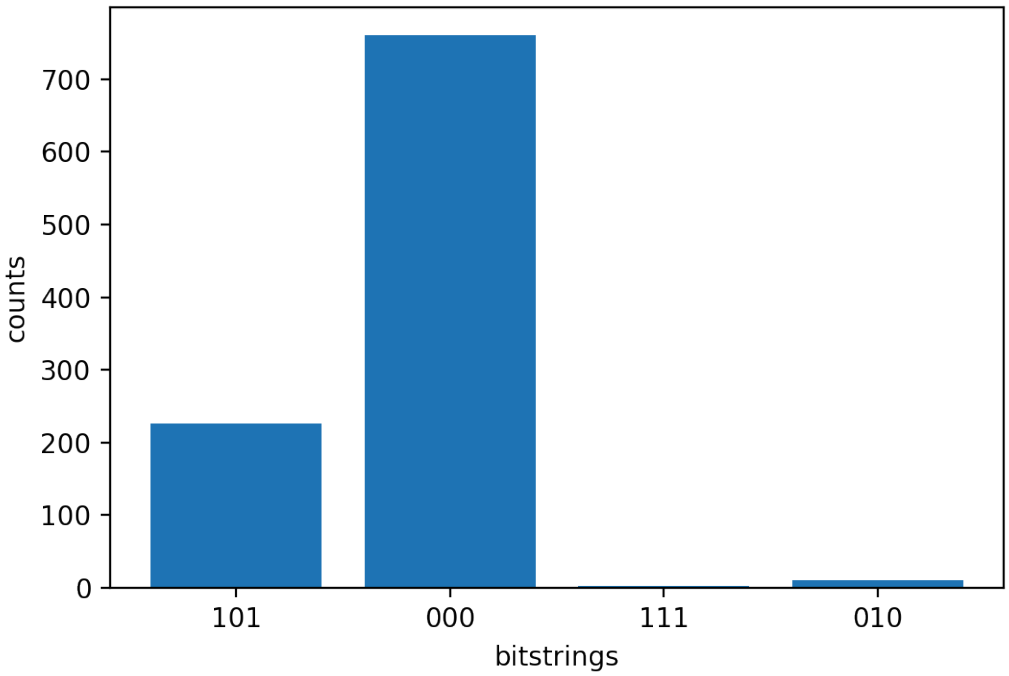
```
Counts for collapsed states:
 Counter({'000': 761, '101': 226, '010': 10, '111': 3})
Probabilities for collapsed states:
 {'101': 0.226, '000': 0.761, '111': 0.003, '010': 0.01}
```

예제 결과 보기

도 지정했으므로 반환된 결과를 볼 수 있습니다. ResultType 결과 유형은 회로에 추가된 순서대로 표시됩니다.

```
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]
```



QPU에 양자 작업 제출

AmazonBraket을 사용하면 QPU 장치에서 양자 회로를 실행할 수 있습니다. 다음 예제는 우리 장치에 양자 작업을 제출하는 방법을 보여줍니다. Rigetti IonQ

Rigetti Aspen-M-3디바이스를 선택한 다음 관련 연결 그래프를 살펴보세요.

```
# 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']}}
```

위 사전에는 현재 Rigetti 장치의 연결에 대한 정보가 `connectivityGraph` 들어 있습니다.

장치를 선택합니다. IonQ Harmony

IonQ Harmony기기의 경우, 다음 예와 같이 `connectivityGraph` 기기가 All-to-All 연결을 제공하므로 이 비어 있습니다. 따라서 자세한 내용은 `connectivityGraph` 필요하지 않습니다.

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

다음 예와 같이 기본 버킷 이외의 위치를 지정하기로 선택한 경우 shots (기본값=1000), `poll_timeout_seconds` (기본값 = 432000 = 5일), `poll_interval_seconds` (기본값 = 1s`s3_location`) 및 결과를 저장할 S3 버킷 () 의 위치를 조정할 수 있습니다.

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

IonQ및 Rigetti 디바이스는 제공된 회로를 각각의 네이티브 게이트 세트로 자동 컴파일하고 추상 qubit 인덱스를 각 QPU의 물리적 qubits 인덱스에 매핑합니다.

> **Note**
>
> QPU 디바이스는 용량이 제한되어 있습니다. 용량에 도달하면 대기 시간이 더 길어질 수 있습니다.

AmazonBraket은 특정 가용성 기간 내에서 QPU 양자 작업을 실행할 수 있지만 모든 해당 데이터와 메타데이터가 적절한 S3 버킷에 안정적으로 저장되므로 언제든지 (연중무휴) 양자 작업을 제출할 수 있습니다. 다음 섹션에서 볼 수 있듯이 고유한 양자 작업 ID를 사용하여 `AwsQuantumTask` 양자 작업을 복구할 수 있습니다.

로컬 시뮬레이터로 양자 작업 실행

신속한 프로토타이핑 및 테스트를 위해 양자 작업을 로컬 시뮬레이터로 직접 보낼 수 있습니다. 이 시뮬레이터는 로컬 환경에서 실행되므로 Amazon S3 위치를 지정할 필요가 없습니다. 결과는 세션에서 직접 계산됩니다. 로컬 시뮬레이터에서 양자 작업을 실행하려면 shots 파라미터만 지정해야 합니다.

> **Note**
>
> 로컬 시뮬레이터가 처리할 수 있는 실행 속도 및 최대 수는 Amazon Braket 노트북 인스턴스 유형이나 로컬 하드웨어 사양에 따라 다릅니다. qubits

다음 명령은 모두 동일하며 상태 벡터 (노이즈 프리) 로컬 시뮬레이터를 인스턴스화합니다.

```
# 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")
```

그런 다음 다음을 사용하여 양자 작업을 실행합니다.

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

로컬 밀도 매트릭스 (노이즈) 시뮬레이터를 인스턴스화하기 위해 고객은 다음과 같이 백엔드를 변경합니다.

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

로컬 시뮬레이터에서 특정 큐비트 측정

로컬 상태 벡터 시뮬레이터와 로컬 밀도 매트릭스 시뮬레이터는 회로 큐비트의 하위 집합을 측정할 수 있는 실행 회로를 지원합니다. 이를 부분 측정이라고도 합니다.

예를 들어 다음 코드에서는 2큐비트 회로를 만들고 대상 큐비트가 포함된 `measure` 명령을 회로 끝에 추가하여 첫 번째 큐비트만 측정할 수 있습니다.

```
# 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)
```

양자 태스크 배칭

퀀텀 태스크 배칭은 로컬 시뮬레이터를 제외한 모든 Amazon Braket 기기에서 사용할 수 있습니다. 일괄 처리는 온디맨드 시뮬레이터 (TN1또는SV1) 에서 실행하는 양자 작업에 특히 유용합니다. 여러 양자 작업을 병렬로 처리할 수 있기 때문입니다. 다양한 양자 작업을 설정하는 데 도움이 되도록 Amazon Braket은 예제 노트북을 제공합니다.

일괄 처리를 통해 양자 작업을 병렬로 시작할 수 있습니다. 예를 들어 10개의 양자 작업이 필요하고 이러한 양자 작업의 회로가 서로 독립적이라면 일괄 처리를 사용하는 것이 좋습니다. 이렇게 하면 다른 작업이 시작되기 전에 하나의 양자 작업이 완료될 때까지 기다릴 필요가 없습니다.

다음 예제는 일련의 양자 작업을 실행하는 방법을 보여줍니다.

```
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
```

자세한 내용은 일괄 처리에 대한 보다 구체적인 정보가 있는 Quantum 작업 일괄 처리에 대한 GitHub Amazon Braket 예제를 참조하십시오.

양자 작업 일괄 처리 및 비용 정보

양자 작업 일괄 처리 및 청구 비용과 관련하여 염두에 두어야 할 몇 가지 주의 사항:

- 기본적으로 양자 태스크 배칭은 양자 태스크를 모두 재시도하거나 양자 태스크가 3번 실패합니다.
- 34 qubits for SV1 와 같이 오래 실행되는 양자 작업을 일괄 처리하면 비용이 많이 들 수 있습니다. 일괄 양자 작업을 시작하기 전에 `run_batch` 할당 값을 주의 깊게 다시 한 번 확인하세요. with는 사용하지 TN1 않는 것이 좋습니다`run_batch`.
- TN1실패한 리허설 단계 작업으로 인해 비용이 발생할 수 있습니다 (자세한 내용은 TN1 설명 참조). 자동 재시도는 비용을 가중시킬 수 있으므로 사용 시 일괄 처리의 'max_retry' 횟수를 0으로 설정하는 것이 좋습니다 TN1 (Quantum Task Batching, 186행 참조).

양자 태스크 배칭 및 PennyLane

다음 예와 같이 Amazon Amazon Braket 장치를 인스턴스화할 `parallel = True` 때 설정하여 PennyLane Braket에서 사용할 때 일괄 처리를 활용하십시오.

```
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,)
```

일괄 처리에 대한 자세한 내용은 양자 회로의 병렬화 PennyLane 최적화를 참조하십시오.

태스크 배칭 및 파라미터화된 회로

파라미터화된 회로를 포함하는 양자 작업 배치를 제출할 때는 `list` 배치의 모든 양자 작업에 사용되는 `inputs` 사전을 제공하거나 입력 사전을 제공할 수 있습니다. 이 경우 다음 예와 같이 -번째 사전이 `i` -번째 작업과 쌍을 이룹니다. `i`

```
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)
```

단일 파라메트릭 회로의 입력 사전 목록을 준비하여 양자 작업 배치로 제출할 수도 있습니다. 목록에 N개의 입력 사전이 있는 경우 배치에는 N개의 양자 작업이 포함됩니다. `i`-번째 양자 작업은 `i` -번째 입력 사전을 사용하여 실행되는 회로에 해당합니다.

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

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

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

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

SNS 알림 설정 (선택 사항)

Amazon Simple Notification Service (SNS) 를 통해 알림을 설정하여 Braket 퀀텀 작업이 완료될 때 Amazon 알림을 받을 수 있습니다. 활성 알림은 대기 시간이 길어질 것으로 예상되는 경우 유용합니다. 예를 들어 대규모 양자 작업을 제출하거나 디바이스의 가용성 기간이 지난 후 양자 작업을 제출할 때 유용합니다. 양자 작업이 완료될 때까지 기다리지 않으려면 SNS 알림을 설정할 수 있습니다.

AmazonBraket 노트북은 설정 단계를 안내합니다. 자세한 내용은 Amazon Braket 예제 GitHub 및 특히 알림 설정을 위한 예제 노트북을 참조하십시오.

컴파일된 회로 검사

회로가 하드웨어 장치에서 실행되는 경우 회로를 QPU에서 지원하는 네이티브 게이트로 변환하는 것과 같이 적절한 형식으로 컴파일해야 합니다. 실제 컴파일된 출력을 검사하는 것은 디버깅 목적으로 매우 유용할 수 있습니다. 아래 코드를 사용하여 두 OQC 장치 Rigetti 모두에서 이 회로를 볼 수 있습니다.

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

> **Note**
>
> 현재는 컴파일된 IonQ 장치 회로를 볼 수 없습니다.

OpenQASM 3.0으로 회로를 실행하십시오.

Amazon브라켓은 이제 게이트 기반 양자 장치 및 시뮬레이터를 위한 OpenQASM 3.0을 지원합니다. 이 사용자 안내서는 브라켓에서 지원하는 OpenQASM 3.0의 하위 집합에 대한 정보를 제공합니다. 이

제 브라켓 고객은 SDK를 사용하여 브라켓 회로를 제출하거나 Amazon Braket API 및 Amazon Braket Python SDK를 사용하여 모든 게이트 기반 디바이스에 OpenQASM 3.0 문자열을 직접 제공하여 브라켓 회로를 제출할 수 있습니다.

이 가이드의 항목에서는 다음과 같은 양자 작업을 완료하는 방법에 대한 다양한 예를 안내합니다.

- 다양한 Braket 디바이스에서 OpenQASM 양자 태스크를 생성하고 제출하십시오.
- 지원되는 작업 및 결과 유형에 액세스할 수 있습니다.
- OpenQASM을 사용하여 노이즈를 시뮬레이션하십시오.
- OpenQASM에서는 축어적 컴파일을 사용하십시오.
- OpenQASM 문제를 해결하세요.

또한 이 안내서는 Braket에서 OpenQASM 3.0으로 구현할 수 있는 특정 하드웨어 관련 기능에 대한 소개와 추가 리소스에 대한 링크를 제공합니다.

이 섹션:

- 오픈QASM 3.0이란 무엇입니까?
- 오픈QASM 3.0을 사용해야 하는 경우
- OpenQASM 3.0의 작동 방식
- 사전 조건
- 브라켓은 어떤 OpenQASM 기능을 지원하나요?
- 예제 OpenQASM 3.0 퀀텀 태스크 생성 및 제출
- 다양한 브라켓 디바이스에서의 OpenQASM 지원
- OpenQASM 3.0으로 노이즈를 시뮬레이션하십시오.
- QubitOpenQASM 3.0으로 다시 배선
- OpenQASM 3.0을 사용한 축어적 컴파일
- 브라켓 콘솔
- 추가 리소스
- OpenQASM 3.0을 사용한 그라디언트 컴퓨팅
- OpenQasm 3.0으로 특정 큐비트를 측정합니다.

오픈QASM 3.0이란 무엇입니까?

개방형 양자 어셈블리 언어 (OpenQASM) 는 양자 명령어를 위한 중간 표현입니다. OpenQASM은 오픈 소스 프레임워크이며 게이트 기반 장치의 양자 프로그램 사양에 널리 사용됩니다. OpenQASM을 통해 사용자는 양자 계산의 구성 요소를 구성하는 양자 게이트 및 측정 작업을 프로그래밍할 수 있습니다. 이전 버전의 OpenQASM (2.0) 은 여러 양자 프로그래밍 라이브러리에서 간단한 프로그램을 설명하는 데 사용되었습니다.

새 버전의 OpenQASM (3.0) 은 최종 사용자 인터페이스와 하드웨어 설명 언어 간의 격차를 해소하기 위해 펄스 레벨 제어, 게이트 타이밍 및 기존 제어 흐름과 같은 더 많은 기능을 포함하도록 이전 버전을 확장합니다. 현재 버전 3.0에 대한 세부 정보 및 사양은 OpenQASM 3.x 라이브 사양에서 확인할 수 있습니다. GitHub OpenQASM의 향후 개발은 OpenQASM 3.0 기술 운영 위원회가 관리합니다. OpenQASM 3.0 기술 운영 위원회는 IBM, Microsoft, 인스브루크 대학교와 함께 회원으로 활동하고 있습니다. AWS

오픈QASM 3.0을 사용해야 하는 경우

OpenQASM은 아키텍처에 국한되지 않는 저수준 제어를 통해 양자 프로그램을 지정하는 표현적인 프레임워크를 제공하므로 여러 게이트 기반 장치에서 사용하기에 적합합니다. OpenQASM에 대한 Braket 지원을 통해 게이트 기반 양자 알고리즘 개발에 대한 일관된 접근 방식으로 채택이 확대되어 사용자가 여러 프레임워크에서 라이브러리를 배우고 유지 관리할 필요가 줄어듭니다.

OpenQASM 3.0에 기존 프로그램 라이브러리가 있는 경우 이러한 회로를 완전히 다시 작성하는 대신 Braket과 함께 사용할 수 있도록 조정할 수 있습니다. 또한 연구자와 개발자는 OpenQASM의 알고리즘 개발을 지원하는 타사 라이브러리의 수가 늘어나는 이점을 누릴 수 있습니다.

OpenQASM 3.0의 작동 방식

브라켓의 OpenQASM 3.0에 대한 지원은 현재의 중간 표현과 동일한 기능을 제공합니다. 즉, Braket을 사용하는 하드웨어 장치 및 온디맨드 시뮬레이터에서 현재 수행할 수 있는 모든 작업을 Braket을 사용하면 OpenQASM으로 수행할 수 있습니다. API 현재 Braket의 장치에 회로가 공급되는 방식과 유사한 방식으로 모든 게이트 기반 장치에 OpenQASM 문자열을 직접 공급하여 OpenQASM 3.0 프로그램을 실행할 수 있습니다. 브라켓 사용자는 OpenQASM 3.0을 지원하는 타사 라이브러리를 통합할 수도 있습니다. 이 가이드의 나머지 부분에서는 Braket과 함께 사용할 OpenQASM 표현을 개발하는 방법을 자세히 설명합니다.

사전 조건

Amazon브라켓에서 OpenQASM 3.0을 사용하려면 Amazon Braket Python 스키마 v1.8.0 버전과 Amazon Braket Python SDK v1.17.0 이상이 있어야 합니다.

브라켓을 처음 사용하는 경우 브라켓을 활성화해야 합니다. Amazon Amazon 지침은 Amazon Braket 활성화를 참조하십시오.

브라켓은 어떤 OpenQASM 기능을 지원하나요?

다음 섹션에는 브라켓에서 지원하는 OpenQASM 3.0 데이터 유형, 명령문 및 프래그마 지침이 나와 있습니다.

이 섹션:

- 지원되는 OpenQASM 데이터 유형
- 지원되는 OpenQASM 명령문
- 브라켓: OpenQASM 프라그마
- 로컬 시뮬레이터의 OpenQASM에 대한 고급 기능 지원
- 지원되는 연산 및 문법: OpenPulse

지원되는 OpenQASM 데이터 유형

브라켓은 다음과 같은 OpenQASM 데이터 유형을 지원합니다. Amazon

- 음수가 아닌 정수는 (가상 및 물리적) 큐비트 인덱스에 사용됩니다.
  - `cnot q[0], q[1];`
  - `h $0;`
- 게이트 회전 각도에는 부동 소수점 숫자 또는 상수를 사용할 수 있습니다.
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi는 OpenQASM에 내장된 상수이며 파라미터 이름으로 사용할 수 없습니다.

- 복소수 배열 (허수부에 OpenQASM `im` 표기법 사용) 은 일반 에르미트 관측값을 정의하기 위한 결과 유형 프라그마와 단일 프라그마에서 사용할 수 있습니다.
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

지원되는 OpenQASM 명령문

브라켓은 다음과 같은 OpenQASM 명령문을 지원합니다. Amazon

- `Header: OPENQASM 3;`
- 클래식 비트 선언:
  - `bit b1;`(동등하게,) `creg b1;`
  - `bit[10] b2;`(동등하게,) `creg b2[10];`
- 큐비트 선언:
  - `qubit b1;`(동등하게,) `qreg b1;`
  - `qubit[10] b2;`(동등하게,) `qreg b2[10];`
- 배열 내 인덱싱: `q[0]`
- 입력: `input float alpha;`
- 물리적 사양qubits: `$0`
- 디바이스에서 지원되는 게이트 및 작동:
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> 장치의 지원되는 게이트는 OpenQASM 작업의 장치 속성에서 찾을 수 있습니다. 이러한 게이트를 사용하는 데 게이트 정의가 필요하지 않습니다.

- 박스 명령문 축어. 현재는 박스 지속 시간 표기법을 지원하지 않습니다. 축어적 qubits 상자에는 네이티브 게이트와 피지컬이 필요합니다.

```
#pragma braket verbatim
```

```
box{
    rx(0.314) $0;
}
```

- 레지스터 전체 qubits 또는 전체 qubit 레지스터에 측정 및 측정 할당.
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`

> **Note**
>
> pi는 OpenQASM에 내장된 상수이며 매개변수 이름으로 사용할 수 없습니다.

브라켓: OpenQASM 프라그마

브라켓은 다음과 같은 OpenQASM 프라그마 지침을 지원합니다. Amazon

- 노이즈 프래그마
  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- 축어적 프래그마
  - `#pragma braket verbatim`
- 결과 유형: 프래그마
  - 기본 불변 결과 유형:
    - 상태 벡터: `#pragma braket result state_vector`
    - 밀도 매트릭스: `#pragma braket result density_matrix`
  - 그래디언트 계산 프래그마:
    - 인접 그래디언트: `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Z 기반 결과 유형:

  - 진폭: `#pragma braket result amplitude "01"`
  - 확률: `#pragma braket result probability q[0], q[1]`
- 기준 순환 결과 유형
  - 기대치: `#pragma braket result expectation x(q[0]) @ y([q1])`
  - 차이: `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
  - 샘플: `#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0은 OpenQASM 2.0과 역호환되므로 2.0을 사용하여 작성한 프로그램을 브라켓에서 실행할 수 있습니다. 하지만 브라켓에서 지원하는 OpenQASM 3.0의 기능에는 vs, vs 같은 사소한 구문 차이가 있습니다. `qreg creg qubit bit` 측정 구문에도 차이가 있으며 이러한 차이는 올바른 구문으로 지원되어야 합니다.

로컬 시뮬레이터의 OpenQASM에 대한 고급 기능 지원

브라켓의 QPU 또는 온디맨드 시뮬레이터의 일부로 제공되지 않는 고급 OpenQASM 기능을 `LocalSimulator` 지원합니다. 다음 기능 목록은 에서만 지원됩니다. `LocalSimulator`

- 게이트 모디파이어
- OpenQASM 빌트인 게이트
- 클래식 변수
- 클래식 오퍼레이션
- 커스텀 게이트
- 클래식 컨트롤
- QASM 파일
- 서브루틴

각 고급 기능의 예는 이 샘플 노트북을 참조하십시오. 전체 OpenQASM 사양은 OpenQASM 웹 사이트를 참조하십시오.

지원되는 연산 및 문법: OpenPulse

지원되는 OpenPulse 데이터 유형

Cal 블록:

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

데칼 블록:

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

프레임:

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

파형:

```
// 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};
```

커스텀 게이트 캘리브레이션 예제:

```
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;
```

임의 펄스 예제:

```
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);
```

```
}
```

예제 OpenQASM 3.0 퀀텀 태스크 생성 및 제출

Amazon브라켓 파이썬 SDK, Boto3 또는 를 사용하여 OpenQASM 3.0 퀀텀 태스크를 브라켓 AWS CLI 디바이스에 제출할 수 있습니다. Amazon

이 섹션:

- OpenQASM 3.0 프로그램 예시.
- Python SDK를 사용하여 OpenQASM 3.0 퀀텀 태스크를 생성하십시오
- Boto3를 사용하여 OpenQASM 3.0 퀀텀 태스크를 생성하십시오.
- 를 AWS CLI 사용하여 OpenQASM 3.0 작업을 생성할 수 있습니다.

OpenQASM 3.0 프로그램 예시.

OpenQASM 3.0 태스크를 생성하려면 다음 예와 같이 GHZ 상태를 준비하는 간단한 OpenQASM 3.0 프로그램 (ghz.qasm) 으로 시작하면 됩니다.

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

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

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

c = measure q;
```

Python SDK를 사용하여 OpenQASM 3.0 퀀텀 태스크를 생성하십시오

Amazon Amazon Braket Python SDK를 사용하여 다음 코드를 사용하여 브라켓 디바이스에 이 프로그램을 제출할 수 있습니다.

```
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,
)
```

Boto3를 사용하여 OpenQASM 3.0 퀀텀 태스크를 생성하십시오.

다음 예제와 같이 브라켓용AWS Python SDK (Boto3) 를 사용하여 OpenQASM 3.0 문자열을 사용하여 양자 작업을 생성할 수도 있습니다. 다음 코드 스니펫은 위와 같이 GHZ 상태를 준비하는 ghz.qasm을 참조합니다.

```
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,
)
```

를 AWS CLI 사용하여 OpenQASM 3.0 작업을 생성할 수 있습니다.

다음 예와 같이 AWS Command Line Interface (CLI) 를 사용하여 OpenQASM 3.0 프로그램을 제출할 수도 있습니다.

```
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)
    }'
```

다양한 브라켓 디바이스에서의 OpenQASM 지원

OpenQASM 3.0을 지원하는 장치의 경우 `action` 필드는 `GetDevice` 응답을 통해 새로운 작업을 지원합니다. 이는 및 장치에 대한 다음 예와 같습니다. Rigetti IonQ

```
//OpenQASM as available with the Rigetti device capabilities
{
    "braketSchemaHeader": {
```

```
        "name": "braket.device_schema.rigetti.rigetti_device_capabilities",
        "version": "1"
    },
    "service": {...},
    "action": {
        "braket.ir.jaqcd.program": {...},
        "braket.ir.openqasm.program": {
            "actionType": "braket.ir.openqasm.program",
            "version": [
                "1"
            ],
            ….
        }
    }
}

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

펄스 제어를 지원하는 장치의 경우 `pulse` 필드가 응답에 `GetDevice` 표시됩니다. 다음 예는 Rigetti 및 OQC 장치의 이 `pulse` 필드를 보여줍니다.

```
// 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
    }
  }
}
```

이전 필드에는 다음 내용이 자세히 설명되어 있습니다.

포트:

지정된 포트의 관련 속성 외에도 QPU에 선언된 미리 만들어진 외부 (`extern`) 장치 포트에 대해 설명합니다. 이 구조에 나열된 모든 포트는 사용자가 제출한 `OpenQASM 3.0` 프로그램 내에서 유효한 식별자로 미리 선언됩니다. 포트의 추가 속성은 다음과 같습니다.

- 포트 ID (포트 ID)
  - OpenQASM 3.0에서 식별자로 선언된 포트 이름입니다.
- 방향 (방향)
  - 포트의 방향. 드라이브 포트는 펄스 (방향 “tx”) 를 전송하고 측정 포트는 펄스 (방향 “rx”) 를 수신합니다.
- 포트 유형 (포트 유형)
  - 이 포트가 담당하는 동작 유형 (예: 드라이브, 캡처 또는 ff - 패스트 플럭스)
- Dt (dt)
  - 지정된 포트의 단일 샘플 타임스텝을 나타내는 시간 (초) 입니다.
- 큐비트 매핑 (큐비트 매핑)
  - 지정된 포트와 관련된 큐비트.
- 중심 주파수 (중앙 주파수)
  - 포트에서 사전 선언되거나 사용자가 정의한 모든 프레임에 대한 관련 중심 주파수 목록. 자세한 내용은 프레임을 참조하십시오.
- QHP 특정 속성 (SpecificPropertiesqhp)
  - QHP 전용 포트에 대한 기존 속성을 자세히 설명하는 선택적 맵입니다.

프레임:

QPU에 선언된 미리 만들어진 외부 프레임과 프레임에 대한 관련 속성에 대해 설명합니다. 이 구조에 나열된 모든 프레임은 사용자가 제출한 `OpenQASM 3.0` 프로그램 내에서 유효한 식별자로 미리 선언됩니다. 프레임의 추가 속성은 다음과 같습니다.

- 프레임 ID (프레임 ID)
  - OpenQASM 3.0에서 식별자로 선언된 프레임 이름입니다.
- 포트 ID (포트 ID)
  - 프레임에 연결된 하드웨어 포트입니다.
- 주파수 (주파수)
  - 프레임의 기본 초기 주파수입니다.

- 중심 주파수 (중앙 주파수)
  - 프레임 주파수 대역폭의 중심입니다. 일반적으로 프레임은 중앙 주파수 주변의 특정 대역폭으로만 조정할 수 있습니다. 따라서 주파수 조정은 중심 주파수의 지정된 델타 범위 내에서 유지되어야 합니다. 검증 매개변수에서 대역폭 값을 찾을 수 있습니다.
- 페이즈 (페이즈)
  - 프레임의 기본 초기 단계입니다.
- 어소시에이티드 게이트 (어소시에이티드 게이트)
  - 지정된 프레임과 관련된 게이트.
- 큐비트 매핑 (큐비트 매핑)
  - 주어진 프레임과 관련된 큐비트.
- QHP 특정 속성 (qhp) SpecificProperties
  - QHP 전용 프레임에 대한 기존 속성을 자세히 설명하는 선택적 맵입니다.

SupportsDynamic프레임:

OpenPulse`newframe`함수를 통해 `cal` 또는 `defcal` 블록에서 프레임을 선언할 수 있는지 여부를 설명합니다. 이 값이 false인 경우 프레임 구조에 나열된 프레임만 프로그램 내에서 사용할 수 있습니다.

SupportedFunctions:

지정된 OpenPulse 함수의 관련 인수, 인수 유형 및 반환 유형 외에도 장치에 지원되는 함수를 설명합니다. OpenPulse함수 사용 예를 보려면 OpenPulse사양을 참조하십시오. 현재 Braket은 다음을 지원합니다.

- 시프트_페이즈
  - 프레임의 위상을 지정된 값만큼 이동합니다.
- 세트_페이즈
  - 프레임의 위상을 지정된 값으로 설정합니다.
- 시프트_프리퀀시
  - 프레임의 주파수를 지정된 값만큼 이동합니다.
- 세트_프리퀀시
  - 프레임 주파수를 지정된 값으로 설정합니다.
- 플레이
  - 파형을 스케줄링합니다.

- 캡처_v0
  - 캡처 프레임의 값을 비트 레지스터로 반환합니다.

SupportedQhpTemplateWaveforms:

장치에서 사용할 수 있는 사전 빌드된 파형 함수와 관련 인수 및 유형에 대해 설명합니다. 기본적으로 Braket Pulse는 다음과 같은 모든 장치에서 사전 빌드된 파형 루틴을 제공합니다.

상수

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

τ는 파형의 길이이며 복소수입니다. `iq`

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

가우시안

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

τ는 파형의 길이, 는 가우스의 폭, σ 는 진폭입니다. `A` `ZaE`로 `True` 설정하면 가우시안이 오프셋되고 파형의 시작과 끝에서 0이 되고 최대값에 도달하도록 크기가 조정됩니다. `A`

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

드래그 가우시안

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

τ는 파형의 길이, 는 가우스의 폭, 자유 파라미터, σ 는 진폭입니다. β `A` `ZaE`로 `True` 설정하면 단열 게이트에 의한 미분 제거 (DRAG) 가우시안이 파형의 시작과 끝에서 0이 되고 실제 부분이 최대에 도달하도록 오프셋되고 배율이 조정됩니다. `A` DRAG 파형에 대한 자세한 내용은 약한 비선형 큐비트의 누설 제거를 위한 단순 펄스 논문을 참조하십시오.

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

SupportsLocalPulseElements:

포트, 프레임, 파형과 같은 펄스 요소를 블록 단위로 로컬로 정의할 수 있는지 여부를 설명합니다. `defcal` 값이 `false` 인 경우 요소를 블록 단위로 `cal` 정의해야 합니다.

SupportsNonNativeGatesWithPulses:

비네이티브 게이트를 펄스 프로그램과 함께 사용할 수 있는지 여부를 설명합니다. 예를 들어, 사용된 큐비트의 게이트 스루를 먼저 정의하지 않으면 프로그램의 H 게이트처럼 네이티브가 아닌 게이트를 `defcal` 사용할 수 없습니다. 디바이스 기능에서 네이티브 게이트 `nativeGateSet` 키 목록을 찾을 수 있습니다.

ValidationParameters:

다음을 포함하여 펄스 요소 검증 경계를 설명합니다.

- 파형의 최대 스케일/최대 진폭 값 (임의 및 사전 제작)
- 제공된 중심 주파수의 최대 주파수 대역폭 (Hz)
- 최소 펄스 길이/지속 시간 (초)
- 최대 펄스 길이/지속 시간 (초)

OpenQasm에서 지원되는 작업, 결과 및 결과 유형

각 장치가 지원하는 OpenQASM 3.0 기능을 확인하려면 장치 기능 출력의 `action` 필드에 있는 `braket.ir.openqasm.program` 키를 참조할 수 있습니다. 예를 들어 Braket State Vector 시뮬레이터에서 사용할 수 있는 지원되는 작업 및 결과 유형은 다음과 같습니다. SV1

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

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

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

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

OpenQASM 3.0으로 노이즈를 시뮬레이션하십시오.

OpenQASM3로 노이즈를 시뮬레이션하려면 pragma 명령어를 사용하여 노이즈 연산자를 추가합니다. 예를 들어, 이전에 제공된 GHZ 프로그램의 잡음이 많은 버전을 시뮬레이션하려면 다음 OpenQASM 프로그램을 제출할 수 있습니다.

```
// 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;
```

지원되는 모든 프라그마 노이즈 연산자의 사양은 다음 목록에 나와 있습니다.

```
#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
```

크라우스 오퍼레이터

크라우스 연산자를 생성하려면 행렬 목록을 반복하여 행렬의 각 요소를 복잡한 표현식으로 인쇄할 수 있습니다.

Kraus 연산자를 사용할 때는 다음 사항을 기억하십시오.

- 개수는 qubits 2를 초과할 수 없습니다. 스키마의 현재 정의는 이 제한을 설정합니다.
- 인수 목록의 길이는 8의 배수여야 합니다. 즉, 2x2 행렬로만 구성되어야 합니다.
- 총 길이는 2*num_qubits 행렬 2개를 초과할 수 없습니다. 즉, 1에는 행렬 4개, 2에는 16개 행렬이 있습니다. qubit qubits
- 제공된 모든 행렬은 완전 양성 추적 보존 (CPTP) 입니다.
- 전치 공액을 포함한 크라우스 연산자의 곱을 더하면 단위 행렬이 될 수 있습니다.

QubitOpenQASM 3.0으로 다시 배선

Amazon브라켓은 Rigetti 기기의 OpenQASM 내에서 물리적 qubit 표기법을 지원합니다 (자세한 내용은 이 페이지 참조). 간단한 재배선 qubits 전략으로 물리적 장치를 사용하는 경우 선택한 장치에 연결되어 있는지 확인하십시오. qubits 또는 qubit 레지스터를 대신 사용하는 경우 PARTIAL 재배선 전략이 디바이스에서 기본적으로 활성화됩니다. Rigetti

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

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

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

OpenQASM 3.0을 사용한 축어적 컴파일

Rigetti,OQC, 를 사용하여 양자 컴퓨터에서 양자 회로를 실행하면 컴파일러가 회로를 수정하지 않고 정의된 대로 정확하게 실행하도록 지시할 수 있습니다. IonQ 이 기능을 축어 컴파일이라고 합니다. Rigetti 장치를 사용하면 전체 회로 또는 특정 부분만 보존할 대상을 정확하게 지정할 수 있습니다. 회로의 특정 부분만 보존하려면 보존 영역 내에서 네이티브 게이트를 사용해야 합니다. 현재는 전체 회로에 대해 축어 OQC 컴파일만 지원하므로 회로의 모든 명령을 축어 상자에 넣어야 합니다. IonQ

OpenQASM을 사용하면 하드웨어의 저수준 컴파일 루틴에 의해 변경되지 않고 최적화되지 않은 코드 상자 주위에 축어적인 프래그마를 지정할 수 있습니다. 다음 코드 예제는 를 사용하는 방법을 보여줍니다. `#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;
```

축어 컴파일에 대한 자세한 내용은 Verbatim 컴파일 샘플 노트북을 참조하십시오.

브라켓 콘솔

OpenQASM 3.0 태스크를 사용할 수 있으며 브라켓 콘솔 내에서 관리할 수 있습니다. Amazon 콘솔에서는 기존 양자 작업을 제출할 때와 마찬가지로 OpenQASM 3.0에서 양자 작업을 제출한 경험이 동일합니다.

추가 리소스

OpenQASM은 모든 브라켓 지역에서 사용할 수 있습니다. Amazon

[브라켓에서 OpenQASM을 시작하기 위한 예제 노트북은 Amazon 브라켓 튜토리얼을 참조하십시오. GitHub]

OpenQASM 3.0을 사용한 그라디언트 컴퓨팅

Amazon Braket은 애드조인트 차별화 방법을 사용하여 온디맨드 및 로컬 시뮬레이터 모두에서 `shots=0` (정확한) 모드의 컴퓨팅 그래디언트를 지원합니다. 다음 예와 같이 적절한 프래그마를 제공하여 계산하려는 그래디언트를 지정할 수 있습니다.

```
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
```

모든 파라미터를 개별적으로 나열하는 대신 `all` 프래그마에서 지정할 수도 있습니다. 이렇게 하면 나열된 모든 `input` 매개변수에 대한 기울기가 계산됩니다. 이는 파라미터 수가 매우 많을 때 편리할 수 있습니다. 이 경우 프라그마는 다음 예와 같이 표시됩니다.

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

개별 연산자, 텐서 곱, 에르미트 옵저버블 등을 포함한 모든 옵저버블 유형이 지원됩니다. `Sum` 기울기를 계산하는 데 사용할 연산자는 `expectation()` 캡슐화기에 포함되어야 하며 각 항이 작용하는 큐비트를 지정해야 합니다.

OpenQasm 3.0으로 특정 큐비트를 측정합니다.

로컬 상태 벡터 시뮬레이터와 로컬 밀도 매트릭스 시뮬레이터는 회로 큐비트의 하위 집합을 측정할 수 있는 OpenQASM 프로그램 제출을 지원합니다. 이를 종종 부분 측정이라고 합니다. 예를 들어 다음 코드에서는 2큐비트 회로를 만들고 첫 번째 큐비트만 측정할 수 있습니다.

```
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];
"""
```

두 개의 큐비트가 `q[1]` 있지만 `q[0]` 여기서는 큐비트 0만 측정합니다. `b[0] = measure q[0]` 이제 로컬 상태 벡터 시뮬레이터에서 다음을 실행합니다.

```
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)
```

장치의 동작 속성에서 `requiresAllQubitsMeasurement` 필드를 검사하여 장치가 부분 측정을 지원하는지 여부를 확인할 수 있습니다. 지원되는 경우 부분 측정이 지원됩니다. `False`

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

여기서는 모든 큐비트를 측정할 필요는 없음을 나타냅니다. `requiresAllQubitsMeasurement` `False`

QuEra's Aquila를 사용하여 아날로그 프로그램을 제출하십시오.

이 페이지는 Aquila 머신의 기능에 대한 포괄적인 문서를 제공합니다. QuEra 여기에서 다루는 세부 사항은 다음과 같습니다. 1) 파라미터화된 해밀턴식 시뮬레이션Aquila, 2) AHS 프로그램 매개변수, 3)

AHS 결과 내용, 4) 기능 매개변수. Aquila Ctrl+F 텍스트 검색을 사용하여 질문과 관련된 매개변수를 찾는 것이 좋습니다.

해밀턴식

의 Aquila 기계는 기본적으로 다음과 같은 (시간에 따라 달라지는) 해밀턴 QuEra 동작을 시뮬레이션합니다.

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

> **Note**
>
> 로컬 디튜닝에 대한 액세스는 실험적 기능이며 요청 시 Braket Direct를 통해 이용할 수 있습니다.

여기서 각 항목은 다음과 같습니다.

- H drive,k (t) =($^{1}/_{2}$Ω (t) e $^{i(t)}$ S $_{-,k}$ + $^{1}/_{2}$Ω (t) e $^{-i(t) S) + (-Δ (t_{+,k})}$ n), global k
  - Ω (t) 는 시간에 따른 글로벌 구동 진폭 (Rabi 주파수라고도 함) 이며, 단위는 (rad/s) 입니다.
  - (t) 는 시간에 따른 글로벌 위상이며 라디안 단위로 측정됩니다.
  - $_{-,k}$S와 S는 $_{+,k}$ 원자 k의 스핀 하강 및 상승 연산자입니다 (기준 |↓# =|g#, |↑γ=|r ㅣ , S =|g ㅣ r|, S =(S) † =|r ㅣ g|) - + -
  - Δ (t) 는 global 시간에 따른 글로벌 디튜닝입니다.
  - $_{k}$n은 원자 k의 리드베리 상태에 대한 투영 연산자입니다 (예: n=|r ㅣ r|)
- local detuning,kH ($_{local}$t) =-Δ (t) h n $_{k}$ $_{k}$
  - Δ local (t) 는 로컬 주파수 이동의 시간 종속 인자로, 단위 (rad/s)
  - $_{k}$h는 사이트 종속 인자로, 0.0에서 1.0 사이의 무차원 수입니다.
- V vdw,k,l =C_6/(d) 6 n n, $_{k,l}$ k l
  - $_{6}$C는 (rad/s) * (m) ^6 단위로 표시되는 반 데르 발스 계수입니다.
  - $_{k,l}$d는 미터 단위로 측정한 원자 k와 l 사이의 유클리드 거리입니다.

사용자는 Braket AHS 프로그램 스키마를 통해 다음 파라미터를 제어할 수 있습니다.

- 쌍별 원자 거리 $_{k,l}$ d를 k, l=1,2,... N으로 제어하는 2차원 원자 배열 (각 원자의 x $_{k}$ 및 y $_{k}$ 좌표, um 단위, um 단위)

- Ω (t), 시간에 따른 글로벌 Rabi 주파수, 단위 (rad/s)
- (t), 시간에 따른 글로벌 위상, 단위 (rad)
- Δ_{global} (t), 시간 종속 글로벌 디튜닝, 단위 (rad/s)
- Δ_{local} (t), 로컬 디튜닝 규모의 시간 종속 (글로벌) 계수, 단위 (rad/s)
- h_k, 로컬 디튜닝 규모의 (정적) 사이트 종속 인자, 0.0에서 1.0 사이의 무차원 수

> **Note**
>
> 사용자는 관련된 수준 (예: S_-, S_+, n 연산자가 고정됨) 이나 Rydberg-Rydberg 상호 작용 계수 (C) 의 강도를 제어할 수 없습니다. [6]

브라켓 AHS 프로그램 스키마

Braket.ir.ahs.Program_v1.Program 객체 (예제)

```
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 (예제)

```
{
    "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]
                }
```

```
            }
        ]
    }
}
```

기본 필드

프로그램 필드	type	설명
setup.ahs_register.sites	목록 [목록 [십진수]]	핀셋이 원자를 가두는 2차원 좌표 목록
setup.ahs_register.filling	리스트 [int]	트랩 사이트를 차지하는 원자를 1로 표시하고 빈 사이트를 0으로 표시합니다.
해밀턴.DrivingFields [] .amplitude.time_series.times	목록 [십진수]	구동 진폭의 시점, 오메가 (t)
해밀턴. 드라이빙 필드 [] .amplitude.time_series.values	목록 [십진수]	구동 진폭 값, 오메가 (t)
해밀턴. 드라이빙 필드 [] .amplitude.pattern	str	구동 진폭의 공간 패턴, 오메가 (t), '균일'해야 함
해밀턴. 드라이빙 필드 [] .phase.time_series.times	목록 [십진수]	구동 단계의 시점, phi (t)
해밀턴. 드라이빙 필드 [] .phase.time_series.values	목록 [십진수]	구동 단계 값, phi (t)
해밀턴. 드라이빙 필드 []. 페이즈. 패턴	str	구동 단계의 공간 패턴, phi (t); '균일'해야 함
해밀턴. 드라이빙 필드 [] .detuning.time_series.times	목록 [십진수]	드라이빙 디튜닝 시점, 델타_글로벌 (t)

프로그램 필드	type	설명
해밀턴. 드라이빙 필드 [] .디튜닝.time_series.values	목록 [십진수]	드라이빙 디튜닝 값, 델타_글로벌 (t)
해밀턴. 드라이빙 필드 [] .디튜닝. 패턴	str	드라이빙 디튜닝의 공간 패턴, Delta_Global (t); '균일'이어야 함
해밀턴.LocaldeTuning [] .magnitude.time_series.times. times	목록 [십진수]	로컬 디튜닝 규모의 시간 종속 인자의 시점, Delta_Local (t)
해밀턴.LocalDetuning [] .magnitude.time_series.values	목록 [십진수]	로컬 디튜닝 규모의 시간 종속 인자 값인 델타_로컬 (t)
해밀턴.LocaldeTuning [] .magnitude.pattern	목록 [십진수]	로컬 디튜닝 규모의 사이트 종속 인자 h_k (값은 setup.ahs_register.sites의 사이트에 해당)

메타데이터 필드

프로그램 필드	type	설명
브래킷 이름 SchemaHeader	str	스키마 이름. 'braket.ir.ahs.program'이어야 합니다.
브라켓. 버전 SchemaHeader	str	스키마 버전

브라켓 AHS 태스크 결과 스키마

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.
AnalogHamiltonianSimulationQuantumTaskResult(예시)

```
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 (예시)

```
{
    "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
        }
    }
```

```
}
```

기본 필드

태스크 결과 필드	type	설명
측정 []. 샷 결과. 프리시퀀스	리스트 [int]	각 샷의 사전 시퀀스 측정 비트 (원자 사이트당 하나): 사이트가 비어 있으면 0, 사이트가 채워지면 1, 양자 진화를 실행하는 펄스 시퀀스 이전에 측정
측정 [] .샷 결과. 사후 시퀀스	리스트 [int]	각 샷의 사후 시퀀스 측정 비트: 원자가 Rydberg 상태이거나 위치가 비어 있는 경우 0, 원자가 바닥 상태인 경우 1, 양자 진화를 실행하는 펄스 시퀀스 끝에서 측정

메타데이터 필드

작업 결과 필드	type	설명
브래킷 이름 SchemaHeader	str	스키마 이름. 'braket.task_result.analog_hamiltonian_simulation_task_result'이어야 합니다.
브라켓. 버전 SchemaHeader	str	스키마 버전
태스크메타데이터.브라켓 .name SchemaHeader	str	스키마 이름. '브라켓.태스크_결과.태스크_

작업 결과 필드	type	설명
		메타데이터'여야 합니다.
태스크메타데이터.브라켓 .version SchemaHeader	str	스키마 버전
태스크메타데이터.id	str	퀀텀 태스크의 ID. AWS 양자 작업의 경우 이는 양자 작업 ARN입니다.
태스크메타데이터. 샷	int	퀀텀 태스크를 위한 샷 수
태스크메타데이터.샷. 디바이스 ID	str	양자 태스크가 실행된 디바이스의 ID. AWS 디바이스의 경우 이는 디바이스 ARN입니다.
태스크 메타데이터. 샷. 생성됨	str	생성 타임스탬프, 형식은 ISO-8601/RFC3339 문자열 형식 YYYY-MM-ddthh:mm:ss.sssz여야 합니다. 기본값은 없음입니다.

작업 결과 필드	type	설명
태스크 메타데이터.shots.end at	str	퀀텀 태스크가 종료된 시점의 타임스탬프. 형식은 ISO-8601/RFC3339 문자열 형식 YYYY-MM-ddthh:mm:ss.sssz 형식이어야 합니다. 기본값은 없음입니다.
태스크메타데이터. 샷. 상태	str	퀀텀 태스크 상태 (생성, 대기 중, 실행 중, 완료, 실패) 기본값은 없음입니다.
태스크 메타데이터. 샷. 실패 이유	str	퀀텀 태스크의 실패 이유. 기본값은 없음입니다.
추가 메타데이터. 액션	Braket.ir.ahs.Program_v1.Program	(브라켓 AHS 프로그램 스키마 섹션 참조)
추가 메타데이터.액션.브라켓Schema Header. 쿼리 메타데이터.이름	str	스키마 이름. 'braket.task_result.quera_metadata '여야 합니다.

작업 결과 필드	type	설명
추가 SchemaHeader 메타데이터.Act ion.Braket .querMetadata.version	str	스키마 버전
추가 메타데이터.Action.num SuccessfulShots	int	완전히 성공한 샷 수. 요청된 샷 수와 같아야 합니다.
측정 []. 샷 메타데이터. 샷 상태	int	샷 상태 (성공, 부분 성공, 실패) 는 “성공”이어야 합니다.

QuEra 디바이스 속성 스키마

braket.device_schema.quera.quera_device_capabilities_v1. QueraDevice기능 (예시)

```
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 (예시)

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

```
}
```

서비스 속성 필드

서비스 속성 필드	type	설명
서비스. 실행 Windows []. 실행일	ExecutionDay	실행 기간의 요일은 '매일', '평일', '주말', '월요일', '화요일', '수요일', '목요일', '금요일', '토요일' 또는 '일요일'이어야 합니다.
서비스. 실행창 [] .window StartHour	날짜/시간.	실행 창이 시작되는 시간의 UTC 24시간 형식
서비스. 실행윈도우 [] .window EndHour	날짜/시간.	실행 기간이 종료되는 시간을 UTC 24시간 형식으로 표시합니다.
Service.qpu_Capabilities.Service.ShotsRange	튜플 [int, int]	디바이스의 최소 및 최대 샷 수
서비스.QPU_기능.서비스.장치비용.가격	float	디바이스 가격 (미국 달러 기준)
서비스.QPU_기능. 서비스.장치비용.단위	str	가격 청구 단위 (예: '분', '시간', '샷', '작업')

메타데이터 필드

메타데이터 필드	type	설명
액션 [] .버전	str	AHS 프로그램 스키마 버전
액션 []. 액션 유형	ActionType	AHS 프로그램 스키마 이름. 'braket.ir.ahs.program'이어야 합니다.

메타데이터 필드	type	설명
SchemaHeader서비스. 브라켓 .이름	str	스키마 이름. 'braket.device_schema.device_service_properties'여야 합니다.
SchemaHeader서비스.브라켓 .version	str	스키마 버전
서비스. 장치 설명서. 이미지 URL	str	디바이스 이미지의 URL
서비스. 장치 설명서. 요약	str	장치에 대한 간략한 설명
서비스. 장치 설명서. 외부 DocumentationUrl	str	외부 설명서 URL
서비스. 장치 위치	str	디바이스의 지리적 위치
서비스. 업데이트 날짜	datetime	디바이스 속성이 마지막으로 업데이트된 시간

Boto3로 작업하기

Boto3는 파이썬을 AWS 위한 SDK입니다. Boto3를 사용하면 Python 개발자가 Braket과 같은 Amazon 것을 만들고, 구성하고 AWS 서비스, 관리할 수 있습니다. Boto3는 Braket에 대한 저수준 액세스뿐만 아니라 객체 지향 API 액세스를 제공합니다. Amazon

Boto3 퀵스타트 가이드의 지침에 따라 Boto3 설치 및 구성 방법을 알아보십시오.

Boto3는 Amazon Braket Python SDK와 함께 작동하는 핵심 기능을 제공하여 양자 작업을 구성하고 실행하는 데 도움이 됩니다. Python 고객은 항상 Boto3를 설치해야 합니다. Boto3가 핵심 구현이기 때문입니다. 추가 도우미 메서드를 사용하려면 Braket SDK도 설치해야 합니다Amazon.

예를 들어`CreateQuantumTask`, 호출하면 Amazon Braket SDK가 요청을 Boto3에 제출하고 Boto3가 요청을 제출한 다음 Boto3를 호출합니다. AWS API

이 섹션:

아마존 브라켓 Boto3 클라이언트를 켜세요

Boto3를 Amazon Braket과 함께 사용하려면 Boto3를 가져온 다음 Broket에 연결하는 데 사용할 클라이언트를 정의해야 합니다. Amazon API 다음 예제에서는 Boto3 클라이언트의 이름을 지정합니다. `braket`

> **Note**
>
> 이전 버전과의 이전 버전과의 호환성을 위해 OpenQASM 정보는 호출에서 생략됩니다. BraketSchemas `GetDevice` API 이 정보를 얻으려면 사용자 에이전트가 최신 버전 (1.8.0 이상) 을 제시해야 합니다 BraketSchemas . Braket SDK는 이를 자동으로 보고합니다. Braket SDK를 사용할 때 `GetDevice` 응답에 OpenQASM 결과가 표시되지 않는 경우 `AWS_EXECUTION_ENV` 환경 변수를 설정하여 사용자 에이전트를 구성해야 할 수 있습니다. AWS CLI, Boto3, Go, Java,/SDK에서 이 작업을 수행하는 방법은 OpenQASM 결과를 GetDevice 반환하지 않음 오류 항목에 제공된 코드 예제를 참조하십시오. JavaScript TypeScript

```
import boto3
import botocore

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

이제 `braket` 클라이언트가 설정되었으므로 Braket 서비스에서 요청을 하고 응답을 처리할 수 있습니다. Amazon 요청 및 응답 데이터에 대한 자세한 내용은 API Reference에서 확인할 수 있습니다.

다음 예제는 장치 및 양자 작업을 사용하는 방법을 보여줍니다.

- 디바이스 검색
- 기기 검색
- 퀀텀 태스크 생성하기
- 양자 태스크 불러오기

- 양자 태스크 검색
- 양자 태스크 취소

디바이스 검색

- `search_devices(**kwargs)`

지정된 필터를 사용하여 장치를 검색합니다.

```
# 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'])
```

기기 검색

- `get_device(deviceArn)`

AmazonBraket에서 사용할 수 있는 장치를 검색하십시오.

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

퀀텀 태스크 생성하기

- `create_quantum_task(**kwargs)`

퀀텀 태스크 생성하기.

```
# 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")
```

양자 태스크 불러오기

- `get_quantum_task(quantumTaskArn)`

지정된 양자 태스크를 검색합니다.

```
# 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'])
```

양자 태스크 검색

- `search_quantum_tasks(**kwargs)`

지정된 필터 값과 일치하는 양자 작업을 검색합니다.

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

양자 태스크 취소

- `cancel_quantum_task(quantumTaskArn)`

지정된 양자 작업을 취소합니다.

```
# Pass the quantum task ARN when sending the request and capture the response
response = braket.cancel_quantum_task(quantumTaskArn='arn:aws:braket:us-
west-1:123456789012:quantum-task/ce78c429-cef5-45f2-88da-123456789012')

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

Boto3 및 아마존 브라켓 SDK용 AWS CLI 프로필 구성

AmazonBraket SDK는 명시적으로 달리 지정하지 않는 한 기본 AWS CLI 자격 증명을 사용합니다. 노트북 인스턴스를 시작할 권한이 있는 IAM 역할을 제공해야 하므로 관리형 Amazon Braket 노트북에서 실행할 때는 기본값을 유지하는 것이 좋습니다.

선택적으로 코드를 로컬 (예: Amazon EC2 인스턴스) 에서 실행하는 경우 명명된 AWS CLI 프로필을 설정할 수 있습니다. 기본 프로필을 정기적으로 덮어쓰지 않고 각 프로필에 다른 권한 집합을 부여할 수 있습니다.

이 섹션에서는 이러한 `profile` CLI를 구성하는 방법과 해당 프로필을 Amazon Braket에 통합하여 해당 프로필의 권한으로 API 호출이 이루어지도록 하는 방법에 대해 간략하게 설명합니다.

이 섹션:

- 1단계: 로컬 구성 AWS CLIprofile
- 2단계: Boto3 세션 개체 설정
- 3단계: Boto3 세션을 브라켓에 통합 AwsSession

1단계: 로컬 구성 AWS CLI**`profile`**

사용자를 만드는 방법과 기본이 아닌 프로필을 구성하는 방법을 설명하는 것은 이 문서의 범위를 벗어납니다. 이러한 주제에 대한 자세한 내용은 다음을 참조하십시오.

- 시작하기
- AWS CLI 사용하도록 구성 AWS IAM Identity Center

AmazonBraket을 사용하려면 이 사용자 및 관련 `profile` CLI에 필요한 Braket 권한을 제공해야 합니다. 예를 들어 정책을 첨부할 수 있습니다. AmazonBraketFullAccess

2단계: Boto3 세션 개체 설정

Boto3 세션 객체를 설정하려면 다음 코드 예제를 활용하십시오.

```
from boto3 import Session

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

> **Note**
>
> 예상 API 통화에 `profile` 기본 지역과 일치하지 않는 지역 기반 제한이 있는 경우 다음 예와 같이 Boto3 세션의 지역을 지정할 수 있습니다.

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

로 지정된 인수의 경우 Amazon Braket을 사용할 수 있는 값 중 하나에 해당하는 값 (예:`region`, 등`us-east-1`) 을 대체하십시오. AWS 리전 `us-west-1`

3단계: Boto3 세션을 브라켓에 통합 AwsSession

다음 예제는 Boto3 Braket 세션을 초기화하고 해당 세션에서 기기를 인스턴스화하는 방법을 보여줍니다.

```
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)
```

이 설정이 완료되면 예를 들어 명령을 호출하여 인스턴스화된 `AwsDevice` 객체에 양자 작업을 제출할 수 있습니다. `device.run(…)` 해당 디바이스에서 이루어진 모든 API 호출은 이전에 지정한 CLI 프로파일과 연결된 IAM 자격 증명을 활용할 수 있습니다. `profile`

Amazon Braket의 펄스 컨트롤

이 섹션에서는 Amazon Braket의 다양한 QPU에서 펄스 제어를 사용하는 방법을 설명합니다.

이 섹션:

브라켓 펄스

펄스는 양자 컴퓨터의 큐비트를 제어하는 아날로그 신호입니다. Amazon Braket의 특정 디바이스에서는 펄스 제어 기능에 액세스하여 펄스를 사용하여 회로를 제출할 수 있습니다. 브라켓 SDK를 통해, OpenQASM 3.0을 사용하거나, 브라켓 API를 통해 직접 펄스 제어에 액세스할 수 있습니다. 먼저 브라켓의 펄스 제어에 대한 몇 가지 주요 개념을 소개해 보겠습니다.

Frames(프레임)

프레임은 양자 프로그램 내에서 클록과 위상 역할을 모두 수행하는 소프트웨어 추상화입니다. 클럭 시간은 각 사용량과 주파수로 정의된 스테이트풀 캐리어 신호에 따라 증가합니다. 큐비트로 신호를 전송할 때 프레임은 큐비트의 반송파 주파수, 위상 오프셋 및 파형 포락선이 방출되는 시간을 결정합니다. Braket Pulse에서 프레임을 구성하는 것은 장치, 주파수 및 위상에 따라 달라집니다. 기기에 따라 사전 정의된 프레임을 선택하거나 포트를 제공하여 새 프레임을 인스턴스화할 수 있습니다.

```
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)
```

포트

포트는 큐비트를 제어하는 입력/출력 하드웨어 구성 요소를 나타내는 소프트웨어 추상화입니다. 이를 통해 하드웨어 공급업체는 사용자가 상호 작용하여 큐비트를 조작하고 관찰할 수 있는 인터페이스를

제공할 수 있습니다. 포트는 커넥터 이름을 나타내는 단일 문자열로 구분됩니다. 또한 이 문자열은 파형을 얼마나 세밀하게 정의할 수 있는지를 지정하는 최소 시간 증분을 나타냅니다.

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

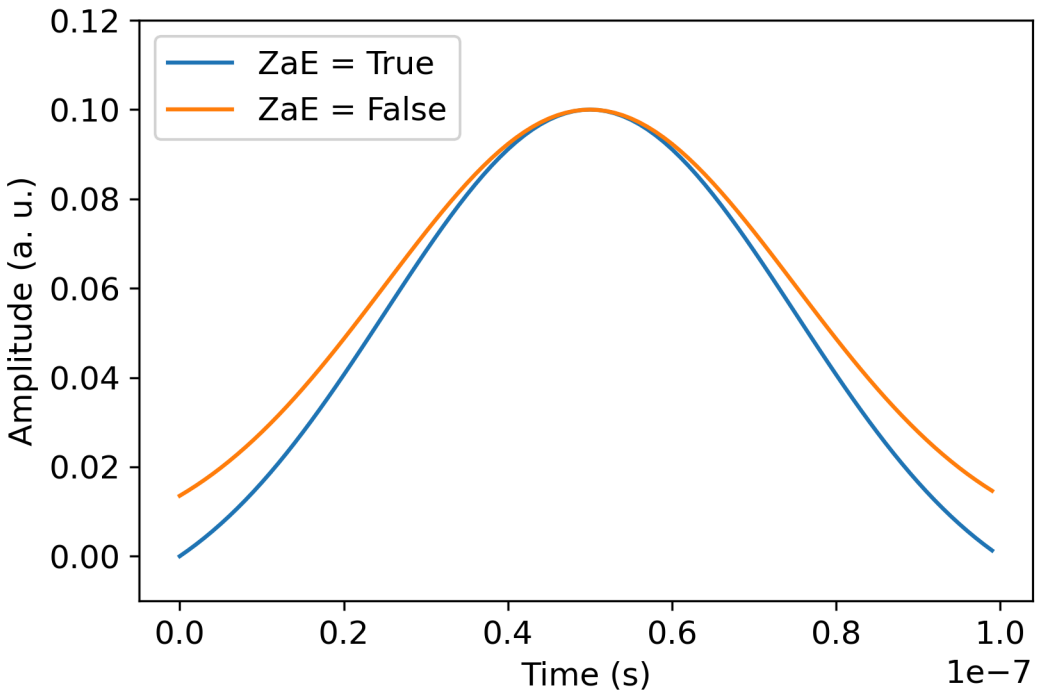
파형

파형은 시간에 따라 달라지는 엔벨로프로, 출력 포트에서 신호를 방출하거나 입력 포트를 통해 신호를 캡처하는 데 사용할 수 있습니다. 복소수 목록을 통해 직접 파형을 지정하거나 파형 템플릿을 사용하여 하드웨어 제공업체로부터 목록을 생성하여 파형을 지정할 수 있습니다.

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

Braket Pulse는 상수 파형, 가우스 파형, 단열 게이트에 의한 미분 제거 (DRAG) 파형을 포함한 표준 파형 라이브러리를 제공합니다. 다음 예제와 같이 함수를 통해 파형 데이터를 검색하여 파형의 모양을 그릴 수 있습니다. `sample`

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



위 이미지는 에서 생성된 가우스 파형을 보여줍니다. `GaussianWaveform` 펄스 길이는 100ns, 너비는 25ns, 진폭은 0.1 (임의 단위) 로 선택했습니다. 파형은 펄스 창 중앙에 위치합니다. `GaussianWaveform`부울 인수 `zero_at_edges` (범례에서는 zAe) 를 받아들입니다. 로 `True` 설정하면 이 인수는 t=0과 t=에 있는 점이 0이 되도록 가우스 파형을 오프셋하고 최대값이 인수와 일치하도록 진폭을 재조정합니다. `length` `amplitude`

이제 펄스 레벨 액세스의 기본 개념을 다루었으니 이제 게이트와 펄스를 사용하여 회로를 구성하는 방법을 살펴보겠습니다.

프레임과 포트의 역할

이 섹션에서는 각 장치에 사용할 수 있는 사전 정의된 프레임 및 포트에 대해 설명합니다. 또한 특정 프레임에서 펄스가 재생될 때 관련된 메커니즘에 대해서도 간략하게 설명합니다.

리게티

Frames(프레임)

Rigetti디바이스는 관련 큐비트와 공진하도록 주파수 및 위상이 보정된 사전 정의된 프레임을 지원합니다. `q{i}[_q{j}]_{role}_frame`여기서 명명 규칙은 첫 번째 큐비트 번호를 `{i}` 참조하고, 프레임

이 2-큐비트 상호 작용을 활성화하는 경우 두 번째 큐비트 번호를 나타내며, 프레임의 역할을 {j} 나타냅니다. {role} 역할은 다음과 같습니다.

- rf큐비트의 0-1 전환을 구동하는 프레임입니다. 펄스는 이전에 및 함수를 통해 제공된 주파수 및 위상의 마이크로파 과도 신호로 전송됩니다. set shift 시간에 따른 신호 진폭은 프레임에서 재생되는 파형에 따라 결정됩니다. 프레임은 단일 큐비트의 비대각선 상호 작용을 연결합니다. 자세한 내용은 Krantz 등을 참조하십시오. 그리고 라하밈 외. .
- rf_f12은 (와) rf 비슷하며 파라미터는 1-2 트랜지션을 대상으로 합니다.
- ro_rx연결된 동일 평면 도파관을 통해 큐비트를 분산적으로 판독하는 데 사용됩니다. 판독 파형의 주파수, 위상 및 전체 파라미터 세트는 사전 보정됩니다. 현재는 를 통해 사용되며 프레임 식별자 capture_v0 이외의 인수는 필요하지 않습니다.
- ro_tx공진기에서 신호를 전송하는 데 사용됩니다. 현재는 사용되지 않습니다.
- cz2-큐비트 게이트를 활성화하도록 보정된 프레임입니다. cz 포트와 관련된 모든 프레임과 마찬가지로, ff 포트 쌍과의 공진 시 조정 가능한 큐비트를 변조하여 플럭스 라인을 통해 얽힌 상호 작용을 활성화합니다. 얽힘 메커니즘에 대한 자세한 내용은 Reagor et al. 을 참조하십시오. , 콜드웰 외. , 디디에 외. .
- cphase2-큐비트 cphaseshift 게이트를 활성화하도록 보정된 프레임이며 포트에 연결되어 있습니다. ff 얽힘 메커니즘에 대한 자세한 내용은 프레임 설명을 참조하십시오. cz
- xy2큐비트 XY (β) 게이트를 사용할 수 있도록 보정된 프레임이며 포트에 연결되어 있습니다. ff 얽힘 메커니즘 및 XY 게이트 구현 방법에 대한 자세한 내용은 프레임 설명 및 Abrams et al. 을 참조하십시오. cz .

ff포트 기반 프레임이 조정 가능한 큐비트의 주파수를 이동함에 따라 큐비트와 관련된 다른 모든 구동 프레임은 진폭 및 주파수 이동 지속 시간과 관련된 양만큼 디페이즈됩니다. 따라서 인접 큐비트의 프레임에 해당하는 위상 변이를 추가하여 이 효과를 보정해야 합니다.

포트

Rigetti디바이스는 디바이스 기능을 통해 검사할 수 있는 포트 목록을 제공합니다. 포트 이름은 규약을 따릅니다. q{i}_{type} 여기서 {i} 포트 이름은 큐비트 번호를 {type} 나타내며 포트 유형을 나타냅니다. 모든 큐비트에 완전한 포트 세트가 있는 것은 아니라는 점에 유의하십시오. 포트 유형은 다음과 같습니다.

- rf단일 큐비트 전환을 주도하는 기본 인터페이스를 나타냅니다. 및 프레임과 연결되어 있습니다rf. rf_f12 큐비트에 용량 방식으로 연결되어 기가헤르츠 범위의 마이크로파 구동이 가능합니다.

- `ro_tx`큐비트에 용량 방식으로 연결된 판독 공진기에 신호를 전송하는 역할을 합니다. 판독 신호 전달은 팔각형으로 8배 멀티플렉싱됩니다.
- `ro_rx`큐비트에 연결된 리드아웃 공진기로부터 신호를 수신하는 역할을 합니다.
- `ff`큐비트에 유도 방식으로 연결된 고속 플럭스 라인을 나타냅니다. 이를 사용하여 트랜스미온의 주파수를 조정할 수 있습니다. 튜닝 가능성이 높도록 설계된 큐비트에만 포트가 있습니다. `ff` 이 포트는 인접 트랜스 몬의 각 쌍 사이에 정적 용량 결합이 있기 때문에 큐비트-큐비트 상호 작용을 활성화하는 역할을 합니다.

아키텍처에 대한 자세한 내용은 Valery et al. 을 참조하십시오. .

표준 품질

Frames(프레임)

OQC디바이스는 관련 큐비트와 공진하도록 주파수 및 위상이 보정된 사전 정의된 프레임을 지원합니다. 이러한 프레임의 명명 규칙은 다음과 같습니다.

- 구동 프레임: `q{i}[_q{j}]_{role}` 여기서 `{i}` 는 첫 번째 큐비트 번호를 `{j}` 의미하고, 프레임이 2-큐비트 상호 작용을 활성화하는 경우 두 번째 큐비트 번호를 `{role}` 의미하며, 아래에 설명된 프레임의 역할을 나타냅니다.
- 큐비트 리드아웃 프레임: `r{i}_{role}` 여기서 `{i}` 말하는 프레임은 큐비트 넘버를 말하며, 아래 설명과 같이 프레임의 역할을 `{role}` 나타냅니다.

다음과 같이 각 프레임을 설계된 역할에 맞게 사용하는 것이 좋습니다.

- `drive`큐비트의 0-1 전환을 구동하는 메인 프레임으로 사용됩니다. 펄스는 이전에 및 함수를 통해 제공된 주파수 및 위상의 마이크로파 과도 신호로 전송됩니다. `set shift` 시간에 따른 신호 진폭은 프레임에서 재생되는 파형에 따라 결정됩니다. 프레임은 단일 큐비트의 비대각선 상호 작용을 연결합니다. 자세한 내용은 Krantz 등을 참조하십시오. 그리고 라하밈 외. .
- `second_state`drive프레임과 동일하지만 주파수는 1-2 트랜지션에서의 공진에 따라 조정됩니다.
- `measure`판독용입니다. 판독 파형의 주파수, 위상 및 전체 파라미터 세트는 사전 보정됩니다. 현재는 를 통해 사용되고 `capture_v0` 있으며 프레임 식별자 이외의 인수는 필요하지 않습니다.
- `acquire`공진기에서 신호를 캡처하는 데 사용됩니다. 현재는 사용되지 않습니다.
- `cross_resonance`대상 큐비트의 전환 주파수에서 제어 큐비트를 `i j` 구동하여 큐비트 간의 교차 공명 상호 작용을 활성화합니다. `i j` 따라서 프레임 주파수는 대상 큐비트의 주파수를 사용하여 설

정됩니다. 상호 작용은 이 교차 공진 드라이브의 진폭에 비례하는 속도로 발생합니다. 크로스토크 유형이 다르면 원치 않는 효과가 발생하므로 수정이 필요합니다. 패터슨 외 참조. 동축 모양의 트랜스몬 큐비트 ('coaxmon') 와의 교차 공명 상호 작용에 대한 자세한 내용은 여기를 참조하십시오.

- `cross_resonance_cancellation`교차 공명 상호 작용이 활성화될 때 크로스토크로 인한 유해한 영향을 억제하기 위한 수정을 추가할 수 있습니다. 초기 프레임 주파수는 제어 큐비트의 전환 주파수로 설정됩니다. `i` 취소 방법에 대한 자세한 내용은 Patterson 등을 참조하십시오. .

포트

OQC디바이스는 디바이스 기능을 통해 검사할 수 있는 포트 목록을 제공합니다. 앞서 설명한 프레임은 ID로 식별되는 포트와 연결됩니다. `channel_{N}` 여기서 `{N}` 0은 정수입니다. 포트는 코액스몬에 연결된 제어 라인 (방향`tx`) 및 판독 공진기 (방향`rx`) 에 대한 인터페이스입니다. 각 큐비트는 하나의 제어 라인과 하나의 판독 공진기에 연결됩니다. 전송 포트는 단일 큐비트 및 2-큐비트 조작을 위한 인터페이스입니다. 수신 포트는 큐비트 판독에 사용됩니다.

헬로 펄스

여기서는 펄스로 직접 간단한 벨 페어를 구성하고 Rigetti 장치에서 이 펄스 프로그램을 실행하는 방법을 알아봅니다. 벨 페어는 첫 번째 큐비트의 하다마드 게이트와 첫 번째 큐비트와 두 번째 큐비트 사이의 `cnot` 게이트로 구성된 2큐비트 회로입니다. 펄스로 얽힌 상태를 만들려면 하드웨어 유형과 장치 아키텍처에 따라 달라지는 특정 메커니즘이 필요합니다. 기본 메커니즘을 사용하여 게이트를 만들지는 않겠습니다. `cnot` 대신 `cz` 게이트를 기본적으로 활성화하는 특정 파형과 프레임을 사용하겠습니다. 이 예제에서는 단일 큐비트 네이티브 게이트를 사용하여 Hadamard 게이트를 만들고 펄스를 사용하여 게이트를 `rx` 표현합니다. `rz` `cz`

먼저 필요한 라이브러리를 가져와 보겠습니다. 이제 `Circuit` 클래스 외에도 클래스도 가져와야 합니다. `PulseSequence`

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

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

다음으로, 디바이스의 Amazon 리소스 이름 (ARN) 을 사용하여 새 Braket 디바이스를 인스턴스화합니다. Rigetti Aspen-M-3 디바이스의 레이아웃을 보려면 Amazon Braket 콘솔의 Rigetti Aspen-M-3 디바이스 페이지를 참조하십시오.

```
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")
```

Hadamard 게이트는 Rigetti 디바이스의 기본 게이트가 아니므로 펄스와 함께 사용할 수 없습니다. 따라서 이를 및 네이티브 게이트의 시퀀스로 분해해야 합니다. `rx` `rz`

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

`cz`게이트의 경우 캘리브레이션 단계에서 하드웨어 공급업체가 미리 정한 파라미터 (진폭, 상승/하강 시간, 지속 시간) 가 포함된 임의 파형을 사용합니다. 이 파형은 에 적용됩니다. `q10_q113_cz_frame` 여기에 사용된 임의 파형의 최신 버전은 웹 사이트의 QCS를 참조하십시오. Rigetti QCS 계정을 만들어야 할 수도 있습니다.

```
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, 0.00543941944632366, 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,
```

```
 0.4122811110344471, 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')
```

그러면 다음과 같은 결과가 반환됩니다.

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

이제 방금 정의한 파형을 사용하여 cz 게이트를 구성할 수 있습니다. 제어 큐비트가 해당 상태에 있는 경우 cz 게이트는 대상 큐비트의 위상 플립으로 구성된다는 점을 기억하세요. |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_wfm파형은 패스트 플럭스 포트에 연결된 프레임에서 재생됩니다. 그 역할은 큐비트 주파수를 이동시켜 큐비트-큐비트 상호 작용을 활성화하는 것입니다. 자세한 내용은 프레임 및 포트의 역할을 참조하십시오. 주파수가 변함에 따라 큐비트 프레임은 그대로 유지되는 단일 큐비트 프레임과 다른 속도로 회전합니다. 후자의 큐비트 rf 프레임은 디페이즈되고 있습니다. 이러한 위상 변화는 사전에 Ramsey 시퀀스를 통해 보정되었으며, 여기서는 및 (전체 기간) 를 통해 하드코딩된 정보로 제공됩니다. phase_shift_a phase_shift_b 프레임에 있는 지침을 사용하여 이러한 디페이징을 수정합니다. shift_phase rf 이 시퀀스는 a 큐비트와 관련된 XY 프레임이 없고 사용되는 프로그램에서만 작동한다는 점에 유의하십시오. 이러한 프레임에서 발생하는 위상 변이를 보상하지 않기 b 때문입니다. 이것은 rf 및 cz 프레임만 사용하는 이 단일 벨 페어 프로그램의 경우입니다. 자세한 내용은 콜드웰 등을 참조하십시오. .

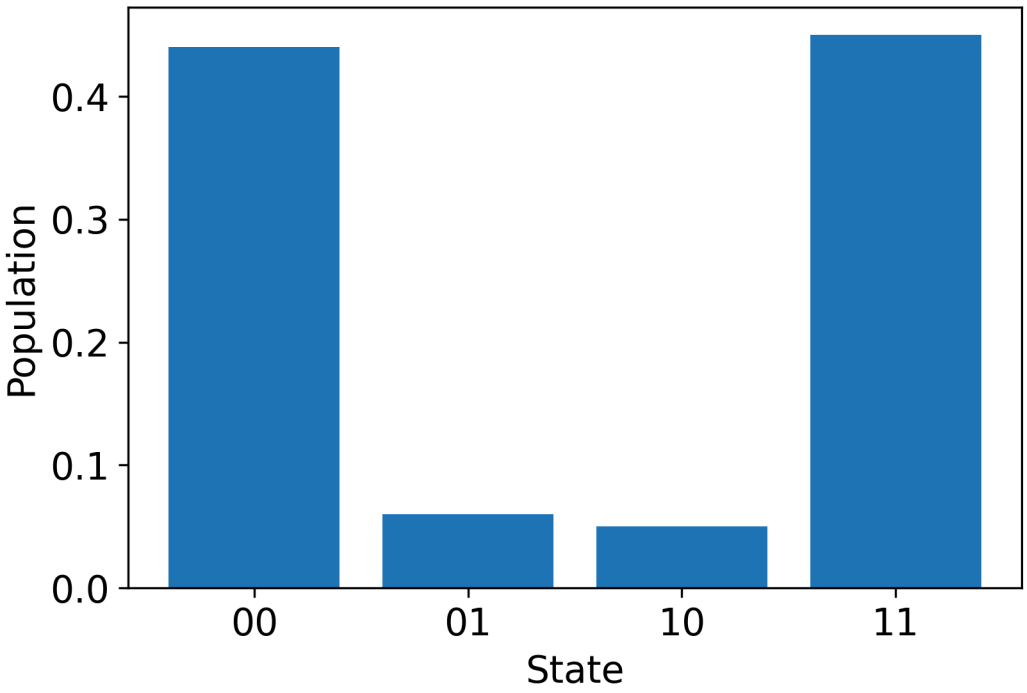
이제 펄스가 있는 벨 페어를 만들 준비가 되었습니다.

```
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    |
```

이 벨 페어를 Rigetti 기기에서 실행해 봅시다. 이 코드 블록을 실행하면 요금이 부과된다는 점에 유의하세요. 이러한 비용에 대한 자세한 내용은 Amazon Braket 가격 책정 페이지를 참조하십시오. 샷 수를 늘리기 전에 소량의 샷을 사용하여 회로를 테스트하여 디바이스에서 회로를 실행할 수 있는지 확인하는 것이 좋습니다.

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

counts = task.result().measurement_counts

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



헬로 펄스 사용 OpenPulse

OpenPulse일반 양자 장치의 펄스 레벨 제어를 지정하는 데 사용되는 언어이며 OpenQASM 3.0 사양의 일부입니다. Amazon Braket은 OpenPulse OpenQASM 3.0 표현을 사용하여 펄스를 직접 프로그래밍할 수 있도록 지원합니다.

Braket은 네이티브 OpenPulse 명령어로 펄스를 표현하기 위한 기본 중간 표현으로 사용합니다. OpenPulse`defcal`(“캘리브레이션 정의”의 줄임말) 선언의 형태로 명령 캘리브레이션을 추가할 수 있도록 지원합니다. 이러한 선언을 사용하여 하위 수준의 제어 문법 내에서 게이트 명령의 구현을 지정할 수 있습니다.

이 예제에서는 OpenQASM 3.0을 사용하고 주파수 조정 가능한 트랜스몬을 사용하는 OpenPulse 기기에서 Bell 회로를 구성해 보겠습니다. Bell 회로는 첫 번째 큐비트의 Hadamard 게이트와 두 큐비트 사이의 게이트로 구성된 2큐비트 회로라는 점을 기억하세요. `cnot` `cnot`게이트는 기저 변환을 통해서만 `cz` 게이트와 다르므로 여기서는 Hadamard와 gates를 대신 사용하여 벨 쌍을 정의하겠습니다. 이 데모에서는 장치가 `cz` 게이트를 생성하는 더 간단한 방법을 제공하므로 여기서는 Hadamard와 gates를 대신 사용하여 벨 쌍을 정의하겠습니다. `cz`

먼저 디바이스의 네이티브 게이트를 사용하여 Hadamard 게이트를 정의해 보겠습니다.

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

```
defcal h $10 {
    rz(pi) $10;
    rx(pi/2) $10;
    rz(pi/2) $10;
    rx(-pi/2) $10;
}
defcal h $113 {
    rz(pi) $113;
    rx(pi/2) $113;
    rz(pi/2) $113;
    rx(-pi/2) $113;
}
```

CZ 게이트의 경우 사전에 미리 결정된 매개변수 (진폭, 상승/하강 시간 및 지속 시간) 가 포함된 임의의 파형을 사용합니다. `q10_q113_cz_frame`이 파형은 에 적용됩니다.

```
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,
 0.484438680618381 7, 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, 0.48443981200754405, 0.484439812007528 4,
    0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
    0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.484432196472809 6,
    0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.481411707540660 3,
    0.4753811409710734, 0.46121311095968553, 0.4331554251320285, 0.38631750509562957,
    0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
```

```
 0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
 0.0008644760431374626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
}
defcal cz $10, $113 {
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
    play(q10_q113_cz_frame, q10_q113_cz_wfm);
    delay[124ns] q10_rf_frame;
    shift_phase(q10_rf_frame, 1.1733407221086924);
    delay[124ns] q113_rf_frame;
    shift_phase(q113_rf_frame, 6.269846678712192);
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
}
```

q10_q113_cz_wfm파형의 지속 시간은 124개 샘플이며, 이는 최소 시간 증분이 1ns이므로 124ns에 해당합니다. dt

q10_q113_cz_wfm파형은 패스트 플럭스 포트에 바인딩된 프레임에서 재생됩니다. 큐비트 주파수를 이동시켜 큐비트-큐비트 상호 작용을 활성화하는 역할을 합니다. 자세한 내용은 프레임 및 포트의 역할을 참조하십시오. 주파수가 변함에 따라 큐비트 프레임은 그대로 유지되는 단일 큐비트 프레임과 다른 속도로 회전합니다. 후자의 큐비트 rf 프레임은 디페이즈되고 있습니다. 이러한 디페이징은 캘리브레이션 단계에서 Ramsey 시퀀스를 사용하여 측정할 수 있으며 명령어 및 프레임으로 보정할 수 있습니다. shift_phase rf xy 자세한 내용은 Caldwell 등을 참조하십시오. .

이제 Hadamard와 cnot 게이트 두 개를 사용하여 게이트를 분해한 벨 페어 회로를 실행할 수 있습니다. cz

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

네이티브 게이트와 펄스의 조합을 사용하여 구성된 Bell 회로의 전체 OpenQASM 3.0 표현은 다음과 같습니다.

```
// 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, 0.461213110959685 9, 0.4753811409710734, 0.4814117075406603,
0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
 0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
 0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.484432196472809б,
 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;
```

이제 다음 코드를 사용하여 Braket SDK를 사용하여 디바이스에서 이 OpenQASM 3.0 프로그램을 실행할 수 있습니다. Rigetti

```
# 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,
)
```

펄스를 사용하여 네이티브 게이트에 액세스합니다.

연구자들은 특정 QPU에서 지원하는 네이티브 게이트가 어떻게 펄스로 구현되는지 정확히 알아야 하는 경우가 많습니다. 펄스 시퀀스는 하드웨어 제공업체가 세심하게 보정하지만, 연구자는 펄스 시퀀스에 액세스하여 더 나은 게이트를 설계하거나 특정 게이트의 펄스를 스트레칭하여 제로 노이즈 추론과 같은 오류 완화를 위한 프로토콜을 탐색할 수 있습니다.

Amazon Braket은 Rigetti의 네이티브 게이트에 프로그래밍 방식으로 액세스할 수 있도록 지원합니다.

```
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**
>
> 하드웨어 공급자는 QPU를 정기적으로 보정하며, 보통 하루에 한 번 이상 조정합니다. Braket SDK를 사용하면 최신 게이트 캘리브레이션을 얻을 수 있습니다.

```
device.refresh_gate_calibrations()
```

RX 또는 XY 게이트와 같은 지정된 네이티브 게이트를 검색하려면 원하는 `Gate` 객체와 큐비트를 전달해야 합니다. 예를 들어, 0에 적용된 RX (π/2) 의 펄스 구현을 검사할 수 있습니다. qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

함수를 사용하여 필터링된 교정 세트를 만들 수 있습니다. `filter` 게이트 목록 또는 게이트 목록을 전달합니다. `QubitSet` 다음 코드는 RX (π/2) 와 0에 대한 모든 교정을 포함하는 두 세트를 만듭니다. qubit

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

```
q0_calibrations = calibrations.filter(qubits=QubitSet([0])
```

이제 사용자 지정 보정 세트를 연결하여 네이티브 게이트의 동작을 제공하거나 수정할 수 있습니다. 예를 들어, 다음 회로를 고려해 보십시오.

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

PulseSequence객체 사전을 gate_definitions 키워드 인수에 qubit 0 전달하여 게이트 온에 대한 사용자 지정 rx 게이트 보정을 사용하여 실행할 수 있습니다. pulse_sequencesGateCalibrations객체의 속성으로 사전을 만들 수 있습니다. 지정되지 않은 모든 게이트는 양자 하드웨어 공급자의 펄스 보정으로 대체됩니다.

```
nb_shots = 50
custom_calibration = GateCalibrations({rx_pi_2_q0: pulse_sequence_rx_pi_2_q0})
task=device.run(bell_circuit, gate_definitions=custom_calibration.pulse_sequences,
 shots=nb_shots)
```

Amazon Braket 하이브리드 채용 사용자 가이드

이 섹션에서는 Amazon Braket에서 하이브리드 작업을 설정하고 관리하는 방법에 대한 지침을 제공합니다.

다음을 사용하여 Braket에서 하이브리드 작업에 액세스할 수 있습니다.

- 아마존 브라켓 파이썬 SDK.
- 아마존 브라켓 콘솔.
- 더 Amazon API 브라켓.

이 섹션:

- 하이브리드 Job이란 무엇입니까?
- Amazon Braket 하이브리드 채용을 사용하는 경우
- 로컬 코드를 하이브리드 작업으로 실행하십시오.
- Amazon Braket 하이브리드 작업을 사용하여 하이브리드 작업을 실행하십시오.
- 첫 번째 하이브리드 Job 생성
- 입력, 출력, 환경 변수 및 도우미 함수
- 작업 결과 저장
- 체크포인트를 사용하여 하이브리드 작업을 저장하고 다시 시작합니다.
- 알고리즘 스크립트의 환경을 정의하세요.
- 하이퍼파라미터 사용
- 알고리즘 스크립트를 실행하도록 하이브리드 작업 인스턴스를 구성하십시오.
- 하이브리드 작업 취소
- 파라메트릭 컴파일을 사용하여 하이브리드 작업의 속도를 높입니다.
- 아마존 PennyLane 브라켓과 함께 사용
- Amazon Braket 하이브리드 작업을 사용하고 QAOA PennyLane 알고리즘을 실행하십시오.
- 의 임베디드 시뮬레이터로 하이브리드 워크로드를 가속화하십시오. PennyLane
- 로컬 모드로 하이브리드 작업을 빌드하고 디버그하세요.
- 자체 컨테이너 가져오기 (BYOC)
- 에서 기본 버킷을 구성하십시오. AwsSession
- 를 사용하여 하이브리드 작업과 직접 상호 작용할 수 있습니다. API

하이브리드 Job이란 무엇입니까?

Amazon Braket Hybrid Jobs는 기존 AWS 리소스와 양자 처리 장치 (QPU) 가 모두 필요한 하이브리드 양자-고전 알고리즘을 실행할 수 있는 방법을 제공합니다. Hybrid Jobs는 요청된 클래식 리소스를 가동하고, 알고리즘을 실행하고, 완료 후 인스턴스를 릴리스하도록 설계되었으므로 사용한 만큼만 비용을 지불하면 됩니다.

Hybrid Jobs는 기존 리소스와 양자 리소스를 모두 포함하는 장기 실행 반복 알고리즘에 적합합니다. 실행할 알고리즘을 제출하면 Braket은 확장 가능한 컨테이너식 환경에서 알고리즘을 실행하고 알고리즘이 완료되면 결과를 검색합니다.

또한 하이브리드 작업에서 생성된 양자 작업은 대상 QPU에 대한 우선 순위가 높은 대기열을 통해 이점을 얻을 수 있습니다. 이렇게 하면 양자 작업이 대기열에 있는 다른 작업보다 먼저 처리되고 실행될 수 있습니다. 이는 이전 양자 작업의 결과에 따라 후속 작업이 달라지는 반복적 하이브리드 알고리즘에 특히 유용합니다. 이러한 알고리즘의 예로는 양자 근사 최적화 알고리즘 (QAOA), 변이 양자 고유 해석기 또는 양자 기계 학습이 있습니다. 또한 알고리즘 진행 상황을 거의 실시간으로 모니터링하여 비용, 예산 또는 학습 손실이나 기대값과 같은 사용자 지정 메트릭을 추적할 수 있습니다.

Amazon Braket 하이브리드 채용을 사용하는 경우

Amazon Braket Hybrid Jobs를 사용하면 기존 컴퓨팅 리소스와 양자 컴퓨팅 디바이스를 결합하여 오늘날 양자 시스템의 성능을 최적화하는 가변 양자 고유 솔버 (VQE) 및 양자 근사 최적화 알고리즘 (QAOA) 과 같은 하이브리드 양자-고전 알고리즘을 실행할 수 있습니다. Amazon Braket 하이브리드 잡스는 다음과 같은 세 가지 주요 이점을 제공합니다.

1. 성능: Amazon Braket Hybrid Jobs는 자체 환경에서 하이브리드 알고리즘을 실행하는 것보다 더 나은 성능을 제공합니다. 작업이 실행되는 동안에는 선택한 대상 QPU에 우선적으로 액세스할 수 있습니다. 작업의 작업이 장치의 대기열에 있는 다른 작업보다 먼저 실행됩니다. 그 결과 하이브리드 알고리즘의 런타임이 짧아지고 예측 가능해집니다. Amazon Braket 하이브리드 잡스는 파라메트릭 컴파일도 지원합니다. 무료 파라미터를 사용하여 회로를 제출하면 Braket이 회로를 한 번 컴파일하므로 이후에 동일한 회로에 대한 파라미터 업데이트를 위해 다시 컴파일할 필요 없이 회로를 한 번 컴파일하므로 런타임이 훨씬 빨라집니다.
2. 편리성: Amazon Braket Hybrid Jobs는 컴퓨팅 환경의 설정 및 관리를 단순화하고 하이브리드 알고리즘이 실행되는 동안 계속 실행되도록 합니다. 알고리즘 스크립트를 제공하고 실행할 양자 장치 (양자 처리 장치 또는 시뮬레이터) 를 선택하기만 하면 됩니다. Amazon Braket은 대상 디바이스를 사용할 수 있을 때까지 기다렸다가 기존 리소스를 가동하고 사전 구축된 컨테이너 환경에서 워크로드를 실행하고 Amazon Simple Storage Service (Amazon S3) 에 결과를 반환하고 컴퓨팅 리소스를 릴리스합니다.

3. 지표: Amazon Braket Hybrid Jobs는 실행 중인 알고리즘에 on-the-fly 대한 통찰력을 제공하고 사용자 지정 가능한 알고리즘 지표를 CloudWatch Amazon과 Amazon Braket 콘솔에 거의 실시간으로 제공하므로 알고리즘 진행 상황을 추적할 수 있습니다.

로컬 코드를 하이브리드 작업으로 실행하십시오.

Amazon Braket Hybrid Jobs는 Amazon EC2 컴퓨팅 리소스를 Amazon Braket QPU (양자 처리 장치) 액세스와 결합하여 하이브리드 양자-고전 알고리즘의 완전 관리형 오케스트레이션을 제공합니다. 하이브리드 작업에서 생성된 양자 작업은 개별 양자 작업보다 우선 순위가 높아 양자 작업 대기열의 변동으로 인해 알고리즘이 중단되지 않습니다. 각 QPU는 별도의 하이브리드 작업 대기열을 유지하므로 한 번에 하나의 하이브리드 작업만 실행할 수 있습니다.

이 섹션:

- 로컬 Python 코드에서 하이브리드 작업 생성
- 추가 Python 패키지 및 소스 코드 설치
- 하이브리드 작업 인스턴스로 데이터를 저장하고 로드합니다.
- 하이브리드 잡 데코레이터를 위한 베스트 프랙티스

로컬 Python 코드에서 하이브리드 작업 생성

로컬 Python 코드를 Amazon Braket 하이브리드 Job으로 실행할 수 있습니다. 다음 코드 예제와 같이 `@hybrid_job` 데코레이터로 코드에 주석을 달면 이 작업을 수행할 수 있습니다. 사용자 지정 환경의 경우 Amazon Elastic 컨테이너 레지스트리 (ECR) 의 사용자 지정 컨테이너를 사용하도록 선택할 수 있습니다.

> **Note**
>
> 기본적으로 Python 3.10만 지원됩니다.

`@hybrid_job`데코레이터를 사용하여 함수에 주석을 달 수 있습니다. Braket은 데코레이터 내부의 코드를 Braket 하이브리드 작업 알고리즘 스크립트로 변환합니다. 그런 다음 하이브리드 작업은 Amazon EC2 인스턴스의 데코레이터 내에서 함수를 호출합니다. Braket 콘솔을 `job.state()` 사용하거나 Braket 콘솔을 사용하여 작업 진행 상황을 모니터링할 수 있습니다. 다음 코드 예제는 에서 다섯 가지 상태의 시퀀스를 실행하는 방법을 보여줍니다. State Vector Simulator (SV1) device

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

device_arn = Devices.Amazon.SV1


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

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

    theta = 0.0  # initial parameter

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

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

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

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

일반적인 Python 함수처럼 함수를 호출하여 하이브리드 작업을 생성합니다. 하지만 데코레이터 함수는 함수의 결과가 아닌 하이브리드 작업 핸들을 반환합니다. 완료 후 결과를 검색하려면 를 사용하십시오`job.result()`.

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

`@hybrid_job`데코레이터의 기기 인수는 하이브리드 작업이 우선적으로 액세스할 수 있는 기기 (이 경우에는 SV1 시뮬레이터) 를 지정합니다. QPU 우선순위를 얻으려면 함수 내에서 사용된 기기 ARN이

데코레이터에 지정된 것과 일치하는지 확인해야 합니다. 편의를 위해 helper 함수를 사용하여 에서 선언한 기기 `get_job_device_arn()` ARN을 캡처할 수 있습니다. `@hybrid_job`

> **Note**
>
> Amazon EC2에 컨테이너식 환경을 생성하므로 각 하이브리드 작업의 시작 시간은 최소 1분입니다. 따라서 단일 회로나 회로 배치와 같이 매우 짧은 워크로드의 경우 양자 작업을 사용하는 것으로 충분할 수 있습니다.

하이퍼파라미터

이 `run_hybrid_job()` 함수는 `num_tasks` 인수를 받아 생성되는 양자 작업의 수를 제어합니다. 하이브리드 작업은 이를 하이퍼파라미터로 자동 캡처합니다.

> **Note**
>
> 하이퍼파라미터는 Braket 콘솔에 2500자로 제한되는 문자열로 표시됩니다.

지표 및 로깅

`run_hybrid_job()`함수 내에는 반복 알고리즘의 메트릭이 함께 `log_metrics` 기록됩니다. 지표는 하이브리드 작업 탭 아래의 Braket 콘솔 페이지에 자동으로 플로팅됩니다. Braket 비용 추적기로 하이브리드 작업을 실행하는 동안 메트릭을 사용하여 양자 작업 비용을 거의 실시간으로 추적할 수 있습니다. 위 예시에서는 결과 유형에서 첫 번째 확률을 기록하는 지표 이름 “확률”을 사용합니다.

결과 검색

하이브리드 작업이 완료된 후 하이브리드 작업 결과를 검색하는 `job.result()` 데 사용합니다. 반환 명령문의 모든 객체는 Braket에 의해 자동으로 캡처됩니다. 참고로, 함수에서 반환되는 객체는 튜플이어야 하며 각 요소는 직렬화 가능해야 합니다. 예를 들어, 다음 코드는 작동하는 예제와 실패한 예제를 보여줍니다.

```
@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
```

Job 이름

기본적으로 이 하이브리드 작업의 이름은 함수 이름에서 유추됩니다. 최대 50자까지 사용자 정의 이름을 지정할 수도 있습니다. 예를 들어, 다음 코드에서 작업 이름은 “my-job-name”입니다.

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

로컬 모드

로컬 작업은 데코레이터에 인수를 `local=True` 추가하여 생성됩니다. 이렇게 하면 랩톱과 같은 로컬 컴퓨팅 환경의 컨테이너화된 환경에서 하이브리드 작업이 실행됩니다. 로컬 작업에는 양자 작업에 대한 우선 순위 대기열이 없습니다. 다중 노드 또는 MPI와 같은 고급 사례의 경우 로컬 작업에서 필수 Braket 환경 변수에 액세스할 수 있습니다. 다음 코드는 디바이스를 SV1 시뮬레이터로 사용하는 로컬 하이브리드 작업을 생성합니다.

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

다른 모든 하이브리드 작업 옵션이 지원됩니다. 옵션 목록은 braket.jobs.quantum_job_create 모듈을 참조하십시오.

추가 Python 패키지 및 소스 코드 설치

선호하는 Python 패키지를 사용하도록 런타임 환경을 사용자 정의할 수 있습니다. `requirements.txt`파일, 패키지 이름 목록 또는 자체 컨테이너 가져오기 (BYOC) 를 사용할 수 있습니다. `requirements.txt`파일을 사용하여 런타임 환경을 사용자 지정하려면 다음 코드 예제를 참조하십시오.

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

예를 들어, `requirements.txt` 파일에는 설치할 다른 패키지가 포함될 수 있습니다.

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

또는 다음과 같이 패키지 이름을 Python 목록으로 제공할 수 있습니다.

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

추가 소스 코드는 다음 코드 예제와 같이 모듈 목록 또는 단일 모듈로 지정할 수 있습니다.

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

하이브리드 작업 인스턴스로 데이터를 저장하고 로드합니다.

입력 교육 데이터 지정

하이브리드 작업을 생성할 때 Amazon Simple Storage Service (Amazon S3) 버킷을 지정하여 입력 교육 데이터세트를 제공할 수 있습니다. 로컬 경로를 지정할 수도 있습니다. 그러면 Braket이 Amazon S3에서 자동으로 데이터를 업로드합니다. `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>` 로컬 경로를 지정하는 경우 채널 이름의 기본값은 “input”입니다. 다음 코드는 로컬 경로의 numpy 파일을 보여줍니다. `data/file.npy`

```
@hybrid_job(device=Devices.Amazon.SV1, input_data="data/file.npy")
def run_hybrid_job(num_tasks = 1):
    data = np.load("data/file.npy")
    return ...
```

S3의 경우 `get_input_data_dir()` 도우미 함수를 사용해야 합니다.

```
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")
```

채널 값 사전과 S3 URI 또는 로컬 경로를 제공하여 여러 입력 데이터 소스를 지정할 수 있습니다.

```
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**
>
> 입력 데이터가 크면 (>1GB) 작업이 생성되기까지 대기 시간이 오래 걸립니다. 이는 로컬 입력 데이터를 S3 버킷에 처음 업로드한 후 S3 경로가 작업 요청에 추가되기 때문입니다. 마지막으로 작업 요청이 Braket 서비스에 제출됩니다.

S3에 결과 저장

데코레이트된 함수의 return 문에 포함되지 않은 결과를 저장하려면 모든 파일 쓰기 작업에 올바른 디렉토리를 추가해야 합니다. 다음 예제는 numpy 배열과 matplotlib 그림을 저장하는 방법을 보여줍니다.

```
@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 ...
```

모든 결과는 라는 이름의 파일로 압축됩니다. model.tar.gz Python 함수를 job.result() 사용하거나 Braket 관리 콘솔의 하이브리드 작업 페이지에서 결과 폴더로 이동하여 결과를 다운로드할 수 있습니다.

체크포인트에서 저장 및 재개

장기 실행 하이브리드 작업의 경우 알고리즘의 중간 상태를 주기적으로 저장하는 것이 좋습니다. 내장된 save_job_checkpoint() 도우미 함수를 사용하거나 파일을 경로에 저장할 수 있습니다. AMZN_BRAKET_JOB_RESULTS_DIR 나중 함수는 헬퍼 함수와 함께 사용할 수 있습니다. get_job_results_dir()

다음은 하이브리드 작업 데코레이터를 사용하여 체크포인트를 저장하고 로드하는 간단한 작업 예제입니다.

```
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()
```

첫 번째 하이브리드 작업에서는 save_job_checkpoint() 저장하려는 데이터가 들어 있는 사전을 사용하여 를 호출합니다. 기본적으로 모든 값은 텍스트로 직렬화할 수 있어야 합니다. numpy 배열과 같은 더 복잡한 Python 객체를 체크포인팅하기 위해 설정할 수 있습니다. data_format = PersistedJobDataFormat.PICKLED_V4 이 코드는 하이브리드 작업 아티팩트의 “checkpoints”라는 하위 폴더 아래에 있는 기본 이름을 <jobname>.json 가진 체크포인트 파일을 만들고 덮어씁니다.

체크포인트에서 계속할 새 하이브리드 작업을 생성하려면 이전 작업의 하이브리드 작업 job_arn ARN이 copy_checkpoints_from_job=job_arn 어디에 있는지 전달해야 합니다. 그런 다음 체크포인트에서 load_job_checkpoint(job_name) 로드하는 데 사용합니다.

하이브리드 잡 데코레이터를 위한 베스트 프랙티스

비동기성 수용하기

데코레이터 어노테이션으로 생성된 하이브리드 작업은 비동기식이므로 기존 리소스와 양자 리소스를 사용할 수 있게 되면 실행됩니다. Braket Management Console또는 Amazon을 사용하여 알고리즘 진행 상황을 CloudWatch 모니터링합니다. 실행을 위해 알고리즘을 제출하면 Braket은 확장 가능한 컨테이너식 환경에서 알고리즘을 실행하고 알고리즘이 완료되면 결과가 검색됩니다.

반복적 변형 알고리즘 실행

하이브리드 작업은 반복적인 양자-고전 알고리즘을 실행할 수 있는 도구를 제공합니다. 순수 양자 문제의 경우 양자 작업 또는 일련의 양자 작업을 사용하십시오. 특정 QPU에 대한 우선 순위 액세스는 QPU를 여러 번 반복해서 호출하고 그 사이에 클래식 처리를 적용해야 하는 장기 실행 변형 알고리즘에 가장 유용합니다.

로컬 모드를 사용하여 디버그합니다.

QPU에서 하이브리드 작업을 실행하기 전에 먼저 시뮬레이터 SV1에서 실행하여 예상대로 실행되는지 확인하는 것이 좋습니다. 소규모 테스트의 경우 빠른 반복 및 디버깅을 위해 로컬 모드로 실행할 수 있습니다.

BYOC (Bring Your Own Container) 로 재현성을 개선하세요.

소프트웨어와 해당 종속성을 컨테이너화된 환경 내에 캡슐화하여 재현 가능한 실험을 만들어 보세요. 모든 코드, 종속성 및 설정을 컨테이너에 패키징하여 잠재적 충돌 및 버전 관리 문제를 방지할 수 있습니다.

멀티인스턴스 분산 시뮬레이터

많은 회로를 실행하려면 내장된 MPI 지원을 사용하여 단일 하이브리드 작업 내의 여러 인스턴스에서 로컬 시뮬레이터를 실행하는 것이 좋습니다. 자세한 내용은 임베디드 시뮬레이터를 참조하십시오.

파라메트릭 회로 사용

하이브리드 작업에서 제출하는 파라메트릭 회로는 파라메트릭 컴파일을 사용하여 특정 QPU에서 자동으로 컴파일되어 알고리즘의 런타임을 개선합니다.

정기적으로 체크포인트를 지정합니다.

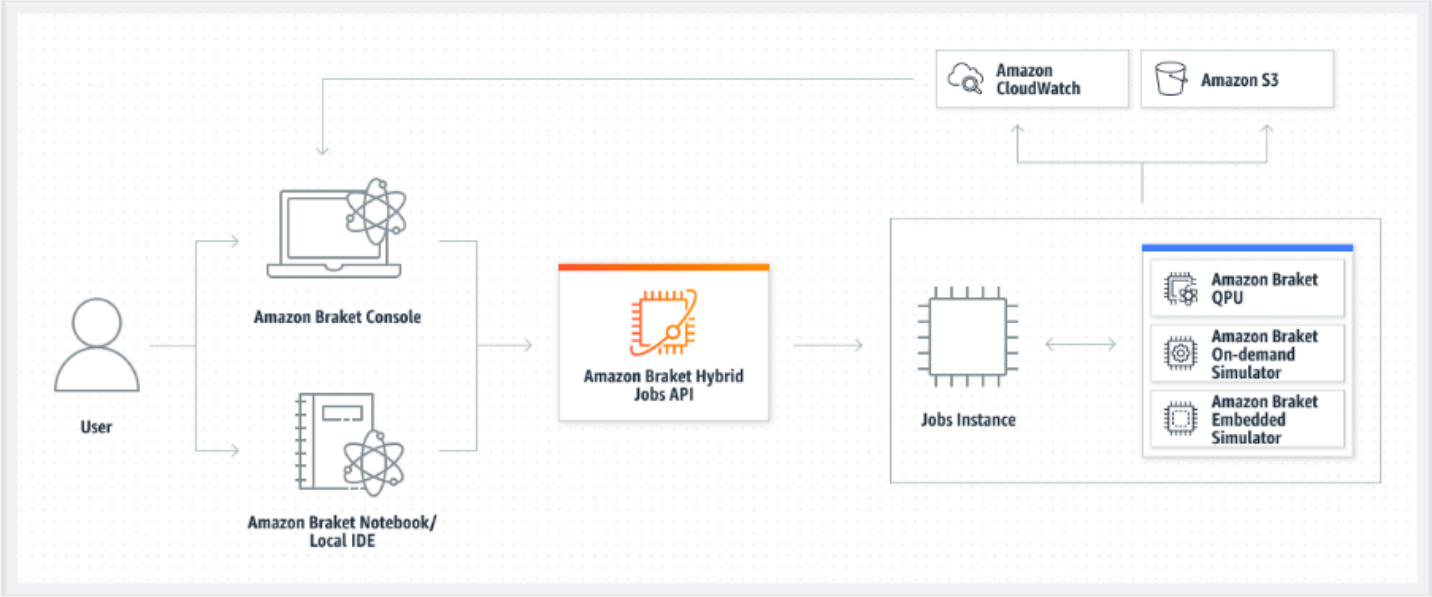
장기 실행 하이브리드 작업의 경우 알고리즘의 중간 상태를 주기적으로 저장하는 것이 좋습니다.

추가 예제, 사용 사례 및 모범 사례는 Amazon GitHub Braket 예제를 참조하십시오.

Amazon Braket 하이브리드 작업을 사용하여 하이브리드 작업을 실행하십시오.

Amazon브라켓 하이브리드 잡스로 하이브리드 작업을 실행하려면 먼저 알고리즘을 정의해야 합니다. Amazon Braket Python PennyLaneSDK를 사용하거나 알고리즘 스크립트를 작성하고 선택적으로 기타 종속성 파일을 작성하여 정의할 수 있습니다. 다른 (오픈 소스 또는 독점) 라이브러리를 사용하려는 경우 이러한 라이브러리가 포함된 Docker를 사용하여 사용자 지정 컨테이너 이미지를 정의할 수 있습니다. 자세한 내용은 BYOC (기존 컨테이너 가져오기) 를 참조하십시오.

어느 경우든, 다음에는 Amazon API Braket을 사용하여 하이브리드 작업을 생성합니다. 여기서 알고리즘 스크립트 또는 컨테이너를 제공하고 하이브리드 작업에 사용할 대상 양자 디바이스를 선택한 다음 다양한 옵션 설정 중에서 선택합니다. 이러한 선택적 설정에 제공된 기본값은 대부분의 사용 사례에서 작동합니다. Hybrid Job을 실행할 대상 장치의 경우 QPU, 온디맨드 시뮬레이터 (예: DM1 또는TN1) 또는 기존 하이브리드 작업 인스턴스 중 하나를 선택할 수 있습니다. SV1 온디맨드 시뮬레이터 또는 QPU를 사용하면 하이브리드 작업 컨테이너가 원격 장치로 API를 호출합니다. 내장된 시뮬레이터를 사용하면 시뮬레이터가 알고리즘 스크립트와 동일한 컨테이너에 내장됩니다. 의 라이트닝 PennyLane 시뮬레이터에는 사용자가 사용할 수 있도록 기본 사전 빌드된 하이브리드 작업 컨테이너가 내장되어 있습니다. 임베디드 PennyLane 시뮬레이터 또는 사용자 지정 시뮬레이터를 사용하여 코드를 실행하는 경우 인스턴스 유형과 사용하려는 인스턴스 수를 지정할 수 있습니다. 각 선택과 관련된 비용은 Amazon Braket 가격 책정 페이지를 참조하십시오.



대상 디바이스가 온디맨드 또는 임베디드 시뮬레이터인 경우 Amazon Braket은 하이브리드 작업을 즉시 실행하기 시작합니다. 하이브리드 작업 인스턴스를 가동하고 (API호출에서 인스턴스 유형을 사용자

지정할 수 있음), 알고리즘을 실행하고, 결과를 Amazon S3에 쓰고, 리소스를 릴리스합니다. 이번 리소스 릴리스에서는 사용한 만큼만 비용을 지불하면 됩니다.

양자 처리 장치 (QPU) 당 동시 하이브리드 작업의 총 수는 제한됩니다. 현재는 QPU에서 한 번에 하나의 하이브리드 작업만 실행할 수 있습니다. 큐는 허용된 제한을 초과하지 않도록 실행할 수 있는 하이브리드 작업 수를 제어하는 데 사용됩니다. 대상 장치가 QPU인 경우 하이브리드 작업은 먼저 선택한 QPU의 작업 대기열에 들어갑니다. Amazon Braket은 필요한 하이브리드 작업 인스턴스를 가동하고 디바이스에서 하이브리드 작업을 실행합니다. 알고리즘 기간 동안 하이브리드 작업에 우선 액세스 권한이 부여됩니다. 즉, 하이브리드 작업의 양자 작업이 디바이스에 대기중인 다른 Braket 양자 작업보다 먼저 실행됩니다. 단, 작업 양자 작업이 몇 분에 한 번씩 QPU에 제출되어야 합니다. 하이브리드 작업이 완료되면 리소스가 해제되므로 사용한 만큼만 비용을 지불하면 됩니다.

> **Note**
>
> 디바이스는 지역별로 제공되며 하이브리드 작업은 기본 AWS 리전 디바이스와 동일하게 실행됩니다.

시뮬레이터와 QPU 대상 시나리오 모두에서 해밀턴의 에너지와 같은 사용자 지정 알고리즘 메트릭을 알고리즘의 일부로 정의할 수 있습니다. 이러한 지표는 Amazon에 자동으로 CloudWatch 보고되며, Amazon Braket 콘솔에 거의 실시간으로 표시됩니다.

> **Note**
>
> GPU 기반 인스턴스를 사용하려면 Braket의 임베디드 시뮬레이터와 함께 제공되는 GPU 기반 시뮬레이터 중 하나를 사용해야 합니다 (예:). `lightning.gpu` CPU 기반 임베디드 시뮬레이터 중 하나 (예: 또는`braket:default-simulator`) 를 선택하면 GPU가 사용되지 않으므로 불필요한 비용이 발생할 수 있습니다. `lightning.qubit`

첫 번째 하이브리드 Job 생성

이 섹션에서는 Python 스크립트를 사용하여 Hybrid Job을 생성하는 방법을 보여줍니다. 또는 선호하는 통합 개발 환경 (IDE) 또는 Braket 노트북과 같은 로컬 Python 코드에서 하이브리드 작업을 만들려면 을 참조하십시오로컬 코드를 하이브리드 작업으로 실행하십시오..

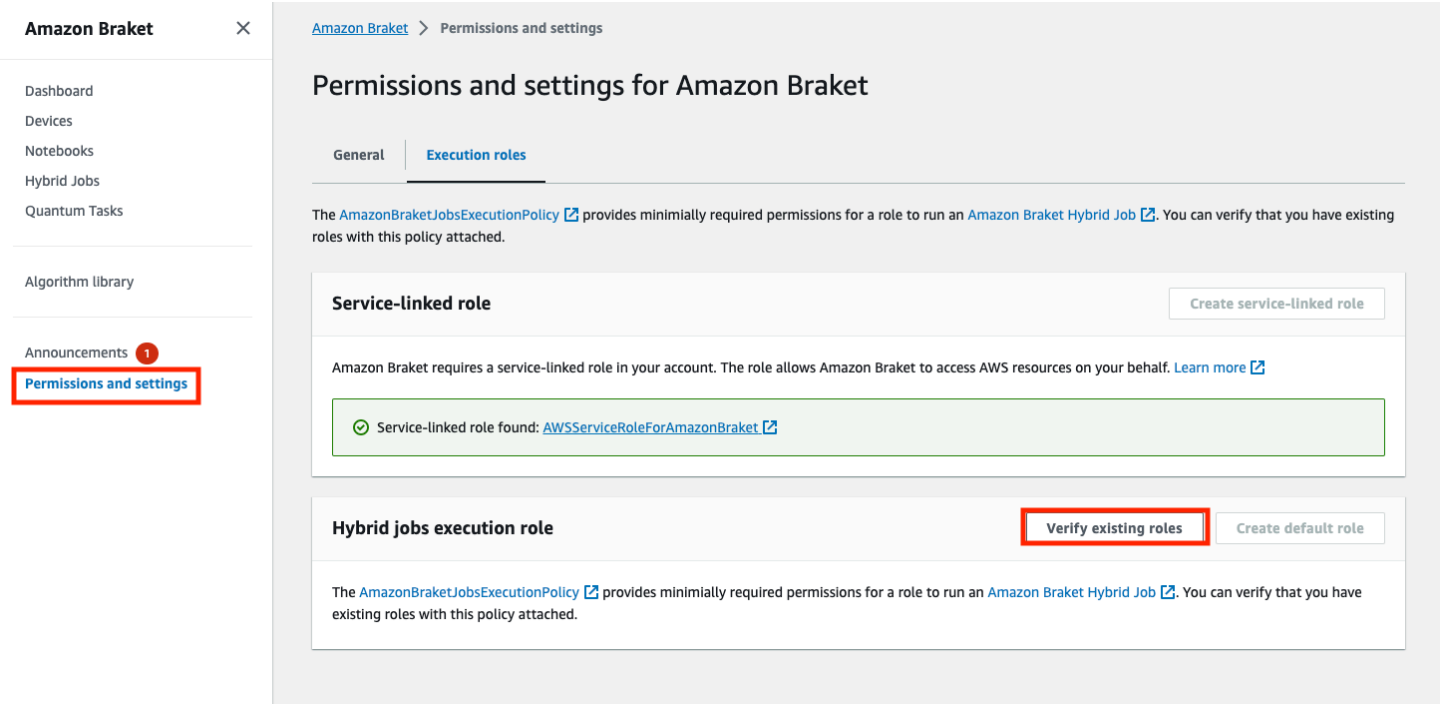
이 섹션:

- 권한 설정

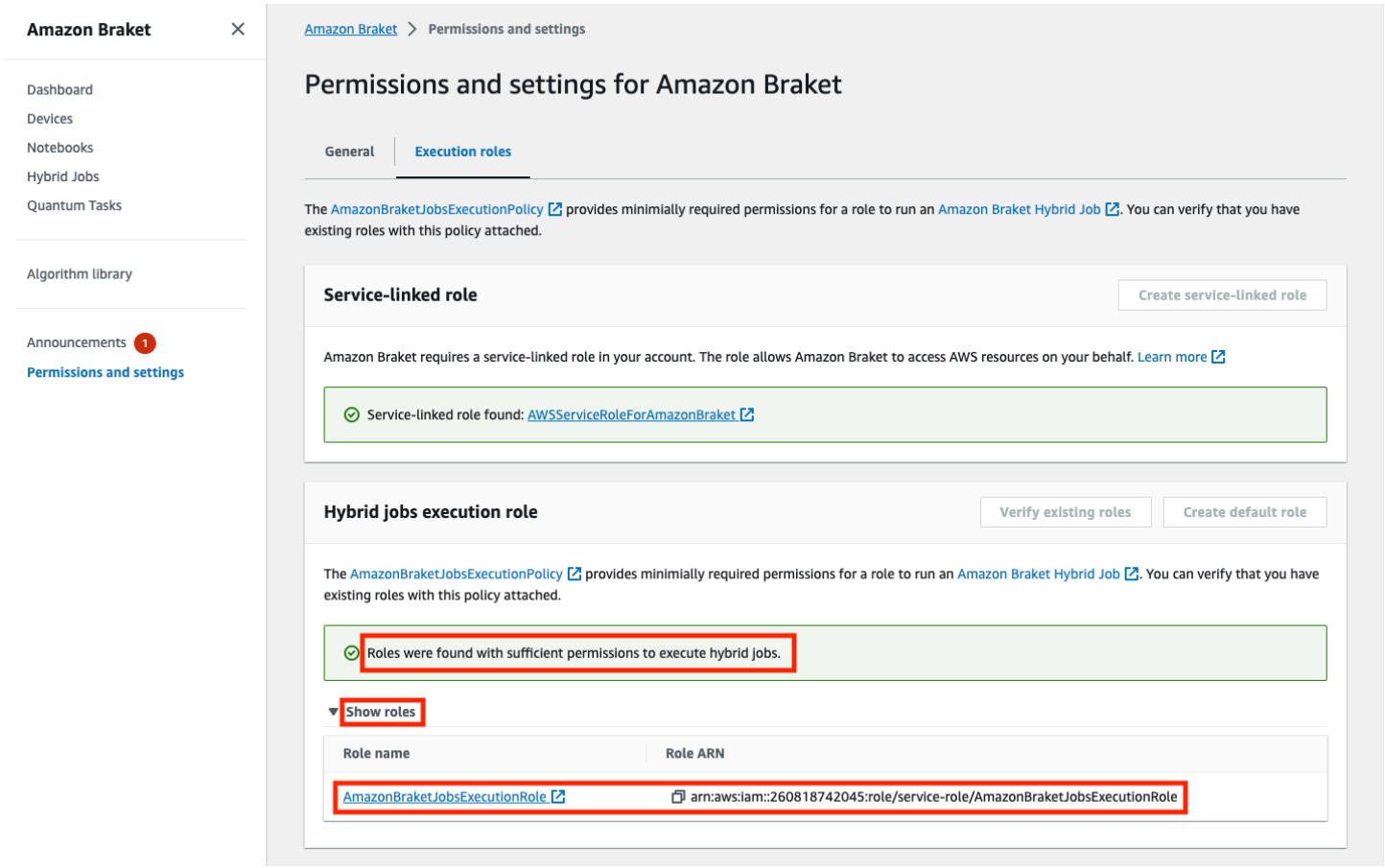
- 생성 및 실행
- 모니터 결과

권한 설정

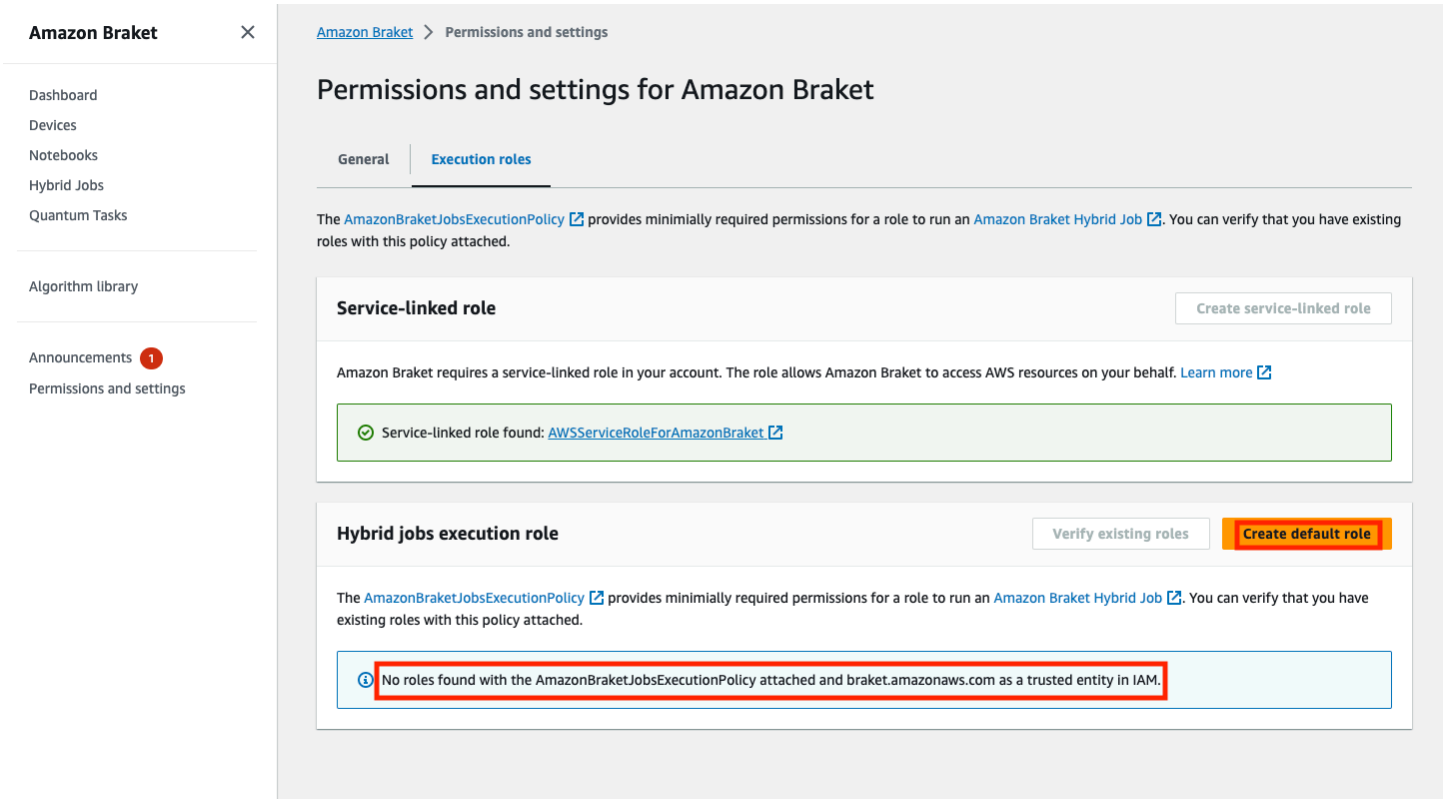
첫 번째 하이브리드 작업을 실행하기 전에 이 작업을 진행할 수 있는 충분한 권한이 있는지 확인해야 합니다. 권한이 올바른지 확인하려면 Braket 콘솔 왼쪽의 메뉴에서 권한을 선택하십시오. Amazon Braket의 권한 관리 페이지는 기존 역할 중 하나에 하이브리드 작업을 실행하기에 충분한 권한이 있는지 확인하는 데 도움이 되며, 하이브리드 작업을 실행하는 데 사용할 수 있는 기본 역할을 아직 갖고 있지 않은 경우 하이브리드 작업을 실행하는 데 사용할 수 있는 기본 역할을 생성하는 방법을 안내합니다.



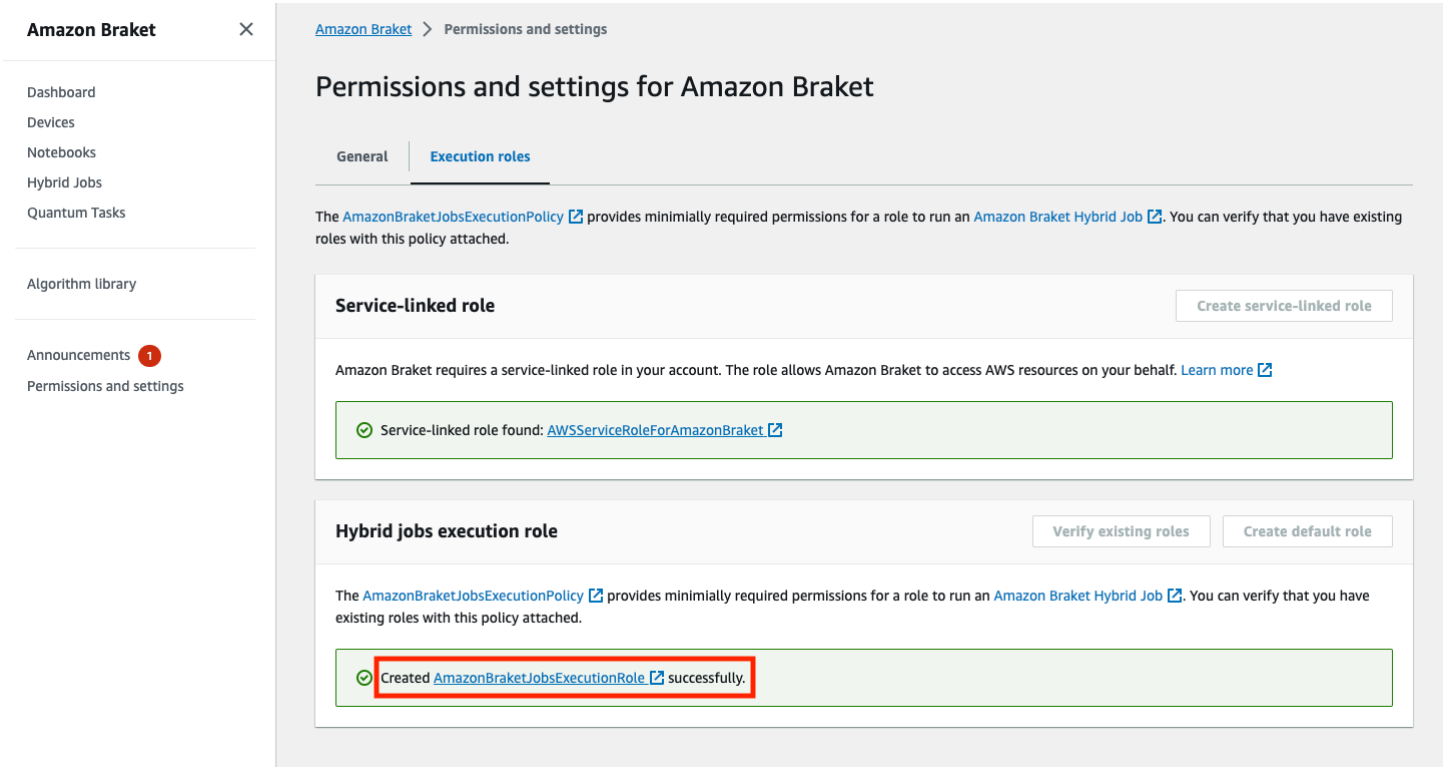
하이브리드 작업을 실행할 수 있는 충분한 권한을 가진 역할이 있는지 확인하려면 기존 역할 확인 버튼을 선택하십시오. 그럴 경우 역할을 찾았다는 메시지가 나타납니다. 역할 이름과 역할 ARN을 보려면 역할 보기 버튼을 선택합니다.



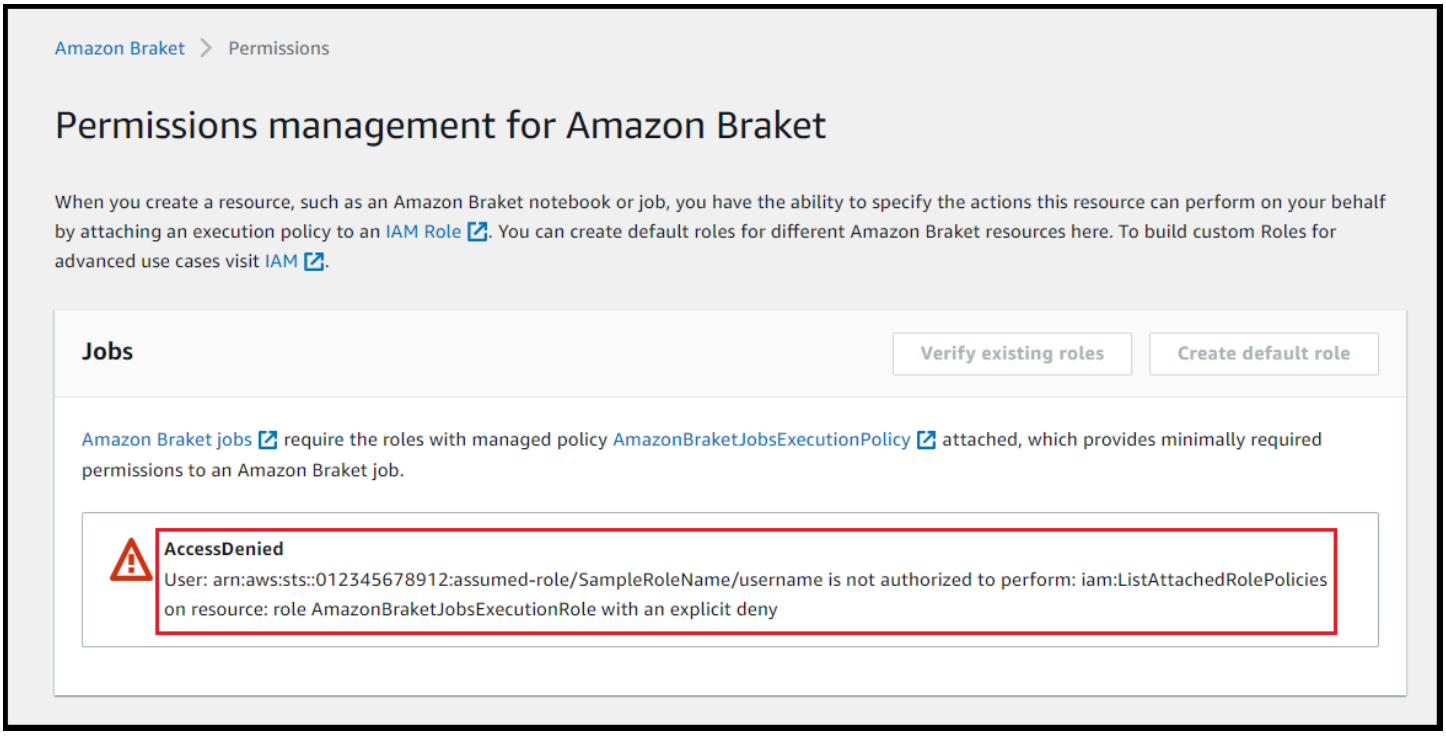
하이브리드 작업을 실행할 수 있는 충분한 권한을 가진 역할이 없는 경우 해당 역할을 찾을 수 없다는 메시지가 표시됩니다. 충분한 권한이 있는 역할을 가져오려면 기본 역할 만들기 버튼을 선택합니다.



역할이 성공적으로 생성되면 이를 확인하는 메시지가 나타납니다.

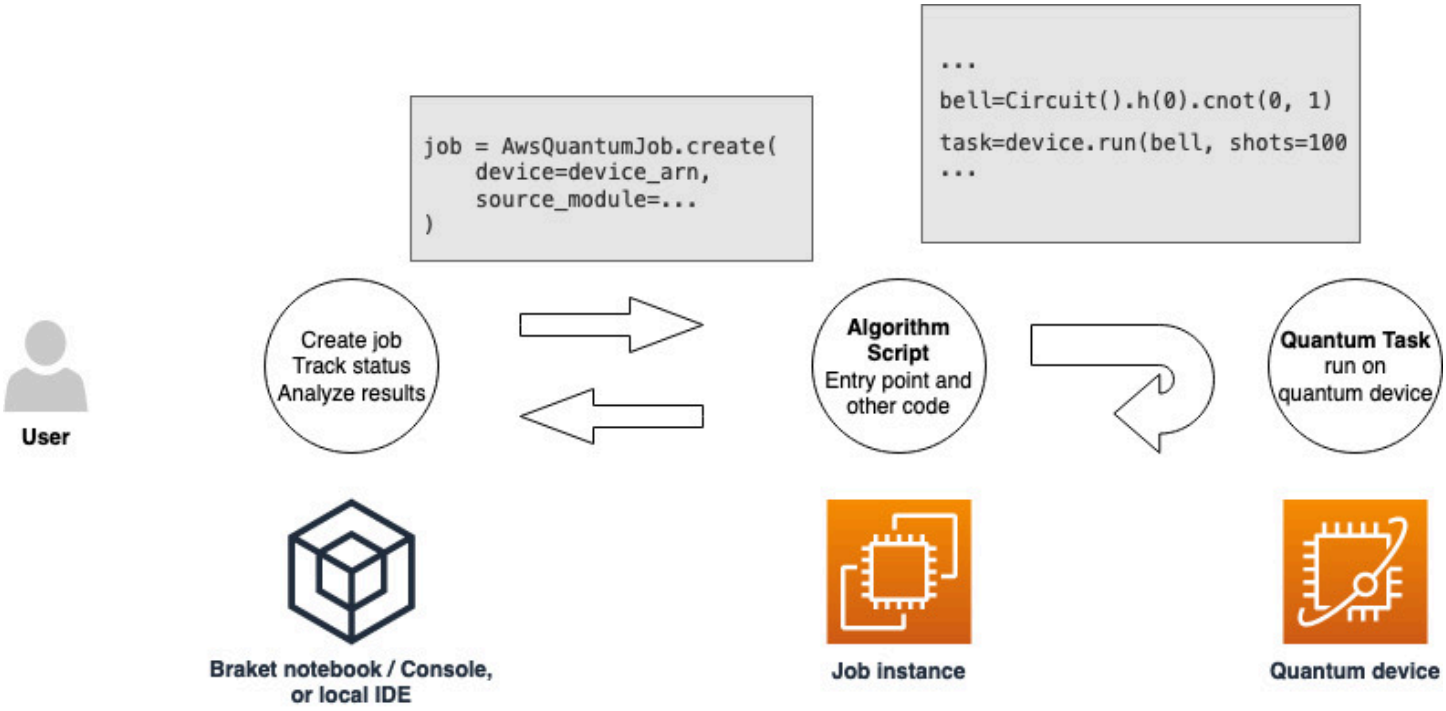


이 문의를 할 권한이 없는 경우 액세스가 거부됩니다. 이 경우 내부 AWS 관리자에게 문의하세요.



생성 및 실행

하이브리드 작업을 실행할 권한이 있는 역할을 갖게 되면 작업을 계속할 준비가 된 것입니다. 첫 번째 Braket 하이브리드 작업의 핵심은 알고리즘 스크립트입니다. 실행하려는 알고리즘을 정의하며 알고리즘의 일부인 기존 논리 및 양자 작업을 포함합니다. 알고리즘 스크립트 외에도 다른 종속성 파일을 제공할 수 있습니다. 알고리즘 스크립트와 종속 항목을 함께 소스 모듈이라고 합니다. 진입점은 하이브리드 작업이 시작될 때 소스 모듈에서 실행할 첫 번째 파일 또는 함수를 정의합니다.



먼저, 5가지 벨 상태를 생성하고 해당 측정 결과를 출력하는 알고리즘 스크립트의 다음 기본 예를 살펴보겠습니다.

```
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!")
```

Braket 노트북 또는 로컬 환경의 현재 작업 디렉토리에 algorithm_script.py 라는 이름으로 이 파일을 저장하십시오. algorithm_script.py 파일이 시작 `start_here()` 지점으로 계획되어 있습니다.

다음으로, algorithm_script.py 파일과 같은 디렉토리에 Python 파일 또는 Python 노트북을 생성합니다. 이 스크립트는 하이브리드 작업을 시작하고 원하는 상태나 주요 결과를 인쇄하는 등의 모든 비동기 처리를 처리합니다. 이 스크립트는 최소한 하이브리드 작업 스크립트와 기본 장치를 지정해야 합니다.

> **Note**
>
> Braket 노트북을 생성하거나 노트북과 동일한 디렉터리에 algorithm_script.py 파일과 같은 파일을 업로드하는 방법에 대한 자세한 내용은 Amazon Braket Python SDK를 사용하여 첫 번째 회로 실행을 참조하십시오.

이 기본적인 첫 번째 사례에서는 시뮬레이터를 대상으로 합니다. 시뮬레이터 또는 실제 양자 처리 장치 (QPU) 중 어떤 유형을 대상으로 하든, 다음 스크립트에서 지정하는 장치는 하이브리드 작업을 예약하는 데 사용되며 알고리즘 스크립트에서 환경 변수로 사용할 수 있습니다. `device AMZN_BRAKET_DEVICE_ARN`

> **Note**
>
> 하이브리드 작업에서 사용할 수 있는 장치만 사용할 수 있습니다. AWS 리전 Amazon Braket SDK는 이를 자동으로 선택합니다. AWS 리전예를 들어 us-east-1의 하이브리드 작업은, SV1DM1, TN1 및 장치를 IonQ 사용할 수 있지만 장치는 사용할 수 없습니다. Rigetti

시뮬레이터 대신 양자 컴퓨터를 선택하면 Braket은 우선 액세스가 가능한 모든 양자 작업을 실행하도록 하이브리드 작업을 예약합니다.

```
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
)
```

이 파라미터는 작업이 실행 중인 실제 작업의 출력을 인쇄하도록 세부 정보 표시 모드를 `wait_until_complete=True` 설정합니다. 다음 예제와 비슷한 출력이 표시될 것입니다.

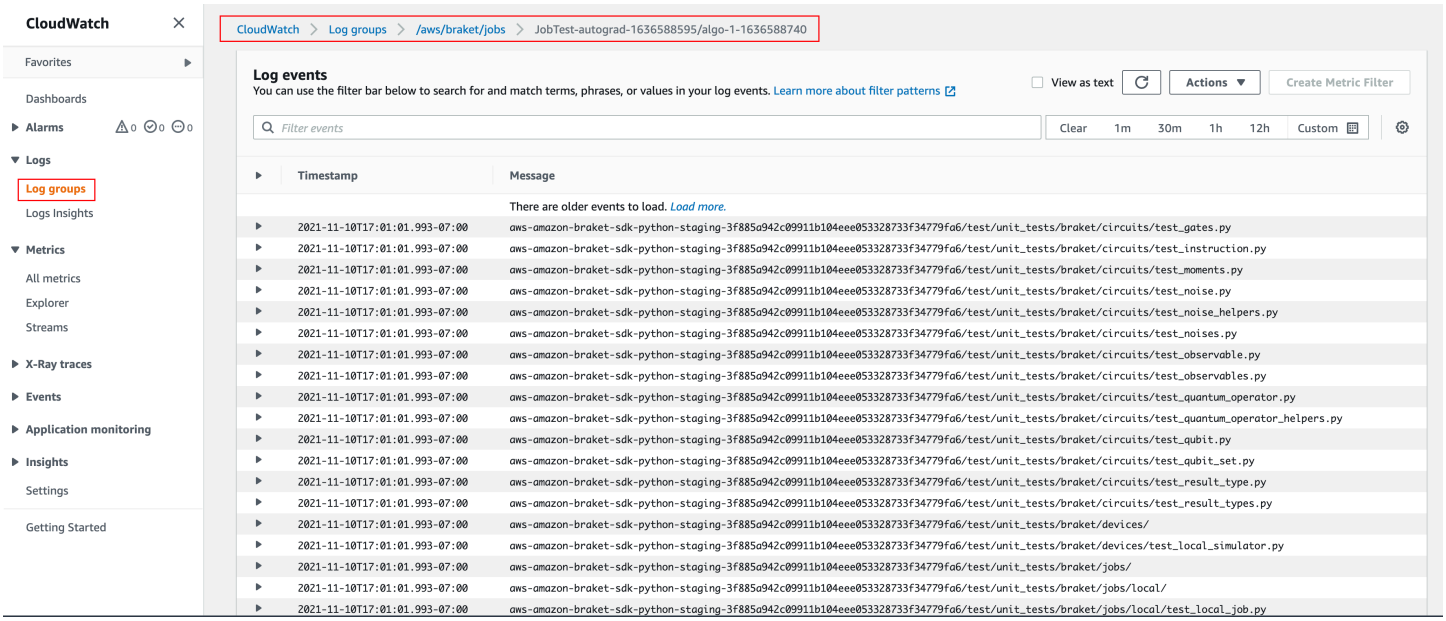
```
job = AwsQuantumJob.create(
```

```
    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
```

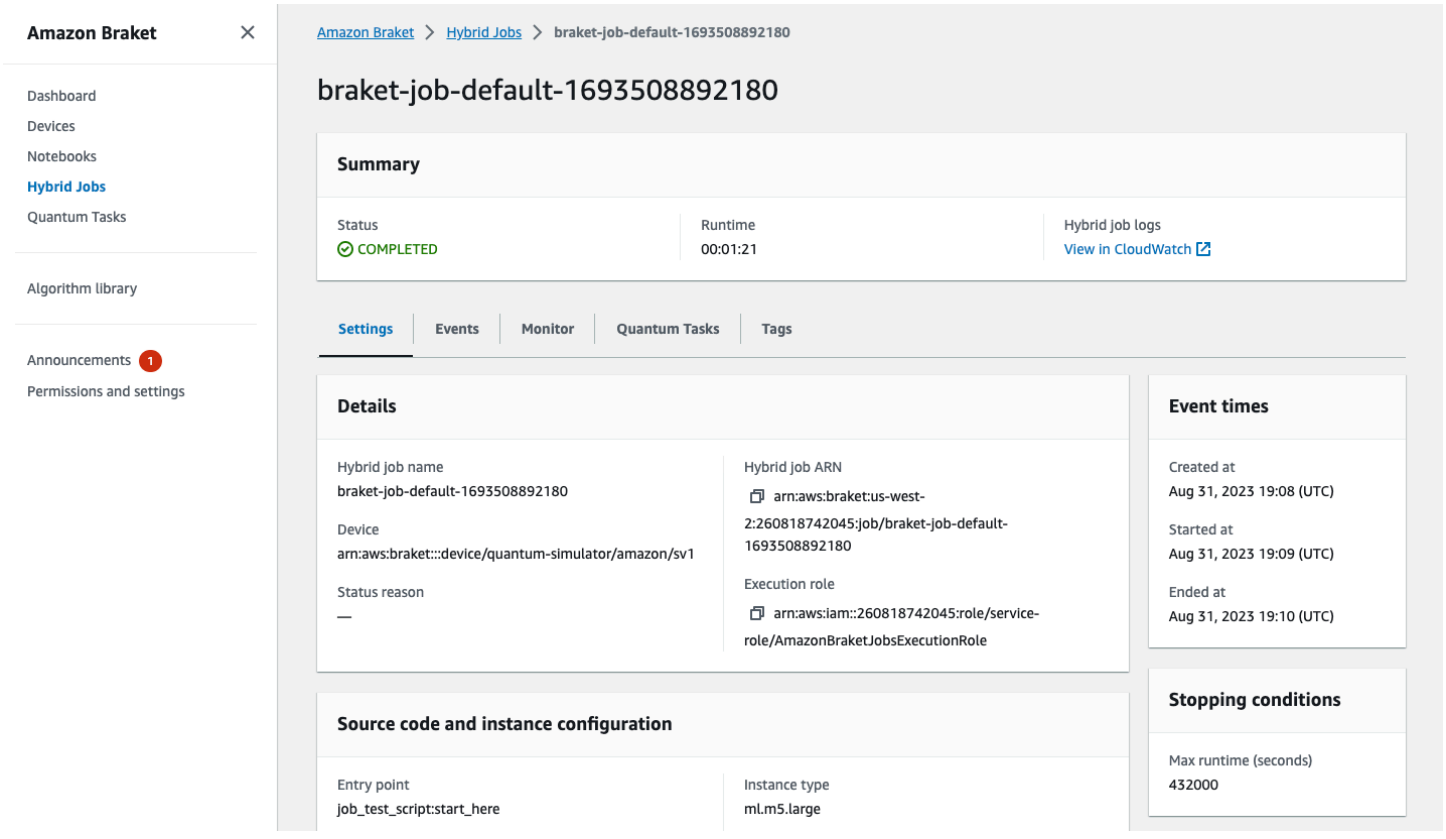
> **Note**
>
> 또한 위치 (로컬 디렉토리 또는 파일 경로 또는 tar.gz 파일의 S3 URI) 를 전달하여 AwsQuantumjob.create 메서드와 함께 사용자 지정 모듈을 사용할 수도 있습니다. 실제 예제는 Amazon Braket 예제 Github 리포지토리의 하이브리드 작업 폴더에 있는 Parallelize_Training_for_QML.iPynb 파일을 참조하십시오.

모니터 결과

또는 Amazon에서 로그 출력에 액세스할 수 CloudWatch 있습니다. 이렇게 하려면 작업 세부 정보 페이지의 왼쪽 메뉴에 있는 로그 그룹 탭으로 이동하여 로그 그룹을 aws/braket/jobs 선택한 다음 작업 이름이 포함된 로그 스트림을 선택합니다. 위 예시에서는 다음과 같습니다`braket-job-default-1631915042705/algo-1-1631915190`.



하이브리드 작업 페이지를 선택한 다음 설정을 선택하여 콘솔에서 하이브리드 작업의 상태를 볼 수도 있습니다.



하이브리드 작업은 실행 중에 Amazon S3에서 일부 아티팩트를 생성합니다. 기본 S3 버킷 이름은 `amazon-braket-<region>-<accountid>` 이며 콘텐츠는 `jobs/<jobname>/<timestamp>` 디

렉터리에 있습니다. Braket Python SDK로 하이브리드 작업을 생성할 `code_location` 때 다른 위치를 지정하여 이러한 아티팩트가 저장되는 S3 위치를 구성할 수 있습니다.

> **Note**
>
> 이 S3 버킷은 작업 스크립트와 AWS 리전 동일한 위치에 있어야 합니다.

`jobs/<jobname>/<timestamp>`디렉토리에는 엔트리 포인트 스크립트의 출력이 `model.tar.gz` 파일에 포함된 하위 폴더가 있습니다. 알고리즘 스크립트 아티팩트가 `script` 파일에 들어 있는 디렉토리도 있습니다. `source.tar.gz` 실제 양자 작업의 결과는 이름이 지정된 `jobs/<jobname>/tasks` 디렉터리에 있습니다.

입력, 출력, 환경 변수 및 도우미 함수

전체 알고리즘 스크립트를 구성하는 파일 외에도 하이브리드 작업에는 추가 입력 및 출력이 있을 수 있습니다. 하이브리드 작업이 시작되면 Amazon Braket은 하이브리드 작업 생성의 일부로 제공된 입력을 알고리즘 스크립트를 실행하는 컨테이너에 복사합니다. 하이브리드 작업이 완료되면 알고리즘 중에 정의된 모든 출력이 지정된 Amazon S3 위치에 복사됩니다.

> **Note**
>
> 알고리즘 지표는 실시간으로 보고되며 이 출력 절차를 따르지 않습니다.

Amazon또한 Braket은 컨테이너 입력 및 출력과의 상호 작용을 단순화하는 여러 환경 변수와 도우미 함수를 제공합니다.

이 섹션에서는 Amazon Braket Python SDK에서 제공하는 `AwsQuantumJob.create` 함수의 주요 개념과 컨테이너 파일 구조로의 매핑에 대해 설명합니다.

이 섹션:

- 입력
- 결과
- 환경 변수
- 헬퍼 함수

입력

입력 데이터: 사전으로 설정된 입력 데이터 파일을 인수와 함께 지정하여 하이브리드 알고리즘에 입력 데이터를 제공할 수 있습니다. `input_data` 사용자는 SDK의 `input_data` `AwsQuantumJob.create` 함수 내에 인수를 정의합니다. 이렇게 하면 입력 데이터가 환경 `"AMZN_BRAKET_INPUT_DIR"` 변수로 지정된 위치의 컨테이너 파일 시스템에 복사됩니다. 하이브리드 알고리즘에서 입력 데이터를 사용하는 방법에 대한 몇 가지 예를 보려면 Amazon Braket Hybrid Jobs PennyLane 및 Amazon Braket Hybrid Jobs Jupyter 노트북의 Quantum 기계 학습을 포함하는 QAOA를 참조하십시오.

> **Note**
>
> 입력 데이터가 큰 경우 (>1GB) 하이브리드 작업이 제출되기까지 대기 시간이 오래 걸립니다. 이는 로컬 입력 데이터가 먼저 S3 버킷에 업로드된 후 S3 경로가 하이브리드 작업 요청에 추가되고 마지막으로 하이브리드 작업 요청이 Braket 서비스에 제출되기 때문입니다.

하이퍼파라미터: 전달하면 환경 변수에서 사용할 수 있습니다. `hyperparameters` `"AMZN_BRAKET_HP_FILE"`

> **Note**
>
> 하이퍼파라미터 및 입력 데이터를 만든 다음 이 정보를 하이브리드 작업 스크립트에 전달하는 방법에 대한 자세한 내용은 하이퍼파라미터 사용 섹션 및 이 github 페이지를 참조하십시오.

체크포인트: 새 하이브리드 `job-arn` 작업에서 사용할 체크포인트를 지정하려면 명령을 사용합니다. `copy_checkpoints_from_job` 이 명령은 체크포인트 데이터를 새 하이브리드 작업에 복사하여 작업이 실행되는 `AMZN_BRAKET_CHECKPOINT_DIR` 동안 환경 변수가 지정한 경로에서 사용할 수 있도록 합니다. `checkpoint_configs3Uri` 기본값은 입니다. 즉None, 다른 하이브리드 작업의 체크포인트 데이터는 새 하이브리드 작업에서 사용되지 않습니다.

결과

양자 태스크: 퀀텀 태스크 결과는 S3 위치에 `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks` 저장됩니다.

Job results: 알고리즘 스크립트가 환경 변수로 지정된 디렉토리에 저장하는 모든 내용이 에서 지정한 S3 위치로 `"AMZN_BRAKET_JOB_RESULTS_DIR"` 복사됩니다`output_data_config`. 이 값을 지정

하지 않으면 기본값은 입니다. `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data` 알고리즘 스크립트에서 호출할 때 사전 형태로 결과를 편리하게 저장하는 데 사용할 수 있는 SDK 헬퍼 함수를 **`save_job_result`**제공합니다.

체크포인트: 체크포인트를 사용하려는 경우 환경 변수로 지정된 디렉터리에 체크포인트를 저장할 수 있습니다. `"AMZN_BRAKET_CHECKPOINT_DIR"` SDK 헬퍼 함수를 대신 사용할 수도 있습니다. `save_job_checkpoint`

알고리즘 지표: 하이브리드 작업이 실행되는 동안 CloudWatch Amazon으로 전송되고 Amazon Braket 콘솔에 실시간으로 표시되는 알고리즘 스크립트의 일부로 알고리즘 지표를 정의할 수 있습니다. 알고리즘 지표를 사용하는 방법에 대한 예는 Amazon Braket 하이브리드 작업을 사용하여 QAOA 알고리즘 실행을 참조하십시오.

환경 변수

AmazonBraket은 컨테이너 입력 및 출력과의 상호 작용을 단순화하는 몇 가지 환경 변수를 제공합니다. 다음 코드는 Braket이 사용하는 환경 변수를 나열합니다.

```
# 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"]
```

헬퍼 함수

AmazonBraket은 컨테이너 입력 및 출력과의 상호 작용을 단순화하는 여러 도우미 함수를 제공합니다. 이러한 도우미 함수는 Hybrid Job을 실행하는 데 사용되는 알고리즘 스크립트 내에서 호출됩니다. 다음 예제는 사용 방법을 보여줍니다.

```
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
```

작업 결과 저장

알고리즘 스크립트에서 생성된 결과를 저장하여 하이브리드 작업 스크립트의 하이브리드 작업 객체와 Amazon S3의 출력 폴더 (model.tar.gz 라는 tar-zip 파일) 에서 사용할 수 있도록 할 수 있습니다.

출력은 JavaScript 객체 표기법 (JSON) 형식을 사용하여 파일에 저장해야 합니다. numpy 배열의 경우처럼 데이터를 텍스트로 쉽게 직렬화할 수 없는 경우 피클된 데이터 형식을 사용하여 직렬화하는 옵션을 전달할 수 있습니다. 자세한 내용은 braket.jobs.data_persistence 모듈을 참조하십시오.

하이브리드 작업의 결과를 저장하려면 #ADD 주석이 달린 다음 줄을 알고리즘 스크립트에 추가합니다.

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

그런 다음 #ADD **print(job.result())**주석이 달린 줄을 추가하여 작업 스크립트의 작업 결과를 표시할 수 있습니다.

```
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
```

이 예제에서는 자세한 정보 출력을 `wait_until_complete=True` 억제하도록 제거했습니다. 디버깅을 위해 다시 추가할 수 있습니다. 이 하이브리드 작업을 실행하면 하이브리드 작업이 완료될 때까지 10초마다 하이브리드 작업의 상태와 식별자가 출력되고 `job-arn`COMPLETED, 이어서 벨 회로의 결과가 표시됩니다. 다음 예를 참조하세요.

```
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}]}
```

체크포인트를 사용하여 하이브리드 작업을 저장하고 다시 시작합니다.

체크포인트를 사용하여 하이브리드 작업의 중간 반복을 저장할 수 있습니다. 이전 섹션의 알고리즘 스크립트 예제에서 #ADD 주석이 달린 다음 줄을 추가하여 체크포인트 파일을 생성할 수 있습니다.

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

하이브리드 작업을 실행하면 체크포인트 디렉터리의 <jobname>하이브리드 작업 아티팩트에 기본 경로를 사용하여 -checkpoint-1.json 파일이 생성됩니다. `/opt/jobs/checkpoints` 이 기본 경로를 변경하지 않는 한 하이브리드 작업 스크립트는 변경되지 않습니다.

이전 하이브리드 작업에서 생성된 체크포인트에서 하이브리드 작업을 로드하려는 경우 알고리즘 스크립트가 사용합니다. `from braket.jobs import load_job_checkpoint` 알고리즘 스크립트에 로드되는 로직은 다음과 같습니다.

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

이 체크포인트를 로드한 후 로드된 콘텐츠를 기반으로 로직을 계속할 수 있습니다. `checkpoint-1`

> **Note**
>
> checkpoint_file_suffix는 체크포인트를 만들 때 이전에 지정한 접미사와 일치해야 합니다.

오케스트레이션 스크립트는 이전 하이브리드 작업의 내용을 #ADD 주석이 달린 줄에 지정해야 합니다`job-arn`.

```
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
    )
```

알고리즘 스크립트의 환경을 정의하세요.

AmazonBraket은 알고리즘 스크립트용으로 컨테이너별로 정의된 세 가지 환경을 지원합니다.

- 기본 컨테이너 (`image_uri`지정되지 않은 경우 기본값)

- 텐서플로와 텐서플로가 포함된 컨테이너 PennyLane
- 및 가 있는 컨테이너 PyTorch PennyLane

다음 표에는 컨테이너와 컨테이너에 포함된 라이브러리에 대한 세부 정보가 나와 있습니다.

아마존 브라켓 컨테이너

유형	PennyLane 너비 TensorFlow	PennyLane 와 PyTorch	페니레인
Base	292282985366.dkr.ecr.us-east-1.amazonaws.com /amazon-braket-tensor-flow-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: 최신
상속된 라이브러리	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
추가 라이브러리	• 아마존-브레이크-디폴트 시뮬레이터 • 아마존-브레이크-페니레인-플러그인 • 아마존 브래킷 스키마 • 아마존-브라켓-sdk • 아이피 커널 • 케라스 • 매트플롯립 • 네트워크 • 오픈바벨 • PennyLane • 프로토버프 • psi4	• 아마존 브레이크 디폴트 시뮬레이터 • 아마존-브레이크-페니레인-플러그인 • 아마존 브래킷 스키마 • 아마존-브라켓-sdk • 아이피 커널 • 케라스 • 매트플롯립 • 네트워크 • 오픈바벨 • PennyLane • 프로토버프 • psi4	• 아마존 브레이크 디폴트 시뮬레이터 • 아마존-브레이크-페니레인-플러그인 • 아마존 브래킷 스키마 • 아마존-브라켓-sdk • awscli • boto3 • 아이피 커널 • 매트플롯 라이브러리 • 네트워크 • numpy • 오픈바벨 • pandas

유형	PennyLane 너비 TensorFlow	PennyLane 와 PyTorch	페니레인
	• rsa • PennyLane- 라이트닝 - GPU • 큐 퀀텀	• rsa • PennyLane- 라이트닝 - GPU • 큐 퀀텀	• PennyLane • 프로토버프 • psi4 • rsa • scipy

AWS/amazon-braket-containers에서 오픈 소스 컨테이너 정의를 보고 액세스할 수 있습니다. 사용 사례에 가장 적합한 컨테이너를 선택하십시오. 컨테이너는 하이브리드 작업을 AWS 리전 호출하는 소스 내에 있어야 합니다. 하이브리드 작업을 생성할 때 하이브리드 작업 스크립트의 `create(…)` 호출에 다음 세 가지 인수 중 하나를 추가하여 컨테이너 이미지를 지정합니다. AmazonBraket 컨테이너는 인터넷에 연결되어 있으므로 시작 또는 런타임 비용을 지불하고 런타임 시 선택한 컨테이너에 추가 종속성을 설치할 수 있습니다. 다음 예는 us-west-2 지역을 위한 것입니다.

- 기본 이미지 이미지_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /amazon-braket-base-jobs:1.0-cpu-py39-ubuntu22.04”
- 텐서플로우 이미지 이미지_uri="292282985366.dkr. ecr.us-east-1.amazonaws.com /amazon-braket-tensorflow-jobs:2.11.0-gpu-py39-cu112-ubuntu20.04”
- PyTorch 이미지 이미지_uri="292282985366.dkr. ecr.us-west-2.amazonaws.com /amazon-braket-pytorch-jobs:1.13.1-gpu-py39-cu117-ubuntu20.04”

`image-uris`브라켓 SDK의 함수를 사용하여 검색할 수도 있습니다. `retrieve_image()` Amazon 다음 예제는 AWS 리전 us-west-2에서 검색하는 방법을 보여줍니다.

```
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")
```

하이퍼파라미터 사용

하이브리드 작업을 생성할 때 학습률이나 단계 크기 등 알고리즘에 필요한 하이퍼파라미터를 정의할 수 있습니다. 하이퍼파라미터 값은 일반적으로 알고리즘의 다양한 측면을 제어하는 데 사용되며, 알고

리즘의 성능을 최적화하도록 조정할 수도 있습니다. Braket 하이브리드 작업에서 하이퍼파라미터를 사용하려면 하이퍼파라미터의 이름과 값을 사전으로 명시적으로 지정해야 합니다. 단, 값은 문자열 데이터 유형이어야 합니다. 최적의 값 세트를 검색할 때 테스트하려는 하이퍼파라미터 값을 지정합니다. 하이퍼파라미터를 사용하는 첫 번째 단계는 하이퍼파라미터를 딕셔너리로 설정하고 정의하는 것입니다. 이 내용은 다음 코드에서 확인할 수 있습니다.

```
#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
}
```

그런 다음 선택한 알고리즘에서 사용할 수 있도록 위에 제공된 코드 스니펫에 정의된 하이퍼파라미터를 다음과 같은 형식으로 전달합니다.

```
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**
>
> 입력 데이터에 대해 자세히 알아보려면 입력 섹션을 참조하십시오.

그러면 다음 코드를 사용하여 하이퍼파라미터를 하이브리드 작업 스크립트에 로드합니다.

```
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**
>
> 입력 데이터 및 장치 arn과 같은 정보를 하이브리드 작업 스크립트에 전달하는 방법에 대한 자세한 내용은 이 github 페이지를 참조하십시오.

하이퍼파라미터 사용 방법을 배우는 데 매우 유용한 몇 가지 가이드는 Amazon Braket Hybrid Jobs가 포함된 PennyLaneQAOA와 Amazon Braket Hybrid Jobs 자습서의 Quantum 기계 학습을 제공합니다.

알고리즘 스크립트를 실행하도록 하이브리드 작업 인스턴스를 구성하십시오.

알고리즘에 따라 요구 사항이 다를 수 있습니다. 기본적으로 Amazon Braket은 `ml.m5.large` 인스턴스에서 알고리즘 스크립트를 실행합니다. 하지만 다음 가져오기 및 구성 인수를 사용하여 하이브리드 작업을 생성할 때 이 인스턴스 유형을 사용자 지정할 수 있습니다.

```
from braket.jobs.config import InstanceConfig

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

```
    ...
    ),
```

임베디드 시뮬레이션을 실행 중이고 장치 구성에서 로컬 장치를 지정한 경우 InstanceCount를 지정하고 둘 이상으로 설정하여 에서 두 개 이상의 인스턴스를 추가로 요청할 수 있습니다. InstanceConfig 상한은 5입니다. 예를 들어 다음과 같이 인스턴스 3개를 선택할 수 있습니다.

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

여러 인스턴스를 사용하는 경우 data parallel 기능을 사용하여 하이브리드 작업을 배포해 보세요. 이 Braket 예제를 보는 방법에 대한 자세한 내용은 다음 예제 노트북을 참조하십시오.

다음 세 표에는 표준, 컴퓨팅 최적화 및 가속화된 컴퓨팅 인스턴스에 사용할 수 있는 인스턴스 유형과 사양이 나와 있습니다.

> **Note**
>
> 하이브리드 작업의 기본 클래식 컴퓨팅 인스턴스 할당량을 보려면 이 페이지를 참조하십시오.

표준 인스턴스	vCPU	메모리
ml.m5.large (기본값)	2	8GiB
ml.m5.xlarge	4	16GiB
ml.m5.2xlarge	8	32GiB
ml.m5.4xlarge	16	64GiB
ml.m5.12xlarge	48	192GiB
ml.m5.24xlarge	96	384 GiB

표준 인스턴스	vCPU	메모리
ml.m4.xlarge	4	16GiB
ml.m4.2xlarge	8	32GiB
ml.m4.4xlarge	16	64GiB
ml.m4.10xlarge	40	256GiB

컴퓨팅 최적화 인스턴스	vCPU	메모리
ml.c4.xlarge	4	7.5GiB
ml.c4.2xlarge	8	15GiB
ml.c4.4xlarge	16	30 기가비트
ml.c4.8xlarge	36	192GiB
ml.c5.xlarge	4	8GiB
ml.c5.2xlarge	8	16GiB
ml.c5.4xlarge	16	32GiB
ml.c5.9xlarge	36	72 기가비트
ml.c5.18xlarge	72	144 기가바이트
ml.c5n.xlarge	4	10.5 GiB
ml.c5n.2xlarge	8	21 GiB
ml.c5n.4xlarge	16	42기가바이트
ml.c5n.9xlarge	36	96GiB
ml.c5n.18 xlarge	72	192GiB

액셀러레이티드 컴퓨팅 인스턴스	vCPU	메모리
ml.p2.xlarge	4	61GiB
ml.p2.8xlarge	32	488GiB
ml.p2.16xlarge	64	732GiB
ml.p3.2xlarge	8	61GiB
ml.p3.8xlarge	32	244GiB
ml.p3.16xlarge	64	488GiB
ml.g4dn.xlarge	4	16GiB
ml.g4dn.2xlarge	8	32GiB
ml.g4dn.4xlarge	16	64GiB
ml.g4dn.8xlarge	32	128GiB
ml.g4dn.12xlarge	48	192GiB
ml.g4dn.16xlarge	64	256GiB

> **Note**
>
> us-west-1에서는 p3 인스턴스를 사용할 수 없습니다. 하이브리드 작업에서 요청된 ML 컴퓨팅 파워를 프로비저닝할 수 없는 경우 다른 지역을 사용하세요.

각 인스턴스는 30GB의 기본 데이터 스토리지 (SSD) 구성을 사용합니다. 하지만 구성한 것과 같은 방식으로 스토리지를 조정할 수 `instanceType` 있습니다. 다음 예제는 총 스토리지를 50GB로 늘리는 방법을 보여줍니다.

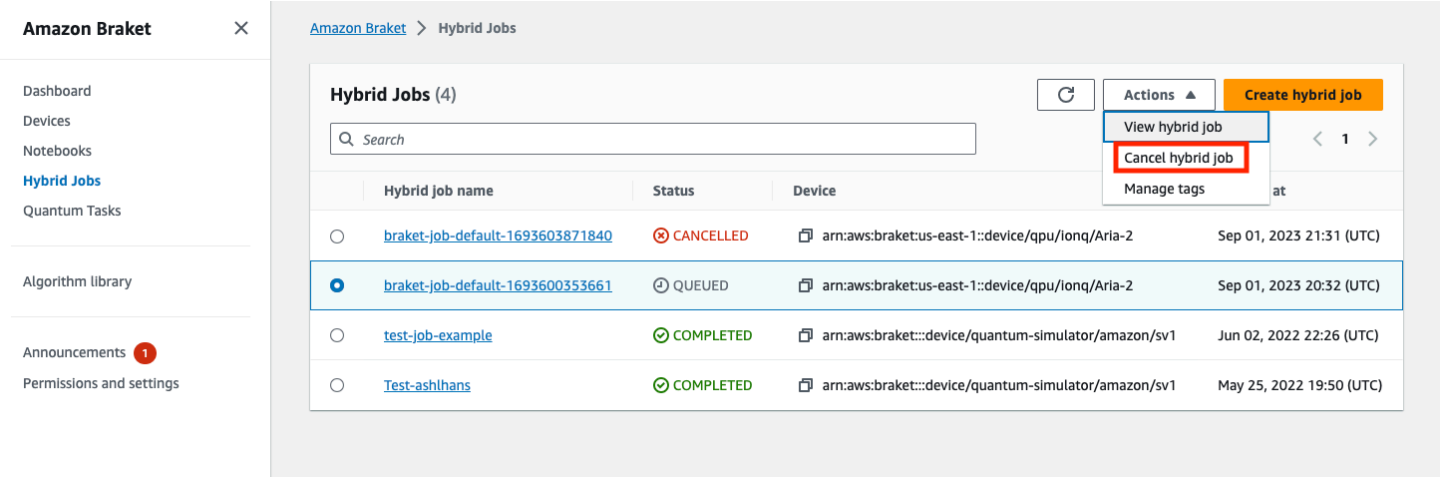
```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
```

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

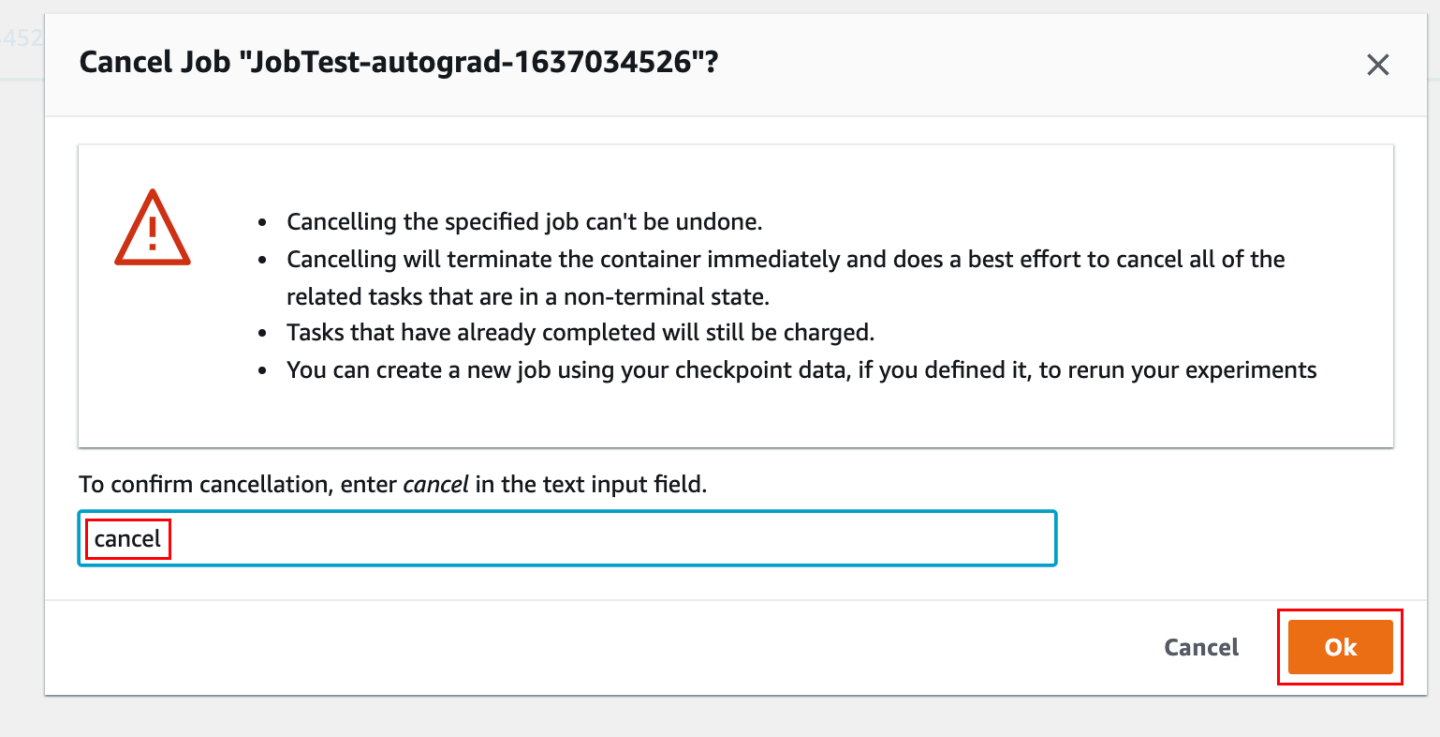
하이브리드 작업 취소

비터미널 상태에서는 하이브리드 작업을 취소해야 할 수 있습니다. 콘솔에서 또는 코드를 사용하여 이 작업을 수행할 수 있습니다.

콘솔에서 하이브리드 작업을 취소하려면 하이브리드 작업 페이지에서 취소할 하이브리드 작업을 선택한 다음 작업 드롭다운 메뉴에서 하이브리드 작업 취소를 선택합니다.



취소를 확인하려면 메시지가 표시되면 입력 필드에 취소를 입력한 다음 확인을 선택합니다.



Braket Python SDK의 코드를 사용하여 하이브리드 작업을 `job_arn` 취소하려면 다음 코드와 같이 를 사용하여 하이브리드 작업을 식별한 다음 해당 작업에 대한 `cancel` 명령을 호출합니다.

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

이 `cancel` 명령은 기존 하이브리드 작업 컨테이너를 즉시 종료하고 아직 비터미널 상태에 있는 관련 양자 작업을 모두 취소하기 위해 최선을 다합니다.

파라메트릭 컴파일을 사용하여 하이브리드 작업의 속도를 높입니다.

Amazon Braket은 특정 QPU에서 파라메트릭 컴파일을 지원합니다. 이를 통해 하이브리드 알고리즘의 모든 반복에 대해 회로를 한 번만 컴파일하지 않고 한 번만 컴파일하여 계산 비용이 많이 드는 컴파일 단계와 관련된 오버헤드를 줄일 수 있습니다. 이렇게 하면 각 단계에서 회로를 다시 컴파일할 필요가 없으므로 하이브리드 작업의 런타임을 크게 개선할 수 있습니다. 지원되는 QPU 중 하나에 파라미터화된 회로를 Braket Hybrid Job으로 제출하기만 하면 됩니다. 장기간 실행되는 하이브리드 작업의 경우 Braket은 회로를 컴파일할 때 하드웨어 제공업체의 업데이트된 보정 데이터를 자동으로 사용하여 최고 품질의 결과를 보장합니다.

파라메트릭 회로를 만들려면 먼저 파라미터를 알고리즘 스크립트의 입력으로 제공해야 합니다. 이 예제에서는 작은 파라메트릭 회로를 사용하고 각 반복 사이의 기존 처리는 무시합니다. 일반적인 워크로

드의 경우 많은 회로를 일괄적으로 제출하고 각 반복에서 매개 변수를 업데이트하는 것과 같은 일반적인 처리를 수행합니다.

```
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.")
```

다음 작업 스크립트와 함께 Hybrid Job으로 실행할 알고리즘 스크립트를 제출할 수 있습니다. 파라메트릭 컴파일을 지원하는 QPU에서 Hybrid Job을 실행하는 경우 회로는 처음 실행할 때만 컴파일됩니다. 다음 실행에서는 컴파일된 회로가 재사용되므로 추가 코드 줄 없이 Hybrid Job의 런타임 성능이 향상됩니다.

```
from braket.aws import AwsQuantumJob

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

> **Note**
>
> 파라메트릭 컴파일은 펄스 레벨 프로그램을 제외한 모든 초전도 게이트 기반 QPU에서 Rigetti Computing 지원됩니다. Oxford Quantum Circuits

아마존 PennyLane 브라켓과 함께 사용

하이브리드 알고리즘은 고전적 명령어와 양자적 명령을 모두 포함하는 알고리즘입니다. 기존 명령은 기존 하드웨어 (EC2 인스턴스 또는 랩톱) 에서 실행되고 양자 명령은 시뮬레이터 또는 양자 컴퓨터에서 실행됩니다. 하이브리드 작업 기능을 사용하여 하이브리드 알고리즘을 실행하는 것이 좋습니다. 자세한 내용은 Amazon Braket 작업을 사용하는 경우를 참조하십시오.

AmazonBraket을 사용하면 Braket PennyLane 플러그인 또는 Amazon Amazon Braket Python SDK 및 예제 노트북 리포지토리를 사용하여 하이브리드 양자 알고리즘을 설정하고 실행할 수 있습니다. Amazon SDK를 기반으로 하는 Braket 예제 노트북을 사용하면 플러그인 없이 특정 하이브리드 알고리즘을 설정하고 실행할 수 있습니다. PennyLane 하지만 더 풍부한 경험을 PennyLane 제공하므로 사용하는 것이 좋습니다.

하이브리드 양자 알고리즘에 대해

현대의 양자 컴퓨팅 장치는 일반적으로 잡음이 발생하여 오류가 발생하기 때문에 하이브리드 양자 알고리즘은 오늘날 업계에서 중요합니다. 계산에 양자 게이트가 추가될 때마다 잡음이 추가될 가능성이 높아집니다. 따라서 장시간 실행되는 알고리즘은 잡음에 압도되어 계산 오류가 발생할 수 있습니다.

Shor's (양자 위상 추정 예제) 나 Grover (Grover의 예제) 와 같은 순수 양자 알고리즘에는 수천 또는 수백만 번의 연산이 필요합니다. 이러한 이유로 일반적으로 잡음이 많은 중간 규모 양자 (NISQ) 장치라고 하는 기존 양자 장치에는 실용적이지 않을 수 있습니다.

하이브리드 양자 알고리즘에서 양자 처리 장치 (QPU) 는 기존 CPU의 보조 프로세서 역할을 하며, 특히 기존 알고리즘의 특정 계산 속도를 높이기 위한 것입니다. 회로 실행 시간이 훨씬 짧아져 오늘날 장치의 기능을 감당할 수 있을 정도로 줄어듭니다.

아마존 브라켓 포함 PennyLane

AmazonBraket은 양자 미분 PennyLane프로그래밍 개념을 중심으로 구축된 오픈 소스 소프트웨어 프레임워크를 지원합니다. 이 프레임워크를 사용하여 양자 화학, 양자 기계 학습 및 최적화의 계산 문제에 대한 솔루션을 찾기 위해 신경망을 훈련시키는 것과 같은 방식으로 양자 회로를 훈련할 수 있습니다.

PennyLane 라이브러리는 PyTorch 및 TensorFlow 를 비롯한 친숙한 기계 학습 도구에 대한 인터페이스를 제공하여 양자 회로를 빠르고 직관적으로 훈련할 수 있도록 합니다.

- PennyLane 라이브러리 -- Amazon Braket 노트북에 사전 PennyLane 설치되어 있습니다. 에서 Amazon PennyLane Braket 장치에 액세스하려면 노트북을 열고 다음 명령을 사용하여 PennyLane 라이브러리를 가져오십시오.

```
import pennylane as qml
```

튜토리얼 노트북은 빠르게 시작하는 데 도움이 됩니다. 또는 원하는 IDE의 PennyLane Amazon Braket에서 사용할 수도 있습니다.

- AmazonBraket PennyLane 플러그인 — 자체 IDE를 사용하려면 Amazon Braket PennyLane 플러그인을 수동으로 설치하면 됩니다. 플러그인은 Amazon Braket Python PennyLane SDK와 연결되므로 브라켓 PennyLane 디바이스에서 Amazon 회로를 실행할 수 있습니다. PennyLane 플러그인을 설치하려면 다음 명령을 사용하십시오.

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

다음 예제는 Amazon Braket 장치에 대한 액세스를 설정하는 방법을 보여줍니다. 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)
```

자습서 예제 및 이에 대한 PennyLane 자세한 내용은 Amazon Braket 예제 리포지토리를 참조하십시오.

AmazonBraket PennyLane 플러그인을 사용하면 한 줄의 코드로 Amazon Braket QPU와 임베디드 시뮬레이터 디바이스 간에 PennyLane 전환할 수 있습니다. 함께 사용할 수 있는 두 개의 Amazon Braket 양자 장치를 제공합니다. PennyLane

- `braket.aws.qubit`QPU 및 시뮬레이터를 포함한 Amazon Braket 서비스의 양자 장치와 함께 실행하는 데 적합합니다.
- `braket.local.qubit`AmazonBraket SDK의 로컬 시뮬레이터로 실행하는 경우

AmazonBraket PennyLane 플러그인은 오픈 소스입니다. PennyLane 플러그인 GitHub 저장소에서 설치할 수 있습니다.

에 대한 PennyLane 자세한 내용은 PennyLane 웹 사이트의 설명서를 참조하십시오.

Amazon Braket 예제 노트북의 하이브리드 알고리즘

AmazonBraket은 하이브리드 알고리즘 실행을 위해 PennyLane 플러그인을 사용하지 않는 다양한 예제 노트북을 제공합니다. QAOA (양자 근사치 최적화 알고리즘) 또는 VQE (변량 양자 고유해석기) 와 같은 변형 방법을 설명하는 Amazon Braket 하이브리드 예제 노트북 중 아무거나 사용하여 시작할 수 있습니다.

Amazon브라켓 예제 노트북은 아마존 브라켓 파이썬 SDK를 사용합니다. SDK는 Braket을 통해 양자 컴퓨팅 하드웨어 디바이스와 상호 작용할 수 있는 프레임워크를 제공합니다. Amazon 하이브리드 워크플로의 양자 부분을 지원하도록 설계된 오픈 소스 라이브러리입니다.

예제 노트북을 통해 Amazon Braket을 더 자세히 살펴볼 수 있습니다.

시뮬레이터가 내장된 하이브리드 알고리즘 PennyLane

Amazon브라켓 하이브리드 잡스는 이제 의 고성능 CPU 및 GPU 기반 임베디드 시뮬레이터와 함께 제공됩니다. PennyLane 이 임베디드 시뮬레이터 제품군은 하이브리드 작업 컨테이너에 직접 내장될 수 있으며 고속 상태 벡터 `lightning.qubit` 시뮬레이터, NVIDIA의 CuQuantum 라이브러리를 사용하여 가속화된 `lightning.gpu` 시뮬레이터 등을 포함합니다. 이러한 임베디드 시뮬레이터는 인접 차별화 방법과 같은 고급 방법을 활용할 수 있는 양자 기계 학습과 같은 변형 알고리즘에 매우 적합합니다. 하나 이상의 CPU 또는 GPU 인스턴스에서 이러한 임베디드 시뮬레이터를 실행할 수 있습니다.

Hybrid Jobs를 사용하면 이제 기존 보조 프로세서와 QPU, Amazon Braket 온디맨드 시뮬레이터 등의 SV1 조합을 사용하거나 에서 내장된 시뮬레이터를 직접 사용하여 변형 알고리즘 코드를 실행할 수 있습니다. PennyLane

임베디드 시뮬레이터는 이미 Hybrid Jobs 컨테이너와 함께 사용할 수 있습니다. 기본 Python 함수를 `@hybrid_job` 데코레이터로 장식하기만 하면 됩니다. PennyLane `lightning.gpu`시뮬레이터를 사용하려면 다음 코드 스니펫과 `InstanceConfig` 같이 GPU 인스턴스도 지정해야 합니다.

```
import pennylane as qml
from braket.jobs import hybird_job
from braket.jobs.config import InstanceConfig


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

Hybrid Jobs와 함께 PennyLane 임베디드 시뮬레이터 사용을 시작하려면 예제 노트북을 참조하십시오.

Amazon Braket 시뮬레이터를 PennyLane 사용한 인더조인트 그래디언트

Amazon Braket용 PennyLane 플러그인을 사용하면 로컬 상태 벡터 시뮬레이터 또는 SV1에서 실행할 때 보조 미분 방법을 사용하여 그래디언트를 계산할 수 있습니다.

참고: 인접 미분 방법을 사용하려면 다음과 같이 지정하고 지정하지 않아야 합니다. **`diff_method='device'` `qnode`** `diff_method='adjoint'` 다음 예를 참조하세요.

```
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**
>
> 현재는 QAOA 해밀턴의 그룹화 지수를 계산하고 이를 사용하여 해밀턴을 여러 기대값으로 분할할 PennyLane 예정입니다. 에서 PennyLane QAOA를 실행할 때 SV1의 보조 미분 기능을 사용하려면 다음과 같이 그룹화 인덱스를 제거하여 비용 해밀턴을 재구성해야 합니다. `cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False) cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

Amazon Braket 하이브리드 작업을 사용하고 QAOA PennyLane 알고리즘을 실행하십시오.

이 섹션에서는 학습한 내용을 바탕으로 파라메트릭 컴파일을 PennyLane 사용하여 실제 하이브리드 프로그램을 작성해 보겠습니다. 알고리즘 스크립트를 사용하여 양자 근사 최적화 알고리즘 (QAOA) 문제를 해결합니다. 이 프로그램은 일반적인 Max Cut 최적화 문제에 해당하는 비용 함수를 만들고, 파라미터화된 양자 회로를 지정하고, 비용 함수가 최소화되도록 간단한 경사하강법 방법을 사용하여 파라미터를 최적화합니다. 이 예제에서는 단순화를 위해 알고리즘 스크립트에서 문제 그래프를 생성하지만, 보다 일반적인 사용 사례에서는 입력 데이터 구성의 전용 채널을 통해 문제 사양을 제공하는 것이 가장 좋습니다. 플래그는 `parametrize_differentiable` 기본적으로 로 `True` 설정되므로 지원되는 QPU에서 파라메트릭 컴파일을 통해 향상된 런타임 성능의 이점을 자동으로 얻을 수 있습니다.

```
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**
>
> 파라메트릭 컴파일은 펄스 레벨 프로그램을 제외한 모든 초전도 게이트 기반 QPU에서 Rigetti Computing 지원됩니다. Oxford Quantum Circuits

의 임베디드 시뮬레이터로 하이브리드 워크로드를 가속화하십시오. PennyLane

Amazon Braket Hybrid Jobs의 임베디드 시뮬레이터를 사용하여 하이브리드 워크로드를 실행하는 방법을 살펴보겠습니다. PennyLane Pennylane의 GPU 기반 임베디드 시뮬레이터는 Nvidia CuQuantum `lightning.gpu` 라이브러리를 사용하여 회로 시뮬레이션을 가속화합니다. 임베디드 GPU 시뮬레이터는 사용자가 즉시 사용할 수 있는 모든 Braket 작업 컨테이너에 사전 구성되어 있습니다. 이 페이지에서는 하이브리드 워크로드의 속도를 높이는 `lightning.gpu` 데 사용하는 방법을 보여줍니다.

양자 근사치 최적화 알고리즘 **`lightning.gpu`** 워크로드에 사용

이 노트북에 수록된 양자 근사 최적화 알고리즘 (QAOA) 예제를 고려해 보십시오. 임베디드 시뮬레이터를 선택하려면 `device` 인수를 다음과 같은 형식의 문자열로 지정합니다. `"local:<provider>/<simulator_name>"` 예를 들어, 를 다음과 같이 설정합니다 `"local:pennylane/lightning.gpu"``lightning.gpu`. Hybrid Job을 시작할 때 제공하는 장치 문자열은 작업에 환경 변수로 `"AMZN_BRAKET_DEVICE_ARN"` 전달됩니다.

```
device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
prefix, device_name = device_string.split("/")
device = qml.device(simulator_name, wires=n_wires)
```

이 페이지에서는 두 개의 내장된 PennyLane 상태 벡터 시뮬레이터 `lightning.qubit` (CPU 기반)와 `lightning.gpu` (GPU 기반) 를 비교해 보겠습니다. 다양한 그래디언트를 계산하려면 시뮬레이터에 몇 가지 사용자 지정 게이트 분해를 제공해야 합니다.

이제 하이브리드 작업 시작 스크립트를 준비할 준비가 되었습니다. 두 가지 인스턴스 유형인 및 을 사용하여 QAOA 알고리즘을 실행합니다. `m5.2xlarge` `p3.2xlarge` `m5.2xlarge`인스턴스 유형은 표준 개발자 랩톱과 비슷합니다. `p3.2xlarge`이 인스턴스는 16GB 메모리의 NVIDIA Volta GPU 1개를 탑재한 가속화된 컴퓨팅 인스턴스입니다.

모든 하이브리드 작업의 `hyperparameters` 용도는 동일합니다. 다른 인스턴스와 시뮬레이터를 시험해 보려면 다음과 같이 두 줄을 바꾸기만 하면 됩니다.

```
# 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')
```

또는:

```
# 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**
>
> 를 GPU 기반 인스턴스를 사용하여 **`instance_config`** as를 지정하고 를 임베디드 CPU 기반 시뮬레이터 (**`lightning.qubit`**) **`device`** 로 선택하면 GPU는 사용되지 않습니다. GPU를 대상으로 하려면 반드시 내장된 GPU 기반 시뮬레이터를 사용하십시오!

먼저 두 개의 하이브리드 작업을 생성하고 꼭짓점이 18개인 그래프에서 QAOA를 사용하여 Max-Cut을 풀 수 있습니다. 이는 18큐비트 회로로 해석됩니다. 이 회로는 비교적 작아서 노트북이나 인스턴스에서 빠르게 실행할 수 있습니다. `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,
)
```

인스턴스의 평균 반복 시간은 약 25초인 반면, `m5.2xlarge` 인스턴스의 경우 약 12초입니다. `p3.2xlarge` 이 18큐비트 워크플로의 경우 GPU 인스턴스는 속도를 2배 향상시킵니다. Amazon Braket Hybrid Jobs 요금 페이지를 보면 인스턴스의 분당 비용이 0.00768 달러인 반면 `m5.2xlarge` 인스턴스의 경우 0.06375 달러임을 알 수 있습니다. `p3.2xlarge` 여기서 수행한 것처럼 총 5회 반복을 실행하려면 CPU 인스턴스를 사용할 경우 0.016달러, GPU 인스턴스를 사용하면 0.06375달러가 듭니다. 둘 다 상당히 저렴합니다.

이제 문제를 더 어렵게 만들고 24큐비트로 변환되는 24개의 꼭짓점 그래프에서 Max-Cut 문제를 풀어 보겠습니다. 동일한 두 인스턴스에서 하이브리드 작업을 다시 실행하고 비용을 비교하십시오.

> **Note**
>
> CPU 인스턴스에서 이 하이브리드 작업을 실행하는 데 걸리는 시간은 약 5시간이라는 것을 알 수 있습니다!

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

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

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

```
)
```

m5.2xlarge인스턴스의 평균 반복 시간은 약 1시간이고, p3.2xlarge 인스턴스의 경우 약 2분입니다. 이 더 큰 문제의 경우 GPU 인스턴스가 훨씬 더 빠릅니다! 이 속도 향상의 이점을 누리려면 코드 두 줄을 변경하여 인스턴스 유형과 사용된 로컬 시뮬레이터를 교체하기만 하면 되었습니다. 여기에서와 같이 총 5회 반복을 실행하려면 CPU 인스턴스를 사용할 경우 약 2.27072 USD, GPU 인스턴스를 사용할 경우 약 0.775625 USD의 비용이 듭니다. CPU 사용량은 더 비쌀 뿐만 아니라 실행하는 데 더 많은 시간이 걸립니다. CuQuantumNVIDIA가 지원하는 임베디드 시뮬레이터를 사용하여 사용 가능한 GPU 인스턴스로 이 워크플로를 가속화하면 총 비용을 줄이고 시간을 단축하면서 중간 큐비트 수 (20~30) PennyLane 의 워크플로를 실행할 수 있습니다. AWS즉, 노트북이나 비슷한 크기의 인스턴스에서 빠르게 실행하기에는 너무 큰 문제라도 양자 컴퓨팅을 실험해 볼 수 있습니다.

양자 기계 학습 및 데이터 병렬화

워크로드 유형이 데이터세트를 기반으로 학습하는 QML (양자 기계 학습) 인 경우 데이터 병렬화를 사용하여 워크로드를 더욱 가속화할 수 있습니다. QML의 모델에는 하나 이상의 양자 회로가 포함되어 있습니다. 모델에는 기존 신경망이 포함될 수도 있고 포함되지 않을 수도 있습니다. 데이터셋으로 모델을 훈련시키는 경우 모델의 파라미터가 업데이트되어 손실 함수를 최소화합니다. 일반적으로 손실 함수는 단일 데이터 포인트에 대해 정의되고 전체 데이터셋의 평균 손실에 대한 총 손실은 정의됩니다. QML에서 손실은 일반적으로 그래디언트 계산을 위해 총 손실로 평균화하기 전에 순차적으로 계산됩니다. 이 절차는 특히 수백 개의 데이터 포인트가 있는 경우 시간이 많이 걸립니다.

한 데이터 포인트의 손실은 다른 데이터 포인트에 의존하지 않으므로 손실을 병렬로 평가할 수 있습니다! 여러 데이터 포인트와 관련된 손실과 기울기를 동시에 평가할 수 있습니다. 이를 데이터 병렬화라고 합니다. 분산 데이터 병렬 라이브러리를 사용하는 SageMaker Amazon Braket Hybrid Jobs를 사용하면 데이터 병렬화를 보다 쉽게 활용하여 교육을 가속화할 수 있습니다.

잘 알려진 UCI 리포지토리의 Sonar 데이터 세트를 이진 분류의 예로 사용하는 다음과 같은 데이터 병렬 처리를 위한 QML 워크로드를 고려해 보십시오. Sonar 데이터셋에는 208개의 데이터 포인트가 있으며, 각 데이터 포인트는 물질에서 반사되는 소나 신호에서 수집한 60개의 특징을 가지고 있습니다. 각 데이터 포인트는 광산의 경우 “M”, 암석의 경우 “R”로 표시됩니다. QML 모델은 입력 계층, 은닉 계층인 양자 회로, 출력 계층으로 구성되어 있습니다. 입력 및 출력 계층은 에서 PyTorch 구현되는 고전적인 신경망입니다. 양자 회로는 PennyLane 의 PyTorch qml.qnn 모듈을 사용하여 신경망과 통합됩니다. 워크로드에 대한 자세한 내용은 예제 노트북을 참조하십시오. 위의 QAOA 예제와 마찬가지로, 와 같은 임베디드 GPU 기반 시뮬레이터를 사용하여 GPU의 성능을 활용하여 임베디드 CPU 기반 시뮬레이터보다 성능을 개선할 수 있습니다. PennyLane `lightning.gpu`

하이브리드 작업을 만들려면 키워드 인수를 통해 알고리즘 스크립트, 장치 `AwsQuantumJob.create` 및 기타 구성을 호출하고 지정할 수 있습니다.

```
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,
    ...
)
```

데이터 병렬화를 사용하려면 SageMaker 분산 라이브러리의 알고리즘 스크립트에서 몇 줄의 코드를 수정하여 학습을 올바르게 병렬화해야 합니다. 먼저 여러 GPU와 여러 인스턴스에 워크로드를 분산하는 데 필요한 대부분의 번거로운 작업을 수행하는 `smdistributed` 패키지를 가져옵니다. 이 패키지는 브라켓과 컨테이너에 사전 구성되어 있습니다. PyTorch TensorFlow 이 `dist` 모듈은 훈련(`world_size`) 에 사용할 총 GPU 수와 GPU `local_rank` 코어의 끝을 알고리즘 스크립트에 알려줍니다. `rank` `rank`는 모든 인스턴스에 대한 GPU의 절대 인덱스이고, `local_rank` 는 인스턴스 내 GPU의 절대 인덱스입니다. 예를 들어, 각각 8개의 GPU가 훈련용으로 할당된 인스턴스가 4개 있는 경우 `rank` 범위는 0에서 31이고 범위는 0에서 `local_rank` 7까지입니다.

```
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)
```

다음으로, `world_size` `rank` 및 `DistributedSampler` 에 따라 a를 정의한 다음 데이터 로더에 전달합니다. 이 샘플러를 사용하면 GPU가 동일한 데이터세트 조각에 액세스하는 것을 방지할 수 있습니다.

```
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,
)
```

다음으로 DistributedDataParallel 클래스를 사용하여 데이터 병렬화를 활성화합니다.

```
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"])
```

위 내용은 데이터 병렬화를 사용하는 데 필요한 변경 사항입니다. QML에서는 결과를 저장하고 학습 진행 상황을 인쇄하려는 경우가 많습니다. 각 GPU가 저장 및 인쇄 명령을 실행하면 로그에 반복되는 정보가 넘쳐나고 결과가 서로 겹쳐쓰게 됩니다. 이를 방지하려면 0인 GPU에서만 저장하고 인쇄할 수 있습니다. rank

```
if dp_info["rank"]==0:
    print('elapsed time: ', elapsed)
    torch.save(model.state_dict(), f"{output_dir}/test_local.pt")
    save_job_result({"last loss": loss_before})
```

Amazon Braket 하이브리드 잡스는 SageMaker 분산 데이터 병렬 라이브러리의 ml.p3.16xlarge 인스턴스 유형을 지원합니다. 하이브리드 작업의 InstanceConfig 인수를 통해 인스턴스 유형을 구성합니다. SageMaker 분산 데이터 병렬 라이브러리가 데이터 병렬화가 활성화되었는지 확인하려면 사용 중인 인스턴스 유형으로 "sagemaker_distributed_dataparallel_enabled" 설정하고 설정하는 두 개의 추가 하이퍼파라미터를 추가해야 합니다. "true" "sagemaker_instance_type" 이 두 하이퍼파라미터는 패키지에서 사용됩니다. smdistributed 알고리즘 스크립트는 이를 명시적으로 사용할 필요가 없습니다. Amazon Braket SDK에서는 편리한 키워드 인수를 제공합니다. distribution 하이브리드 작업 distribution="data_parallel" 생성의 경우 Amazon Braket

SDK는 두 개의 하이퍼파라미터를 자동으로 삽입합니다. Amazon Braket API를 사용하는 경우 이러한 두 하이퍼파라미터를 포함해야 합니다.

인스턴스 및 데이터 병렬화가 구성되었으므로 이제 하이브리드 작업을 제출할 수 있습니다. 인스턴스에는 8개의 GPU가 있습니다. `ml.p3.16xlarge` 설정하면 `instanceCount=1` 인스턴스의 8개 GPU에 워크로드가 분산됩니다. 두 개 `instanceCount` 이상으로 설정하면 모든 인스턴스에서 사용 가능한 GPU에 워크로드가 분산됩니다. 여러 인스턴스를 사용하는 경우 사용 시간에 따라 각 인스턴스에 요금이 부과됩니다. 예를 들어 인스턴스 4개를 사용하는 경우 워크로드를 동시에 실행하는 인스턴스가 4개이기 때문에 청구 가능 시간은 인스턴스당 실행 시간의 4배입니다.

```
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**
>
> 위의 하이브리드 작업 생성에서는 `train_dp.py` 데이터 병렬화를 사용하기 위한 수정된 알고리즘 스크립트입니다. 데이터 병렬화는 위 섹션에 따라 알고리즘 스크립트를 수정할 때만 제대로 작동한다는 점에 유의하세요. 올바르게 수정된 알고리즘 스크립트 없이 데이터 병렬화 옵션을 활성화하면 하이브리드 작업에서 오류가 발생하거나 각 GPU가 동일한 데이터 슬라이스를 반복적으로 처리하여 비효율적일 수 있습니다.

위에서 언급한 이진 분류 문제를 해결하기 위해 26큐비트 양자 회로로 모델을 학습시키는 예제를 통해 실행 시간과 비용을 비교해 보겠습니다. 이 `ml.p3.16xlarge` 예제에 사용된 인스턴스의 요금은 분당

0.4692 USD입니다. 데이터 병렬화를 사용하지 않는 경우 시뮬레이터가 모델을 1 에포크 (즉, 208개 이상의 데이터 포인트) 동안 학습시키는 데 약 45분이 걸리며 비용은 약 20달러입니다. 인스턴스 1개와 인스턴스 4개에 걸친 데이터 병렬화를 사용하면 각각 6분, 1.5분밖에 걸리지 않으며, 이는 두 인스턴스 모두 약 2.8 USD에 해당합니다. 4개 인스턴스에서 데이터 병렬화를 사용하면 실행 시간이 30배 향상될 뿐만 아니라 비용도 몇 배나 절감할 수 있습니다!

로컬 모드로 하이브리드 작업을 빌드하고 디버그하세요.

새 하이브리드 알고리즘을 빌드하는 경우 로컬 모드를 사용하면 알고리즘 스크립트를 디버깅하고 테스트할 수 있습니다. 로컬 모드는 Amazon Braket Hybrid Jobs에서 사용하려는 코드를 실행할 수 있는 기능이지만 하이브리드 작업을 실행하기 위한 인프라를 Braket에서 관리할 필요가 없습니다. 대신 Braket Notebook 인스턴스 또는 랩톱 또는 데스크톱 컴퓨터와 같은 기본 클라이언트에서 로컬로 하이브리드 작업을 실행합니다. 로컬 모드에서는 여전히 실제 장치로 양자 작업을 전송할 수 있지만 로컬 모드에서 실제 QPU를 사용하여 실행하면 성능상의 이점을 얻을 수 없습니다.

로컬 모드를 사용하려면 발생 `LocalQuantumJob` 위치에 `AwsQuantumJob` 맞게 수정하세요. 예를 들어, 첫 번째 하이브리드 작업 생성의 예제를 실행하려면 다음과 같이 하이브리드 작업 스크립트를 편집하십시오.

```
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**
>
> 이 기능을 사용하려면 Amazon Braket 노트북에 이미 사전 설치되어 있는 Docker를 로컬 환경에 설치해야 합니다. Docker 설치 지침은 여기에서 확인할 수 있습니다. 또한 모든 매개변수가 로컬 모드에서 지원되는 것은 아닙니다.

자체 컨테이너 가져오기 (BYOC)

Amazon Braket Hybrid Jobs는 다양한 환경에서 코드를 실행할 수 있도록 사전 구축된 세 개의 컨테이너를 제공합니다. 이러한 컨테이너 중 하나가 사용 사례를 지원하는 경우 하이브리드 작업을 생성할

때 알고리즘 스크립트만 제공하면 됩니다. 누락된 사소한 종속성은 알고리즘 스크립트나 를 사용하는 `requirements.txt` `pip` 파일에서 추가할 수 있습니다.

이러한 컨테이너 중 사용 사례를 지원하는 컨테이너가 없거나 해당 컨테이너를 확장하려는 경우 Braket Hybrid Jobs는 자체 사용자 지정 Docker 컨테이너 이미지 또는 BYOC (Bring Your Own Container) 를 사용하여 하이브리드 작업을 실행할 수 있도록 지원합니다. 하지만 본격적으로 살펴보기 전에 이 기능이 실제로 사용 사례에 적합한 기능인지 확인해 보겠습니다.

자체 컨테이너를 가져오는 것이 언제 올바른 결정일까요?

자체 컨테이너 (BYOC) 를 Braket Hybrid Jobs로 가져오면 자체 소프트웨어를 패키지 환경에 설치하여 유연하게 사용할 수 있습니다. 특정 요구 사항에 따라 전체 BYOC Docker 빌드 (Amazon ECR 업로드) - 사용자 지정 이미지 URI 주기를 거치지 않고도 동일한 유연성을 얻을 수 있는 방법이 있을 수 있습니다.

> **Note**
>
> 공개적으로 사용할 수 있는 소수의 추가 Python 패키지 (일반적으로 10개 미만) 를 추가하려는 경우 BYOC가 올바른 선택이 아닐 수 있습니다. 예를 들어, 를 사용하는 경우. PyPi

이 경우 사전 빌드된 Braket 이미지 중 하나를 사용한 다음 작업 제출 시 소스 디렉터리에 `requirements.txt` 파일을 포함시킬 수 있습니다. 파일이 자동으로 읽히고 `pip` 지정된 버전의 패키지가 정상적으로 설치됩니다. 많은 수의 패키지를 설치하는 경우 작업 런타임이 크게 늘어날 수 있습니다. Python 및 해당하는 경우 사용하려는 사전 빌드된 컨테이너의 CUDA 버전을 확인하여 소프트웨어가 작동하는지 테스트하십시오.

BYOC는 작업 스크립트에 파이썬이 아닌 언어 (예: C++ 또는 Rust) 를 사용하거나 Braket의 사전 빌드된 컨테이너를 통해 사용할 수 없는 Python 버전을 사용하려는 경우에 필요합니다. 다음과 같은 경우에도 좋은 선택입니다.

- 라이선스 키가 있는 소프트웨어를 사용하고 있는데 소프트웨어를 실행하려면 라이선스 서버에 대해 해당 키를 인증해야 합니다. BYOC를 사용하면 Docker 이미지에 라이센스 키를 포함하고 인증 코드를 포함할 수 있습니다.
- 공개되지 않은 소프트웨어를 사용하고 있습니다. 예를 들어, 액세스하려면 특정 SSH 키가 필요한 사설 GitLab 또는 GitHub 저장소에서 소프트웨어가 호스팅됩니다.
- Braket에서 제공하는 컨테이너에 패키지되지 않은 대규모 소프트웨어 제품군을 설치해야 합니다. BYOC를 사용하면 소프트웨어 설치로 인한 하이브리드 작업 컨테이너의 긴 시작 시간을 없앨 수 있습니다.

또한 BYOC를 사용하면 소프트웨어로 Docker 컨테이너를 구축하고 사용자가 사용할 수 있도록 함으로써 고객이 맞춤형 SDK 또는 알고리즘을 사용할 수 있도록 할 수 있습니다. Amazon ECR에서 적절한 권한을 설정하여 이 작업을 수행할 수 있습니다.

> **Note**
>
> 모든 해당 소프트웨어 라이선스를 준수해야 합니다.

자체 컨테이너 반입 레시피

이 섹션에서는 Braket Hybrid Jobs에 bring your own container (BYOC) 필요한 사항에 대한 step-by-step 가이드를 제공합니다. 즉, 사용자 지정 Docker 이미지를 시작하고 실행하기 위해 이를 결합하는 스크립트, 파일 및 단계를 설명합니다. 다음과 같은 두 가지 일반적인 경우에 대한 레시피를 제공합니다.

1. Docker이미지에 추가 소프트웨어를 설치하고 작업에는 Python 알고리즘 스크립트만 사용하십시오.
2. 하이브리드 작업에서는 Python이 아닌 언어로 작성된 알고리즘 스크립트 또는 x86 이외의 CPU 아키텍처를 사용하십시오.

케이스 2의 경우 컨테이너 입력 스크립트를 정의하는 것이 더 복잡합니다.

Braket은 하이브리드 작업을 실행할 때 요청된 개수와 유형의 Amazon EC2 인스턴스를 시작한 다음 이미지 URI 입력으로 지정된 이미지를 Docker 실행하여 해당 인스턴스에서 작업을 생성합니다. BYOC 기능을 사용할 때는 읽기 액세스 권한이 있는 프라이빗 Amazon ECR 리포지토리에 호스팅되는 이미지 URI를 지정합니다. Braket Hybrid Jobs는 해당 사용자 지정 이미지를 사용하여 작업을 실행합니다.

하이브리드 작업과 함께 사용할 수 있는 Docker 이미지를 만드는 데 필요한 특정 구성 요소 작성 및 `Dockerfiles` 빌드에 익숙하지 않은 경우 이 지침을 읽는 동안 필요에 따라 Dockerfile 설명서 및 Amazon ECR CLI설명서를 참조하는 것이 좋습니다.

필요한 사항에 대한 개요는 다음과 같습니다.

- Dockerfile의 기본 이미지
- (선택 사항) 수정된 컨테이너 진입점 스크립트
- 필요한 소프트웨어를 모두 설치하고 컨테이너 스크립트를 Dockerfile 포함하는 A.

Dockerfile의 기본 이미지

Python을 사용하고 Braket에서 제공하는 컨테이너 위에 소프트웨어를 설치하려는 경우 기본 이미지 옵션은 GitHub 리포지토리와 Amazon ECR에서 호스팅되는 Braket 컨테이너 이미지 중 하나입니다. 이미지를 가져와서 그 위에 빌드하려면 Amazon ECR에 인증해야 합니다. 예를 들어 BYOC Docker 파일의 첫 번째 줄은 다음과 같을 수 있습니다. `FROM [IMAGE_URI_HERE]`

그런 다음 나머지 Dockerfile 내용을 입력하여 컨테이너에 추가할 소프트웨어를 설치하고 설정합니다. 사전 빌드된 Braket 이미지에는 적절한 컨테이너 진입점 스크립트가 이미 포함되어 있으므로 포함에 대해 걱정할 필요가 없습니다.

C++, Rust 또는 Julia와 같은 비Python 언어를 사용하거나 ARM과 같이 x86이 아닌 CPU 아키텍처용 이미지를 빌드하려는 경우 베어본 공개 이미지를 기반으로 빌드해야 할 수 있습니다. Amazon Elastic 컨테이너 레지스트리 공개 갤러리에서 이러한 이미지를 많이 찾을 수 있습니다. CPU 아키텍처에 적합한 것을 선택하고 필요한 경우 사용하려는 GPU를 선택하십시오.

(선택 사항) 수정된 컨테이너 진입점 스크립트

> **Note**
>
> 사전 빌드된 Braket 이미지에 추가 소프트웨어만 추가하는 경우에는 이 섹션을 건너뛰어도 됩니다.

하이브리드 작업의 일부로 Python이 아닌 코드를 실행하려면 컨테이너 진입점을 정의하는 Python 스크립트를 수정해야 합니다. Amazon Braket `braket_container.py` Github에 있는 파이썬 스크립트를 예로 들어 보겠습니다. Braket에서 미리 빌드한 이미지가 알고리즘 스크립트를 시작하고 적절한 환경 변수를 설정하는 데 사용하는 스크립트입니다. 컨테이너 진입점 스크립트 자체는 Python으로 되어 있어야 하지만 Python이 아닌 스크립트를 시작할 수 있습니다. 사전 빌드된 예제에서 Python 알고리즘 스크립트가 Python 하위 프로세스로 시작되거나 완전히 새로운 프로세스로 실행되는 것을 볼 수 있습니다. 이 로직을 수정하면 엔트리 포인트 스크립트에서 Python이 아닌 알고리즘 스크립트를 실행하도록 할 수 있습니다. 예를 들어, 파일 확장자 끝에 따라 Rust 프로세스를 시작하도록 `thekick_off_customer_script()`함수를 수정할 수 있습니다.

완전히 새로 작성할 수도 `braket_container.py` 있습니다. Amazon S3의 입력 데이터, 소스 아카이브 및 기타 필요한 파일을 컨테이너로 복사하고 적절한 환경 변수를 정의해야 합니다.

필요한 소프트웨어를 모두 설치하고 컨테이너 스크립트를 **Dockerfile** 포함하는 A.

> **Note**
>
> 미리 빌드된 Braket 이미지를 Docker 기본 이미지로 사용하는 경우 컨테이너 스크립트가 이미 있습니다.

이전 단계에서 수정된 컨테이너 스크립트를 만든 경우 이를 컨테이너에 복사하고 환경 변수를 SAGEMAKER_PROGRAM 정의하거나 새 컨테이너 진입점 스크립트의 이름을 정의해야 합니다. `braket_container.py`

다음은 GPU 가속 작업 인스턴스에서 Julia를 사용할 수 `Dockerfile` 있는 예제입니다.

```
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
```

이 예제는 에서 제공하는 스크립트를 다운로드하고 AWS 실행하여 모든 관련 오픈 소스 라이선스를 준수하는지 확인합니다. 예를 들어, a에 의해 관리되는 설치된 코드의 출처를 적절하게 지정하는 경우를 들 수 있습니다. MIT license

비공개 코드 (예: 비공개 GitHub 또는 GitLab 리포지토리에서 호스팅되는 코드) 를 포함해야 하는 경우 이미지에 액세스하기 위해 SSH 키를 이미지에 내장하지 마십시오. Docker 대신 Docker Compose 빌드할 때 사용하여 SSH가 구축된 호스트 시스템에서 SSH에 액세스할 수 있도록 Docker 하세요. 자세

한 내용은 Docker에서 SSH 키를 안전하게 사용하여 비공개 Github 리포지토리에 액세스하기 가이드를 참조하세요.

이미지 빌드 및 업로드 Docker

이제 적절하게 Dockerfile 정의되었으므로 프라이빗 Amazon ECR 리포지토리가 아직 없는 경우 단계를 따라 프라이빗 Amazon ECR 리포지토리를 생성할 준비가 된 것입니다. 또한 컨테이너 이미지를 빌드하고 태그를 지정하고 리포지토리에 업로드할 수 있습니다.

이제 이미지를 빌드하고 태그를 지정하고 푸시할 준비가 되었습니다. 옵션에 대한 전체 docker build설명과 몇 가지 예는 Docker 빌드 설명서를 참조하십시오.

위에서 정의한 샘플 파일의 경우 다음을 실행할 수 있습니다.

```
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
```

적절한 Amazon ECR 권한 할당

Braket Hybrid Jobs Docker이미지는 프라이빗 Amazon ECR 리포지토리에서 호스팅되어야 합니다. 기본적으로 비공개 Amazon ECR 리포지토리는 공동 작업자 Braket Hybrid Jobs IAM role 또는 학생과 같이 이미지를 사용하려는 다른 사용자에게 읽기 권한을 제공하지 않습니다. 적절한 권한을 부여하려면 리포지토리 정책을 설정해야 합니다. 일반적으로 이미지에 액세스하려는 특정 사용자와 IAM 역할에게만 권한을 부여하십시오. 단, 권한이 있는 사람이 이미지를 image URI 가져오도록 허용하지 마십시오.

자체 컨테이너에서 Braket 하이브리드 작업 실행

자체 컨테이너로 하이브리드 작업을 생성하려면 image_uri 지정된 인수를 AwsQuantumJob.create() 사용하여 호출하십시오. 온디맨드 시뮬레이터인 QPU를 사용하거나 Braket Hybrid Jobs에서 사용할 수 있는 클래식 프로세서에서 로컬로 코드를 실행할 수 있습니다. 실제 QPU에서 실행하기 전에 SV1, DM1 또는 TN1과 같은 시뮬레이터에서 코드를 테스트해 보는 것이 좋습니다.

기존 프로세서에서 코드를 실행하려면 를 업데이트하여 사용하는 instanceType 와 instanceCount 를 지정하십시오. InstanceConfig 참고로 instance_count > 1을 지정하는 경

우 코드가 여러 호스트에서 실행될 수 있는지 확인해야 합니다. 선택할 수 있는 인스턴스 수의 상한은 5개입니다. 예:

```
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**
>
> 기기 ARN을 사용하여 하이브리드 작업 메타데이터로 사용한 시뮬레이터를 추적할 수 있습니다. 허용되는 값은 형식을 `device = "local:<provider>/<simulator_name>"` 따라야 합니다. `<provider>`및 는 문자, 숫자,, 등으로만 `<simulator_name>` 구성되어야 한다는 점을 기억하십시오`.. _` - 문자열은 256자로 제한됩니다.
> BYOC를 사용할 계획이고 Braket SDK를 사용하여 양자 작업을 생성하지 않을 경우 요청의 파라미터에 환경 변수 `AMZN_BRAKET_JOB_TOKEN` 값을 전달해야 합니다. `jobToken` `CreateQuantumTask` 그렇지 않으면 양자 작업에 우선 순위가 부여되지 않고 일반 독립형 양자 작업으로 요금이 청구됩니다.

에서 기본 버킷을 구성하십시오. **AwsSession**

자체 버킷을 제공하면 `AwsSession` 유연성을 높일 수 있습니다 (예: 기본 버킷 위치). 기본적으로 의 기본 버킷 위치는 `f"amazon-braket-{id}-{region}"` 입니다. `AwsSession` 하지만 생성 시 이 기본값을 재정의할 수 있습니다. `AwsSession` 사용자는 다음 코드 예제와 `aws_session` 같이 매개 변수 이름을 `AwsQuantumJob.create` 사용하여 `AwsSession` 개체를 선택적으로 전달할 수 있습니다.

```
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
)
```

를 사용하여 하이브리드 작업과 직접 상호 작용할 수 있습니다. API

를 사용하여 Amazon Braket 하이브리드 작업에 직접 액세스하고 이와 상호 작용할 수 있습니다. API 하지만 를 직접 사용할 때는 기본값 및 편리한 방법을 사용할 수 없습니다. API

> **Note**
>
> Amazon Braket Python SDK를 사용하여 Amazon Braket 하이브리드 작업과 상호 작용하는 것이 좋습니다. 하이브리드 작업을 성공적으로 실행하는 데 도움이 되는 편리한 기본값 및 보호 기능을 제공합니다.

이 항목에서는 사용의 기본 사항을 다룹니다. API API를 사용하기로 선택한 경우 이 접근 방식이 더 복잡할 수 있다는 점을 염두에 두고 하이브리드 작업을 실행하려면 몇 번의 반복을 거쳐야 합니다.

API를 사용하려면 계정에 `AmazonBraketFullAccess` 관리형 정책과 관련된 역할이 있어야 합니다.

> **Note**
>
> `AmazonBraketFullAccess`관리형 정책으로 역할을 얻는 방법에 대한 자세한 내용은 Amazon Braket 활성화 페이지를 참조하십시오.

또한 실행 역할도 필요합니다. 이 역할은 서비스에 전달됩니다. Amazon Braket 콘솔을 사용하여 역할을 생성할 수 있습니다. 권한 및 설정 페이지의 실행 역할 탭을 사용하여 하이브리드 작업에 대한 기본 역할을 생성할 수 있습니다.

하이브리드 작업에 필요한 모든 매개 변수를 지정해야 합니다. `CreateJob` API Python을 사용하려면 알고리즘 스크립트 파일을 tar 번들 (예: input.tar.gz 파일) 로 압축하고 다음 스크립트를 실행합니다. 하이브리드 작업이 시작되는 경로, 파일, 메서드를 지정하는 계정 정보 및 진입점과 일치하도록 각진 괄호 (<>) 안의 코드 부분을 업데이트하십시오.

```
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
    },
)
```

하이브리드 작업을 생성한 후에는 GetJob API 또는 콘솔을 통해 하이브리드 작업 세부 정보에 액세스할 수 있습니다. 이전 예제와 같이 createJob 코드를 실행한 Python 세션에서 하이브리드 작업 세부 정보를 가져오려면 다음 Python 명령을 사용합니다.

```
getJob = client.get_job(jobArn=job["jobArn"])
```

하이브리드 작업을 취소하려면 Amazon Resource Name of the job ('JobArn') 을 CancelJob API 사용하여 를 호출하십시오.

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

checkpointConfig파라미터 createJob API 사용의 일부로 체크포인트를 지정할 수 있습니다.

```
    checkpointConfig = {
        "localPath" : "/opt/omega/checkpoints",
        "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
    },
```

> **Note**
>
> 의 localPath는/opt/ml, /opt/braket/tmp, 또는 예약된 경로로 시작할 checkpointConfig 수 없습니다. /usr/local/nvidia

오류 완화

양자 오류 완화는 양자 컴퓨터의 오류 영향을 줄이기 위한 일련의 기술입니다.

양자 장치는 수행되는 계산 품질을 저하시키는 환경 잡음의 영향을 받기 쉽습니다. 내결함성 양자 컴퓨팅은 이 문제에 대한 해결책을 제시하지만, 현재의 양자 장치는 큐비트 수와 상대적으로 높은 오류율로 인해 제한됩니다. 단기적으로 이 문제를 해결하기 위해 연구자들은 잡음이 많은 양자 계산의 정확도를 개선하는 방법을 연구하고 있습니다. 양자 오류 완화로 알려진 이 접근 방식에는 잡음이 있는 측정 데이터에서 다양한 기술을 사용하여 최상의 신호를 추출하는 것이 포함됩니다.

오류 완화: IonQ Aria

오류 완화에는 여러 물리적 회로를 실행하고 측정치를 결합하여 결과를 개선하는 것이 포함됩니다. 이 IonQ Aria 장치에는 디바이어싱이라는 오류 완화 방법이 있습니다.

디바이어싱은 회로를 다양한 큐비트 순열 또는 다양한 게이트 분해에 작용하는 여러 변형으로 매핑합니다. 이를 통해 측정 결과를 편향시킬 수 있는 다양한 회로 구현을 사용함으로써 게이트 과회전 또는 단일 결함 큐비트와 같은 시스템 오류의 영향을 줄일 수 있습니다. 이로 인해 여러 큐비트와 게이트를 교정하는 데 드는 추가 오버헤드가 발생합니다.

편향 제거에 대한 자세한 내용은 대칭화를 통한 양자 컴퓨터 성능 향상을 참조하십시오.

> **Note**
>
> 디바이어싱을 사용하려면 최소 2500장의 사진이 필요합니다.

다음 코드를 사용하여 IoQ Aria 디바이스에서 디바이어싱과 함께 퀀텀 작업을 실행할 수 있습니다.

```
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
```

양자 작업이 완료되면 양자 작업의 측정 확률과 결과 유형을 확인할 수 있습니다. 모든 변이의 측정 확률과 개수가 단일 분포로 집계됩니다. 기대값과 같이 회로에 지정된 모든 결과 유형은 총 측정 카운트를 사용하여 계산됩니다.

선명화

샤프닝이라는 다른 후처리 전략을 사용하여 계산된 측정 확률에 액세스할 수도 있습니다. 선명하게 하기 기능을 사용하면 각 변형의 결과를 비교한 후 일치하지 않는 사진을 삭제하여 여러 변형에서 가장 가능성이 높은 측정 결과를 얻을 수 있습니다. 자세한 내용은 대칭화를 통한 양자 컴퓨터 성능 향상을 참조하십시오.

중요한 점은 샤프닝은 출력 분포의 형태가 희박하고 확률이 0인 상태가 많으며 희박한 것으로 가정한다는 것입니다. 이 가정이 유효하지 않을 경우 확률 분포가 왜곡될 수 있습니다.

Braket Python SDK의 `additional_metadata` 필드에서 예리한 분포를 통해 확률에 액세스할 수 있습니다. `GateModelTaskResult` 참고로 샤프닝은 측정 카운트를 반환하지 않고 대신 다시 정규화된 확률 분포를 반환합니다. 다음 코드 스니펫은 샤프닝 후 분포에 액세스하는 방법을 보여줍니다.

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

브라켓 다이렉트

Braket Direct를 사용하면 원하는 다양한 양자 장치에 대한 전용 액세스를 예약하고, 양자 컴퓨팅 전문가와 연결하여 워크로드에 대한 지침을 받고, 가용성이 제한된 새로운 양자 장치와 같은 차세대 기능을 조기에 이용할 수 있습니다.

이 섹션:

예약

예약을 통해 원하는 퀀텀 디바이스를 독점적으로 이용할 수 있습니다. 편한 시간에 예약을 예약할 수 있으므로 워크로드 실행이 시작되고 종료되는 시점을 정확히 알 수 있습니다. 예약은 1시간 단위로 가능하며 최대 48시간 전에 추가 비용 없이 취소할 수 있습니다. 예정된 예약을 위해 퀀텀 태스크 및 하이브리드 작업을 미리 대기열에 넣거나 예약 중에 워크로드를 제출하도록 선택할 수 있습니다.

전용 장치 액세스 비용은 QPU (Quantum Processing Unit) 에서 실행하는 양자 작업 및 하이브리드 작업의 수와 상관없이 예약 기간을 기준으로 합니다.

예약할 수 있는 양자 컴퓨터는 다음과 같습니다.

- 아이온큐의 아리아
- IQM의 가넷
- QuEra아퀼라
- 리게티의 아스펜-M-3

예약 사용 시기

예약 시 전용 디바이스 액세스를 활용하면 퀀텀 워크로드의 실행 시작 및 종료 시점을 정확히 알 수 있는 편리함과 예측 가능성을 확보할 수 있습니다. 온디맨드 방식으로 작업과 하이브리드 작업을 제출하는 것과 비교하면 다른 고객 작업을 기다릴 필요가 없습니다. 예약하는 동안에는 디바이스에 독점적으로 액세스할 수 있으므로 예약 기간 내내 디바이스에서 워크로드만 실행됩니다.

연구의 설계 및 프로토타이핑 단계에서는 알고리즘을 빠르고 비용 효율적으로 반복할 수 있도록 온디맨드 액세스를 사용하는 것이 좋습니다. 최종 실험 결과를 산출할 준비가 되면 프로젝트 또는 출판 마감일을 맞출 수 있도록 편한 시간에 기기를 예약하는 것을 고려해 보세요. 또한 양자 컴퓨터에서 라이브 데모나 워크샵을 진행하는 경우와 같이 특정 시간에 작업을 실행하려는 경우에도 예약을 사용하는 것이 좋습니다.

이 섹션:

- 예약 생성하기
- 예약을 통해 워크로드를 처리하세요.
- 기존 예약 취소 또는 일정 변경

예약 생성하기

예약을 생성하려면 다음 단계에 따라 Braket 팀에 문의하세요.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 창에서 Braket Direct를 선택한 다음 예약 섹션에서 디바이스 예약을 선택합니다.
3. 예약하려는 기기를 선택합니다.
4. 이름 및 이메일을 포함한 연락처 정보를 입력합니다. 정기적으로 확인하는 유효한 이메일 주소를 제공해야 합니다.
5. 워크로드에 대해 알려주세요에서 예약을 사용하여 실행할 워크로드에 대한 세부 정보를 입력하세요. 원하는 예약 기간, 관련 제약 조건, 원하는 일정 등을 예로 들 수 있습니다.
6. 예약이 확정된 후 예약 준비 세션을 위해 Braket 전문가와 연결하려면 선택적으로 준비 세션에 관심이 있습니다를 선택하십시오.

다음 단계에 따라 당사에 연락하여 예약을 생성할 수도 있습니다.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 창에서 디바이스를 선택하고 예약하려는 디바이스를 선택합니다.
3. 요약 섹션에서 기기 예약을 선택합니다.
4. 이전 절차의 4~6단계를 따르십시오.

양식을 제출하면 Braket 팀으로부터 예약을 생성하는 다음 단계가 포함된 이메일을 받게 됩니다. 예약이 확인되면 이메일을 통해 예약 ARN을 받게 됩니다.

> **Note**
>
> 예약 ARN을 받은 후에만 예약이 확정됩니다.

예약은 최소 1시간 단위로 가능하며, 특정 디바이스에는 추가 예약 길이 제한 (최소 및 최대 예약 기간 포함) 이 있을 수 있습니다. Braket 팀은 예약을 확인하기 전에 모든 관련 정보를 귀하와 공유합니다.

예약 준비 세션에 관심이 있는 경우 Braket 팀이 이메일을 통해 연락하여 Braket 전문가와의 30분 세션을 주선합니다.

예약을 통해 워크로드를 처리하세요.

예약 중에는 워크로드만 디바이스에서 실행됩니다. 디바이스 예약 중에 실행할 퀀텀 태스크 및 하이브리드 작업을 지정하려면 유효한 예약 ARN을 사용해야 합니다.

> **Note**
>
> 예약은 AWS 계정 및 기기별로 다릅니다. 예약을 생성한 AWS 계정만 예약 ARN을 사용할 수 있습니다. 또한 예약 ARN은 선택한 시작 및 종료 시간에 예약된 디바이스에서만 유효합니다.

예약된 시간을 최대한 활용하려면 예약 전에 작업과 작업을 대기열에 넣도록 선택할 수 있습니다. 이러한 워크로드는 예약이 시작될 때까지 QUEUED 상태를 유지합니다. 예약이 시작되면 대기 중인 모든 워크로드가 제출된 순서대로 실행됩니다. Job 태스크는 독립형 퀀텀 태스크보다 우선 순위가 정해집니다.

> **Note**
>
> 예약 중에는 워크로드만 실행되므로 예약 ARN으로 제출된 작업 및 작업에 대해서는 대기열이 표시되지 않습니다.

예약을 위한 퀀텀 태스크를 생성하기 위한 코드 예제:

1. OpenQASM 형식으로 GHZ 상태를 준비하기 위한 회로를 정의합니다.

```
// 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. 회로와 예약 ARN을 사용하여 양자 작업을 생성하십시오.

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

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

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

program = Program(source=ghz_qasm_string)

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

# You can also specify a particular Amazon S3 bucket location
# and the desired number of shots, when running the program.
```

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

Braket Direct 예약을 위한 하이브리드 작업을 생성하기 위한 코드 예제:

1. 알고리즘 스크립트를 정의하세요.

```
//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. 알고리즘 스크립트와 예약 ARN을 사용하여 하이브리드 작업을 생성합니다.

```
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. 원격 데코레이터를 사용하여 하이브리드 작업을 생성합니다.

```
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
```

예약이 종료되면 어떻게 되나요?

예약이 종료된 후에는 더 이상 해당 기기에 대한 전용 액세스 권한이 없습니다. 이 예약과 함께 대기 중인 나머지 워크로드는 자동으로 취소됩니다.

> **Note**
>
> 예약이 종료될 때 RUNNING 상태에 있었던 모든 작업은 취소됩니다. 체크포인트를 사용하여 편리한 시간에 작업을 저장하고 다시 시작하는 것이 좋습니다.

예약 시작 후 및 예약 종료 전과 같이 진행 중인 예약은 연장할 수 없습니다. 각 예약은 독립형 전용 장치 액세스를 나타내므로 연장할 수 없습니다. 예를 들어 두 개의 back-to-back 예약은 분리된 것으로 간주되며 첫 번째 예약에서 보류 중인 작업은 자동으로 취소됩니다. 두 번째 예약에서는 재개되지 않습니다.

> **Note**
>
> 예약은 AWS 계정 전용 기기 액세스를 의미합니다. 기기가 유휴 상태로 유지되더라도 다른 고객은 사용할 수 없습니다. 따라서 사용 시간에 상관없이 예약된 시간만큼 요금이 부과됩니다.

기존 예약 취소 또는 일정 변경

예정된 예약 시작 시간보다 최소 48시간 전에 예약을 취소할 수 있습니다. 취소하려면 취소 요청과 함께 받은 예약 확인 이메일에 회신하십시오.

일정을 변경하려면 기존 예약을 취소한 다음 새 예약을 해야 합니다.

전문가 조언

Braket 관리 콘솔에서 직접 양자 컴퓨팅 전문가와 연결하여 워크로드에 대한 추가 지침을 얻으십시오.

Braket Direct를 통해 전문가 조언 옵션을 탐색하려면 Braket 콘솔을 열고 왼쪽 창에서 Braket Direct를 선택한 다음 전문가 조언 섹션으로 이동하십시오. 다음과 같은 전문가 조언 옵션을 사용할 수 있습니다.

- 브라켓 근무 시간: 브라켓 근무 시간은 1:1 세션으로, 선착순으로 매월 진행됩니다. 이용 가능한 각 근무 시간은 30분이며 무료입니다. Braket 전문가와 상담하면 use-case-to-device 적합성을 탐색하고, Braket을 알고리즘에 가장 잘 활용할 수 있는 옵션을 식별하고, Amazon Braket Hybrid Jobs, Braket Pulse 또는 아날로그 해밀턴식 시뮬레이션과 같은 특정 브라켓 기능을 사용하는 방법에 대한 권장 사항을 받아 아이디어 구상에서 실행까지 더 빠르게 진행할 수 있습니다.
  - Braket 근무 시간에 등록하려면 등록을 선택하고 연락처 정보, 워크로드 세부 정보 및 원하는 토론 주제를 작성하십시오.
  - 이메일을 통해 다음 예약 가능 시간대에 대한 캘린더 초대장을 받게 됩니다.

  > **Note**
  >
  > 긴급한 문제나 빠른 문제 해결 질문이 있는 경우 AWS Support에 문의하는 것이 좋습니다. 긴급하지 않은 질문의 경우 AWS re:Post 포럼 또는 Quantum Computing Stack Exchange를 사용하여 이전에 답변한 질문을 찾아보고 새로운 질문을 할 수도 있습니다.

- 양자 하드웨어 제공업체 제품: IonQ, Oxford Quantum Circuits QuEra, 및 Rigetti는 각각 를 통해 전문 서비스를 제공합니다. AWS Marketplace

- 해당 제품을 살펴보려면 Connect를 선택하고 해당 목록을 찾아보십시오.
- 에서 제공하는 전문 서비스에 대해 자세히 AWS Marketplace알아보려면 전문 서비스 제품을 참조하십시오.

- AmazonQuantum Solutions Lab (QSL): QSL은 양자 컴퓨팅을 효과적으로 탐색하고 이 기술의 현재 성능을 평가하는 데 도움을 줄 수 있는 양자 컴퓨팅 전문가로 구성된 공동 연구 및 전문 서비스 팀입니다.
  - QSL에 문의하려면 Connect를 선택하고 연락처 정보와 사용 사례 세부 정보를 입력합니다.
  - QSL 팀에서 다음 단계를 안내하는 이메일을 통해 연락을 드릴 것입니다.

실험적 기능

연구 워크로드를 향상시키려면 새로운 혁신 기능에 빠르게 액세스하는 것이 중요합니다. Braket Direct를 사용하면 가용성이 제한된 새로운 양자 장치와 같은 사용 가능한 실험 기능에 대한 액세스를 Braket 콘솔에서 직접 요청할 수 있습니다.

일부 실험 기능은 표준 장치 사양을 벗어나서 작동하므로 사용 사례에 맞는 실습 지침이 필요합니다. 워크로드가 성공할 수 있도록 설정하려면 요청 시 Braket Direct를 통해 액세스할 수 있습니다.

예약만 하면 IonQ 포르테를 이용할 수 있습니다.

브라켓 다이렉트를 이용하면 예약만 하면 아이온큐 포르테 QPU를 이용할 수 있습니다. 한정된 수량으로 인해 이 장치는 브라켓 다이렉트를 통해서만 구입할 수 있습니다.

IonQ Forte에 대해 자세히 알아보고 액세스를 요청하려면 다음 단계를 따르십시오.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 메뉴에서 브라켓 다이렉트를 선택한 다음 실험용 기능에서 IonQ Forte로 이동합니다. 기기 보기를 선택합니다.
3. Forte 장치 세부 정보 페이지의 요약에서 장치 예약을 선택합니다.
4. 이름 및 이메일을 포함한 연락처 정보를 입력합니다. 정기적으로 확인하는 유효한 이메일 주소를 제공하십시오.
5. 워크로드에 대해 알려주세요에서 원하는 예약 기간, 관련 제약 조건, 원하는 일정 등 예약을 사용하여 실행할 워크로드에 대한 세부 정보를 제공하십시오.
6. (선택 사항) 예약이 확정된 후 예약 준비 세션을 위해 Braket 전문가와 연결하려면 준비 세션에 관심이 있습니다를 선택하십시오.

양식이 제출되면 Braket 팀에서 다음 단계를 안내해 드릴 것입니다.

> **Note**
>
> 기기 가용성이 제한되어 있기 때문에 Forte에 대한 액세스가 제한됩니다. 자세한 내용은 당사로 문의하세요.

Aquila의 로컬 디튜닝 이용 QuEra

Braket Direct를 사용하면 QPU에서 프로그래밍할 때 로컬 디튜닝을 제어하기 위한 액세스를 요청할 수 있습니다. QuEra Aquila 이 기능을 사용하면 드라이빙 필드가 각 특정 큐비트에 미치는 영향을 조정할 수 있습니다.

자세히 알아보고 이 기능에 대한 액세스를 요청하려면 다음 단계를 따르십시오.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 메뉴에서 Braket Direct를 선택한 다음 실험 기능에서 QuEra Aquila - 로컬 디튜닝으로 이동합니다. [액세스 권한 가져오기] 를 선택합니다.
3. 이름과 이메일을 포함한 연락처 정보를 입력합니다. 정기적으로 확인하는 유효한 이메일 주소를 제공하십시오.
4. 워크로드에 대해 알려주세요에서 워크로드에 대한 세부 정보와 이 기능을 사용할 계획을 알려주세요.

Aquila에서 키가 큰 지오메트리에 액세스할 수 있습니다. QuEra

Braket Direct를 사용하면 QPU에서 프로그래밍할 때 확장된 지오메트리에 대한 액세스를 요청할 수 있습니다. QuEra Aquila 이 기능을 사용하면 표준 장치 기능 이상으로 실험하고 격자 높이가 증가된 지오메트리를 지정할 수 있습니다.

자세히 알아보고 이 기능에 대한 액세스를 요청하려면 다음 단계를 따르세요.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 메뉴에서 Braket Direct를 선택한 다음 실험 기능에서 QuEra Aquila - 키가 큰 기하학으로 이동합니다. '액세스하기'를 선택합니다.
3. 이름 및 이메일을 포함한 연락처 정보를 입력합니다. 정기적으로 확인하는 유효한 이메일 주소를 제공하십시오.

4. 워크로드에 대해 알려주세요에서 워크로드에 대한 세부 정보와 이 기능을 사용할 계획을 알려주세요.

Aquila의 좁은 지오메트리에 접근할 수 있습니다. QuEra

Braket Direct를 사용하면 QPU에서 프로그래밍할 때 확장된 지오메트리에 대한 액세스를 요청할 수 있습니다. QuEra Aquila 이 기능을 사용하면 표준 장치 기능 이상으로 실험하고 격자 행을 더 좁은 수직 간격으로 정렬할 수 있습니다.

자세히 알아보고 이 기능에 대한 액세스를 요청하려면 다음 단계를 따르세요.

1. 아마존 브라켓 콘솔을 엽니다.
2. 왼쪽 메뉴에서 Braket Direct를 선택한 다음 실험 기능에서 QuEra Aquila - 키가 큰 기하학으로 이동합니다. '액세스하기'를 선택합니다.
3. 이름 및 이메일을 포함한 연락처 정보를 입력합니다. 정기적으로 확인하는 유효한 이메일 주소를 제공하십시오.
4. 워크로드에 대해 알려주세요에서 워크로드에 대한 세부 정보와 이 기능을 사용할 계획을 알려주세요.

로깅 및 모니터링

양자 작업을 제출한 후에는 Amazon Braket SDK 및 콘솔을 통해 상태를 추적할 수 있습니다. 양자 작업이 완료되면 Braket은 지정된 Amazon S3 위치에 결과를 저장합니다. 대기열 길이에 따라 완료하는 데 시간이 걸릴 수 있으며, 특히 QPU 디바이스의 경우 시간이 걸릴 수 있습니다. 상태 유형에는 다음이 포함됩니다.

- `CREATED`— 아마존 브라켓이 당신의 퀀텀 태스크를 받았습니다.
- `QUEUED`— Amazon Braket은 양자 작업을 처리했으며 이제 디바이스에서 실행되기를 기다리고 있습니다.
- `RUNNING`— 양자 작업은 QPU 또는 온디맨드 시뮬레이터에서 실행됩니다.
- `COMPLETED`— QPU 또는 온디맨드 시뮬레이터에서 양자 작업 실행이 완료되었습니다.
- `FAILED`— 양자 작업을 실행하려고 했지만 실패했습니다. 양자 과제가 실패한 이유에 따라 양자 과제를 다시 제출해 보세요.
- `CANCELLED`— 양자 작업을 취소했습니다. 퀀텀 태스크가 실행되지 않았어요.

이 섹션:

- Amazon Braket SDK에서 양자 작업 추적하기
- Amazon Braket 콘솔을 통한 양자 작업 모니터링
- Amazon Braket 리소스에 태그 지정
- 아마존 브라켓 (Amazon Braket) 관련 이벤트 및 자동 액션 EventBridge
- 아마존과 함께하는 아마존 브라켓 모니터링 CloudWatch
- Amazon Braket API를 사용한 로깅 CloudTrail
- 를 사용하여 Amazon Braket 노트북 인스턴스를 생성합니다. AWS CloudFormation
- 고급 로깅

Amazon Braket SDK에서 양자 작업 추적하기

이 명령은 고유한 양자 작업 ID를 사용하여 양자 작업을 `device.run(…)` 정의합니다. 다음 예와 `task.state()` 같이 상태를 쿼리하고 추적할 수 있습니다.

참고: `task = device.run()` 비동기 작업이므로 시스템이 백그라운드에서 양자 작업을 처리하는 동안에도 계속 작업할 수 있습니다.

결과 검색

전화를 `task.result()` 걸면 SDK는 Amazon Braket을 폴링하여 양자 작업이 완료되었는지 확인하기 시작합니다. SDK는 사용자가 정의한 폴링 매개변수를 사용합니다. `.run()` 양자 작업이 완료되면 SDK는 S3 버킷에서 결과를 검색하고 이를 객체로 반환합니다. `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
```

퀀텀 태스크 취소

양자 작업을 취소하려면 다음 예제와 같이 `cancel()` 메서드를 호출하십시오.

```
# cancel quantum task
task.cancel()
status = task.state()
```

```
print('Status of task:', status)
```

```
Status of task: CANCELLING
```

메타데이터를 확인하세요.

다음 예와 같이 완료된 양자 작업의 메타데이터를 확인할 수 있습니다.

```
# 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
```

양자 작업 또는 결과 검색

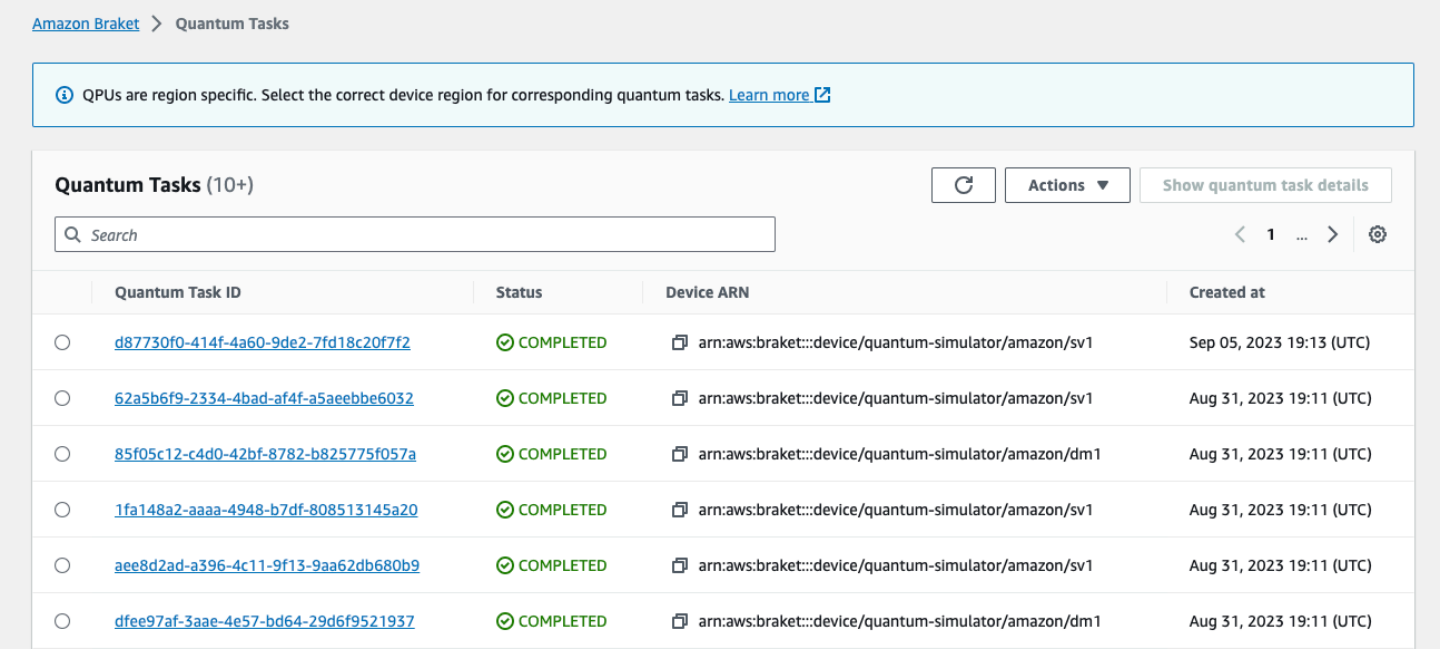
양자 작업을 제출한 후 커널이 종료되거나 노트북이나 컴퓨터를 닫은 경우 고유한 ARN (양자 작업 ID)으로 `task` 객체를 재구성할 수 있습니다. 그런 다음 `task.result()` 호출하여 저장된 S3 버킷에서 결과를 가져올 수 있습니다.

```
from braket.aws import AwsSession, AwsQuantumTask
```

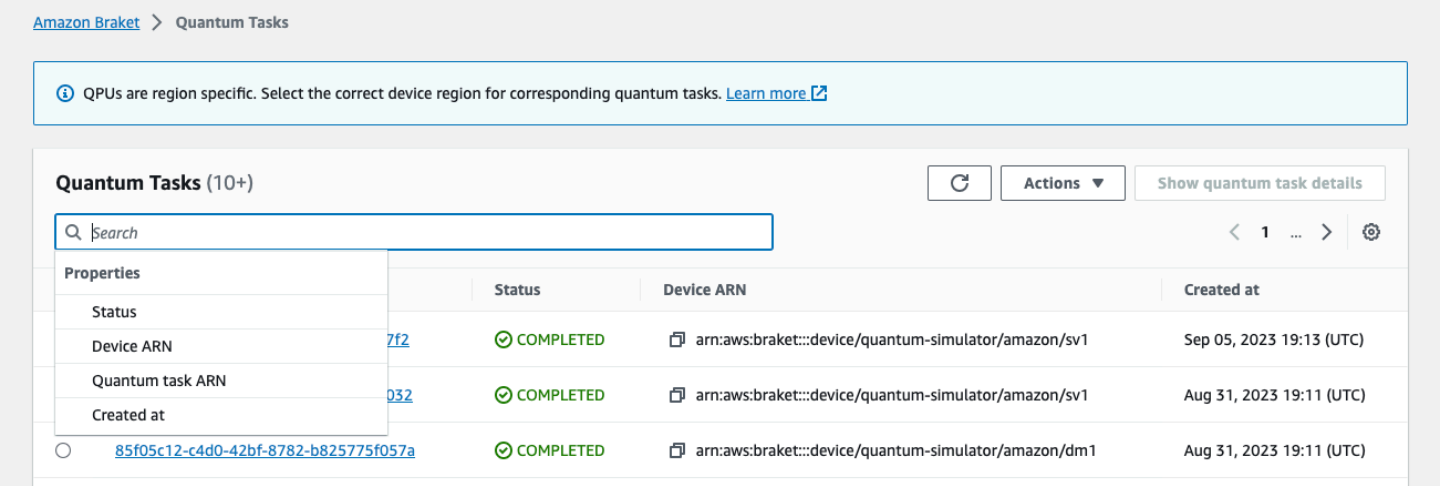
```
# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
result = task_load.result()
```

Amazon Braket 콘솔을 통한 양자 작업 모니터링

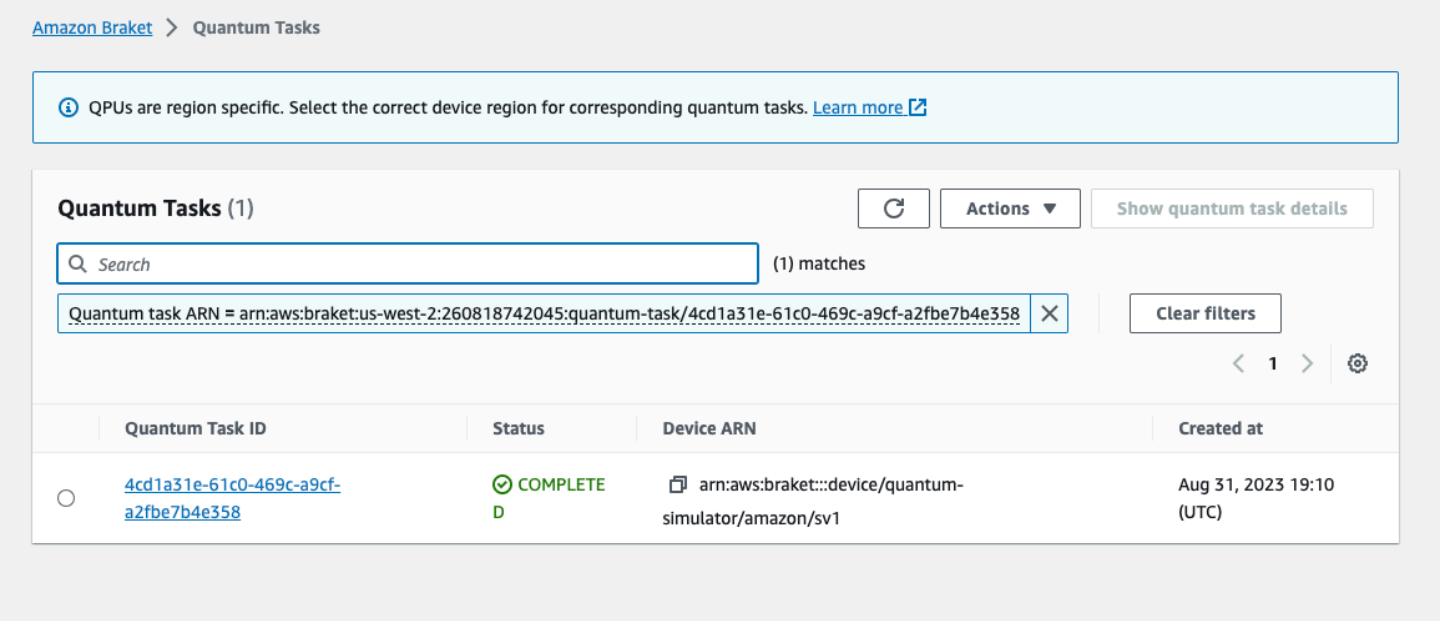
AmazonBraket은 Amazon Braket 콘솔을 통해 양자 작업을 모니터링하는 편리한 방법을 제공합니다. 제출된 모든 양자 작업은 다음 그림과 같이 Quantum Tasks 필드에 나열됩니다. 이 서비스는 지역별로 다르므로 특정 지역에서 생성된 양자 작업만 볼 수 있습니다. AWS 리전



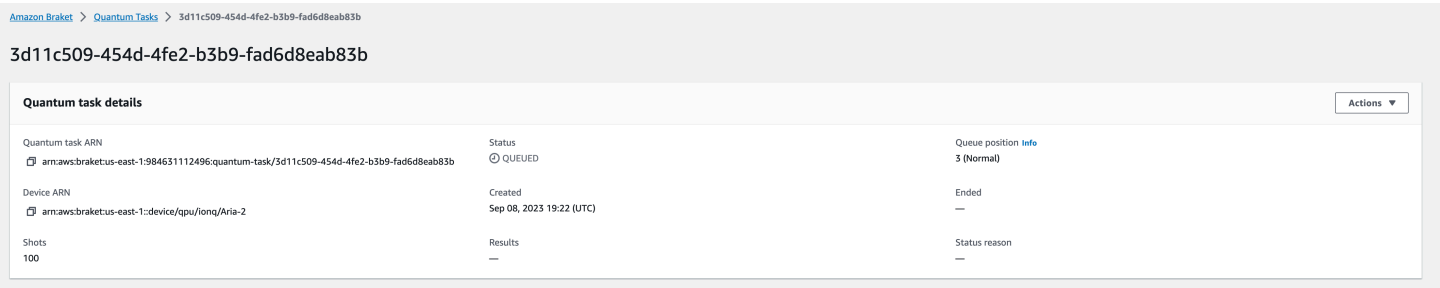
탐색 표시줄을 통해 특정 양자 작업을 검색할 수 있습니다. 검색은 퀀텀 태스크 ARN (ID), 상태, 장치 및 생성 시간을 기반으로 할 수 있습니다. 다음 예와 같이 내비게이션 바를 선택하면 옵션이 자동으로 나타납니다.



다음 이미지는 고유한 양자 작업 ID를 기반으로 양자 작업을 검색하는 예를 보여줍니다. 이 ID는 호출을 `task.id` 통해 얻을 수 있습니다.



또한 아래 그림에서 볼 수 있듯이 양자 작업이 상태에 있는 동안 상태를 모니터링할 수 있습니다. QUEUED 양자 작업 ID를 클릭하면 세부 정보 페이지가 표시됩니다. 이 페이지에는 처리할 장치와 관련된 양자 작업의 동적 대기열 위치가 표시됩니다.



하이브리드 작업의 일부로 제출된 Quantum 작업은 대기열에 있을 때 우선 순위를 가집니다. 하이브리드 작업 외부에서 제출된 Quantum 작업은 일반적인 대기열 우선 순위를 갖습니다.

Braket SDK를 쿼리하려는 고객은 프로그래밍 방식으로 퀀텀 태스크 및 하이브리드 작업 대기열 위치를 확인할 수 있습니다. 자세한 내용은 내 작업이 언제 실행되나요? 페이지를 참조하십시오.

Amazon Braket 리소스에 태그 지정

태그는 사용자가 할당하거나 리소스에 할당하는 사용자 지정 속성 레이블입니다. AWS AWS 태그는 리소스에 대해 자세히 알려주는 메타데이터입니다. 각 태그는 키와 값으로 구성됩니다. 태그 키와 태그 값을 합해서 키-값 페어라고 합니다. 사용자가 할당하는 태그에 대해 키와 값을 정의합니다.

AmazonBraket 콘솔에서 양자 작업 또는 노트북으로 이동하여 관련 태그 목록을 볼 수 있습니다. 태그를 추가하거나, 태그를 제거하거나, 태그를 수정할 수 있습니다. 퀀텀 태스크 또는 노트북을 만들 때 태그를 지정한 다음 콘솔 AWS CLI, 또는 를 통해 관련 태그를 관리할 수 API 있습니다.

태그 사용

태그를 사용하면 리소스를 유용한 범주로 구성할 수 있습니다. 예를 들어, “부서” 태그를 할당하여 이 리소스를 소유하는 부서를 지정할 수 있습니다.

각 태그에는 다음 두 가지 부분이 있습니다.

- 태그 키 (예: 환경 또는 프로젝트) CostCenter 태그 키는 대소문자를 구별합니다.
- 태그 값이라고 하는 선택적 필드 (예: 111122223333 또는 프로덕션). 태그 값을 생략하는 것은 빈 문자열을 사용하는 것과 같습니다. 태그 키처럼 태그 값은 대/소문자를 구별합니다.

태그를 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- AWS 리소스를 식별하고 정리하세요. 많은 서비스가 태그 지정을 AWS 서비스 지원하므로 서로 다른 서비스의 리소스에 동일한 태그를 할당하여 리소스가 관련되어 있음을 나타낼 수 있습니다.
- AWS 비용을 추적하세요. AWS Billing and Cost Management 대시보드에서 이러한 태그를 활성화합니다. AWS 태그를 사용하여 비용을 분류하고 월별 비용 할당 보고서를 제공합니다. 자세한 내용은 AWS Billing and Cost Management 사용 설명서의 비용 할당 태그 사용을 참조하세요.
- AWS 리소스에 대한 액세스를 제어하세요. 자세한 내용은 태그를 사용한 액세스 제어를 참조하십시오.

AWS 및 태그에 대한 자세한 내용

- 이름 지정 및 사용 규칙을 비롯한 태깅에 대한 일반 정보는 일반 참조의 태그 지정 AWS 리소스를 참조하십시오.AWS
- 태깅 제한에 대한 자세한 내용은 일반 참조의 태그 이름 지정 제한 및 요구 사항을 참조하십시오.AWS
- 모범 사례 및 태그 지정 전략은 태그 지정 모범 사례 및 태그 지정 전략을 참조하십시오.AWS
- 태그 사용을 지원하는 서비스 목록은 Resource Groups Tagging API 참조를 참조하세요.

다음 섹션에서는 Braket의 태그에 대한 보다 구체적인 정보를 제공합니다. Amazon

Amazon Braket에서 지원되는 리소스

AmazonBraket의 다음 리소스 유형은 태그 지정을 지원합니다.

- `quantum-task` 리소스
- 리소스 이름: `AWS::Service::Braket`
- ARN 정규식: `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

참고: 노트북은 실제로 Amazon Amazon 리소스이지만 콘솔을 사용하여 노트북 리소스로 이동하여 Amazon Braket 콘솔에서 Braket 노트북에 태그를 적용하고 관리할 수 있습니다. SageMaker 자세한 내용은 설명서의 노트북 인스턴스 메타데이터를 참조하십시오. SageMaker

태그 제한

AmazonBraket 리소스의 태그에는 다음과 같은 기본 제한이 적용됩니다.

- 리소스에 할당할 수 있는 최대 태그 수: 50
- 최대 키 길이: 유니코드 128자
- 최대 값 길이: 유니코드 256자
- 키와 값에 유효한 문자: `a-z, A-Z, 0-9, space` 및 다음 문자: `_ . : / = + -` 및 `@`
- 키와 값은 대/소문자를 구분합니다.
- 키의 `aws` 접두사로 사용하지 마십시오. 이 접두사는 AWS 사용하도록 예약되어 있습니다.

아마존 브라켓의 태그 관리

태그를 리소스의 속성으로 설정합니다. AmazonBraket 콘솔, Braket 또는 를 통해 태그를 보고, 추가하고, 수정하고, 나열하고, 삭제할 수 있습니다. Amazon API AWS CLI자세한 내용은 아마존 브라켓 API 레퍼런스를 참조하십시오.

태그 추가

다음과 같은 경우에 태그가 가능한 리소스에 태그를 추가할 수 있습니다.

- 리소스를 생성할 때: 콘솔을 사용하거나 API에 `Create` 작업과 함께 `Tags` 파라미터를 포함시키십시오.AWS
- 리소스를 생성한 후: 콘솔을 사용하여 Quantum 태스크 또는 노트북 리소스로 이동하거나 AWS API에서 `TagResource` 작업을 호출하십시오.

리소스를 생성할 때 리소스에 태그를 추가하려면 지정된 유형의 리소스를 생성할 수 있는 권한도 필요합니다.

태그 보기

콘솔을 사용하여 작업 또는 노트북 리소스로 이동하거나 작업을 호출하여 Amazon Braket에서 태그가 가능한 모든 리소스의 태그를 볼 수 있습니다. AWS `ListTagsForResource` API

다음 AWS API 명령을 사용하여 리소스의 태그를 볼 수 있습니다.

- AWS API: `ListTagsForResource`

태그 편집

콘솔을 사용하여 Quantum 태스크 또는 노트북 리소스로 이동하여 태그를 편집하거나 다음 명령을 사용하여 태깅 가능한 리소스에 연결된 태그의 값을 수정할 수 있습니다. 이미 존재하는 태그 키를 지정하면 해당 키의 값을 덮어씁니다.

- AWS API: `TagResource`

태그 제거

제거할 키를 지정하거나, 콘솔을 사용하여 퀀텀 태스크 또는 노트북 리소스로 이동하거나, 작업을 호출할 때 리소스에서 태그를 제거할 수 있습니다. `UntagResource`

- AWS API: UntagResource

아마존 브라켓의 CLI 태깅 예제

AWS CLI를 사용하는 경우 다음은 QPU의 파라미터 SV1 설정으로 생성하는 양자 작업에 적용되는 태그를 생성하는 방법을 보여주는 예제 명령입니다. Rigetti 예시 명령어 끝에 태그가 지정되어 있는 것을 확인할 수 있습니다. 이 경우 Key에는 값이 state 제공되고 Value에는 값이 제공됩니다Washington.

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

Amazon Braket API로 태그 지정

- Amazon API Braket을 사용하여 리소스에 태그를 설정하는 경우 를 호출하십시오. TagResourceAPI

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

- 리소스에서 태그를 제거하려면 를 호출하십시오. UntagResourceAPI

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- 특정 리소스에 연결된 모든 태그를 나열하려면 를 호출하십시오 `ListTagsForResource`API.

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

아마존 브라켓 (Amazon Braket) 관련 이벤트 및 자동 액션 EventBridge

Amazon은 Amazon Braket 양자 작업에서 상태 변경 이벤트를 EventBridge 모니터링합니다. AmazonBraket의 이벤트는 거의 실시간으로 전송됩니다 EventBridge. 이벤트가 규칙과 일치할 때 수행할 자동화된 작업을 포함하여 관심 있는 이벤트를 나타내는 간단한 규칙을 작성할 수 있습니다. 트리거될 수 있는 자동 액션은 다음과 같습니다.

- 함수 호출 AWS Lambda
- 스테이트 머신 활성화 AWS Step Functions
- Amazon SNS 주제 알림

EventBridge 다음과 같은 Amazon Braket 상태 변경 이벤트를 모니터링합니다.

- 퀀텀 태스크의 상태가 바뀝니다.

Amazon브라켓은 양자 작업 상태 변경 이벤트의 전달을 보장합니다. 이러한 이벤트는 최소 한 번 전달되지만 순서가 맞지 않을 수 있습니다.

자세한 내용은 의 이벤트 및 이벤트 패턴을 참조하십시오 EventBridge.

이 섹션:

- 다음을 통해 양자 작업 상태를 모니터링할 수 있습니다. EventBridge
- 아마존 브라켓 이벤트 EventBridge 예시

다음을 통해 양자 작업 상태를 모니터링할 수 있습니다. EventBridge

를 사용하면 Braket이 Amazon Braket 양자 작업과 EventBridge 관련된 상태 변경 알림을 보낼 때 취할 조치를 정의하는 규칙을 만들 수 있습니다. 예를 들어, 양자 작업의 상태가 변경될 때마다 이메일 메시지를 보내는 규칙을 만들 수 있습니다.

1. 사용 권한이 있는 EventBridge 계정과 Amazon Braket을 사용하여 로그인하십시오. AWS
2. https://console.aws.amazon.com/events/ 에서 아마존 EventBridge 콘솔을 엽니다.
3. 다음 값을 사용하여 EventBridge 규칙을 생성합니다.
   - 규칙 유형(Rule type)에서 이벤트 패턴이 있는 규칙(Rule with an event pattern)을 생성합니다.
   - 이벤트 소스(Event source)에서 기타(Other)를 선택합니다.
   - 이벤트 패턴 섹션에서 사용자 지정 패턴(JSON 편집기)을 선택하고 다음 이벤트 패턴을 텍스트 영역에 붙여 넣습니다.

     ```
     {
       "source": [
         "aws.braket"
       ],
       "detail-type": [
         "Braket Task State Change"
       ]
     }
     ```

     AmazonBraket에서 모든 이벤트를 캡처하려면 다음 코드에 표시된 대로 `detail-type` 섹션을 제외하십시오.

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

   - 대상 유형에서 을 선택하고 AWS 서비스, 대상 선택에서 Amazon SNS 주제 또는 AWS Lambda 기능과 같은 대상을 선택합니다. 대상은 Amazon Braket으로부터 양자 작업 상태 변경 이벤트가 수신될 때 트리거됩니다.

     예를 들어, 이벤트 발생 시 Amazon Simple Notification Service (SNS) 주제를 사용하여 이메일 또는 텍스트 메시지를 보낼 수 있습니다. 이를 위해서는 먼저 Amazon SNS 콘솔을 사용하여 Amazon SNS 주제를 생성하십시오. 자세한 내용은 사용자 알림에 Amazon SNS 사용을 참조하세요.

규칙 생성에 대한 자세한 내용은 이벤트에 반응하는 Amazon EventBridge 규칙 생성을 참조하십시오.

아마존 브라켓 이벤트 EventBridge 예시

AmazonBraket Quantum 작업 상태 변경 이벤트의 필드에 대한 자세한 내용은 의 이벤트 및 이벤트 패턴을 참조하십시오. EventBridge

JSON '세부 정보' 필드에는 다음과 같은 속성이 표시됩니다.

- **quantumTaskArn**(str): 이 이벤트가 생성된 퀀텀 태스크입니다.
- **status**(선택 사항 [str]): 양자 과제가 전환된 상태입니다.
- **deviceArn**(str): 이 양자 태스크가 생성된 데 사용할 사용자가 지정한 장치입니다.
- shots(int): 사용자가 shots 요청한 건수입니다.
- **outputS3Bucket**(str): 사용자가 지정한 출력 버킷입니다.
- **outputS3Directory**(str): 사용자가 지정한 출력 키 접두사입니다.
- **createdAt**(str): 양자 태스크 생성 시간을 ISO-8601 문자열로 나타낸 것입니다.
- **endedAt**(선택 사항 [str]): 양자 작업이 최종 상태에 도달한 시간. 이 필드는 양자 작업이 최종 상태로 전환된 경우에만 표시됩니다.

다음 JSON 코드는 Amazon 브라켓 퀀텀 태스크 상태 변경 이벤트의 예를 보여줍니다.

```
{
    "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"
      }
}
```

아마존과 함께하는 아마존 브라켓 모니터링 CloudWatch

원시 데이터를 수집하여 읽기 쉬운 거의 실시간 지표로 처리하는 Amazon을 사용하여 Amazon CloudWatch Braket을 모니터링할 수 있습니다. Amazon CloudWatch 콘솔에서 최대 15개월 전에 생성된 기록 정보 또는 지난 2주 동안 업데이트된 지표를 확인하여 Amazon Braket의 실적을 더 잘 파악할 수 있습니다. 자세한 내용은 CloudWatch 지표 사용을 참조하십시오.

아마존 브라켓 지표 및 치수

지표는 의 기본 개념입니다. CloudWatch 지표는 게시되는 시간 순서대로 정렬된 데이터 요소 집합을 CloudWatch 나타냅니다. 모든 지표는 차원 집합으로 특징 지어집니다. 의 측정항목 측정기준에 대해 자세히 알아보려면 측정기준을 참조하십시오CloudWatch . CloudWatch

Amazon Braket은 아마존 브라켓과 관련된 다음과 같은 지표 데이터를 아마존 메트릭으로 전송합니다. CloudWatch

퀀텀 태스크 메트릭스

양자 작업이 있는 경우 메트릭을 사용할 수 있습니다. 콘솔의 AWS/Braket/By Device 아래에 표시됩니다. CloudWatch

지표	설명
개수	양자 태스크 수.
지연 시간	이 지표는 양자 작업이 완료될 때 출력됩니다. 이는 양자 작업 초기화부터 완료까지의 총 시간을 나타냅니다.

퀀텀 태스크 메트릭스의 크기

양자 작업 메트릭은 arn:aws:braket: ::device/xxx 형식의 `deviceArn` 파라미터를 기반으로 하는 차원과 함께 게시됩니다.

지원되는 디바이스

지원되는 기기 및 기기 ARN 목록은 브라켓 기기를 참조하십시오.

> **Note**
>
> Amazon 콘솔의 노트북 세부 정보 페이지로 이동하여 Amazon Braket 노트북의 CloudWatch 로그 스트림을 볼 수 있습니다. SageMaker 추가 Amazon Braket 노트북 설정은 콘솔을 통해 사용할 수 있습니다. SageMaker

Amazon Braket API를 사용한 로깅 CloudTrail

AmazonBraket은 사용자 AWS CloudTrail, 역할 또는 AWS 서비스 Amazon Braket에서 수행한 작업의 기록을 제공하는 서비스와 통합됩니다. CloudTrail AmazonBraket에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처된 통화에는 Amazon Braket 콘솔에서의 통화 및 Braket Braket 작업에 대한 코드 호출이 Amazon 포함됩니다. 트레일을 생성하면 Amazon Braket에 대한 CloudTrail 이벤트를 포함하여 Amazon S3 버킷으로 이벤트를 지속적으로 전송할 수 있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 Amazon Braket에 이루어진 요청, 요청한 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 CloudTrail 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

아마존 브라켓 정보 입력 CloudTrail

CloudTrail 계정을 생성할 AWS 계정 때 활성화됩니다. AmazonBraket에서 활동이 발생하면 해당 활동이 CloudTrail 이벤트 기록의 다른 이벤트와 함께 AWS 서비스 이벤트에 기록됩니다. 내 페이지에서 최근 이벤트를 보고, 검색하고, 다운로드할 수 있습니다. AWS 계정자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

AmazonBraket 이벤트를 포함하여 내 이벤트의 진행 중인 기록을 보려면 트레일을 생성하십시오 AWS 계정. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 트레일은 AWS 파티션에 있는 모든 지역의 이벤트를 기록하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 AWS 서비스 취하도록 기타를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하십시오.

- 추적 생성 개요

- CloudTrail 지원되는 서비스 및 통합
- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신 및 여러 계정으로부터 CloudTrail 로그 파일 수신

모든 Amazon Braket 작업은 에 의해 CloudTrail 기록됩니다. 예를 들어, `GetQuantumTask` 또는 `GetDevice` 작업을 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 역할 또는 페더레이션 사용자에 대한 임시 보안 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 CloudTrail 사용자 ID 요소를 참조하십시오.

Amazon Braket 로그 파일 항목의 이해

트레일은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 양자 작업의 세부 정보를 가져오는 작업에 대한 로그 항목입니다. `GetQuantumTask`

```
{
  "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"
}
```

다음은 장치 이벤트의 세부 정보를 반환하는 GetDevice 작업에 대한 로그 항목을 보여줍니다.

```
{
  "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"
}
```


를 사용하여 Amazon Braket 노트북 인스턴스를 생성합니다. AWS CloudFormation

를 AWS CloudFormation 사용하여 Amazon Braket 노트북 인스턴스를 관리할 수 있습니다. 브라켓 노트북 인스턴스는 SageMaker Amazon에서 빌드됩니다. 를 사용하면 의도한 구성을 설명하는 템플릿 파일을 사용하여 노트북 인스턴스를 프로비저닝할 수 있습니다. CloudFormation 템플릿 파일은 JSON 또는 YAML 형식으로 작성됩니다. 순서대로 반복 가능한 방식으로 인스턴스를 생성, 업데이트 및 삭제할 수 있습니다. 이는 여러 Braket 노트북 인스턴스를 관리할 때 유용할 수 있습니다. AWS 계정

Braket 노트북용 CloudFormation 템플릿을 생성한 후에는 리소스를 AWS CloudFormation 배포하는 데 사용합니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 AWS CloudFormation 콘솔에서 스택 만들기를 참조하십시오.

를 사용하여 CloudFormation Braket 노트북 인스턴스를 만들려면 다음 세 단계를 수행하십시오.

1. Amazon 수명 SageMaker 주기 구성 스크립트를 생성합니다.
2. 담당할 AWS Identity and Access Management (IAM) 역할을 생성하십시오. SageMaker
3. 접두사를 사용하여 SageMaker 노트북 인스턴스를 생성합니다. **amazon-braket-**

생성한 모든 Braket 노트북의 수명 주기 구성을 재사용할 수 있습니다. 동일한 실행 권한을 할당한 Braket 노트북의 IAM 역할을 재사용할 수도 있습니다.

1단계: Amazon 수명 SageMaker 주기 구성 스크립트 생성

다음 템플릿을 사용하여 SageMaker 수명 주기 구성 스크립트를 생성합니다. 이 스크립트는 Braket용 SageMaker 노트북 인스턴스를 사용자 지정합니다. 라이프사이클 CloudFormation 리소스의 구성 옵션은 AWS CloudFormation 사용 AWS::SageMaker::NotebookInstanceLifecycleConfig설명서를 참조하십시오.

```
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
```

2단계: Amazon에서 위임하는 IAM 역할 생성 SageMaker

Braket 노트북 인스턴스를 사용하는 경우 사용자를 대신하여 작업을 SageMaker 수행합니다. 예를 들어 지원되는 기기의 회로를 사용하여 Braket 노트북을 실행한다고 가정해 보겠습니다. 노트북 인스턴스 내에서 Braket에서 작업을 자동으로 SageMaker 실행합니다. 노트북 실행 역할은 사용자를 대신하여 실행하도록 허용된 정확한 작업을 정의합니다. SageMaker 자세한 내용은 Amazon SageMaker 개발자 안내서의 SageMaker 역할을 참조하십시오.

다음 예제를 사용하여 필요한 권한이 있는 Braket 노트북 실행 역할을 생성합니다. 필요에 따라 정책을 수정할 수 있습니다.

> **Note**
>
> 이 역할에 접두사가 붙은 Amazon S3 버킷에서의 `s3:ListBucket` 및 `s3:GetObject` 작업에 대한 권한이 있는지 확인하십시오. `braketnotebookcdk-"` 수명 주기 구성 스크립트에 Braket 노트북 설치 스크립트를 복사하려면 이러한 권한이 필요합니다.

```
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: "*"
```

3단계: 접두사를 사용하여 Amazon SageMaker 노트북 인스턴스 생성 **amazon-braket-**

1단계와 2단계에서 생성한 SageMaker 수명 주기 스크립트와 IAM 역할을 사용하여 SageMaker 노트북 인스턴스를 생성합니다. 노트북 인스턴스는 Braket에 맞게 사용자 지정되며 Amazon Braket 콘솔을 사용하여 액세스할 수 있습니다. 이 CloudFormation 리소스의 구성 옵션에 대한 자세한 내용은 AWS CloudFormation 사용 AWS::SageMaker::NotebookInstance설명서를 참조하십시오.

```
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
```

고급 로깅

로거를 사용하여 전체 작업 처리 프로세스를 기록할 수 있습니다. 이러한 고급 로깅 기술을 사용하면 백그라운드 폴링을 확인하고 나중에 디버깅할 수 있도록 레코드를 만들 수 있습니다.

로거를 사용하려면 양자 작업이 오래 실행되고 양자 작업 상태가 지속적으로 기록되고 결과가 파일에 저장되도록 `poll_timeout_seconds` 및 `poll_interval_seconds` 매개변수를 변경하는 것이 좋습니다. 이 코드를 Jupyter 노트북 대신 Python 스크립트로 전송하여 스크립트가 백그라운드에서 프로세스로 실행되도록 할 수 있습니다.

로거를 설정하세요.

먼저, 다음 예제 줄과 같이 모든 로그가 자동으로 텍스트 파일에 기록되도록 로거를 구성합니다.

```
# 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
```

회로 생성 및 실행

이제 회로를 만들고 실행할 장치에 회로를 제출하고 이 예와 같이 어떤 일이 발생하는지 확인할 수 있습니다.

```
# 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
)
```

로그 파일을 확인하세요.

다음 명령을 입력하여 파일에 기록된 내용을 확인할 수 있습니다.

```
# 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})
```

로그 파일에서 ARN 가져오기

이전 예와 같이 반환된 로그 파일 출력에서 ARN 정보를 얻을 수 있습니다. ARN ID를 사용하면 완료된 양자 작업의 결과를 검색할 수 있습니다.

```
# 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()
```

아마존 브라켓의 보안

이 장은 Amazon Braket을 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 보안 및 규정 준수 목표를 충족하도록 Amazon Braket을 구성하는 방법을 보여줍니다. 또한 Amazon Braket 리소스를 모니터링하고 보호하는 데 도움이 AWS 서비스 되는 기타 리소스를 사용하는 방법도 알아봅니다.

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. 여러분은 AWS 고객으로서 보안에 민감한 기관의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다. 데이터의 민감도, 회사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요인에 대한 책임은 귀하에게 있습니다.

보안에 대한 공동 책임

보안은 AWS과 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 클라우드 및 보안의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Braket에 적용되는 규정 준수 프로그램에 대해 알아보려면 규정 준수 프로그램별 범위 내 AWS 서비스를 참조하십시오.
- 클라우드에서의 보안 — 이 AWS 인프라에서 호스팅되는 콘텐츠에 대한 제어를 유지하는 것은 사용자의 책임입니다. 이 콘텐츠에는 사용하는 AWS 서비스 서비스의 보안 구성과 관리 작업이 포함되어 있습니다.

데이터 보호

AWS공동 책임 모델 Amazon Braket의 데이터 보호에 적용됩니다. 이 모델이 설명하는 것처럼 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 Data Privacy FAQ(데이터 프라이버시 FAQ)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS Shared Responsibility Model and GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정보안 인증 정보를 보호하고 AWS IAM Identity Center또는 AWS Identity and Access Management(IAM)를 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이렇게 하

면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 멀티 팩터 인증 설정(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2는 필수이며 TLS 1.3를 권장합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정하세요.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용하세요.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 인증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하세요. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 FIPS(Federal Information Processing Standard) 140-2를 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 Name 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 Amazon Braket 또는 기타 콘솔AWS CLI, API 또는 AWS 서비스 SDK를 사용하여 작업하는 경우가 포함됩니다. AWS 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 보안 인증를 URL에 포함시켜서는 안 됩니다.

데이터 보존

90일이 지나면 Amazon Braket은 양자 작업과 관련된 모든 양자 작업 ID 및 기타 메타데이터를 자동으로 제거합니다. 이 데이터 보존 정책에 따라 이러한 작업과 결과는 S3 버킷에 저장된 상태로 유지되더라도 Amazon Braket 콘솔에서 더 이상 검색으로 검색할 수 없습니다.

90일 이상 S3 버킷에 저장된 과거 양자 작업 및 결과에 액세스해야 하는 경우 작업 ID 및 해당 데이터와 관련된 기타 메타데이터를 별도로 기록해 두어야 합니다. 90일 이전에 정보를 저장해야 합니다. 저장된 정보를 사용하여 과거 데이터를 검색할 수 있습니다.

아마존 브라켓에 대한 액세스 관리

이 장에서는 Amazon Braket을 실행하거나 특정 사용자 및 역할의 액세스를 제한하는 데 필요한 권한을 설명합니다. 계정의 모든 사용자 또는 역할에 필요한 권한을 부여 (또는 거부) 할 수 있습니다. 이렇게 하려면 다음 섹션에 설명된 대로 계정의 해당 사용자 또는 역할에 적절한 Amazon Braket 정책을 연결하십시오.

사전 요구 사항으로 Amazon Braket을 활성화해야 합니다. Braket을 활성화하려면 (1) 관리자 권한이 있거나 (2) AmazonBraketFullAccess정책이 할당되고 Amazon Simple Storage Service (Amazon S3) 버킷을 생성할 권한이 있는 사용자 또는 역할로 로그인해야 합니다.

이 섹션:

아마존 브라켓 리소스

Braket은 양자 작업 리소스라는 한 가지 유형의 리소스를 생성합니다. 이 리소스 유형의 Amazon 리소스 이름 (ARN) 은 다음과 같습니다.

- 리소스 이름:: :서비스AWS: :브라켓
- ARN 정규식: ARN: $ {파티션} :브래킷: $ {지역} :$ {계정} :퀀텀 태스크/$ {} RandomId

노트북 및 역할

브라켓에서 노트북 리소스 유형을 사용할 수 있습니다. 노트북은 Braket이 공유할 수 있는 Amazon SageMaker 리소스입니다. Braket과 함께 노트북을 사용하려면 로 시작하는 이름으로 IAM 역할을 지정해야 합니다. `AmazonBraketServiceSageMakerNotebook`

노트북을 만들려면 관리자 권한이 있거나 다음과 같은 인라인 정책이 첨부된 역할을 사용해야 합니다.

```
{
   "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*"
                    ]
                }
            }
        }
    ]
}
```

역할을 만들려면 전자 필기장 만들기 페이지에 나와 있는 단계를 따르거나 관리자가 대신 생성하도록 하십시오. AmazonBraketFullAccess정책이 첨부되어 있는지 확인하십시오.

역할을 생성한 후에는 향후 시작하는 모든 노트북에 해당 역할을 재사용할 수 있습니다.

정책 정보 AmazonBraketFullAccess

이 AmazonBraketFullAccess정책은 다음 작업에 대한 권한을 포함하여 Amazon Braket 작업에 대한 권한을 부여합니다.

- Amazon Elastic 컨테이너 레지스트리에서 컨테이너 다운로드 — Amazon Braket Hybrid Jobs 기능에 사용되는 컨테이너 이미지를 읽고 다운로드하려면 컨테이너는 “arn:aws:ecr: ::리포지토리/아마존 브라켓” 형식을 준수해야 합니다.
- AWS CloudTrail 로그 보관 — 쿼리 시작 및 중지, 지표 필터 테스트, 로그 이벤트 필터링 외에도 모든 설명, 가져오기 및 나열 작업에 대해 기록합니다. AWS CloudTrail 로그 파일에는 계정에서 발생하는 모든 Amazon Braket API 활동 기록이 들어 있습니다.
- 역할을 활용하여 리소스를 제어 — 계정에서 서비스 연결 역할을 생성합니다. 서비스 연결 역할은 사용자를 대신하여 AWS 리소스에 액세스할 수 있습니다. Amazon Braket 서비스에서만 사용할 수 있습니다. 또한 Amazon `CreateJob` API Braket에 IAM 역할을 전달하고 역할을 생성하고 범위가 지정된 정책을 역할에 연결할 AmazonBraketFullAccess 수 있습니다.
- 계정에 대한 사용 로그 파일을 유지 관리하기 위해 로그 그룹, 로그 이벤트 및 쿼리 로그 그룹을 생성합니다. 계정에서 Amazon Braket 사용에 대한 로깅 정보를 생성, 저장 및 확인합니다. 하이브리드 작업 로그 그룹의 쿼리 지표 적절한 Braket 경로를 포함하고 로그 데이터 입력을 허용하십시오. 메트릭 데이터를 입력하세요. CloudWatch
- Amazon S3 버킷에 데이터 생성 및 저장, 모든 버킷 나열 - S3 버킷을 생성하려면 계정의 S3 버킷을 나열하고, 이름이 amazon-braket-로 시작하는 계정 내 모든 버킷에 객체를 넣고 객체를 가져옵니다. Braket이 처리된 양자 작업의 결과가 포함된 파일을 버킷에 넣고 버킷에서 검색하려면 이러한 권한이 필요합니다.
- IAM 역할 전달 - IAM 역할을 에 전달합니다. `CreateJob` API
- Amazon SageMaker Notebook — “arn:aws:sagemaker: SageMaker ::노트북 인스턴스/amazon-braket-”의 리소스로 범위가 지정된 노트북 인스턴스를 생성하고 관리합니다.
- 서비스 할당량 확인 — SageMaker 노트북 및 Amazon Braket Hybrid 작업을 생성하려면 리소스 수가 계정에 대한 할당량을 초과할 수 없습니다.

정책 내용

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

AmazonBraketJobsExecutionPolicy정책 정보

이 AmazonBraketJobsExecutionPolicy정책은 다음과 같이 Amazon Braket 하이브리드 작업에서 사용되는 실행 역할에 대한 권한을 부여합니다.

- Amazon Elastic 컨테이너 레지스트리에서 컨테이너 다운로드 - Amazon Braket 하이브리드 작업 기능에 사용되는 컨테이너 이미지를 읽고 다운로드할 수 있는 권한입니다. 컨테이너는 “arn:aws:ecr:*:*:리포지토리/아마존 브라켓*” 형식을 준수해야 합니다.

- 계정에 대한 사용 로그 파일을 유지 관리하기 위해 로그 그룹과 로그 이벤트 및 쿼리 로그 그룹을 생성합니다. 계정에서 Amazon Braket 사용에 대한 로깅 정보를 생성, 저장 및 확인합니다. 하이브리드 작업 로그 그룹의 쿼리 지표 적절한 Braket 경로를 포함하고 로그 데이터 입력을 허용하십시오. 메트릭 데이터를 입력하세요. CloudWatch
- Amazon S3 버킷에 데이터 저장 — 계정에 있는 S3 버킷을 나열하고, 이름에 amazon-braket-로 시작하는 계정 내 모든 버킷에 객체를 넣고, 객체를 가져옵니다. Braket이 처리된 양자 작업의 결과가 포함된 파일을 버킷에 넣고 버킷에서 해당 파일을 검색하려면 이러한 권한이 필요합니다.
- IAM 역할 전달 — IAM 역할을 에 전달합니다. CreateJob API 역할은 arn:aws:iam: :** 형식을 준수해야 합니다. :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"
     }
    }
   }
  ]
}
```

특정 디바이스에 대한 사용자 액세스를 제한하세요

특정 사용자의 액세스를 특정 Braket 장치에 제한하려면 특정 역할에 거부 권한 정책을 추가할 수 있습니다. IAM

해당 권한으로 다음 작업을 제한할 수 있습니다.

- `CreateQuantumTask`- 특정 디바이스에서 양자 태스크 생성을 거부할 수 있습니다.
- `CreateJob`- 특정 디바이스에서의 하이브리드 작업 생성을 거부할 수 있습니다.
- `GetDevice`- 지정된 장치의 세부 정보를 가져오는 것을 거부합니다.

다음 예에서는 의 모든 QPU에 대한 액세스를 제한합니다. AWS 계정 123456789012

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
  ]
}
```

이 코드를 수정하려면 이전 예제에 표시된 문자열을 제한된 장치의 Amazon 리소스 번호 (ARN) 로 대체하십시오. 이 문자열은 리소스 값을 제공합니다. Braket에서 디바이스는 양자 작업을 실행하기 위해 호출할 수 있는 QPU 또는 시뮬레이터를 나타냅니다. 사용 가능한 디바이스는 디바이스 페이지에 나열되어 있습니다. 이러한 장치에 대한 액세스를 지정하는 데 사용되는 두 가지 스키마가 있습니다.

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

다음은 다양한 유형의 기기 액세스에 대한 예시입니다.

- 모든 지역의 모든 QPU를 선택하려면: `arn:aws:braket:*:*:device/qpu/*`
- us-west-2 지역에서만 모든 QPU를 선택하려면: `arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- 마찬가지로, us-west-2 지역의 모든 QPU를 선택하는 것도 마찬가지입니다 (디바이스는 고객 리소스가 아니라 서비스 리소스이므로). `arn:aws:braket:us-west-2:* :device/qpu/*`
- 모든 온디맨드 시뮬레이터 장치에 대한 액세스를 제한하려면: `arn:aws:braket:* :123456789012:device/quantum-simulator/*`

- us-east-1 지역의 IonQ Harmony 장치에 대한 액세스를 제한하려면: `arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- 특정 공급자의 기기 (예: 기기) 에 대한 액세스를 제한하려면 RigettiQPU: `arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- TN1기기에 대한 액세스를 제한하려면: `arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

관리형 정책에 대한 AWS Amazon Braket 업데이트

다음 표에는 이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 Braket의 AWS 관리형 정책 업데이트에 대한 세부 정보가 나와 있습니다.

변경 사항	설명	날짜
AmazonBraketFullAccess- Braket에 대한 전체 액세스 정책	정책에 포함할 서비스 할당량: GetServiceQuota 및 cloudwatch: GetMetricData 작업을 추가했습니다. AmazonBraketFullAccess .	2023년 3월 24일
AmazonBraketFullAccess- Braket에 대한 전체 액세스 정책	브라켓 조정 iam: 경로를 포함할 AmazonBraketFullAccess 수 있는 PassRole 권한 `service-role/`	2021년 11월 29일
AmazonBraketJobsExecutionPolicy- 브래킷 하이브리드 작업에 대한 Amazon 하이브리드 작업 실행 정책	Braket은 경로를 `service-role/` 포함하도록 하이브리드 작업 실행 역할 ARN을 업데이트했습니다.	2021년 11월 29일
Braket은 변경 사항을 추적하기 시작했습니다.	Braket은 AWS 관리형 정책의 변경 사항을 추적하기 시작했습니다.	2021년 11월 29일

특정 노트북 인스턴스에 대한 사용자 액세스를 제한합니다.

특정 사용자의 액세스를 특정 Braket 노트북 인스턴스로 제한하려면 특정 역할, 사용자 또는 그룹에 거부 권한 정책을 추가할 수 있습니다.

다음 예제에서는 정책 변수를 사용하여 에서 특정 노트북 인스턴스를 시작, 중지 및 액세스할 수 있는 권한을 효율적으로 제한합니다. 액세스 권한이 있어야 하는 사용자에 따라 이름이 지정됩니다 (예: 사용자는 `Alice` 이름이 지정된 `amazon-braket-Alice` 노트북 인스턴스에 액세스할 수 있음). AWS 계정 `123456789012`

```
{
  "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}*"
      ]
    }
  ]
}
```

특정 S3 버킷에 대한 사용자 액세스를 제한합니다.

특정 사용자의 액세스를 특정 Amazon S3 버킷으로 제한하려면 특정 역할, 사용자 또는 그룹에 거부 정책을 추가할 수 있습니다.

다음 예제는 객체를 검색하고 특정 S3 버킷 (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) 에 배치할 수 있는 권한을 제한하고 해당 객체의 목록도 제한합니다.

```
{
  "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/*"
      ]
    }
  ]
}
```

특정 노트북 인스턴스의 버킷 액세스를 제한하려면 노트북 실행 역할에 이전 정책을 추가할 수 있습니다.

아마존 브래킷 서비스 연결 역할

Amazon Braket을 활성화하면 계정에 서비스 연결 역할이 생성됩니다.

서비스 연결 역할은 이 경우 Amazon Braket에 직접 연결되는 고유한 유형의 IAM 역할입니다. Amazon Braket 서비스 연결 역할은 Braket이 사용자를 대신하여 다른 사람에게 전화를 걸 때 요구하는 모든 권한을 포함하도록 사전 정의되어 있습니다. AWS 서비스

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 Amazon Braket을 더 쉽게 설정할 수 있습니다. Amazon Braket은 서비스 연결 역할의 권한을 정의합니다. 이러한 정의를 변경하지 않는 한 Amazon Braket만이 역할을 맡을 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함됩니다. 권한 정책은 다른 어떤 IAM 엔터티에도 연결할 수 없습니다.

Amazon Braket이 설정하는 서비스 연결 역할은 AWS Identity and Access Management (IAM) 서비스 연결 역할 기능의 일부입니다. 서비스 연결 역할을 AWS 서비스 지원하는 기타 서비스에 대한 자세한 내용은 IAM과 함께 작동하는 AWS 서비스를 참조하고 서비스 연결 역할 열에서 Yes가 표시된 서비스를 찾아보십시오. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

Amazon Braket에 대한 서비스 연결 역할 권한

Amazon Braket은 braket.amazonaws.com 엔티티를 신뢰하는 `AWSServiceRoleForAmazonBraket` 서비스 연결 역할을 사용하여 역할을 수임합니다.

IAM 개체 (예: 그룹 또는 역할) 가 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 권한을 구성해야 합니다. 자세한 내용은 서비스 연결 역할 권한을 참조하십시오.

Amazon Braket의 서비스 연결 역할에는 기본적으로 다음 권한이 부여됩니다.

- Amazon S3 — 계정의 버킷을 나열하고, Amazon-braket-로 시작하는 이름의 계정 내 모든 버킷에 객체를 넣고 해당 버킷에서 객체를 가져올 수 있는 권한.
- Amazon CloudWatch Logs — 로그 그룹을 나열 및 생성하고, 관련 로그 스트림을 생성하고, Amazon Braket용으로 생성된 로그 그룹에 이벤트를 넣을 수 있는 권한입니다.

`AWSServiceRoleForAmazonBraket`서비스 연결 역할에는 다음 정책이 첨부됩니다.

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

아마존 브래킷의 탄력성

AWS글로벌 인프라는 가용 영역을 중심으로 AWS 리전 구축됩니다.

각 지역은 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 이러한 가용 영역 (AZ) 은 지연 시간이 짧고 처리량이 높으며 이중화가 높은 네트워킹을 통해 연결됩니다. 따라서 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성이 높고 내결함성이 뛰어나며 확장성이 뛰어납니다.

AZ 간에 페일오버되는 애플리케이션 및 데이터베이스를 중단 없이 자동으로 설계하고 운영할 수 있습니다.

AWS 리전및 가용 영역에 대한 자세한 내용은 AWS글로벌 인프라를 참조하십시오.

아마존 브래킷에 대한 규정 준수 검증

타사 감사자는 Amazon Braket의 보안 및 규정 준수와 타사 하드웨어 공급자와의 통합을 정기적으로 평가합니다. Braket의 규정 준수 정보 up-to-date 목록은 규정 준수 프로그램별 범위를 참조하십시오 AWS 서비스. 일반 정보는 AWS규정 준수를 참조하십시오.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 에서 보고서 다운로드를 참조하십시오AWS Artifact.

> **Note**
>
> AWS규정 준수 보고서에는 자체 독립 감사를 거칠 수 있는 타사 하드웨어 공급업체의 QPU가 포함되지 않습니다.

Amazon Braket을 사용할 때의 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS규정 준수에 도움이 되는 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- AWS 규정 준수 리소스 – 사용자의 업계와 위치에 해당할 수 있는 워크북 및 안내서 모음입니다.

아마존 브래킷의 인프라 보안

관리형 서비스인 Amazon Braket은 보안 프로세스 개요 백서에 AWS 설명된 AWS 글로벌 네트워크 보안 절차에 의해 보호됩니다.

네트워크를 통해 Amazon Braket에 액세스하려면 게시된 AWS API를 호출해야 합니다. 클라이언트가 전송 계층 보안(TLS) 1.2 이상을 지원해야 합니다. 또한 클라이언트는 임시 디피-헬만 (DHE) 또는 타원 곡선 임시 디피-헬만 (ECDHE) 과 같은 완벽한 순방향 보안 (PFS) 을 갖춘 암호 제품군을 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

Amazon Braket 하드웨어 공급자의 보안

Amazon Braket의 QPU는 타사 하드웨어 공급자가 호스팅합니다. QPU에서 양자 작업을 실행하면 Amazon Braket은 처리를 위해 지정된 QPU로 회로를 보낼 때 DevicEarn을 식별자로 사용합니다.

타사 하드웨어 제공업체 중 한 곳에서 운영하는 양자 컴퓨팅 하드웨어에 액세스하는 데 Amazon Braket을 사용하는 경우 회로와 관련 데이터는 해당 공급자가 운영하는 시설 외부의 하드웨어 공급자에 의해 처리됩니다.AWS. 물리적 위치에 대한 정보AWS각 QPU를 사용할 수 있는 지역은 다음에서 찾을 수 있습니다.장치 세부 정보아마존 브라켓 콘솔의 섹션

콘텐츠는 익명화됩니다. 회로를 처리하는 데 필요한 콘텐츠만 제3자에게 전송됩니다.AWS 계정정보는 제3자에게 전송되지 않습니다.

데이터는 저장 전송 중에 암호화됩니다. 데이터는 처리 목적으로만 복호화됩니다. Amazon Braket 타사 공급자는 회로 처리 이외의 목적으로 콘텐츠를 저장하거나 사용할 수 없습니다. 회로가 완료되면 결과가 Amazon Braket으로 반환되고 S3 버킷에 저장됩니다.

Amazon Braket 타사 양자 하드웨어 공급자의 보안은 정기적으로 감사를 실시하여 네트워크 보안, 액세스 제어, 데이터 보호 및 물리적 보안 표준을 충족하는지 확인합니다.

아마존 브라켓용 아마존 VPC 엔드포인트

인터페이스 VPC 엔드포인트를 생성하여 VPC와 Amazon Braket 간에 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이 AWS PrivateLink, NAT 장치, VPN 연결 또는 연결 없이 Braket API에 액세스할 수 있게 해주는 기술인 기술로 구동됩니다. AWS Direct Connect VPC의 인스턴스는 Braket API와 통신하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다.

를 사용하면 PrivateLink VPC와 Braket 간의 트래픽이 Amazon 네트워크를 떠나지 않으므로 데이터가 퍼블릭 인터넷에 노출되는 것을 줄이므로 클라우드 기반 애플리케이션과 공유하는 데이터의 보안이 강화됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트 (AWS PrivateLink) 를 참조하십시오.

Amazon Braket VPC 엔드포인트에 대한 고려 사항

Braket용 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한을 검토해야 합니다.

Braket은 VPC에서 모든 API 작업에 대한 호출을 지원합니다.

기본적으로 VPC 엔드포인트를 통해 Braket에 대한 전체 액세스가 허용됩니다. VPC 엔드포인트 정책을 지정하면 액세스를 제어할 수 있습니다. 자세한 정보는 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어 섹션을 참조하세요.

브라켓을 설정하고 PrivateLink

AWS PrivateLinkAmazonBraket과 함께 사용하려면 Amazon VPC (가상 사설 클라우드) 엔드포인트를 인터페이스로 생성한 다음 Braket 서비스를 통해 Amazon 엔드포인트에 연결해야 합니다. API

이 프로세스의 일반적인 단계는 다음과 같습니다. 이에 대해서는 이후 섹션에서 자세히 설명합니다.

- Amazon VPC를 구성하고 실행하여 리소스를 호스팅합니다. AWS VPC가 이미 있는 경우에는 이 단계를 건너뛸 수 있습니다.
- 브라켓용 아마존 VPC 엔드포인트 생성
- 엔드포인트를 통해 Braket 퀀텀 태스크를 연결하고 실행하세요

1단계: 필요한 경우 Amazon VPC 시작

계정에 이미 VPC가 운영 중인 경우 이 단계를 건너뛰어도 된다는 점을 기억하세요.

VPC는 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이와 같은 네트워크 설정을 제어합니다. 기본적으로 사용자 지정 가상 네트워크에서 AWS 리소스를 시작하는 것입니다. VPC에 대한 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

Amazon VPC 콘솔을 열고 서브넷, 보안 그룹, 네트워크 게이트웨이가 있는 새 VPC를 생성합니다.

2단계: 브라켓용 인터페이스 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 () 를 사용하여 Braket 서비스에 대한 VPC 엔드포인트를 생성할 수 있습니다. AWS Command Line Interface AWS CLI 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

콘솔에서 VPC 엔드포인트를 생성하려면 Amazon VPC 콘솔을 열고 엔드포인트 페이지를 열고 새 엔드포인트 생성을 진행하십시오. 나중에 참조할 수 있도록 엔드포인트 ID를 기록해 둡니다. APIBraket에 특정 전화를 걸 때 `–endpoint-url` 플래그의 일부로 필요합니다.

다음 서비스 이름을 사용하여 Braket의 VPC 엔드포인트를 생성합니다.

- com.amazonaws.substitute_your_region.braket

참고: 엔드포인트에 프라이빗 DNS를 활성화하면 해당 지역의 기본 DNS 이름 (예:) 을 사용하여 Braket에 API 요청을 보낼 수 있습니다. braket.us-east-1.amazonaws.com

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통해 서비스 액세스 섹션을 참조하세요.

3단계: 엔드포인트를 통해 Braket 퀀텀 태스크를 연결하고 실행하세요

VPC 엔드포인트를 생성한 후 다음 예와 같이 endpoint-url 파라미터가 포함된 CLI 명령을 실행하여 API 또는 런타임에 대한 인터페이스 엔드포인트를 지정할 수 있습니다.

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

VPC 엔드포인트에 프라이빗 DNS 호스트 이름을 활성화하는 경우 CLI 명령에서 엔드포인트를 URL로 지정할 필요가 없습니다. 대신 CLI와 Amazon 브라켓 SDK가 기본적으로 사용하는 브라켓 API DNS 호스트 이름이 VPC 엔드포인트로 확인됩니다. 다음 예제에 표시된 형식은 다음과 같습니다.

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

엔드포인트를 사용하여 AWS PrivateLink Amazon VPC에서 Amazon SageMaker 노트북에 직접 액세스한다는 블로그 게시물에는 Braket 노트북과 유사한 Amazon 엔드포인트를 설정하여 노트북에 안전하게 SageMaker 연결하는 방법의 예시가 나와 있습니다.

블로그 게시물의 단계를 따르고 있다면 SageMakerAmazon을 Amazon Braket이라는 이름으로 대체하는 것을 잊지 마십시오. 해당 지역이 us-east-1이 아닌 경우 서비스 AWS 리전 이름에 해당 문자열에 올바른 이름을 com.amazonaws.us-east-1.braket 입력하거나 대체하십시오.

엔드포인트 생성에 대해 자세히 알아보기

- 프라이빗 서브넷이 있는 VPC를 만드는 방법에 대한 자세한 내용은 프라이빗 서브넷이 있는 VPC 생성을 참조하십시오.
- Amazon VPC 콘솔 또는 AWS CLI를 사용한 엔드포인트 생성 및 구성에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하십시오.
- 를 사용하여 AWS CloudFormation 엔드포인트를 생성하고 구성하는 방법에 대한 자세한 내용은 사용 설명서의 AWS: :EC2: :VPCEndPoint 리소스를 참조하십시오. AWS CloudFormation

Amazon VPC 엔드포인트 정책을 통한 액세스 제어

AmazonBraket에 대한 연결 액세스를 제어하려면 Amazon VPC 엔드포인트에 AWS Identity and Access Management (IAM) 엔드포인트 정책을 연결할 수 있습니다. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체 (사용자 또는 역할)
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 정보는 Amazon VPC 사용 설명서의 VPC 엔드포인트를 통해 서비스에 대한 액세스 제어 섹션을 참조하세요.

예: 브라켓 작업에 대한 VPC 엔드포인트 정책

다음 예는 Braket의 엔드포인트 정책을 보여줍니다. 엔드포인트에 연결할 경우 이 정책은 모든 리소스의 모든 보안 주체에 대해 나열된 Braket 작업에 대한 액세스 권한을 부여합니다.

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

엔드포인트 정책 여러 개를 연결하여 복잡한 IAM 규칙을 만들 수 있습니다. 자세한 내용 및 예는 다음을 참조하십시오.

- Step Functions에 대한 Amazon Virtual Private 클라우드 엔드포인트 정책
- 관리자가 아닌 사용자를 위한 세분화된 IAM 권한 생성
- VPC 엔드포인트로 서비스에 대한 액세스 제어

아마존 브라켓 문제 해결

이 섹션의 문제 해결 정보와 솔루션을 사용하면 Amazon Braket 관련 문제를 해결하는 데 도움이 됩니다.

이 섹션:

AccessDeniedException

Braket을 활성화하거나 사용할 AccessDeniedException때 알림을 받으면 제한된 역할에 액세스할 수 없는 지역에서 Braket을 활성화 또는 사용하려고 시도한 것일 수 있습니다.

이러한 경우 내부 AWS 관리자에게 문의하여 다음 조건 중 어떤 것이 적용되는지 확인해야 합니다.

- 역할 제한으로 인해 지역에 액세스할 수 없는 경우
- 사용하려는 역할이 허용된 경우 Braket을 사용할 수 있습니다.

Braket을 사용할 때 역할에 지정된 지역에 대한 액세스 권한이 없는 경우 해당 지역의 장치를 사용할 수 없습니다.

작업을 호출하는 동안 오류가 발생했습니다 (ValidationException). CreateQuantumTask

다음과 비슷한 오류가 발생하는 경우: 기존 s3_folder를 참조하고 `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-braket-…` 있는지 확인하십시오. Braket은 새 Amazon S3 버킷과 접두사를 자동으로 생성하지 않습니다.

에 API 직접 액세스하여 다음과 비슷한 오류가 발생하는 경우: Amazon S3 버킷 경로에 포함되어 있지 않은지 `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` 확인하십시오. `s3://`

SDK 기능이 작동하지 않습니다.

Python 버전은 3.9 이상이어야 합니다. 아마존 브라켓 하이브리드 작업의 경우, Python 3.10을 권장합니다.

SDK와 스키마가 맞는지 확인하십시오. up-to-date 노트북 또는 Python 편집기에서 SDK를 업데이트하려면 다음 명령어를 실행하세요.

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

스키마를 업데이트하려면 다음 명령어를 실행합니다.

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

자체 클라이언트에서 Amazon Braket에 액세스하는 경우, AWS 지역이 Amazon Braket에서 지원하는 지역으로 설정되어 있는지 확인하십시오.

다음과 같은 이유로 하이브리드 작업이 실패합니다. ServiceQuotaExceededException

Amazon Braket 시뮬레이터를 대상으로 양자 작업을 실행하는 하이브리드 작업은 대상 시뮬레이터 디바이스의 동시 양자 작업 한도를 초과하는 경우 생성되지 않을 수 있습니다. 서비스 한도에 대한 자세한 내용은 할당량 주제를 참조하십시오.

계정의 여러 하이브리드 작업에서 시뮬레이터 디바이스에 대해 동시 작업을 실행하는 경우 이 오류가 발생할 수 있습니다.

특정 시뮬레이터 장치에 대한 동시 양자 작업 수를 확인하려면 다음 코드 예제와 같이 를 사용하십시오. `search-quantum-tasks` API

```
DEVICE_ARN=arn:aws:braket:::device/quantum-simulator/amazon/sv1
task_list=""
for status_value in "CREATED" "QUEUED" "RUNNING" "CANCELLING"; do
    tasks=$(aws braket search-quantum-tasks --filters
 name=status,operator=EQUAL,values=${status_value}
```

```
 name=deviceArn,operator=EQUAL,values=$DEVICE_ARN --max-results 100 --query
 'quantumTasks[*].quantumTaskArn' --output text)
    task_list="$task_list $tasks"
done;
echo "$task_list" | tr -s ' \t' '[\n*]' | sort | uniq
```

Amazon CloudWatch 측정항목 (Braket > 장치별) 을 사용하여 디바이스에 대해 생성된 양자 작업을 볼 수도 있습니다.

이러한 오류가 발생하지 않도록 하려면:

1. 시뮬레이터 디바이스의 동시 양자 작업 수에 대한 서비스 할당량 증가를 요청하세요. 이는 SV1 디바이스에만 적용됩니다.
2. 코드에서 `ServiceQuotaExceeded` 예외를 처리하고 다시 시도하세요.

노트북 인스턴스에서 구성 요소가 작동을 멈췄습니다.

노트북의 일부 구성 요소가 작동을 멈추는 경우 다음을 시도해 보십시오.

1. 만들거나 수정한 노트북을 모두 로컬 드라이브에 다운로드합니다.
2. 노트북 인스턴스를 중지하십시오.
3. 노트북 인스턴스를 삭제합니다.
4. 다른 이름으로 새 노트북 인스턴스를 생성합니다.
5. 노트북을 새 인스턴스에 업로드합니다.

아마존 브라켓 쿼터

다음 표에는 Amazon Braket의 서비스 할당량이 나와 있습니다. 서비스 할당량은 AWS 계정계정의 최대 서비스 리소스 또는 작업 수입니다.

일부 할당량을 늘릴 수 있습니다. 자세한 내용은 AWS 서비스 quotas를 참조하십시오.

- 버스트 레이트 할당량은 늘릴 수 없습니다.
- 조정 가능한 할당량의 최대 속도 증가 (조정할 수 없는 버스트 속도 제외) 는 지정된 기본 속도 제한의 2배입니다. 예를 들어 기본 할당량인 60을 최대 120개로 조정할 수 있습니다.
- 동시 SV1 (DM1) 양자 작업에 대한 조정 가능한 할당량은 개당 최대 AWS 리전 60개까지 허용됩니다.

- 하이브리드 작업에 허용되는 최대 컴퓨팅 인스턴스 수는 5개이며 할당량은 조정 가능합니다.

Resource	설명	Limits	조정 가능
API 요청 비율	현재 리전의 이 계정에서 보낼 수 있는 초당 요청의 최대수입니다.	140	예
요청 버스트율 API	현재 리전의 이 계정에서 한 버스트로 보낼 수 있는 추가 초당 요청(RPS)의 최대수입니다.	600	아니요
`CreateQuantumTask` 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 `CreateQuantumTask` 요청 수입니다.	20	예
요청 버스트율 `CreateQuantumTask`	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 `CreateQuantumTask` 요청(RPS) 의 최대 수입니다.	40	아니요
`SearchQuantumTasks` 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 `SearchQuantumTasks` 요청 수입니다.	5	예
요청 버스트율 `SearchQuantumTasks`	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당	50	아니요

Resource	설명	Limits	조정 가능
	추가 SearchQuantumTasks 요청 (RPS) 의 최대 수입니다.		
GetQuantumTask 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 GetQuantumTask 요청 수입니다.	100	예
요청 버스트율 GetQuantumTask	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 GetQuantumTask 요청 (RPS) 의 최대 수입니다.	500	아니요
CancelQuantumTask 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 CancelQuantumTask 요청 수입니다.	2	예
요청 버스트율 CancelQuantumTask	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 CancelQuantumTask 요청 (RPS) 의 최대 수입니다.	20	아니요
GetDevice 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 GetDevice 요청 수입니다.	5	예

Resource	설명	Limits	조정 가능
요청 버스트율 GetDevice	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 GetDevice 요청 (RPS) 의 최대 수입니다.	50	아니요
SearchDevices 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 SearchDevices 요청 수입니다.	5	예
요청 버스트율 SearchDevices	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 SearchDevices 요청 (RPS) 의 최대 수입니다.	50	아니요
CreateJob 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 CreateJob 요청 수입니다.	1	예
요청 버스트율 CreateJob	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 CreateJob 요청 (RPS) 의 최대 수입니다.	5	아니요
SearchJob 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 SearchJob 요청 수입니다.	5	예

Resource	설명	Limits	조정 가능
요청 버스트율 SearchJob	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 SearchJob 요청 (RPS) 의 최대 수입니다.	50	아니요
GetJob 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 GetJob 요청 수입니다.	5	예
요청 버스트율 GetJob	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 GetJob 요청 (RPS) 의 최대 수입니다.	25	아니요
CancelJob 요청 비율	지역별로 이 계정에서 초당 보낼 수 있는 최대 CancelJob 요청 수입니다.	2	예
요청 버스트율 CancelJob	현재 지역의 이 계정으로 한 번에 전송할 수 있는 초당 추가 CancelJob 요청 (RPS) 의 최대 수입니다.	5	아니요
동시 SV1양자 작업 수	현재 지역의 상태 벡터 시뮬레이터 (SV1) 에서 실행되는 동시 양자 작업의 최대 수입니다.	100 미국 동부 1, 50 미국 서부 1, 100 us-west-2, 50 eu-west-2	아니요

Resource	설명	Limits	조정 가능
동시 양자 작업 수 DM1	현재 지역의 밀도 매트릭스 시뮬레이터 (DM1) 에서 실행되는 동시 양자 작업의 최대 수입니다.	100 미국 동부 1, 50 미국 서부 1, 100 us-west-2, 50 eu-west-2	아니요
동시 양자 작업 수 TN1	현재 지역의 텐서 네트워크 시뮬레이터 (TN1) 에서 실행되는 동시 양자 작업의 최대 수입니다.	10 미국 동부 1, 10 미국 서부 2, 5 유럽-서부-2,	예
동시 하이브리드 작업 수	현재 지역의 최대 동시 하이브리드 작업 수	5	예
하이브리드 작업 런타임 제한	하이브리드 작업을 실행할 수 있는 최대 시간 (일수)	5	아니요

다음은 하이브리드 작업의 기본 클래식 컴퓨팅 인스턴스 할당량입니다. 할당량을 늘리려면 문의하세요. AWS Support또한 각 인스턴스마다 사용 가능한 지역이 지정되어 있습니다.

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.c4.xlarge의 최대 인	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입	5	예	예	예	예	예	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
스턴스 수	ml.c4.xlarge의 인스턴스의 최대수입니다.							
하이브리드 작업에 사용할 수 있는 ml.c4.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.c4.2xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.c4.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c4.4xlarge의 인스턴스의 최대수입니다.	5	예	예	예	예	예	아니요
하이브리드 작업에 사용할 수 있는 ml.c4.8xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c4.8xlarge의 인스턴스의 최대수입니다.	5	예	예	예	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.c5.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5.xlarge의 인스턴스의 최대수입니다.	5	예	예	예	예	예	예
하이브리드 작업의 경우 ml.c5.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5.2xlarge의 인스턴스의 최대수입니다.	5	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.c5.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5.4xlarge의 인스턴스의 최대수입니다.	1	예	예	예	예	예	예
하이브리드 작업의 경우 ml.c5.9xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5.9xlarge의 인스턴스의 최대수입니다.	1	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.c5.18xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5.18xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예
하이브리드 작업에 사용할 수 있는 ml.c5n.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5n.xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.c5n.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5n.2xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	아니요	아니요
하이브리드 작업에 사용할 수 있는 ml.c5n.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5n.4xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.c5n.9xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5n.9xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	아니요	아니요
하이브리드 작업에 사용할 수 있는 ml.c5n.18xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.c5n.18xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.g4dn.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예
하이브리드 작업에 사용할 수 있는 ml.g4dn.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.2xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.g4dn.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.4xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예
하이브리드 작업에 사용할 수 있는 ml.g4dn.8xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.8xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.g4dn.12xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.12xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예
하이브리드 작업에 사용할 수 있는 ml.g4dn.16xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 타입 ml.g4dn.16xlarge의 인스턴스의 최대수입니다.	0	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.m4.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m4.xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	아니요
하이브리드 작업의 경우 ml.m4.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m4.2xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.m4.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m4.4xlarge 타입의 인스턴스의 최대 수입니다.	2	예	예	예	예	예	아니요
하이브리드 작업의 경우 ml.m4.10xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m4.10xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	예	예	예	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.m4.16› large의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m4.16› large 타입의 인스턴스의 최대 수입니다.	0	예	예	예	예	예	아니요
하이브리드 작업의 경우 ml.m5.large의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.large 타입의 인스턴스의 최대수입니다.	5	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.m5.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	예
하이브리드 작업의 경우 ml.m5.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.2xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.m5.4xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.4xlarge 타입의 인스턴스의 최대 수입니다.	5	예	예	예	예	예	예
하이브리드 작업의 경우 ml.m5.12xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.12xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	예	예	예	예

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.m5.24› large의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.m5.24› large 타입의 인스턴스의 최대 수입니다.	0	예	예	예	예	예	예
하이브리드 작업에 사용할 수 있는 ml.p2.xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p2.xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	아니요	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.p2.8xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p2.8xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	아니요	예	아니요	아니요
하이브리드 작업의 경우 ml.p2.16xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p2.16xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	아니요	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.p3.2xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p3.2xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	아니요	예	아니요	아니요
하이브리드 작업에 사용할 수 있는 ml.p4d.24xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p4d.24xlarge 타입의 인스턴스의 최대수입니다.	0	예	예	아니요	예	아니요	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업에 사용할 수 있는 ml.p3dn.24xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p3dn.24xlarge 타입의 인스턴스의 최대수입니다.	0	예	예	아니요	예	아니요	아니요
하이브리드 작업의 경우 ml.p3.8xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p3.8xlarge 타입의 인스턴스의 최대수입니다.	0	예	예	아니요	예	예	아니요

Resource	설명	Limits	조정 가능	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
하이브리드 작업의 경우 ml.p3.16xlarge의 최대 인스턴스 수	이 계정 및 지역의 모든 Amazon Braket Hybrid Jobs에 허용되는 ml.p3.16xlarge 타입의 인스턴스의 최대 수입니다.	0	예	예	아니요	예	예	아니요

한도 업데이트 요청

특정 인스턴스 유형에 대한 ServiceQuotaExceeded 예외를 받지만 사용할 수 있는 인스턴스가 충분하지 않은 경우 콘솔의 Service Quotas 페이지에서 한도 증가를 요청하고 서비스에서 Amazon Braket을 검색할 수 있습니다. AWS AWS

> **Note**
>
> 하이브리드 작업에서 요청된 ML 컴퓨팅 파워를 프로비저닝할 수 없는 경우 다른 지역을 사용하십시오. 또한 테이블에 인스턴스가 표시되지 않으면 하이브리드 작업에서도 해당 인스턴스를 사용할 수 없습니다.

추가 할당량 및 한도

- Amazon Braket 퀀텀 태스크 액션은 크기가 3MB로 제한됩니다.
- SV1DM1, 및 Rigetti 디바이스에 허용되는 작업당 최대 샷 수는 100,000개입니다.

- 작업당 허용되는 최대 샷 수는 TN1 1000개입니다.
- IonQ의 Aria-1 및 Aria-2 장치의 경우 작업당 최대 5,000장의 샷이 가능합니다. IonQ의 하모니 및 포르테 장치 및 장치의 경우 최대 OQC 10,000개입니다.
- 의 경우QuEra, 작업당 허용되는 최대 샷 수는 1000개입니다.
- TN1및 QPU 장치의 경우 작업당 샷 수는 0보다 커야 합니다.

OpenQASM 문제 해결

이 섹션에서는 OpenQASM 3.0을 사용할 때 오류가 발생할 때 유용할 수 있는 문제 해결 지침을 제공합니다.

이 섹션:

명령문 포함 오류

Braket에는 현재 OpenQASM 프로그램에 포함할 표준 게이트 라이브러리 파일이 없습니다. 예를 들어, 다음 예제에서는 파서 오류가 발생합니다.

```
OPENQASM 3;
include "standardlib.inc";
```

이 코드는 오류 메시지를 생성합니다. `No terminal matches '"' in the current parser context, at line 2 col 17.`

비연속 오류 qubits

장치 `true` 기능에서 로 설정된 qubits 장치에서 비연속적인 `requiresContiguousQubitIndices` 장치를 사용하면 오류가 발생합니다.

시뮬레이터 및 에서 양자 작업을 실행할 때 다음 프로그램이 IonQ 오류를 트리거합니다.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

이 코드는 오류 메시지를 생성합니다. `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

물리적 qubits 오류와 가상 qubits 오류 혼용

동일한 프로그램에서 물리적인 qubits qubits 것과 가상을 혼합하는 것은 허용되지 않으며 오류가 발생합니다. 다음 코드는 오류를 생성합니다.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

이 코드는 오류 메시지를 생성합니다. `[line 4] mixes physical qubits and qubits registers.`

동일한 프로그램에서 결과 유형 요청 및 측정 qubits 오류

동일한 프로그램에서 qubits 명시적으로 측정된 결과 유형을 요청하면 오류가 발생합니다. 다음 코드는 오류를 생성합니다.

```
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])
```

이 코드는 오류 메시지를 생성합니다. `Qubits should not be explicitly measured when result types are requested.`

클래식 및 qubit 레지스터 제한 초과 오류

클래식 레지스터 하나와 qubit 레지스터 하나만 허용됩니다. 다음 코드는 오류를 생성합니다.

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

이 코드는 오류 메시지를 생성합니다. `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

상자 앞에 프래그마 오류가 없으면 축어적인 오류가 발생합니다.

모든 상자 앞에는 축어 프래그마가 와야 합니다. 다음 코드는 오류를 생성합니다.

```
box{
rx(0.5) $0;
}
```

이 코드는 오류 메시지를 생성합니다. `In verbatim boxes, native gates are required. x is not a device native gate.`

축어적 상자: 네이티브 게이트 누락 오류

Verbatim 상자에는 네이티브 게이트와 물리적 게이트가 있어야 합니다. qubits 다음 코드는 네이티브 게이트 오류를 생성합니다.

```
#pragma braket verbatim
```

```
box{
x $0;
}
```

이 코드는 오류 메시지를 생성합니다. `In verbatim boxes, native gates are required. x is not a device native gate.`

버바틱 박스 누락 물리적 오류 qubits

축어적 상자에는 물리적인 것이 있어야 합니다. qubits 다음 코드는 누락된 물리적 qubits 오류를 생성합니다.

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

이 코드는 오류 메시지를 생성합니다. `Physical qubits are required in verbatim box.`

축어적인 프래그마에 “braket” 오류가 없습니다.

축어 프래그마에 “브라켓”을 포함해야 합니다. 다음 코드는 오류를 생성합니다.

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

이 코드는 오류 메시지를 생성합니다. `You must include “braket” in the verbatim pragma`

싱글은 인덱싱할 수 qubits 없습니다. 오류

싱글은 qubits 인덱싱할 수 없습니다. 다음 코드는 오류를 생성합니다.

```
OPENQASM 3;

qubit q;
h q[0];
```

이 코드는 오류를 생성합니다. `[line 4] single qubit cannot be indexed.`

하지만 다음과 같이 단일 qubit 배열을 인덱싱할 수 있습니다.

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

두 qubitsqubit 게이트의 물리적으로 연결되지 않았습니다. 오류

물리적 qubits 장치를 사용하려면 먼저 장치를 확인하여 물리적 qubits 장치를 사용하는지 확인한 `device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits` 다음 `device.properties.paradigm.connectivity.connectivityGraph` 또는 를 선택하여 연결 그래프를 확인합니다`device.properties.paradigm.connectivity.fullyConnected`.

```
OPENQASM 3;

cnot $0, $14;
```

이 코드는 오류 메시지를 생성합니다. `[line 3] has disconnected qubits 0 and 14`

GetDevice OpenQASM 결과 오류를 반환하지 않습니다.

Braket SDK를 사용할 때 GetDevice 응답에 OpenQASM 결과가 표시되지 않는 경우 AWS_EXECUTION_ENV 환경 변수를 설정하여 사용자 에이전트를 구성해야 할 수 있습니다. Go 및 Java SDK에서 이 작업을 수행하는 방법은 아래 제공된 코드 예제를 참조하십시오.

사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면: 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]
```

Boto3를 사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면:

```
import boto3
import botocore
```

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

/(SDK v2) 를 사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면: 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'  });
```

/(SDK v3) 를 사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면: 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' });
```

Go SDK를 사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면:

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

Java SDK를 사용할 때 사용자 에이전트를 구성하도록 AWS_EXECUTION_ENV 환경 변수를 설정하려면:

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

로컬 시뮬레이터 지원 경고

QPU 또는 온디맨드 시뮬레이터에서는 사용할 수 없는 OpenQASM의 고급 기능을 `LocalSimulator` 지원합니다. 프로그램에 다음 예와 같이 에만 해당하는 언어 기능이 포함되어 있는 경우 경고 메시지가 표시됩니다. `LocalSimulator`

```
qasm_string = """
```

```
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

이 코드는 다음과 같은 경고를 생성합니다. `이 프로그램은 에서만 지원되는 OpenQASM 언어 기능을 사용합니다. LocalSimulator 이러한 기능 중 일부는 QPU 또는 온디맨드 시뮬레이터에서 지원되지 않을 수 있습니다.

지원되는 OpenQASM 기능에 대한 자세한 내용은 여기를 클릭하십시오.

아마존 브래킷용 API 및 SDK 참조 가이드

Amazon Braket은 노트북 인스턴스를 생성 및 관리하고 모델을 교육 및 배포하는 데 사용할 수 있는 API, SDK 및 명령줄 인터페이스를 제공합니다.

- 아마존 브라켓 파이썬 SDK (권장)
- 아마존 브라켓 API 레퍼런스
- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto)
- AWS SDK for Ruby

Amazon Braket 튜토리얼 GitHub 리포지토리에서 코드 예제를 얻을 수도 있습니다.

- 브래킷 튜토리얼 GitHub

문서 이력

다음 표에는 이번 Amazon Braket 릴리스에 대한 설명서가 설명되어 있습니다.

- API버전: 2022년 4월 28일
- 최신 API 레퍼런스 업데이트: 2023년 9월 25일
- 최신 설명서 업데이트: 2024년 5월 22일

변경 사항	설명	날짜
새 장치 IQM Garnet 및 지역 Europe North 1	IQM Garnet 장치에 대한 지원이 추가되었습니다. 정사각형 격자 토폴로지를 사용하는 20 큐비트 장치. Braket 지원 지역을 유럽 북부 1 (스톡홀름) 으로 확장했습니다.	2024년 5월 22일
로컬 디튜닝 출시	실험 기능에는 이제 Aquila QuEra QPU의 로컬 디튜닝 기능이 포함됩니다.	2024년 4월 11일
노트북 비활성 관리자 출시	노트북 인스턴스를 생성할 때 비활성 관리자를 활성화하고 유휴 시간을 설정하여 Braket 노트북 인스턴스를 자동으로 재설정합니다.	2024년 3월 27일
목차 재작업	Amazon Braket 목차를 재구성하여 AWS 스타일 가이드 요구 사항을 준수하고 고객 경험을 위한 콘텐츠 흐름을 개선했습니다.	2023년 12월 12일
브라켓 다이렉트 릴리스	다음을 포함하여 Braket 다이렉트 기능에 대한 지원이 추가되었습니다.	2023년 11월 27일

	• 예약 • 전문가 조언 • 실험적 기능	
아마존 브라켓 노트북 인스턴스 생성 업데이트됨	Amazon Braket 신규 및 기존 Amazon Braket 고객을 위한 노트북 인스턴스를 생성하기 위한 정보를 추가하도록 설명서를 업데이트했습니다.	2023년 11월 27일
자체 컨테이너 가져오기 (BYOC) 업데이트됨	BYOC 사용 시기, BYOC 레시피, 컨테이너에서의 Braket Hybrid Jobs 실행에 대한 정보를 추가하도록 설명서를 업데이트했습니다.	2023년 10월 18일
하이브리드 잡 데코레이터 출시	로컬 코드를 하이브리드 작업으로 실행하십시오.페이지가 추가되었습니다. 예제 포함: • 로컬 Python 코드에서 하이브리드 작업 생성 • 추가 Python 패키지 및 소스 코드 설치 • 하이브리드 작업 인스턴스에 데이터를 저장하고 로드하세요. • 하이브리드 작업 데코레이터를 위한 모범 사례	2023년 10월 16일

대기열 가시성 추가	개발자 안내서 설명서가 queue depth 및 을 포함하도록 queue position 업데이트되었습니다. 대기열 가시성을 위한 새로운 API 변경 사항을 반영하도록 API 설명서를 업데이트했습니다.	2023년 9월 25일
설명서에서 이름을 표준화하세요.	“작업”을 “하이브리드 작업”으로, “작업”을 “양자 작업”으로 변경하도록 설명서를 업데이트했습니다.	2023년 9월 11일
새 디바이스 IonQ Aria 2	IonQ Aria 2장치에 대한 지원이 추가되었습니다.	2023년 9월 8일
네이티브 게이트 업데이트	Rigetti의 네이티브 게이트에 프로그래밍 방식으로 액세스하는 방법에 대한 정보를 추가하도록 설명서를 업데이트했습니다.	2023년 8월 16일
Xanadu출발	설명서를 업데이트하여 모든 Xanadu 장치를 제거했습니다.	2023년 6월 2일
새 장치 IonQ Aria	IonQ Aria장치에 대한 지원이 추가되었습니다.	2023년 5월 16일
사용 중지된 디바이스 Rigetti	에 대한 지원 중단 Rigetti Aspen-M-2	2023년 5월 2일

업데이트된 AmazonBraketFullAccess정책 정보	서비스 할당량: GetServiceQuota 및 cloudwatch: GetMetricData 작업과 할당량 관련 제한에 대한 정보를 포함하도록 AmazonBraketFullAccess정책 내용을 정의하는 스크립트를 업데이트했습니다.	2023년 4월 19일
가이드 저니 출시	Braket 온보딩을 위한 최신 버전과 간소화된 방법을 반영하도록 설명서를 변경했습니다.	2023년 4월 5일
새 기기 Rigetti Aspen-M-3	Rigetti Aspen-M-3장치에 대한 지원이 추가되었습니다.	2023년 1월 17일
새로운 인접 그래디언트 기능	에서 제공하는 인접 그래디언트 기능에 대한 정보가 추가되었습니다. SV1	2022년 12월 7일
새 알고리즘 라이브러리 기능	사전 구축된 양자 알고리즘 카탈로그를 제공하는 Braket 알고리즘 라이브러리에 대한 정보가 추가되었습니다.	2022년 11월 28일
D-Wave출발	모든 D-Wave 장치를 제거할 수 있도록 설명서가 업데이트되었습니다.	2022년 11월 17일
새 장치 QuEra Aquila	QuEra Aquila장치에 대한 지원이 추가되었습니다.	2022년 10월 31일
브래킷 펄스 지원	및 장치에서 펄스 제어를 사용할 수 있는 브라켓 펄스에 대한 지원이 추가되었습니다. Rigetti OQC	2022년 10월 20일

IonQ 네이티브 게이트 지원	IonQ 디바이스에서 제공하는 네이티브 게이트 세트에 대한 지원이 추가되었습니다.	2022년 9월 13일
새 인스턴스 할당량	하이브리드 작업과 관련된 기본 클래식 컴퓨팅 인스턴스 할당량을 업데이트했습니다.	2022년 8월 22일
새 서비스 대시보드	서비스 대시보드를 포함하도록 콘솔 스크린샷을 업데이트했습니다.	2022년 8월 17일
새 기기 Rigetti Aspen-M-2	Rigetti Aspen-M-2장치에 대한 지원이 추가되었습니다.	2022년 8월 12일
새로운 OpenQASM 기능	로컬 시뮬레이터 (braket_sv 및 braket_dm) 에 대한 OpenQASM 기능 지원 추가	2022년 8월 4일
새로운 비용 추적 절차	시뮬레이터 및 하드웨어 워크로드의 최대 비용 추정치를 거의 실시간으로 산출하는 방법이 추가되었습니다.	2022년 7월 18일
새 디바이스 Xanadu Borealis	Xanadu Borealis장치에 대한 지원이 추가되었습니다.	2022년 6월 2일
새로운 온보딩 간소화 절차	새롭고 간소화된 온보딩 절차의 작동 방식에 대한 정보가 추가되었습니다.	2022년 5월 16일
새 장치 D-Wave Advantage_system6.1	D-WaveAdvantage_system6.1 장치에 대한 지원이 추가되었습니다.	2022년 5월 12일

임베디드 시뮬레이터 지원	하이브리드 작업으로 임베디드 시뮬레이션을 실행하는 방법과 라이트닝 시뮬레이터를 사용하는 방법이 추가되었습니다. PennyLane	2022년 5월 4일
AmazonBraketFullAccess - 아마존 브라켓에 대한 전체 액세스 정책	사용자가 Amazon Braket용으로 생성 및 사용된 버킷을 보고 검사할 수 있는 s3: ListAllMy Buckets 권한이 추가되었습니다.	2022년 3월 31일
OpenQASM에 대한 지원	게이트 기반 양자 장치 및 시뮬레이터에 대한 OpenQASM 3.0 지원 추가	2022년 3월 7일
새로운 퀀텀 하드웨어 제공업체 Oxford Quantum Circuits 및 새로운 지역인 eu-west-2	OQC 및 eu-west-2에 대한 지원이 추가되었습니다	2022년 2월 28일
새 디바이스 Rigetti	Rigetti Aspen M-1에 대한 지원 추가	2022년 2월 15일
새 리소스 제한	최대 동시 SV1 작업 DM1 및 작업 수를 55개에서 100개로 늘렸습니다.	2022년 1월 5일
새 장치 Rigetti	Rigetti Aspen-11에 대한 지원 추가	2021년 12월 20일
사용 중지된 디바이스 Rigetti	장치 지원 중단 Rigetti Aspen-10	2021년 12월 20일
새 결과 유형	로컬 밀도 매트릭스 시뮬레이터 및 DM1 장치에서 지원되는 저밀도 매트릭스 결과 유형	2021년 12월 20일

업데이트된 정책 설명	Amazon Braket은 경로를 포함하도록 역할 ARN을 업데이트했습니다. `servicerole/` 정책 업데이트에 대한 자세한 내용은 AWS 관리형 정책에 대한 Amazon Braket 업데이트 표를 참조하십시오.	2021년 11월 29일
아마존 브라켓 잡스	Amazon Braket 하이브리드 작업 사용 설명서 및 추가됨 API	2021년 11월 29일
새 디바이스 Rigetti	Rigetti Aspen-10에 대한 지원 추가	2021년 11월 20일
사용 중지된 디바이스 D-Wave	QPU에 대한 D-Wave 지원이 중단되었습니다. Advantage_system1	2021년 11월 4일
새 디바이스 D-Wave	추가 D-Wave QPU에 대한 지원이 추가되었습니다. Advantage_system4	2021년 10월 5일
새로운 노이즈 시뮬레이터	최대 17개의 회로를 시뮬레이션할 수 있는 밀도 매트릭스 시뮬레이터 (DM1) qubits 및 로컬 노이즈 시뮬레이터 braket_dm에 대한 지원이 추가되었습니다.	2021년 5월 25일
PennyLane 지원	아마존 PennyLane 브라켓에 대한 지원이 추가되었습니다	2020년 12월 8일
새 시뮬레이터	더 큰 회로를 허용하는 텐서 네트워크 시뮬레이터 (TN1) 에 대한 지원이 추가되었습니다.	2020년 12월 8일
태스크 배칭	브라켓은 고객 작업 배칭을 지원합니다.	2020년 11월 24일

수동 할당 qubit	브라켓은 기기의 수동 qubit 할당을 Rigetti 지원합니다.	2020년 11월 24일
조정 가능한 할당량	브라켓은 작업 리소스에 대한 셀프 서비스 조정 가능한 할당량을 지원합니다.	2020년 10월 30일
에 대한 지원 PrivateLink	Braket 작업을 위한 프라이빗 VPC 엔드포인트를 설정할 수 있습니다.	2020년 10월 30일
태그 지원	Braket은 양자 작업 리소스에 대한 API 기반 태그를 지원합니다.	2020년 10월 30일
D-Wave새 장치	추가 D-Wave QPU에 대한 지원이 추가되었습니다. Advantage_system1	2020년 9월 29일
최초 릴리스	아마존 브라켓 설명서의 최초 릴리스	2020년 8월 12일

AWS 용어집

최신 AWS 용어는 용어집 참조의 AWSAWS용어집을 참조하십시오.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.