



开发人员指南

# Amazon Braket



# Amazon Braket: 开发人员指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

# Table of Contents

什么是 Amazon Braket ? .....	1
亚马逊 Braket 术语和概念 .....	3
AWS Amazon Braket 的术语和小贴士 .....	6
定价 .....	6
近乎实时的成本跟踪 .....	7
节省成本的最佳实践 .....	8
工作方式 .....	10
亚马逊 Braket 量子任务流 .....	10
第三方数据处理 .....	11
Braket 的核心存储库和插件 .....	11
核心存储库 .....	11
插件 .....	12
支持的设备 .....	12
IonQ .....	16
IQM .....	17
Rigetti .....	17
Oxford Quantum Circuits (OQC) .....	18
QuEra .....	18
局部状态向量模拟器 (braket_sv) .....	19
局部密度矩阵模拟器 (braket_dm) .....	19
本地 AHS 模拟器 (braket_ahs) .....	20
状态向量模拟器 (SV1) .....	20
密度矩阵模拟器 (DM1) .....	21
张量网络模拟器 () TN1 .....	22
嵌入式模拟器 .....	23
比较模拟器 .....	23
区域和端点 .....	26
我的量子任务什么时候能运行? .....	27
通过电子邮件或短信发送状态变更通知 .....	27
QPU 可用性窗口和状态 .....	28
队列可见性 .....	28
开始使用 .....	31
启用 Amazon Braket .....	31
先决条件 .....	31

启用 Amazon Braket 的步骤 .....	32
创建 Amazon Braket 笔记本实例 .....	32
使用 Amazon Braket Python SDK 运行你的第一个赛道 .....	34
运行你的第一个量子算法 .....	38
与 Amazon Braket 合作 .....	40
你好 AHS : 运行你的第一个模拟哈密顿仿真 .....	41
啊 .....	41
互动旋转链 .....	41
安排 .....	42
交互 .....	44
驾驶场 .....	45
AHS 计划 .....	47
在本地模拟器上运行 .....	47
分析模拟器结果 .....	48
Run QuEra ning on 的 Aquila QPU .....	51
分析 QPU 结果 .....	52
下一步 .....	53
在 SDK 中构造电路 .....	54
大门和电路 .....	54
部分测量 .....	60
手动qubit分配 .....	61
逐字汇编 .....	61
噪声模拟 .....	62
检查电路 .....	63
结果类型 .....	65
向 QPU 和模拟器提交量子任务 .....	70
Amazon Braket 上的量子任务示例 .....	71
向 QPU 提交量子任务 .....	76
使用本地仿真器运行量子任务 .....	78
量子任务批处理 .....	80
设置 SNS 通知 ( 可选 ) .....	82
检查编译后的电路 .....	82
使用 OpenQasm 3.0 运行你的赛道 .....	82
什么是 OpenQasm 3.0 ? .....	83
何时使用 OpenQasm 3.0 .....	84
OpenQasm 3.0 的工作原理 .....	84

先决条件 .....	84
Braket 支持哪些 OpenQasm 功能？ .....	84
创建并提交示例 OpenQasm 3.0 量子任务 .....	90
在不同的 Braket 设备上支持 OpenQasm .....	93
使用 OpenQasm 3.0 模拟噪声 .....	104
Qubit使用 OpenQasm 3.0 重新布线 .....	105
使用 OpenQasm 3.0 进行逐字编译 .....	106
Braket 控制台 .....	106
更多资源 .....	107
使用 OpenQasm 3.0 计算梯度 .....	107
使用 OpenQasm 3.0 测量特定的量子比特 .....	107
使用 QuEra's Aquila 提交模拟节目 .....	108
Hamiltonian .....	108
支架 AHS 程序架构 .....	110
Braket AHS 任务结果架构 .....	114
QuEra 设备属性架构 .....	120
使用 Boto3 .....	125
打开 Amazon Braket Boto3 客户端 .....	125
AWS CLI 配置 Boto3 和 Amazon Braket SDK 的配置文件 .....	129
Amazon Braket 上的脉冲控制 .....	131
支架脉冲 .....	131
帧 .....	131
端口 .....	131
波形 .....	132
帧和端口的作用 .....	133
Rigetti .....	133
OQC .....	134
你好脉冲 .....	135
你好 Pulse 使用 OpenPulse .....	139
使用脉冲访问原生门 .....	145
亚马逊 Braket 混合职位 .....	148
什么是混合 Job？ .....	149
何时使用 Amazon Braket 混合工作 .....	149
将本地代码作为混合作业运行 .....	149
使用本地 Python 代码创建混合作业 .....	150
安装其他 Python 软件包和源代码 .....	153

将数据保存并加载到混合作业实例中 .....	153
混合工作装饰者的最佳实践 .....	8
使用 Amazon Braket 混合作业运行混合作业 .....	157
创建你的第一个 Hybrid Job .....	158
设置权限 .....	159
创建并运行 .....	162
监控结果 .....	165
输入、输出、环境变量和辅助函数 .....	167
输入 .....	167
输出 .....	168
环境变量 .....	169
辅助函数 .....	170
保存作业结果 .....	170
使用检查点保存并重启混合作业 .....	172
为算法脚本定义环境 .....	173
使用超参数 .....	175
配置混合作业实例以运行算法脚本 .....	177
取消混合 Job .....	180
使用参数化编译加快混合作业的速度 .....	182
PennyLane 与 Amazon Braket 一起使用 .....	183
带有 Amazon Braket PennyLane .....	183
Amazon Braket 中的混合算法示例笔记本 .....	185
带有嵌入式 PennyLane 仿真器的混合算法 .....	185
PennyLane 使用 Amazon Braket 模拟器开启伴随渐变 .....	186
使用 Amazon Braket 混合任务并 PennyLane 运行 QAOA 算法 .....	186
使用来自的嵌入式模拟器加速混合工作负载 PennyLane .....	189
lightning.gpu 用于量子近似优化算法工作负载 .....	190
量子机器学习和数据并行性 .....	192
使用本地模式构建和调试混合作业 .....	196
自带集装箱 (BYOC) .....	197
什么时候自带集装箱才是正确的决定？ .....	197
自带集装箱的食谱 .....	198
在自己的容器中运行 Braket 混合作业 .....	203
在中配置默认存储桶 AwsSession .....	204
使用直接与混合作业互动 API .....	204
错误缓解 .....	208

错误缓解功能已开启 IonQ Aria .....	208
锐化 .....	209
Braket Di .....	210
预留 .....	210
创建预订 .....	211
通过预留来运行您的工作负载 .....	212
取消或重新安排现有预订 .....	215
专家建议 .....	216
实验能力 .....	216
仅限预约访问 ionQ Forte .....	217
在 Aquila 上 QuEra 访问本地停机功能 .....	217
在 Aquila 上 QuEra 访问高大的几何形状 .....	218
在 Aquila 上 QuEra 可以看到紧凑的几何形状 .....	218
日志记录和监控 .....	219
通过 Amazon Braket SDK 跟踪量子任务 .....	219
通过 Amazon Braket 控制台监控量子任务 .....	222
标记资源 .....	223
使用标签 .....	224
更多关于 AWS 和标签 .....	224
Amazon Braket 中支持的资源 .....	224
标签限制 .....	225
在 Amazon Braket 中管理标签 .....	225
Amazon Braket 中的 CLI 标签示例 .....	226
使用 Amazon Braket API 进行标记 .....	227
亚马逊 Braket 活动与 EventBridge .....	227
使用监控量子任务状态 EventBridge .....	228
亚马逊 Braket 活动 EventBridge 示例 .....	229
使用监视器 CloudWatch .....	230
Amazon Braket 的指标和维度 .....	230
支持的设备 .....	231
使用登录 CloudTrail .....	231
Amazon Braket 中的信息 CloudTrail .....	231
了解 Amazon Braket 日志文件条目 .....	232
使用创建一个 Braket 笔记本 CloudFormation .....	234
步骤 1：创建 Amazon SageMaker 生命周期配置脚本 .....	235
步骤 2：创建由亚马逊担任的 IAM 角色 SageMaker .....	235

步骤 3：使用前缀创建一个 Amazon SageMaker 笔记本实例 amazon-braket-	237
高级日志	237
安全性	240
共同承担安全责任	240
数据保护	240
数据留存	241
管理对 Amazon Braket 的访问权限	241
亚马逊 Braket 资源	242
笔记本和角色	242
关于 AmazonBraketFullAccess 政策	243
关于 AmazonBraketJobsExecutionPolicy 政策	248
限制用户访问某些设备	251
Amazon Braket 更新了托管 AWS 政策	252
限制用户访问某些笔记本实例	253
限制用户对某些 S3 存储桶的访问权限	254
服务相关角色	255
亚马逊 Braket 的服务相关角色权限	255
故障恢复能力	256
合规性验证	257
基础设施安全性	257
第三方安全	258
VPC 终端节点 (PrivateLink)	258
Amazon Braket VPC 终端节点的注意事项	258
设置 Braket 然后 PrivateLink	259
有关创建终端节点的更多信息	260
使用 Amazon VPC 终端节点策略控制访问权限	260
故障排除	262
AccessDeniedException	262
调用 CreateQuantumTask 操作时出错 (ValidationException)	262
某个 SDK 功能不起作用	263
由于以下原因，混合作业失败 ServiceQuotaExceededException	263
组件在笔记本实例中停止工作	264
配额	264
其他配额和限制	289
对 OpenQASM 进行故障	289
包含语句错误	290



非连续错误 qubits .....	290
将物理错误qubits与虚拟qubits错误混为一谈 .....	290
请求结果类型并在同一个程序错误qubits中进行测量 .....	291
经典限值和qubit寄存器限值超出误差 .....	291
方框前面没有逐字编译指示错误 .....	291
逐字记录框缺少本机门错误 .....	292
逐字记录框缺少物理错误 qubits .....	292
逐字编译指示缺少“braket” 错误 .....	292
qubits无法为单曲编制索引错误 .....	292
双qubit门qubits中的物理未连接错误 .....	293
GetDevice 不返回 OpenQasm 结果错误 .....	293
本地模拟器支持警告 .....	294
API 和开发工具包参考 .....	296
文档历史记录 .....	297
AWS 术语表 .....	303
.....	ccxiv

# 什么是 Amazon Braket ?

## Tip

通过以下方式学习量子计算的基础 AWS ! 注册 [Amazon Braket 数字学习计划](#) , 完成一系列学习课程和数字评估后 , 即可获得自己的数字徽章。

Amazon Braket 是一款完全托管的软件 AWS 服务 , 可帮助研究人员、科学家和开发人员开始使用量子计算。量子计算有可能解决经典计算机无法解决的计算问题 , 因为它利用量子力学定律以新的方式处理信息。

获取量子计算硬件可能既昂贵又不方便。由于访问权限有限 , 因此很难运行算法、优化设计、评估技术的当前状态以及计划何时投入资源以获得最大收益。Braket 可以帮助您克服这些挑战。

Braket 提供对各种量子计算技术的单点接入。使用 Braket , 您可以 :

- 探索和设计量子算法和混合算法。
- 在不同的量子电路模拟器上测试算法。
- 在不同类型的量子计算机上运行算法。
- 创建概念验证应用程序。

定义量子问题并对量子计算机进行编程来解决这些问题需要一套新的技能。为了帮助您获得这些技能 , Braket 提供了不同的环境来模拟和运行您的量子算法。您可以找到最适合您要求的方法 , 并通过一组名为 notebook 的示例环境快速入门。

Braket 开发分为三个阶段 : 构建、测试和运行 :

**Build -** Braket 提供完全托管的 Jupyter 笔记本环境 , 便于入门。Braket 笔记本预装了示例算法、资源和开发者工具 , 包括 Amazon Braket SDK。使用 Amazon Braket SDK , 您可以构建量子算法 , 然后通过更改一行代码在不同的量子计算机和模拟器上对其进行测试和运行。

**Test -** Braket 提供对完全托管的高性能量子电路模拟器的访问。您可以测试和验证您的电路。Braket 处理所有底层软件组件和亚马逊弹性计算云 (Amazon EC2) 集群 , 从而减轻了在传统高性能计算 (HPC) 基础设施上模拟量子电路的负担。

Run- Braket 提供对不同类型的量子计算机的安全按需访问。您可以从、和 Rigetti 访问基于门的量子计算机 IonQ/QC，也可以访问来自的模拟哈密顿仿真器。QuEra 您也没有预先承诺，也无需通过个别提供商购买访问权限。

## 关于量子计算和 Braket

量子计算正处于早期发展阶段。重要的是要明白，目前不存在通用、容错的量子计算机。因此，某些类型的量子硬件更适合每种用例，因此访问各种计算硬件至关重要。Braket 通过第三方提供商提供各种硬件。

现有的量子硬件由于噪声而受到限制，噪声会带来错误。该行业正处于 Noisy 中级量子量子 (NISQ) 时代。在 NISQ 时代，量子计算设备噪音太大，无法维持纯量子算法，例如肖尔算法或格罗弗算法。在获得更好的量子误差校正之前，最实用的量子计算需要将经典（传统）计算资源与量子计算机相结合来创建混合算法。Braket 可帮助您使用混合量子算法。

在混合量子算法中，量子处理单元 (QPU) 被用作 CPU 的协处理器，从而加快了经典算法中的特定计算。这些算法利用迭代处理，在这种处理中，计算在经典计算机和量子计算机之间移动。例如，量子计算在化学、优化和机器学习中的当前应用基于变分量子算法，变分量子算法是一种混合量子算法。在变分量子算法中，经典优化例程迭代调整参数化量子电路的参数，就像根据机器学习训练集中的误差迭代调整神经网络权重的方式大致相同。Braket 提供对 PennyLane 开源软件库的访问权限，该库可帮助您使用变分量子算法。

量子计算在四个主要领域的计算中越来越受欢迎：

- 数论 — 包括因式分解和密码学（例如，肖尔算法是数论计算的主要量子方法）
- 优化 — 包括约束满足度、求解线性系统和机器学习
- Oracular 计算 — 包括搜索、隐藏子组和排序查找（例如，Grover 算法是预言机计算的主要量子方法）
- 仿真 — 包括直接模拟、节点不变量和量子近似优化算法 (QAOA) 应用

仅举几例，这些计算类别的应用可以在金融服务、生物技术、制造业和制药业中找到。Braket 提供了功能和示例笔记本，除了某些实际问题外，这些功能和示例笔记本电脑已经可以应用于许多概念验证问题。

# 亚马逊 Braket 术语和概念

## Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

Braket 中使用了以下术语和概念：

## 模拟哈密顿仿真

模拟哈密顿模拟 (AHS) 是一种独特的量子计算范式，用于直接仿真多体系统的瞬态量子动力学。在 AHS 中，用户直接指定一个随时间变化的哈密顿量子，量子计算机的调整方式使其可以直接模拟该哈密顿量子下的连续时间演变。AHS 设备通常是特殊用途的设备，而不是像基于门的设备这样的通用量子计算机。它们仅限于他们可以模拟的一类哈密顿人。但是，由于这些 Hamiltonians 是在设备上自然实现的，因此 AHS 不会承受将算法制定为电路和实现门操作所需的开销。

## 支架

我们以 bra -ket 表示法 ([量子力学中的标准表示法](#)) 命名了 Braket 服务。它由保罗·狄拉克于 1939 年推出，用于描述量子系统的状态，它也被称为狄拉克表示法。

## Braket 混合作业

Amazon Braket 有一个名为 Amazon Braket Hybrid Jobs 的功能，它提供混合算法的完全托管执行。Braket 混合作业由三个部分组成：

1. 算法的定义，可以作为脚本、Python 模块或 Docker 容器提供。
2. 基于 Amazon EC2 的混合任务实例，用于运行您的算法。默认值为 ml.m5.xlarge 实例。
3. 用于运行作为算法一部分的量子任务的量子设备。单个混合作业通常包含许多量子任务的集合。

## 设备

在 Amazon Braket 中，设备是可以运行量子任务的后端。设备可以是 QPU 或量子电路模拟器。要了解更多信息，请参阅[支持 Amazon Braket](#) 的设备。

## 基于门的量子计算

在基于门的量子计算 (QC) (也称为基于电路的 QC) 中，计算被分解为基本运算 (门)。某些门集是通用的，这意味着每次计算都可以表示为这些门的有限序列。门是量子电路的基石，类似于经典数字电路的逻辑门。

## Hamiltonian

物理系统的量子动力学由其哈密顿量子决定，哈密顿量子对有关系统各组成部分之间的相互作用和外生驱动力的影响的所有信息进行编码。在经典机器上， $N$  量子比特系统的哈密顿通常表示为复数的  $2^{N \times 2^N}$  矩阵。通过在量子器件上运行模拟哈密顿仿真，可以避免这些指数级的资源需求。

## 脉冲

脉冲是传输到量子比特的瞬态物理信号。它由在帧中播放的波形来描述，该波形充当载波信号的支撑，并绑定到硬件通道或端口。客户可以通过提供调制高频正弦载波信号的模拟包络来设计自己的脉冲。该帧的独特描述是频率和相位，这些频率和相位通常被选为与量子比特的  $|0\rangle$ - 和  $|1\rangle$ - 能级之间的能量分离产生共振。因此，门被设置为具有预定形状和校准参数（例如振幅、频率和持续时间）的脉冲。模板波形未涵盖的用例将通过自定义波形启用，自定义波形将通过提供由固定的物理周期时间分隔的列表以单样本分辨率进行指定。

## 量子电路

量子电路是在基于门的量子计算机上定义计算的指令集。量子电路是一系列量子门，它们是 qubit 寄存器上的可逆变换，还有测量指令。

## 量子电路模拟器

量子电路模拟器是一种在经典计算机上运行并计算量子电路测量结果的计算机程序。对于一般电路，量子仿真的资源需求会随着 qubits 要仿真的数量而呈指数级增长。Braket 提供对托管（通过 Braket 访问 API）和本地（Braket SDK 的一部分）量子 Amazon 电路模拟器的访问权限。

## 量子计算机

量子计算机是一种使用量子力学现象（例如叠加和纠缠）进行计算的物理设备。量子计算 (QC) 有不同的范式，例如基于门的 QC。

## 量子处理单元 (QPU)

QPU 是一种可以在量子任务上运行的物理量子计算设备。QPU 可以基于不同的 QC 范式，例如基于门的 QC。要了解更多信息，请参阅 [支持 Amazon Braket](#) 的设备。

## QPU 原生大门

QPU 原生门可以直接映射到 QPU 控制系统的控制脉冲。无需进一步编译即可在 QPU 设备上运行原生门。QPU 支持的门的子集。您可以在 Braket Amazon t 控制台的“设备”页面和 Braket SDK 中找到设备的原生门。

## 支持 QPU 的大门

QPU 支持的门是 QPU 设备接受的大门。这些门可能无法直接在 QPU 上运行，这意味着它们可能需要分解成原生门。您可以在 Amazon Braket 控制台的“设备”页面和 Braket SDK 中找到设备支持的门。

## 量子任务

在 Braket 中，量子任务是对设备的原子请求。对于基于门的质量控制设备，这包括量子电路（包括测量指令和数量 shots）和其他请求元数据。您可以通过 Amazon Braket SDK 或直接使用该 `CreateQuantumTaskAPI` 操作来创建量子任务。创建量子任务后，它将排队直到请求的设备变为可用为止。您可以在 Amazon Braket 控制台的 Quantum Tasks 页面上或使用 `GetQuantumTask` 或 `SearchQuantumTasksAPI` 操作来查看您的量子任务。

## Qubit

量子计算机中的基本信息单位被称为 qubit（量子比特），就像经典计算中的位一样。A qubit 是一个两能级量子系统，可以通过不同的物理实现来实现，例如超导电路或单个离子和原子。其他 qubit 类型基于光子、电子或核自旋或更奇特的量子系统。

## Queue depth

Queue depth 指排队等候特定设备的量子任务和混合作业的数量。可通过 Braket Software Development Kit (SDK) 或访问设备的量子任务和混合作业队列数 Amazon Braket Management Console。

1. 任务队列深度是指当前等待以正常优先级运行的量子任务总数。
2. 优先任务队列深度是指等待运行的已提交量子任务的总数 Amazon Braket Hybrid Jobs。混合作业启动后，这些任务优先于独立任务。
3. 混合作业队列深度是指设备上当前排队的混合作业总数。Quantum tasks 作为混合作业的一部分提交具有优先级，并汇总在 Priority Task Queue。

## Queue position

Queue position 指您的量子任务或混合任务在相应设备队列中的当前位置。它可以通过或获得，用于量子任务或混合作业 Amazon Braket Management Console。Braket Software Development Kit (SDK)

## Shots

由于量子计算本质上是概率性的，因此任何电路都需要多次评估才能得到准确的结果。单个电路的执行和测量称为镜头。电路的射击次数（重复执行）是根据所需的结果精度来选择的。

# AWS Amazon Braket 的术语和小贴士

## IAM 政策

IAM 策略是允许或拒绝对 AWS 服务和资源的权限的文档。IAM 策略允许您自定义用户对资源的访问级别。例如，您可以允许用户访问您中的所有 Amazon S3 存储桶 AWS 账户，或者仅允许用户访问特定存储桶。

- **最佳实践：**授予权限时遵循最低权限的安全原则。通过遵循这一原则，您可以帮助防止用户或角色拥有的权限超过执行其量子任务所需的权限。例如，如果员工只需要访问特定存储桶，请在 IAM 策略中指定该存储桶，而不是向员工授予访问您 AWS 账户中所有存储桶的权限。

## IAM 角色

IAM 角色是您可以假设的身份，以获得临时权限访问权限。在用户、应用程序或服务担任 IAM 角色之前，必须向他们授予切换到该角色的权限。当有人担任 IAM 角色时，他们会放弃以前在先前角色下拥有的所有权限，并使用新角色的权限。

- **最佳实践：** IAM 角色非常适合需要临时而不是长期授予服务或资源访问权限的情况。

## 亚马逊 S3 存储桶

亚马逊简单存储服务 (Amazon S3) Simple Storage Service 允许您将数据作为对象存储在存储桶中。Amazon S3 存储桶提供无限的存储空间。Amazon S3 存储桶中对象的最大大小为 5 TB。您可以将任何类型的文件数据上传到 Amazon S3 存储桶，例如图像、视频、文本文件、备份文件、网站媒体文件、存档文档以及您的 Braket 量子任务结果。

- **最佳实践：**您可以设置权限以控制对 S3 存储桶的访问权限。有关更多信息，请参阅 Amazon S3 文档中的[存储桶策略和用户策略](#)。

## 亚马逊 Braket 定价

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

借助 Amazon Braket，您可以按需访问量子计算资源，无需预先承诺。您仅需按实际用量付费。要了解有关定价的更多信息，请访问我们的[定价页面](#)。

## 近乎实时的成本跟踪

Braket SDK 为您提供了向量子工作负载添加近乎实时的成本跟踪的选项。我们的每个示例笔记本都包含成本跟踪代码，可为您提供有关Braket量子处理单元 (QPU) 和按需模拟器的最大成本估算。最高成本估算值将以美元显示，不包括任何积分或折扣。

### Note

显示的费用是根据您的 Amazon Braket 模拟器和量子处理单元 (QPU) 任务使用情况估算的费用。显示的预计费用可能与您的实际费用有所不同。预计费用不计入任何折扣或积分，您可能会因使用其他服务（例如亚马逊弹性计算云 (Amazon EC2)）而产生额外费用。

### SV1 的成本跟踪

为了演示如何使用成本跟踪功能，我们将构建一个 Bell State 电路并在我们的 SV1 仿真器上运行它。首先导入 Braket SDK 模块，定义贝尔状态并将Tracker()函数添加到我们的电路中：

```
#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)
```

当你运行 Notebook 时，你可以期待 Bell State 模拟的以下输出。跟踪器功能将显示发送的镜头数量、已完成的量子任务、执行时长、计费的执行持续时间以及以美元为单位的最大成本。每次模拟的执行时间可能会有所不同。

```
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
```

## 使用成本跟踪器设置最高成本

您可以使用成本跟踪器来设置计划的最高成本。对于要在给定项目上花费的金额，您可能有一个最大门槛。通过这种方式，您可以使用成本跟踪器在执行代码中构建成本控制逻辑。以下示例在 Rigetti QPU 上采用相同的电路，并将成本限制为 1 美元。在我们的代码中运行一次电路迭代的成本为 0.37 美元。我们已将逻辑设置为重复迭代，直到总成本超过 1 美元；因此，代码片段将运行三次，直到下一次迭代超过 1 美元。通常，程序会继续迭代，直到达到所需的最大成本，在本例中为三次迭代。

```
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

成本跟踪器不会跟踪失败的 TN1 量子任务的持续时间。在 TN1 模拟过程中，如果您的排练已完成，但收缩步骤失败，则您的排练费用将不会显示在成本跟踪器中。

## 节省成本的最佳实践

请考虑以下使用 Amazon Braket 的最佳实践。节省时间，最大限度地降低成本，并避免常见错误。

### 使用模拟器进行验证

- 在 QPU 上运行仿真器之前，请使用仿真器验证电路，这样您就可以微调电路，而不会因使用 QPU 而产生费用。

- 尽管在仿真器上运行电路的结果可能与在 QPU 上运行电路的结果不同，但您可以使用仿真器识别编码错误或配置问题。

### 限制用户访问某些设备

- 您可以设置限制，防止未经授权的用户在某些设备上提交量子任务。限制访问的推荐方法是使用 AWS IAM。有关如何执行此操作的更多信息，请参阅[限制访问](#)。
- 我们建议您不要使用管理员账户来授予或限制用户访问 Amazon Braket 设备。

### 设置账单警报

- 您可以设置账单警报，以便在账单达到预设限额时通知您。设置闹钟的推荐方法是通过 AWS Budgets。您可以设置自定义预算，并在费用或使用量可能超过预算金额时收到提醒。有关信息，请访问[AWS Budgets](#)。

### 使用低射击计数测试TN1量子任务

- 模拟器的成本低于 qHP，但是如果量子任务的射击次数很高，则某些模拟器可能会很昂贵。我们建议您使用低shot计数来测试您的TN1任务。Shotcount 不影响本地模拟器任务的成本。SV1

### 检查所有区域的量子任务

- 控制台仅显示您当前的量子任务 AWS 区域。在查找已提交的计费量子任务时，请务必查看所有区域。
- 您可以在[支持的](#)设备文档页面上查看设备及其相关区域的列表。

# 亚马逊 Braket 是如何运作的

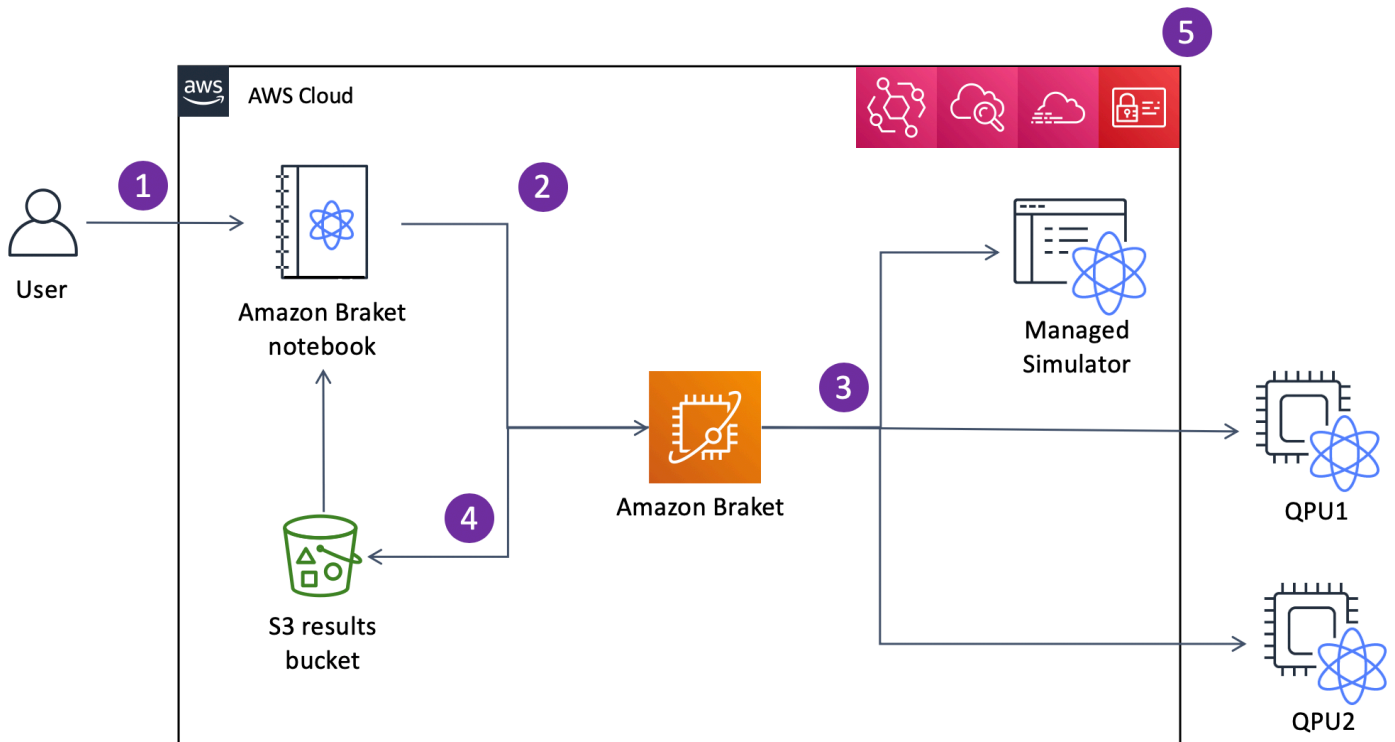
## Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

Amazon Braket 提供对量子计算设备的按需访问，包括按需电路模拟器和不同类型的 QPU。在 Amazon Braket 中，对设备的原子请求是一项量子任务。对于基于门的质量控制设备，此请求包括量子电路（包括测量指令和拍摄次数）和其他请求元数据。对于模拟哈密顿模拟器来说，量子任务包含量子寄存器的物理布局以及操纵场的时间和空间依赖性。

在本节中，我们将学习在 Amazon Braket 上运行量子任务的高级流程。

## 亚马逊 Braket 量子任务流



使用 Jupyter 笔记本电脑，您可以通过 Amazon Braket 控制台或使用 Amazon Braket SDK 方便地定义、提交和监控您的量子任务。您可以直接在 SDK 中构建量子电路。但是，对于模拟哈密顿仿真器，

您可以定义寄存器布局和控制字段。定义量子任务后，您可以选择一台设备来运行该任务，然后将其提交到 Amazon Braket API (2)。根据您选择的设备，量子任务将排队直到设备可用并将任务发送到 QPU 或模拟器进行实施 (3)。Amazon Braket 允许您访问不同类型的 QPU ( IonQ、Oxford Quantum Circuits (OQC)、Rigetti ) QuEra、三个按需模拟器 ( SV1、TN1 ) DM1、两个本地模拟器和一个嵌入式模拟器。要了解更多信息，请参阅[支持 Amazon Braket 的设备](#)。

处理完您的量子任务后，Amazon Braket 会将结果返回到 Amazon S3 存储桶，数据存储到您的 AWS 账户 (4) 中。同时，SDK 会在后台轮询结果，并在量子任务完成时将其加载到 Jupyter 笔记本中。您还可以在 Amazon Braket 控制台的 Quantum Tasks 页面上或使用 Braket 的 GetQuantumTask 操作来查看和管理您的量子任务。Amazon API

Amazon Braket 与 AWS Identity and Access Management (IAM)、Amazon CloudWatch on AWS CloudTrail 和 Amazon 集成，Amazon EventBridge 用于用户访问管理、监控和记录以及基于事件的处理 (5)。

## 第三方数据处理

提交给 QPU 设备的量子任务在位于第三方提供商运营的设施中的量子计算机上处理。要了解有关 Amazon Braket 安全和第三方处理的更多信息，请参阅 [Amazon Braket 硬件提供商的安全](#)。

## Braket 的核心存储库和插件

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

## 核心存储库

下面显示了包含用于 Braket 的密钥包的核心存储库列表：

- [Braket Python SDK](#)-使用 Braket Python SDK 在 Python 编程语言的 Jupyter 笔记本上设置你的代码。设置好 Jupyter 笔记本电脑后，您可以在 Braket 设备和模拟器上运行代码
- [Braket Schemas](#)-Braket SDK 和 Braket 服务之间的合同。
- [Braket 默认模拟器](#) ——我们所有用于 Braket 的本地量子模拟器（状态向量和密度矩阵）。

## 插件

然后是各种插件以及各种设备和编程工具。其中包括 Braket 支持的插件以及第三方支持的插件，如下所示。

亚马逊 Braket 支持：

- [Amazon Braket 算法库](#) ——用 Python 编写的预建量子算法目录。按原样运行它们，或者使用它们作为起点来构建更复杂的算法。
- [Braket-PennyLane 插件](#)-用 PennyLane 作 Braket 上的 QML 框架。

第三方（Braket 团队监控并做出贡献）：

- [Qiskit-Braket 提供商](#)-使用 Qiskit SDK 访问 Braket 资源。
- [Braket-Julia SDK](#)-（实验性）Braket SDK 的 Julia 原生版本

## 支持 Amazon Braket 的设备

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

在 Amazon Braket 中，设备代表一个 QPU 或模拟器，你可以调用它来运行量子任务。Amazon Braket 允许从 IonQ、和三个按需模拟器 IQM Oxford Quantum Circuits QuEra Rigetti、三个本地模拟器和一个嵌入式模拟器访问 QPU 设备。对于所有设备，您可以在 Amazon Braket 控制台的“设备”选项卡上或通过 API 找到更多设备属性，例如设备拓扑、校准数据和原生门禁设置。GetDevice 使用模拟器构建电路时，Amazon Braket 目前要求您使用连续的量子比特或索引。如果您使用的是 Amazon Braket SDK，则可以访问设备属性，如以下代码示例所示。

```
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
```

## 支持的量子硬件提供商

- [IonQ](#)
- [IQM](#)
- [Oxford Quantum Circuits \(OQC\)](#)
- [QuEra Computing](#)
- [Rigetti](#)

## 支持的模拟器

- [局部状态向量模拟器 \(braket\\_sv\) \( '默认模拟器' \)](#)
- [局部密度矩阵模拟器 \(braket\\_dm\)](#)
- [本地 AHS 模拟器](#)
- [状态向量模拟器 \(SV1\)](#)
- [密度矩阵模拟器 \(DM1\)](#)
- [张量网络模拟器 \(\) TN1](#)
- [PennyLane的闪电模拟器](#)

为您的量子任务选择最适合的模拟器

- [比较模拟器](#)

### Note

要查看每台 AWS 区域 设备的可用信息，请在下表中向右滚动。

## 亚马逊 Braket 设备

提供商	设备名称	范式	类型	设备 ARN	区域
IonQ	Aria 1	基于门	QPU	arn: aws: braket: us-east-1:: device/qpu/ionq/aria-1	us-east-1
IonQ	Aria 2	基于门	QPU	arn: aws: braket: us-east-1:: device/qpu/ionq/aria-2	us-east-1
IonQ	Forte 1	基于门	QPU ( 仅限预订 )	arn: aws: braket: us-east-1:: device/qpu/ionq/Forte-1	us-east-1

提供商	设备名称	范式	类型	设备 ARN	区域
IonQ	Harmony	基于门	QPU	arn: aws: braket: us-east-1:: device/qpu/ionq/Harmony	us-east-1
IQM	Garnet	基于门	QPU	arn: aws: braket: eu-north-1:: device/qpu/iqm/Garnet	eu-north-1
Oxford Quantum Circuits	Lucy	基于门	QPU	arn: aws: braket: eu-west-2:: device/qpu/oqc/Lucy	eu-west-2
QuEra	Aquila	模拟哈密顿仿真	QPU	arn: aws: braket: us-east-1:: device/quera/Aquila	us-east-1
Rigetti	Aspen M-3	基于门	QPU	arn: aws: braket: us-west-1:: device/qpu/rigetti/aspen-M-3	us-west-1
AWS	braket_sv	基于门	本地模拟器	不适用 ( Braket SDK 中的本地模拟器 )	不适用
AWS	braket_dm	基于门	本地模拟器	不适用 ( Braket SDK 中的本地模拟器 )	不适用
AWS	SV1	基于门	按需模拟器	arn: aws: braket:: device/quantum-simulator/amazon/sv1	所有提供 Amazon Braket 的地区。
AWS	DM1	基于门	按需模拟器	arn: aws: braket:: device/quantum-simulator/amazon/dm1	所有提供 Amazon Braket 的地区。



提供商	设备名称	范式	类型	设备 ARN	区域
AWS	TN1	基于门	按需模拟器	arn: aws: braket:: device/quantum-simulator/amazon/tn1	us-west-2、us-east-1 和 eu-west-2

### Note

某些 QPU 只能通过 Braket Direct 使用预留进行访问，请参阅[预留](#)。

要查看有关可与 Braket 配合使用的 QPU 的更多详细信息，请参阅 [Amazon Amazon Braket 硬件](#) 提供商。

## IonQ

IonQ 提供基于离子阱技术的基于栅极的 QPU。IonQ 的捕获的离子 qPU 建立在一条捕获的 171Yb+ 离子链上，这些离子通过真空室内的微型表面电极捕集器在空间上受到限制。

IonQ 设备支持以下量子门。

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

通过逐字编译，IonQQPU 支持以下原生门。

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

如果在使用原生 MS 门时仅指定两个相位参数，则会运行一个完全纠缠的 MS 门。完全纠缠的 MS 门始终执行  $\pi/2$  旋转。要指定不同的角度并运行部分缠绕的 MS 门，您可以通过添加第三个参数来指定所需的角。有关更多信息，请参阅 [braket.circuits.gate](#) 模块。

这些原生门只能用于逐字编译。[要了解有关逐字编译的更多信息，请参阅逐字编译。](#)

## IQM

IQM量子处理器是基于超导 transmon 量子比特的通用门控模型器件。该IQM Garnet器件是一款采用方格拓扑结构的 20 量子比特器件。

这些IQM器件支持以下量子门。

```
"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"
```

通过逐字编译，这些IQM设备支持以下本机门。

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

## Rigetti

Rigetti量子处理器是基于全可调超导的通用门控模型机器。qubits这款 79 量子比特Aspen-M-3器件利用其专有的多芯片技术，由 2 个 40 量子比特处理器组装而成。

该Rigetti器件支持以下量子门。

```
'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'
```

通过逐字编译，Rigetti 设备支持以下原生门。

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

Rigetti超导量子处理器只能以  $\pm\pi/2$  或  $\pm\pi$  的角度运行“rx”门。

这些设备提供脉冲电平控制，这些Rigetti设备支持以下类型的一组预定义帧：

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

有关这些帧的更多信息，请参阅[帧和端口的作用](#)。

## Oxford Quantum Circuits (OQC)

OQC量子处理器是通用的门控模型机器，使用可扩展的 Coaxmon 技术构建。该OQC Lucy系统是一种具有环形拓扑结构的8-qubit设备，其中每个设备qubit都与其两个最近的邻居相连。

该Lucy器件支持以下量子门。

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

通过逐字编译，OQC 设备支持以下本机门。

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

设备上可进行脉冲电平控制。OQC这些OQC设备支持一组以下类型的预定义帧：

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

OQC只要您提供有效的端口标识符，设备就支持动态声明帧。有关这些帧和端口的更多信息，请参阅[帧和端口的作用](#)。

### Note

将脉冲控制与一起使用时OQC，程序的最大长度不能超过 90 微秒。单量子比特门的最大持续时间约为 50 纳秒，双量子比特门的最大持续时间约为 1 微秒。这些数字可能因使用的量子比特、器件的电流校准和电路编译而异。

## QuEra

QuEra提供基于中性原子的器件，可以运行模拟哈密顿模拟 (AHS) 量子任务。这些特殊用途的器件忠实地再现了数百个同时相互作用的量子比特的时变量子动力学。

人们可以通过规定量子比特寄存器的布局以及操纵场的时间和空间依赖性，在模拟哈密顿仿真的范式中对这些器件进行编程。Amazon Braket 提供了通过 python SDK 的 AHS 模块构建此类程序的实用工具。braket.ahs

欲了解更多信息，请参阅[模拟哈密顿仿真示例笔记本](#)或[使用的 Aquila 提交模拟程序 QuEra](#)页面。

## 局部状态向量模拟器 (braket\_sv)

本地状态向量模拟器 (braket\_sv) 是在您的环境中本地运行的 Amazon Braket SDK 的一部分。它非常适合在小型电路 (最多 25 个 qubits) 上进行快速原型设计，具体取决于您的 Braket 笔记本实例或本地环境的硬件规格。

本地模拟器支持 Amazon Braket SDK 中的所有门，但 QPU 设备支持的子集较小。您可以在设备属性中找到设备支持的门。

### Note

本地仿真器支持高级 OpenQasm 功能，QPU 设备或其他模拟器可能不支持这些功能。有关支持的功能的更多信息，请参阅 [OpenQasm 本地模拟器](#) 笔记本中提供的示例。

有关如何使用模拟器的更多信息，请参阅 [Amazon Braket](#) 示例。

## 局部密度矩阵模拟器 (braket\_dm)

本地密度矩阵模拟器 (braket\_dm) 是在您的环境中本地运行的 Amazon Braket SDK 的一部分。它非常适合在有噪音 (最多 12 个 qubits) 的小型电路上进行快速原型设计，具体取决于您的 Braket 笔记本电脑实例或本地环境的硬件规格。

您可以使用栅极噪声操作 (例如位翻转和去极化误差) 从头开始构建常见的噪声电路。您还可以将噪声运算应用于现有电路的特定 qubits 和门，这些电路旨在有噪声和无噪声的情况下运行。

给定指定数量，braket\_dm 本地模拟器可以提供以下结果 shots：

- 降低密度矩阵：Shots= 0

### Note

本地仿真器支持高级 OpenQasm 功能，QPU 设备或其他模拟器可能不支持这些功能。有关支持的功能的更多信息，请参阅 [OpenQasm 本地模拟器](#) 笔记本中提供的示例。

要了解有关局部密度矩阵模拟器的更多信息，请参阅 [Braket 噪声模拟器入门示例](#)。

## 本地 AHS 模拟器 (braket\_ahs)

本地 AHS (模拟哈密顿模拟) 模拟器 (braket\_ahs) 是在您的环境中本地运行的 Amazon Braket SDK 的一部分。它可以用来模拟 AHS 程序的结果。它非常适合在小型寄存器 (最多 10-12 个原子) 上进行原型设计, 具体取决于您的 Braket 笔记本实例或本地环境的硬件规格。

本地仿真器支持具有一个均匀驱动场、一个 (非均匀) 移位场和任意原子排列的 AHS 程序。有关详细信息, 请参阅 Braket [AHS 类](#) 和 Braket [AHS 程序架构](#)。

要了解有关本地 AHS 仿真器的更多信息, 请参阅 [Hello AHS : 运行您的第一个模拟哈密顿仿真](#) 页面和 [模拟哈密顿仿真示例笔记本](#)。

## 状态向量模拟器 (SV1)

SV1 是一款按需、高性能、通用状态矢量仿真器。它可以模拟多达 34 的电路 qubits。根据所用门的 34-qubit 类型和其他因素, 您可以预计, 密集的方形电路 (电路深度 = 34) 大约需要 1-2 小时才能完成。带 all-to-all 栅极的电路非常适合 SV1。它以诸如完整状态向量或振幅数组之类的形式返回结果。

SV1 最大运行时间为 6 小时。它的默认并发量子任务为 35 个, 并发量子任务最多为 100 个 (us-west-1 和 eu-west-2 中为 50 个)。

### SV1 结果

SV1 在给定指定数量的情况下, 可以提供以下结果 shots :

- 示例 : Shots > 0
- 期望 : Shots >= 0
- 方差 : Shots >= 0
- 概率 : Shots > 0
- 振幅 : Shots = 0
- 伴随梯度 : Shots = 0

有关结果的更多信息, 请参阅 [结果类型](#)。

SV1 始终可用, 它可以按需运行您的电路, 并且可以并行运行多个电路。运行时随操作数线性扩展, 随操作数呈指数级缩放。qubits 的数量 shots 对运行时间的影响很小。要了解更多信息, 请访问 [比较模拟器](#)。

模拟器支持 Braket SDK 中的所有门，但是 QPU 设备支持的子集较小。你可以在设备属性中找到设备支持的门。

## 密度矩阵模拟器 (DM1)

DM1是一款按需提供的高性能密度矩阵模拟器。它可以模拟多达 17 个的电路qubits。

DM1最大运行时间为 6 小时，默认为 35 个并发量子任务，最多 50 个并发量子任务。

### DM1结果

DM1在给定指定数量的情况下，可以提供以下结果shots：

- 示例：Shots> 0
- 期望：Shots>= 0
- 方差：Shots>= 0
- 概率：Shots> 0
- 降低密度矩阵：Shots= 0，最大值为 8 qubits

有关结果的更多信息，请参阅[结果类型](#)。

DM1始终可用，它可以按需运行您的电路，并且可以并行运行多个电路。运行时随操作数线性扩展，随操作数呈指数级缩放。qubits的数量shots对运行时间的影响很小。要了解更多信息，请参阅[比较模拟器](#)。

### 噪音门和限制

```
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]
```

## 张量网络模拟器 () TN1

TN1是一款按需、高性能、张量网络模拟器。TN1可以模拟某些电路类型，最大为 50，电路深度 qubits 等于 1,000 或更小。TN1对于稀疏电路、带有局部门的电路以及其他具有特殊结构的电路（例如量子傅里叶变换 (QFT) 电路）特别强大。TN1分两个阶段运作。首先，排练阶段会尝试为您的电路确定有效的计算路径，因此TN1可以估计下一阶段（称为收缩阶段）的运行时间。如果估计的收缩时间超过TN1模拟运行时限制，TN1则不要尝试收缩。

TN1运行时间限制为 6 小时。它限制为最多 10 个（在 eu-west-2 中为 5 个）并发量子任务。

### TN1结果

收缩阶段由一系列矩阵乘法组成。一系列乘法一直持续到达到结果或确定无法得出结果为止。

注意：Shots必须大于 0。

结果类型包括：

- 样本
- 期望
- 方差

有关结果的更多信息，请参阅[结果类型](#)。

TN1始终可用，它可以按需运行您的电路，并且可以并行运行多个电路。要了解更多信息，请参阅[比较模拟器](#)。

模拟器支持 Braket SDK 中的所有门，但是 QPU 设备支持的子集较小。你可以在设备属性中找到设备支持的门。

访问 Amazon Braket GitHub 存储库获取 [TN1 示例笔记本](#)，以帮助您入门。TN1

使用的最佳实践 TN1

- 避免 all-to-all 回路。
- 用少量测试一个新的电路或一类电路shots，以了解电路的“硬度” TN1。
- 将大型shot仿真拆分为多个量子任务。

## 嵌入式模拟器

嵌入式仿真器的工作原理是将仿真与算法代码嵌入到同一个容器中，然后直接在混合作业实例上执行仿真。这对于消除与让仿真与远程设备通信相关的瓶颈非常有用。这可以显著降低内存使用量，减少实现预期结果的电路执行次数，并将性能提高十倍或更多。有关嵌入式模拟器的更多信息，请参阅[使用 Amazon Braket 混合作业运行混合作业](#)页面。

### PennyLane的闪电模拟器

您可以在 Braket PennyLane et 上使用闪电模拟器作为 Braket 上的嵌入式模拟器。借助 PennyLane闪电模拟器，您可以利用高级梯度计算方法（例如伴随[微分](#)）来更快地评估梯度。[lightning.qubit 模拟器](#)可通过 Braket NBI 作为设备使用，也可以作为嵌入式模拟器使用，而 lightning.gpu 模拟器需要作为带有 GPU 实例的嵌入式模拟器运行。有关使用 lightning.gpu 的示例，请参阅 [Braket Hybrid Jobs 笔记本中的嵌入式模拟器](#)。

## 比较模拟器

本节通过描述一些概念、限制和用例，帮助您选择最适合您的量子任务的 Amazon Braket 仿真器。

在本地模拟器和按需模拟器之间进行选择 (SV1,,TN1) DM1

本地模拟器的性能取决于托管本地环境的硬件，例如用于运行模拟器的 Braket 笔记本实例。按需模拟器在 AWS 云端运行，旨在超越典型的本地环境。按需模拟器针对较大的电路进行了优化，但是每个量子任务或批量子任务会增加一些延迟开销。如果涉及许多量子任务，这可能意味着权衡取舍。鉴于这些一般性能特征，以下指导可以帮助您选择如何运行模拟，包括带有噪声的仿真。

对于模拟：

- 如果员工人数少于 18 人qubits，请使用本地模拟器。
- 使用 18-24 时qubits，请根据工作负载选择模拟器。



- 当雇用人数超过 24 人时qubits，请使用按需模拟器。

对于噪声模拟：

- 当雇用少于 9 人时qubits，请使用本地模拟器。
- 使用 9-12 时qubits，请根据工作负载选择模拟器。
- 如果雇用人数超过 12 人qubits，请使用DM1。

什么是状态向量模拟器？

SV1是一个通用的状态向量模拟器。它存储量子态的全波函数，并按顺序将栅极运算应用于该状态。它存储了所有可能性，即使是极不可能的也是如此。SV1模拟器执行量子任务的运行时间随着电路中门的数量而线性增加。

什么是密度矩阵模拟器？

DM1用噪声模拟量子电路。它存储系统的全密度矩阵，并按顺序应用电路的栅极和噪声运算。最终的密度矩阵包含有关电路运行后量子态的完整信息。运行时间通常随操作数线性扩展，随操作数呈指数级扩展。qubits

什么是张量网络模拟器？

TN1将量子电路编码成结构化图。

- 图的节点由量子门或qubits。
- 图形的边缘表示大门之间的连接。

由于这种结构，TN1可以为相对较大和复杂的量子电路找到仿真解。

TN1需要两个阶段

通常，TN1采用两阶段方法来模拟量子计算。

- 排练阶段：在这个阶段，TN1想出一种高效地遍历图表的方法，包括访问每个节点，这样你就可以获得你想要的测量结果。作为客户，您看不到此阶段，因为您可以同时TN1执行两个阶段。它完成了第一阶段，并根据实际限制决定是否单独执行第二阶段。在模拟开始后，您对该决定没有任何意见。
- 收缩阶段：此阶段类似于经典计算机中计算的执行阶段。该阶段由一系列矩阵乘法组成。这些乘法的顺序对计算难度有很大的影响。因此，首先要完成排练阶段，以便在图中找到最有效的计算路径。在排练阶段找到收缩路径后，将电路的大门收TN1缩在一起以生成仿真结果。

## TN1图表类似于地图

比喻地说，你可以将底层TN1图表与城市的街道进行比较。在采用规划网格的城市中，使用地图可以很容易地找到通往目的地的路线。在一个街道计划外、街道名称重复等的城市中，通过查看地图可能很难找到通往目的地的路线。

如果TN1没有进行排练阶段，那就像在城市的街道上漫步寻找目的地，而不是先看地图。就步行时间而言，花更多的时间看地图确实可以获得回报。同样，排练阶段也提供了有价值的信息。

你可能会说，它对它所TN1穿越的底层电路的结构有一定的“意识”。它在排练阶段获得了这种意识。

## 问题类型最适合每种类型的模拟器

SV1非常适合于主要依赖于具有一定数量的qubits和门的任何类别的问题。通常，所需时间会随着门数的增加而线性增长，而不取决于门的shots数量。SV1通常比28岁以下TN1的电路快qubits。

SV1对于较高的qubit数字，可能会变慢，因为它实际上模拟了所有可能性，即使是极不可能的也是如此。它无法确定可能出现哪些结果。因此，要进行30-qubit评估，SV1必须计算  $2^{30}$  个配置。由于内存和存储限制，Amazon Braket SV1 模拟器的限制为 34 qubits 是一个实际限制。你可以这样想：每当你qubit向中添加一个SV1，问题就会变得困难两倍。

对于许多类别的问题，与利用图的结构相比SV1，TN1可以在现实时间内评估更大的电路。TN1它本质上是从一开始就跟踪解决方案的演变，并且仅保留有助于高效遍历的配置。换句话说，它保存配置以创建矩阵乘法的顺序，从而简化计算过程。

因为TN1，qubits和门的数量很重要，但图的结构更重要。例如，非常擅长评估栅极为短TN1距离的电路（图表）（也就是说，每个qubit门仅通过栅极连接到其最近的邻居qubits），以及连接（或门）具有相似范围的电路（图表）。的典型范围TN1是让每人只qubit与5点qubits之外的其他qubits人交谈。如果结构的大部分可以分解为更简单的关系，例如这些关系，可以用更多、更小或更均匀的矩阵表示，则可以轻松地TN1进行评估。

## 的局限性 TN1

TN1可能比SV1根据图表的结构复杂度慢一些。对于某些图表，由于以下两个原因之一，会在排练阶段结束后TN1终止仿真并显示状态：FAILED

- 找不到路径 — 如果图形太复杂，则很难找到一条好的遍历路径，模拟器就会放弃计算。TN1无法进行收缩。您可能会看到类似于以下内容的错误消息：No viable contraction path found.
- 收缩阶段太困难了 — 在某些图表中，TN1可以找到遍历路径，但是评估起来非常漫长且非常耗时。在这种情况下，收缩非常昂贵，以至于成本高得令人望而却步，而是在排练阶段之后TN1退出。您

可能会看到类似于以下内容的错误消息：Predicted runtime based on best contraction path found exceeds TN1 limit.

#### Note

TN1即使没有进行收缩并且您会看到状态，您也需要支付排练阶段的费用。FAILED

预测的运行时间也取决于shot计数。在最坏的情况下，TN1收缩时间线性地取决于计数。shot该电路可以用更少shots的钱进行合约。例如，您可以提交一个包含 100 的量子任务shots，该任务TN1决定不可合约，但是如果您只有 10 的量子任务重新提交，则会继续收缩。在这种情况下，要获得 100 个样本，你可以shots为同一电路提交 10 个 10 个量子任务，最后合并结果。

作为最佳实践，我们建议您始终使用几个shots（例如 10）来测试您的电路或电路等级，以了解您的电路有多难TN1，然后再继续使用更高的数量shots。

#### Note

形成收缩阶段的一系列乘法从小的  $n \times N$  矩阵开始。例如，2-qubit门需要一个  $4 \times 4$  矩阵。在被判定为过于困难的收缩期间所需的中间矩阵是巨大的。这样的计算需要几天才能完成。这就是为什么 Amazon Braket 不尝试极其复杂的收缩的原因。

## 并发

所有 Braket 模拟器都允许您同时运行多个电路。并发限制因模拟器和区域而异。有关并发限制的更多信息，请参阅[配额](#)页面。

## 亚马逊 Braket 区域和终端节点

Amazon Braket 有以下几种可供选择：AWS 区域

Amazon Braket 的地区可用性

区域名称	区域	支架端点	QPU
美国东部（弗吉尼亚州北部）	us-east-1	braket.us-east-1.a amazonaws.com	ionQ

区域名称	区域	支架端点	QPU
美国东部 (弗吉尼亚州北部)	us-east-1	braket.us-east-1.amazonaws.com	QuEra
美国西部 (加利福尼亚北部)	us-west-1	braket.us-west-1.amazonaws.com	Rigetti
欧盟北部 1 (斯德哥尔摩)	eu-north-1	braket.eu-north-1.amazonaws.com	IQM
欧盟西部 2 (伦敦)	eu-west-2	braket.eu-west-2.amazonaws.com	OQC

您可以从任何可用的地区运行 Amazon Braket，但每个 QPU 仅在一个地区可用。在 QPU 设备上运行的 Quantum 任务可以在该设备所在区域的 Amazon Braket 控制台中查看。如果您使用的是 Amazon Braket SDK，则无论您在哪个地区工作，都可以向任何 QPU 设备提交量子任务。SDK 会自动为指定的 QPU 创建与该区域的会话。

有关 AWS 如何使用区域和终端节点的一般信息，请参阅 AWS 通用参考中的 [AWS 服务 终端节点](#)。

## 我的量子任务什么时候能运行？

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

当您提交电路时，Amazon Braket 会将其发送到您指定的设备。量子处理单元 (QPU) 和按需仿真器量子任务按接收顺序排队和处理。提交量子任务后处理该任务所需的时间会有所不同，具体取决于其他 Amazon Braket 客户提交的任务的数量和复杂程度以及所选 QPU 的可用性。

## 通过电子邮件或短信发送状态变更通知

当 QPU 的可用性发生变化或量子任务的状态发生变化 EventBridge 时，Amazon Braket 会向亚马逊发送事件。请按照以下步骤通过电子邮件或 SMS 消息接收设备和量子任务状态变更通知：

1. 创建 Amazon SNS 主题并订阅电子邮件或短信。电子邮件或短信的可用性取决于您所在的地区。有关更多信息，请参阅 [Amazon SNS 入门](#) 和 [发送短信](#)。
2. 在中创建一条规则 EventBridge，用于触发您的 SNS 主题的通知。有关更多信息，请参阅使用 [亚马逊监控 Amazon Braket](#)。EventBridge

## 量子任务完成警报

您可以通过亚马逊简单通知服务 (SNS) Simple Notification Service 设置通知，以便在 Amazon Braket 量子任务完成时收到提醒。如果您预计等待时间很长，例如，当您提交大型任务或在设备可用性窗口之外提交任务时，活动通知很有用。如果您不想等待任务完成，则可以设置 SNS 通知。

Amazon Braket 笔记本将引导您完成设置步骤。有关更多信息，请参阅用于 [设置通知的 Amazon Braket 示例笔记本](#)。

## QPU 可用性窗口和状态

QPU 可用性因设备而异。

在 Amazon Braket 控制台的“设备”页面中，您可以看到当前和即将推出的可用窗口以及设备状态。此外，每个设备页面都显示量子任务和混合任务的单独队列深度。

无论可用性窗口如何，如果客户无法使用该设备，则该设备将被视为离线。例如，它可能由于定期维护、升级或操作问题而处于离线状态。

## 队列可见性

在提交量子任务或混合任务之前，您可以通过检查设备队列深度来查看摆在您面前的量子任务或混合任务的数量。

### 队列深度

Queue depth 指排队等候特定设备的量子任务和混合作业的数量。可通过 Braket Software Development Kit (SDK) 或访问设备的量子任务和混合作业队列数 Amazon Braket Management Console。

1. 任务队列深度是指当前等待以正常优先级运行的量子任务总数。
2. 优先任务队列深度是指等待运行的已提交量子任务的总数 Amazon Braket Hybrid Jobs。这些任务在独立任务之前运行。
3. 混合作业队列深度是指设备上当前排队的混合作业总数。Quantum tasks 作为混合作业的一部分提交具有优先级，并汇总在 Priority Task Queue。

希望通过查看队列深度的客户Braket SDK可以修改以下代码片段以获取其量子任务或混合任务的队列位置：

```
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'
```

向 QPU 提交量子任务或混合作业可能会导致您的工作负载处于QUEUED状态。Amazon Braket 让客户可以看到他们的量子任务和混合任务队列位置。

## 队列位置

Queue position指您的量子任务或混合任务在相应设备队列中的当前位置。它可以通过或获得，用于量子任务或混合作业Amazon Braket Management Console。Braket Software Development Kit (SDK)

希望通过查看队列位置的客户Braket SDK可以修改以下代码片段以获取其量子任务或混合任务的队列位置：

```
# 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
```

# 开始使用 Amazon Braket

## Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

按照[启用 Amazon Braket 中的说明进行操作](#)后，就可以开始使用 Braket 了。Amazon

入门步骤包括：

- [启用 Amazon Braket](#)
- [创建 Amazon Braket 笔记本实例](#)
- [使用 Amazon Braket Python SDK 运行你的第一个赛道](#)
- [运行你的第一个量子算法](#)

## 启用 Amazon Braket

## Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

[您可以通过控制台在您的账户中启用 Amazon Braket。AWS](#)

## 先决条件

要启用和运行 Amazon Braket，您必须拥有有权启动 Amazon Braket 操作的用户或角色。这些权限包含在 AmazonBraketFullAccess IAM 策略中（`arn:aws:iam::aws:policy/`）。AmazonBraketFullAccess

## Note

如果您是管理员：



要向其他用户授予访问 Amazon Braket 的权限，请通过附加 AmazonBraketFullAccess 策略或附加您创建的自定义策略向用户授予权限。要详细了解使用 Amazon Braket 所需的权限，请参阅 [管理对亚马逊 Braket 的访问权限](#)。

## 启用 Amazon Braket 的步骤

1. 使用您的账户登录 [Amazon Braket 控制台](#)。AWS 账户
2. 打开 Amazon Braket 控制台。
3. 在 Braket 登录页面上，单击“入门”，进入服务控制面板页面。服务控制面板顶部的警报将引导您完成以下三个步骤：
  - a. 创建 [服务相关角色 \(SLR\)](#)
  - b. 允许访问第三方量子计算机
  - c. 创建新的 Jupyter 笔记本实例

要使用第三方量子设备，您需要同意有关您自己与这些设备之间数据传输的某些条件。AWS 本协议的条款和条件在 Amazon Braket 控制台的“权限和设置”页面的“常规”选项卡上提供。

### Note

无需同意“启用第三方设备”协议即可使用不涉及任何第三方的 Quantum 设备，例如 Braket 本地模拟器或按需模拟器。

如果您要访问第三方硬件，则每个账户只需接受这些条款即可使用第三方设备一次。

## 创建 Amazon Braket 笔记本实例

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

Amazon Braket 提供完全托管的 Jupyter 笔记本供你入门。Amazon Braket 笔记本实例基于 [亚马逊 SageMaker 笔记本](#) 实例。以下说明概述了为新客户和现有客户创建新的笔记本实例所需的步骤。

## 亚马逊 Braket 的新客户

1. 打开 [Amazon Braket 控制台](#)，在左侧窗格中导航至控制面板页面。
2. 在控制面板页面中心的“欢迎使用 Amazon Braket”模式上单击“开始”，提供笔记本名称。这将创建一个默认的 Jupyter 笔记本。
3. 创建笔记本可能需要几分钟。您的笔记本将列在“笔记本”页面上，状态为“待定”。当您的笔记本实例准备就绪可供使用时，状态将更改为 InService。您可能需要刷新页面才能显示笔记本的更新状态。

## 亚马逊 Braket 的现有客户

1. 打开 Amazon Braket 控制台，在左侧窗格中选择笔记本，选择创建笔记本实例。如果您的笔记本为零，请选择标准设置以创建默认 Jupyter 笔记本，然后仅使用字母数字和连字符输入笔记本实例名称，然后选择首选的视觉模式。然后，启用或禁用笔记本电脑的非活动状态管理器。
  - a. 如果启用，请在重置笔记本之前选择所需的空闲持续时间。重置笔记本电脑后，计算费用将停止产生，但存储费用将继续。
  - b. 要查看笔记本实例中的剩余空闲时间，请导航至命令栏并选择 Braket 选项卡，然后选择“不活动管理器”选项卡。

### Note

为了避免您的工作丢失，请考虑[将您的 SageMaker 笔记本实例与 git 存储库集成](#)。或者，将您的工作移到/Braket Algorithms和/Braket Examples文件夹之外可以防止该文件被笔记本实例重启所覆盖。

2. (可选) 使用高级设置，您可以创建具有访问权限、其他配置和网络访问设置的笔记本：
  - a. 在笔记本配置中，选择您的实例类型。默认情况下，选择经济实惠的标准实例类型 ml.t3.medium。要了解有关实例定价的更多信息，请参阅 [Amazon SageMaker 定价](#)。如果要将公共 Github 存储库与笔记本实例相关联，请单击 Git 存储库下拉列表，然后从“存储库”下拉菜单中选择“从 url 中克隆公共 git 存储库”。在 Git 存储库 URL 文本栏中输入存储库的 URL。
  - b. 在权限中，配置任何可选的 IAM 角色、根访问权限和加密密钥。
  - c. 在网络中，为您的 Jupyter Notebook 实例配置自定义网络和访问设置。
3. 查看您的设置，设置任何标签以识别您的笔记本实例，然后单击 Launch。

**Note**

您可以在 Amazon Braket 和亚马逊控制台中查看和管理您的 Amazon Braket 笔记本实例。SageMaker [其他 Amazon Braket 笔记本设置可通过控制台获得](#)。SageMaker

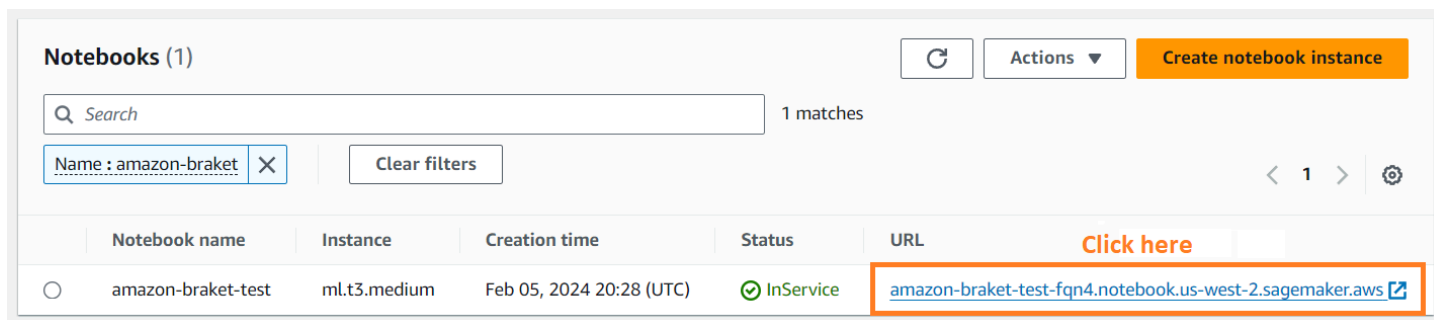
如果你在 Amazon Braket 控制台中 AWS 使用 Amazon Braket SDK，并且插件已预装在你创建的笔记本中。如果您想在自己的计算机上运行，则可以在运行命令 `pip install amazon-braket-sdk` 或运行 `pip install amazon-braket-pennylane-plugin` 用于插件的命令时安装 SDK 和 PennyLane 插件。

## 使用 Amazon Braket Python SDK 运行你的第一个赛道

**Tip**

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

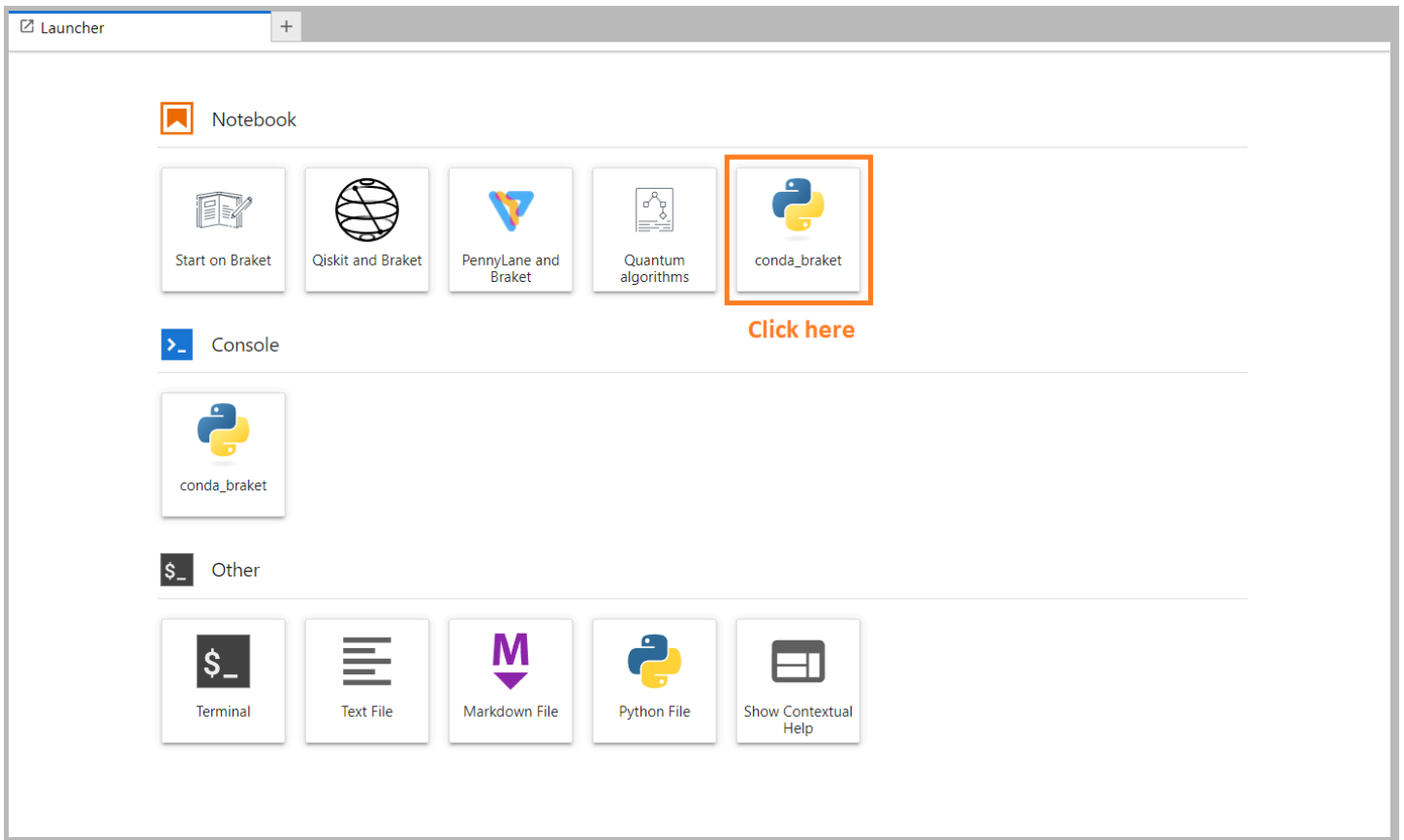
启动笔记本实例后，选择您刚刚创建的笔记本，使用标准 Jupyter 界面打开该实例。



The screenshot shows the Amazon Braket console interface. At the top, there's a search bar with 'Search' text and a '1 matches' indicator. Below the search bar, there's a filter for 'Name : amazon-braket' and a 'Clear filters' button. The main content is a table with columns: Notebook name, Instance, Creation time, Status, and URL. The first row is highlighted with a red box around the URL column. The URL is 'amazon-braket-test-fqn4.notebook.us-west-2.sagemaker.aws' with a 'Click here' link and an external link icon. Above the table, there are buttons for 'Create notebook instance' and 'Actions'.

Notebook name	Instance	Creation time	Status	URL
amazon-braket-test	ml.t3.medium	Feb 05, 2024 20:28 (UTC)	InService	<a href="amazon-braket-test-fqn4.notebook.us-west-2.sagemaker.aws">amazon-braket-test-fqn4.notebook.us-west-2.sagemaker.aws</a>

Amazon Braket 笔记本实例已预先安装了 Amazon Braket SDK 及其所有依赖项。首先创建一个带有 `conda_braket` 内核的新笔记本。



你可以从一个简单的“你好，世界！”开始 示例。首先，构造一个准备贝尔状态的电路，然后在不同的设备上运行该电路以获得结果。

首先导入 Amazon Braket SDK 模块并定义一个简单的 Bell State 电路。

```
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)
```

你可以用这个命令对电路进行可视化：

```
print(bell)
```

在本地模拟器上运行你的电路

接下来，选择运行电路的量子器件。AmazonBraket SDK 附带本地模拟器，用于快速原型设计和测试。对于较小的电路，我们建议使用本地仿真器，最大可达 25 qubits 个（取决于您的本地硬件）。

以下是实例化本地模拟器的方法：

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

然后运行电路：

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

你应该会看到这样的结果：

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

正如预期的那样，你准备的具体贝尔状态是  $|00\rangle$  和  $|11\rangle$  的相等叠加，你会发现测量结果的分布大致相等（最多为 shot 噪声），即 00 和 11。

在按需模拟器上运行您的赛道

AmazonBraket 还提供按需提供的高性能仿真器 SV1，用于运行更大的电路。SV1 是一款按需状态矢量仿真器，允许仿真多达 34 qubits 个量子电路。您可以 SV1 在“[支持的设备](#)”部分和 AWS 控制台找到更多信息。在 SV1（和任 TN1 何 QPU 上）上运行量子任务时，量子任务的结果将存储在您账户的 S3 存储桶中。如果您未指定存储桶，Braket SDK 会 `amazon-braket-{region}-{accountID}` 为您创建一个默认存储桶。要了解更多信息，请参阅[管理对 Amazon Braket 的访问权限](#)。

#### Note

填写您的实际现有存储桶名称，以下示例显示 `example-bucket` 为您的存储桶名称。AmazonBraket 的存储桶名称始终以 `amazon-braket-` 您添加的其他识别字符开头。如果您需要有关如何设置 S3 存储桶的信息，请参阅 [Amazon S3 入门](#)。

```
# 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)
```

要运行电路SV1，您必须提供先前在`.run()`调用中作为位置参数选择的 S3 存储桶的位置。

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

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

Amazon Braket 控制台提供有关您的量子任务的更多信息。导航到控制台中的 Quantum Tasks 选项卡，您的量子任务应该位于列表的顶部。或者，您可以使用唯一的量子任务 ID 或其他条件搜索您的量子任务。

#### Note

90 天后，Amazon Braket 会自动删除与您的量子任务关联的所有量子任务 ID 和其他元数据。有关更多信息，请参阅[数据保留](#)。

## 在 QPU 上运行

使用 Amazon Braket，您只需更改一行代码即可在物理量子计算机上运行前面的量子电路示例。Amazon Braket 允许从 IonQ、Oxford Quantum Circuits QuEra、和访问 QPU 设备。Rigetti 您可以在“[支持的设备](#)”部分和 [AWS 控制台的“设备”](#) 选项卡下找到有关不同设备和可用性窗口的信息。以下示例显示了如何实例化设备。Rigetti

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

使用以下代码选择 IonQ 台设备：

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

选择设备后，在运行工作负载之前，您可以使用以下代码查询设备队列深度，以确定量子任务或混合任务的数量。此外，客户可以在的“设备”页面上查看设备特定的队列深度。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
```

当你运行任务时，Amazon Braket SDK 会轮询结果（默认超时时间为 5 天）。您可以通过修改 `.run()` 命令中的 `poll_timeout_seconds` 参数来更改此默认值，如以下示例所示。请记住，如果您的轮询超时时间太短，则可能无法在轮询时间内返回结果，例如当 QPU 不可用且返回本地超时错误时。您可以通过调用 `task.result()` 函数来重新开始轮询。

```
# 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)
```

此外，提交量子任务或混合任务后，您可以调用该 `queue_position()` 函数来检查队列位置。

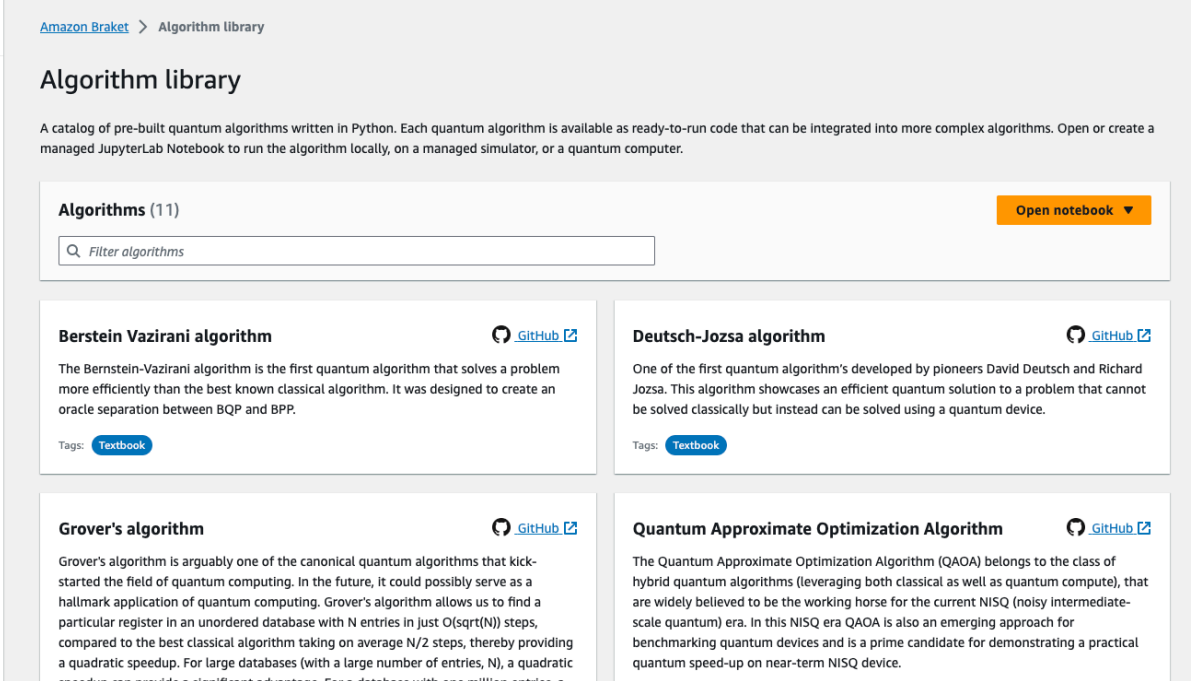
```
print(task.queue_position().queue_position)
# Return the number of quantum tasks queued ahead of you
'2'
```

## 运行你的第一个量子算法

### Tip

通过以下方式学习量子计算的基础 AWS！注册 [Amazon Braket 数字学习计划](#)，完成一系列学习课程和数字评估后，即可获得自己的数字徽章。

Amazon Braket 算法库是用 Python 编写的预建量子算法的目录。您可以按原样运行这些算法，也可以使用它们作为起点来构建更复杂的算法。您可以从 Braket 控制台访问算法库。你也可以在 Github 上访问 Braket 算法库：<https://github.com/aws-samples/amazon-braket-algorithm-library>。



**Amazon Braket** ×

Amazon Braket > Algorithm library

## Algorithm library

A catalog of pre-built quantum algorithms written in Python. Each quantum algorithm is available as ready-to-run code that can be integrated into more complex algorithms. Open or create a managed JupyterLab Notebook to run the algorithm locally, on a managed simulator, or a quantum computer.

Algorithms (11) Open notebook ▾

Filter algorithms

### Bernstein Vazirani algorithm

The Bernstein-Vazirani algorithm is the first quantum algorithm that solves a problem more efficiently than the best known classical algorithm. It was designed to create an oracle separation between BQP and BPP.

Tags: Textbook

[GitHub](#)

### Deutsch-Jozsa algorithm

One of the first quantum algorithm's developed by pioneers David Deutsch and Richard Jozsa. This algorithm showcases an efficient quantum solution to a problem that cannot be solved classically but instead can be solved using a quantum device.

Tags: Textbook

[GitHub](#)

### Grover's algorithm

Grover's algorithm is arguably one of the canonical quantum algorithms that kick-started the field of quantum computing. In the future, it could possibly serve as a hallmark application of quantum computing. Grover's algorithm allows us to find a particular register in an unordered database with  $N$  entries in just  $O(\sqrt{N})$  steps, compared to the best classical algorithm taking on average  $N/2$  steps, thereby providing a quadratic speedup. For large databases (with a large number of entries,  $N$ ), a quadratic speedup can provide a significant advantage. For a database with one million entries, a

[GitHub](#)

### Quantum Approximate Optimization Algorithm

The Quantum Approximate Optimization Algorithm (QAOA) belongs to the class of hybrid quantum algorithms (leveraging both classical as well as quantum compute), that are widely believed to be the working horse for the current NISQ (noisy intermediate-scale quantum) era. In this NISQ era QAOA is also an emerging approach for benchmarking quantum devices and is a prime candidate for demonstrating a practical quantum speed-up on near-term NISQ device.

[GitHub](#)

Braket 控制台提供了算法库中每种可用算法的描述。选择一个 GitHub 链接以查看每种算法的详细信息，或者选择“打开笔记本”以打开或创建包含所有可用算法的笔记本。如果您选择笔记本选项，则可以在笔记本的根文件夹中找到 Braket 算法库。



## 与 Amazon Braket 合作

本节向您展示如何设计量子电路，如何将这些问题作为量子任务提交给设备，以及如何使用 Amazon Braket SDK 监控量子任务。

以下是在 Amazon Braket 上与资源进行交互的主要方式。

- [Amazon Braket 控制台](#) 提供设备信息和状态，以帮助您创建、管理和监控资源和量子任务。
- 通过 [Amazon Braket Python SDK](#) 和控制台提交和运行量子任务。可通过预配置的 Amazon Braket 笔记本电脑访问软件开发工具包。
- [Amazon Braket API](#) 可通过 `Braket` Amazon Python SDK 和笔记本访问。API 如果您正在构建以编程方式使用量子计算的应用程序，则可以直接调用。

本节中的示例演示了如何使用 `Braket Python SDK` 和适用于 Amazon Braket 的 `Amazon Python SDK (Boto AWS 3) API 直接使用 Braket`。

有关 Amazon Braket Python SDK 的更多信息

要使用 Amazon Braket Python SDK，请先安装适用于 Braket 的 AWS Python SDK (Boto3)，这样您就可以与通信。AWS API 您可以将 Amazon Braket Python SDK 看作是量子客户围绕 Boto3 的便捷包装。

- Boto3 包含您需要进入的接口。AWS API ( 请注意，Boto3 是一个大型 Python 软件开发工具包，可以与 AWS API 大多数都 AWS 服务支持 Boto3 接口。 )
- Amazon Braket Python SDK 包含用于电路、门、设备、结果类型和量子任务其他部分的软件模块。每次创建程序时，都要导入该量子任务所需的模块。
- Amazon Braket Python SDK 可通过笔记本访问，笔记本预装了运行量子任务所需的所有模块和依赖项。
- 如果你不想使用笔记本，你可以将模块从 Amazon Braket Python SDK 导入到任何 Python 脚本中。

[安装 Boto3 后，通过 Braket Python SDK 创建量子任务的步骤概述如下所示：](#)

1. ( 可选 ) 打开笔记本电脑。
2. 导入电路所需的 SDK 模块。
3. 指定 QPU 或模拟器。
4. 实例化电路。

5. 运行赛道。
6. 收集结果。

本节中的示例显示了每个步骤的详细信息。

有关更多示例，请参阅上的 [Amazon Braket 示例](#) 存储库。GitHub

本节内容：

- [你好 AHS：运行你的第一个模拟哈密顿仿真](#)
- [在 SDK 中构造电路](#)
- [向 QPU 和模拟器提交量子任务](#)
- [使用 OpenQasm 3.0 运行你的赛道](#)
- [使用 QuEra's Aquila 提交模拟节目](#)
- [使用 Boto3](#)

## 你好 AHS：运行你的第一个模拟哈密顿仿真

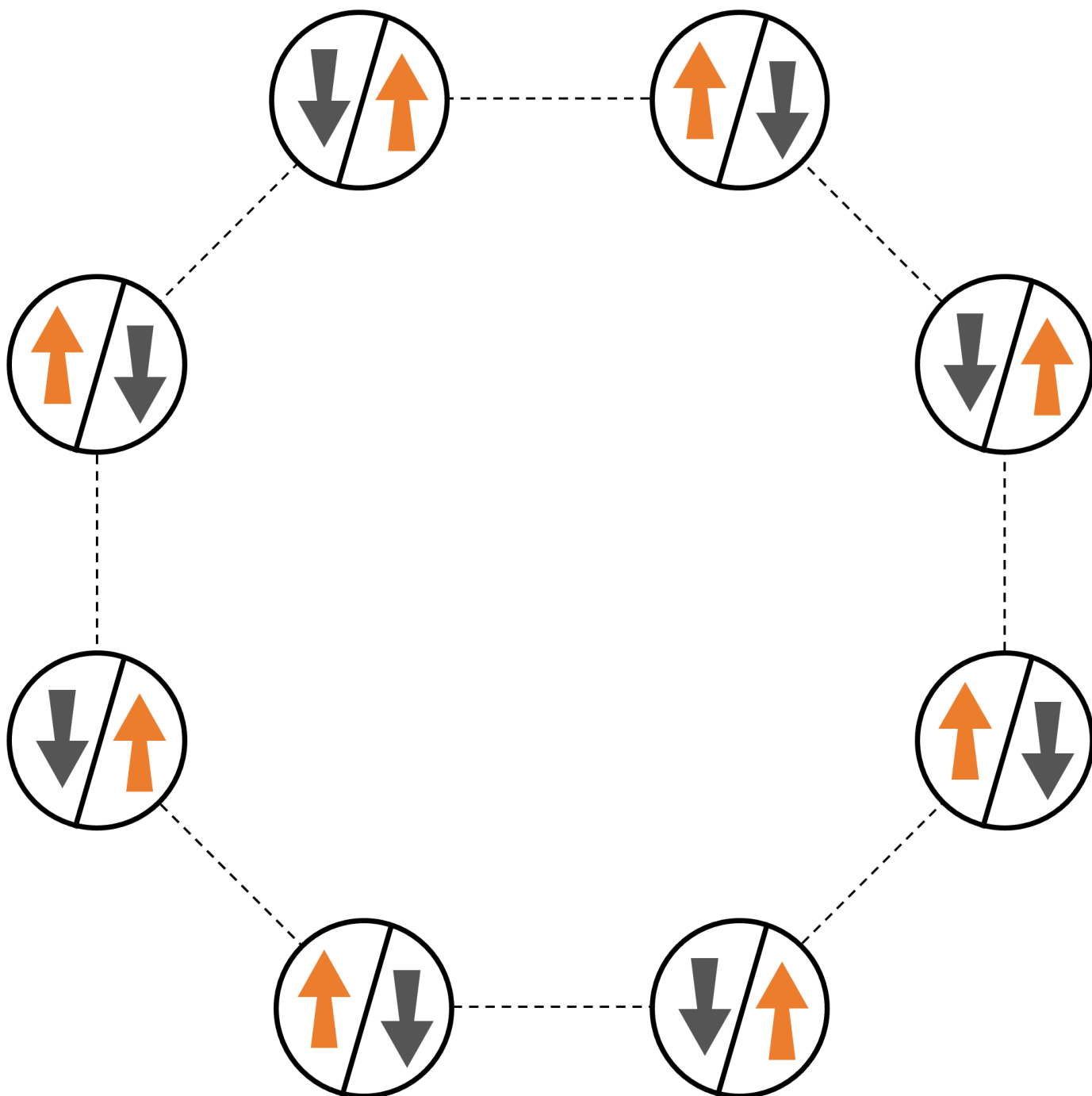
啊

[模拟哈密顿模拟](#) ( AHS ) 是一种量子计算范式，不同于量子电路：AHS程序不是由门序列组成，每个门一次只能作用于几个量子比特，而是由相关哈密顿的时空相关参数来定义。[系统的哈密顿函数](#)对其能级和外力的影响进行编码，它们共同控制着其状态的时间演变。对于 N 量子比特系统，哈密顿可以用  $2^{N \times 2^N}$  的复数方阵表示。

能够执行 AHS 的量子器件将调整其参数（例如相干驱动场的振幅和失调），以近似定制哈密顿下量子系统的时间演变。AHS 范式适用于模拟许多相互作用粒子的量子系统的静态和动态特性。专门构建的 QPU，例如来自 [Aquila 设备](#)，QuEra可以模拟系统的时间演变，而这些系统的尺寸在传统硬件上是无法实现的。

### 互动旋转链

举一个由许多相互作用的粒子组成的系统的典型示例，让我们考虑一个由八个旋转组成的环（每个旋转都可以处于“向上” $\uparrow$ 和“向下” $\downarrow$ 的状态）。尽管规模很小，但该模型系统已经表现出一些自然存在的磁性材料的有趣现象。在这个例子中，我们将展示如何准备一个所谓的反铁磁阶数，即连续的自旋指向相反的方向。



## 安排

我们将使用一个中性原子来代表每次自旋，“向上”和“向下”的自旋态将分别以激发的里德伯格态和原子的基态编码。首先，我们创建二维排列。我们可以用以下代码对上面的旋转圈进行编程。

先决条件：你需要 pip 安装 [Braket](#) SDK。（如果您使用的是 Braket 托管的笔记本实例，则此 SDK 已预先安装在笔记本中。）要重现绘图，你还需要使用 shell 命令单独安装 matplotlib。pip install matplotlib

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

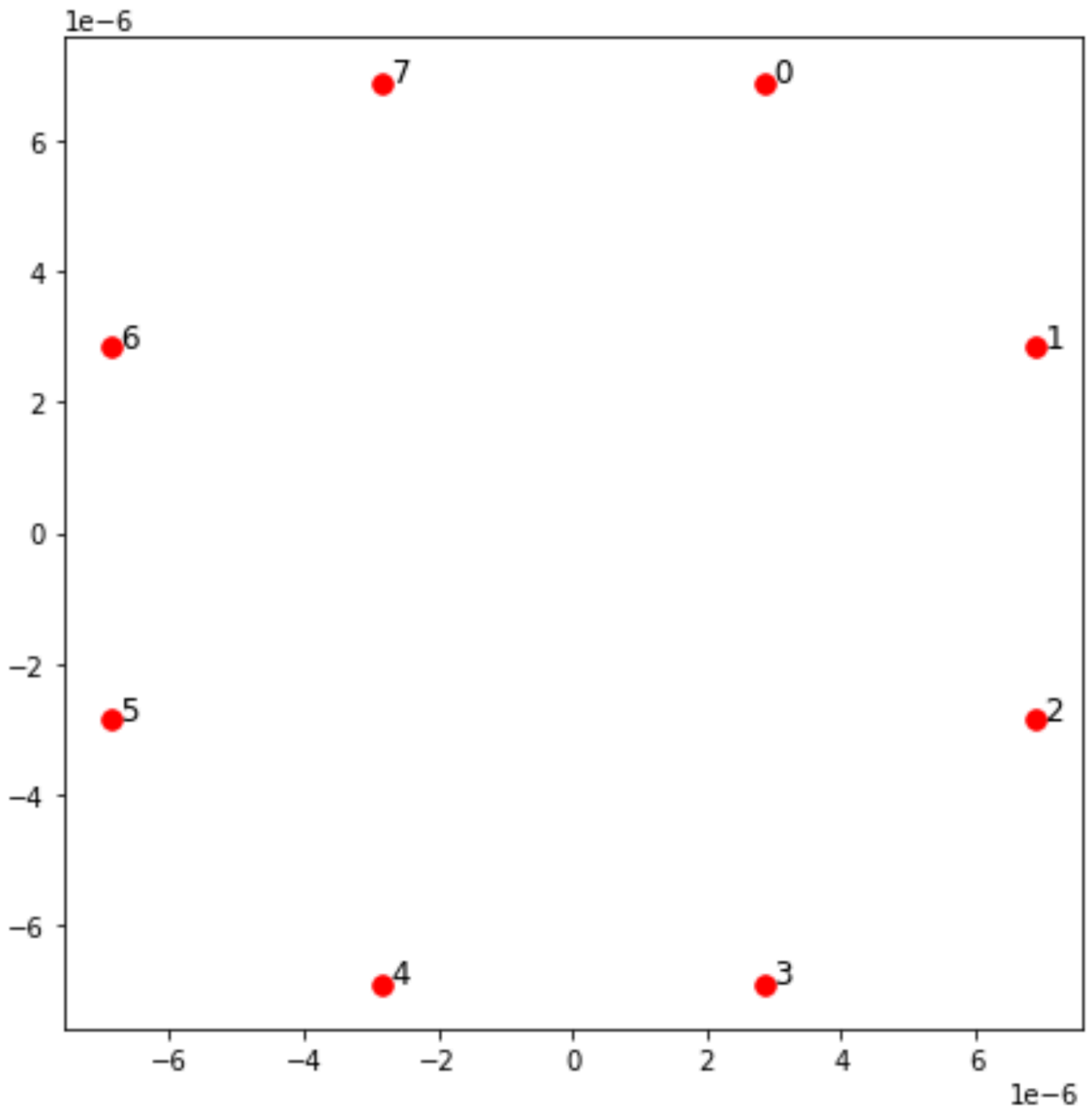
from braket.ahs.atom_arrangement import AtomArrangement

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)
```

我们也可以用它来绘图

```
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
```



## 交互

为了准备反铁磁相，我们需要诱导相邻自旋之间的相互作用。为此，我们使用了 [van der Waals 交互](#)，该交互是由中性原子器件（例如来自 QuEra 的设备）原生实现的 Aquila。使用自旋表示，这种相互作用的哈密顿项可以表示为所有自旋对  $(j, k)$  的总和。

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

这里， $n_j = \frac{1}{2}(S_j^z + 1)$  是一个运算符，仅当  $s_j$  处于“向上”状态时才取值 1，否则取值 0。强度为  $V_{j,k} = C_6/d_{j,k}^6$ ，其中  $C_6$  是固定系数， $d_{j,k}$  是自旋  $j$  和  $k$  之间的欧几里得距离。这个相互作用项的直接影响是，任何自旋  $j$  和自旋  $k$  都是“向上”的状态的能量都会升高（按量  $V_{j,k}$ ）。通过精心设计 AHS 程序的其余部分，这种交互将防止相邻的旋转都处于“向上”状态，这种效果通常被称为“里德伯格封锁”。

## 驾驶场

在 AHS 程序开始时，所有旋转（默认情况下）都以“向下”状态开始，它们处于所谓的铁磁阶段。着眼于准备反铁磁相位的目标，我们指定了一个随时间变化的相干驱动场，该驱动场可以平稳地将自旋从这种状态过渡到首选“向上”状态的多体状态。相应的哈密顿量可以写成

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

其中  $\Omega(t)$ 、 $\phi(t)$ 、 $\Delta(t)$  是随时间变化的全局振幅（又名 [Rabi 频率](#)）、相位和失谐均匀地影响所有自旋的驱动场。这里  $S_{-,k} = \frac{1}{2}(S_k^x - iS_k^y)$  和  $S_{+,k} = \frac{1}{2}(S_k^x + iS_k^y)$  分别是  $s_k$  的降低和升高运算符， $n_k = \frac{1}{2}(S_k^z + 1)$  和以前是同一个运算符。 $\Omega$  部分同时连贯地耦合所有旋转的“向下”和“向上”状态，而  $\Delta$  部分控制“向上”状态的能量奖励。

为了编程从铁磁相向到反铁磁相的平滑过渡，我们使用以下代码指定驱动场。

```
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
)
```

我们可以使用以下脚本可视化驾驶场的时间序列。

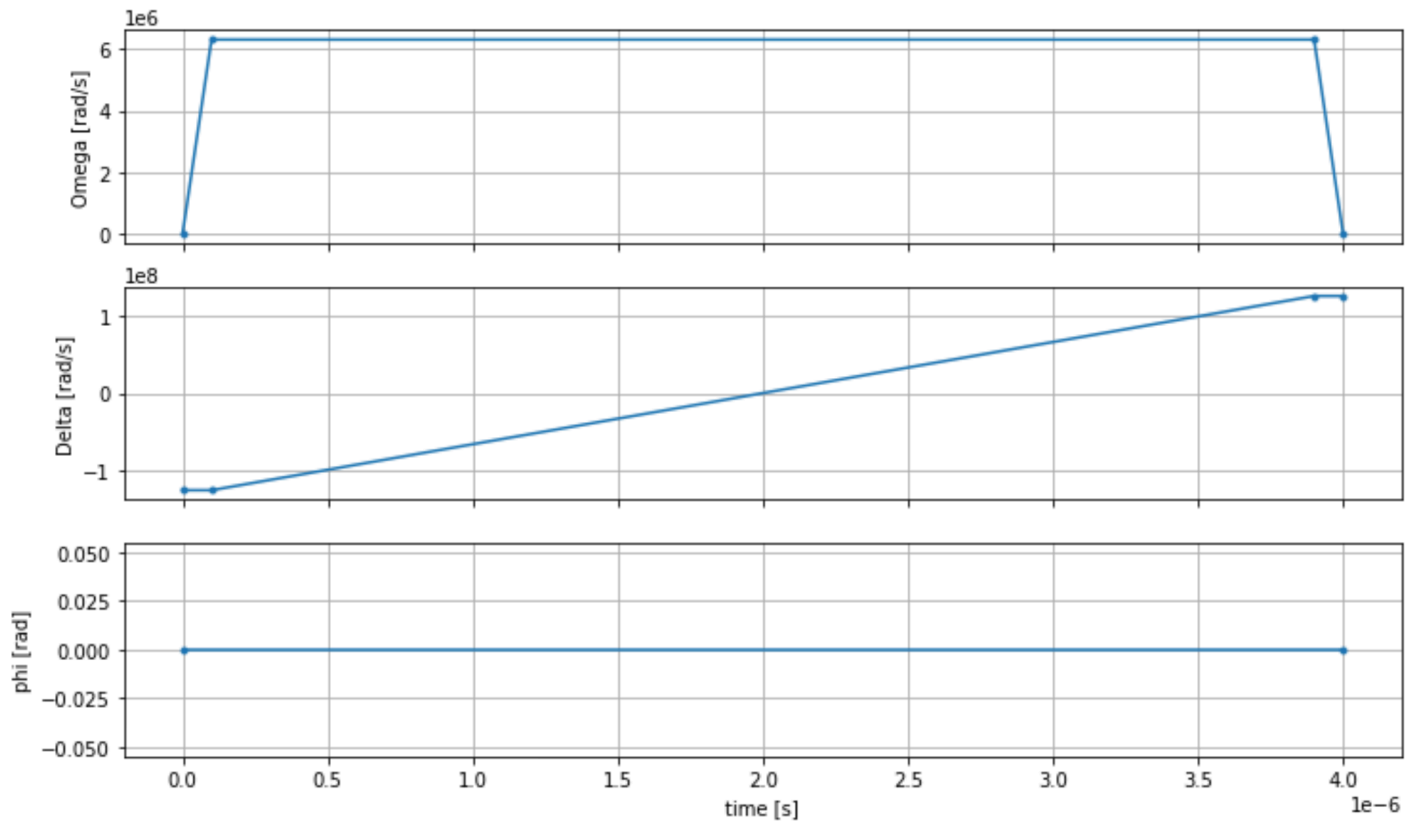
```
fig, axes = plt.subplots(3, 1, figsize=(12, 7), sharex=True)

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 计划

寄存器、驱动场（以及隐式的范德华相互作用）构成了模拟哈密顿仿真程序。ahs\_program

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

## 在本地模拟器上运行

由于此示例很小（少于 15 次旋转），因此在兼容 AHS 的 QPU 上运行之前，我们可以在 Braket SDK 附带的本地 AHS 模拟器上运行它。由于本地模拟器可免费使用 Braket SDK，因此这是确保我们的代码能够正确执行的最佳实践。

在这里，我们可以将镜头数量设置为较高的值（比如 100 万），因为本地模拟器会跟踪量子态的时间演变并从最终状态中抽取样本；因此，可以增加镜头数量，而总运行时间仅略有增加。



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

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

## 分析模拟器结果

我们可以使用以下函数聚合镜头结果，该函数可以推断每次旋转的状态（可以是“d”代表“向下”，“u”表示“向上”，“e”代表空场地），并计算每种配置在镜头中发生的次数。

```
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.AnalogHamiltonianSimulationQuantumTaskResult)

    Returns
        dict: number of times each state configuration is measured

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

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

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

`counts`这是一本字典，它计算了在镜头中观察到每种状态配置的次数。我们还可以使用以下代码将它们可视化。

```
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)))

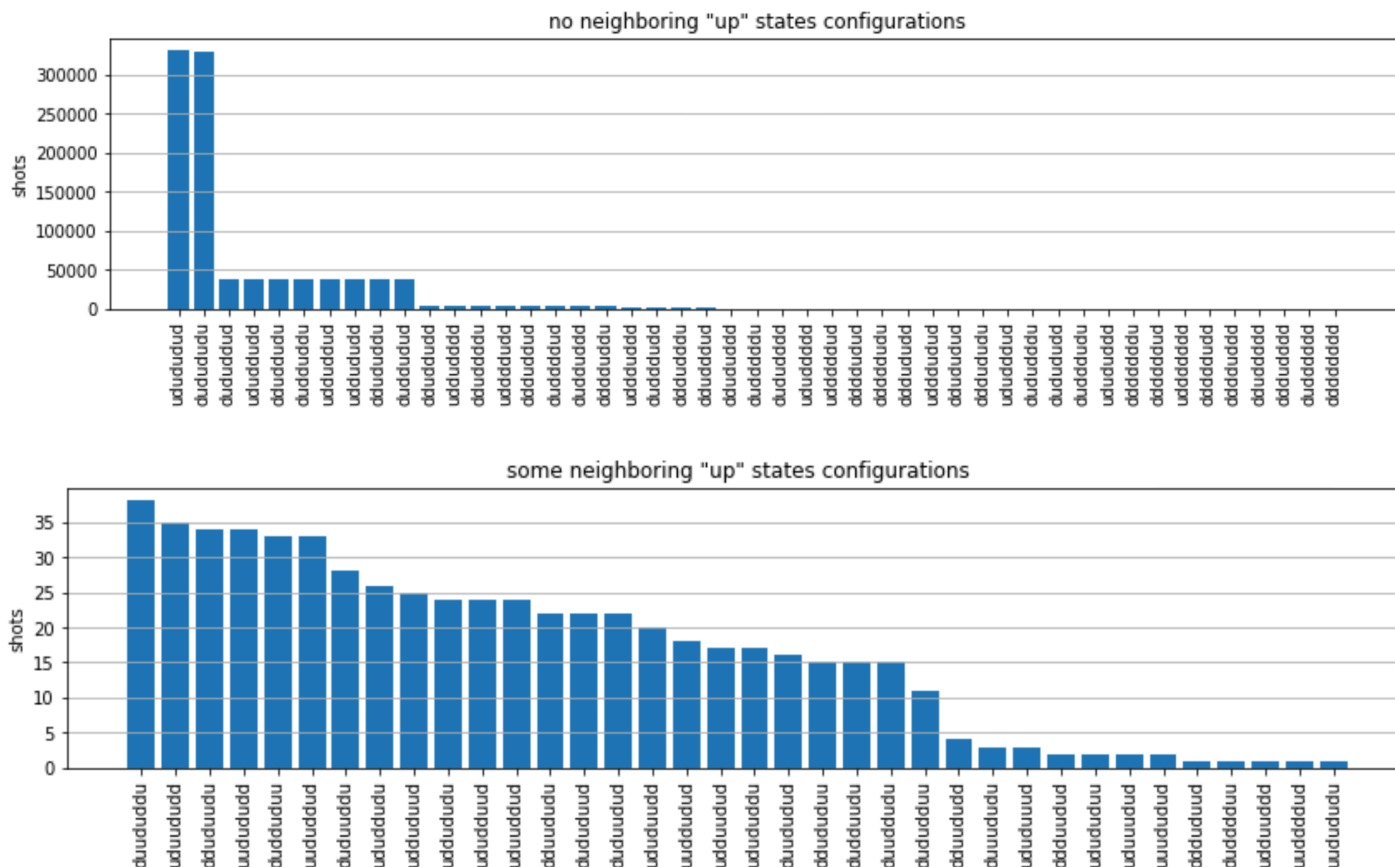
    blockaded.sort(key=lambda _: _[1], reverse=True)
    non_blockaded.sort(key=lambda _: _[1], reverse=True)

    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)
```



从图中我们可以读出以下观察结果，以验证我们成功制备了反铁磁相。

1. 通常，非阻塞状态（没有两个相邻的旋转处于“向上”状态）比至少有一对相邻旋转都处于“向上”状态的状态更为常见。
2. 通常，除非配置被阻止，否则会优先选择激励更多“向上”的状态。
3. 最常见的状态确实是完美的反铁磁态和 "dudududu" "udududud"
4. 第二常见的状态是只有 3 个“向上”激励，连续间隔为 1、2、2 的状态。这表明范德华的相互作用也会对最近的邻居产生影响（尽管要小得多）。

## Run QuEra ning on 的 Aquila QPU

先决条件：除了 pip 安装 Braket [SDK](#) 之外，如果您不熟悉 Amazon Braket，请确保您已完成[必要的入门步骤](#)。

### Note

如果您使用的是 Braket 托管的笔记本实例，则该实例预装了 Braket SDK。

安装完所有依赖项后，我们就可以连接到 Aquila QPU 了。

```
from braket.aws import AwsDevice

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

为了使我们的 AHS 程序适合 QuEra 机器，我们需要对所有值进行四舍五入，以符合 Aquila QPU 允许的精度水平。（这些要求受名称中带有“分辨率”的设备参数的约束。我们可以通过在笔记本 `aquila_qpu.properties.dict()` 中执行来看到它们。有关 Aquila 功能和要求的更多详细信息，请参阅 [Aquila 笔记本简介](#)。）我们可以通过调用 `discretize` 方法来做到这一点。

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

现在我们可以运行该程序（目前只运行 100 张照片）。

### Note

在 Aquila 处理器上运行此程序将产生一定的成本。Amazon Braket SDK 包含一个 [成本追踪器](#)，使客户能够设置成本限额并近乎实时地跟踪成本。

```
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
```

由于量子任务的运行时间差异很大（取决于可用窗口和 QPU 利用率），因此记下量子任务 ARN 是个好主意，这样我们就可以在以后使用以下代码片段检查其状态。

```
# 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
```

状态为“已完成”（也可以从 Amazon Braket [控制台](#) 的量子任务页面进行检查）后，我们可以通过以下方式查询结果：

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

## 分析 QPU 结果

使用与以前相同的 `get_counts` 函数，我们可以计算计数：

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

```
print(counts_aquila)
```

```
*[Output]*
```

```
{'udududud': 24, 'dudududu': 17, 'dududdud': 3, ...}
```

然后用以下方式绘制它们plot\_counts：

```
plot_counts(counts_aquila)
```



请注意，一小部分镜头的场地为空（标有“e”）。这是由于 QPU 的每个原子制备存在 1—2% 的 Aquila 缺陷。除此之外，由于射门数量少，结果在预期的统计波动范围内与仿真相符。

## 下一步

恭喜，你现在已经使用本地 AHS 模拟器和 QPU 在 Amazon Braket 上运行了第一个 AHS 工作负载。Aquila

[要了解有关 Rydberg 物理学、模拟哈密顿仿真和 Aquila 器件的更多信息，请参阅我们的示例笔记本。](#)

## 在 SDK 中构造电路

本节提供了定义电路、查看可用门、扩展电路以及查看每个设备支持的门的示例。它还包含有关如何手动分配 qubits、指示编译器完全按照定义运行电路以及如何使用噪声模拟器构建噪声电路的说明。

你也可以在 Braket 中对具有某些 QPU 的各种门进行脉冲电平工作。有关更多信息，请参阅 [Amazon Braket 上的脉冲控制](#)。

本节内容：

- [大门和电路](#)
- [部分测量](#)
- [手动 qubit 分配](#)
- [逐字汇编](#)
- [噪声模拟](#)
- [检查电路](#)
- [结果类型](#)

### 大门和电路

量子门和电路是在 Amazon Braket Python SDK 的 [braket.circuits](#) 类中定义的。在 SDK 中，你可以通过调用来实例化一个新的电路对象。Circuit()

示例：定义电路

该示例首先定义了一个由四个 qubits（标记为 q0、q1、q2 和 q3）组成的样本电路 q1q2，包括标准的单量子比特 Hadamard 门和两个量子比特 CNOT 门。您可以通过调用 print 函数来可视化此电路，如下例所示。

```
# 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 |

```

### 示例：定义参数化电路

在这个例子中，我们定义了一个电路，其门依赖于自由参数。我们可以指定这些参数的值来创建新电路，或者在提交电路时，在某些设备上作为量子任务运行。

```

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)

```

您可以通过向电路提供单个参数float（这是所有自由参数将采用的值）或关键字参数来指定每个参数的值，从而从参数化电路中创建一个新的非参数化电路，如下所示。

```

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

```

请注意，`my_circuit`这是未修改的，因此您可以使用它来实例化许多具有固定参数值的新电路。

### 示例：修改电路中的门

以下示例定义了一个电路，其门使用控制和功率修改器。您可以使用这些修改来创建新的大门，例如受控Ry门。

```

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)
```

只有本地模拟器支持门控修改器。

示例：查看所有可用的登机口

以下示例说明如何查看 Amazon Braket 中所有可用的门。

```
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)
```

这段代码的输出列出了所有的门。

```
['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']
```

通过调用该类型电路的方法，可以将这些门中的任何一个附加到电路中。例如，你可以打电话 `circ.h(0)`，在第一个门上添加一扇哈达玛德大门。qubit

### Note

门已附加到位，以下示例将上一个示例中列出的所有门添加到同一个电路中。

```
circ = Circuit()
# toffoli gate with q0, q1 the control qubits and q2 the target.
circ.ccnnot(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  $|00\rangle$  state, cphaseshift00(phi) =
diag([exp(1j*phi),1,1,1])
circ.cphaseshift00(0, 1, 0.15)
# controlled-phase gate that phases the  $|01\rangle$  state, cphaseshift01(phi) =
diag([1,exp(1j*phi),1,1])
circ.cphaseshift01(0, 1, 0.15)
# controlled-phase gate that phases the  $|10\rangle$  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)

```

除了预定义的门设置外，您还可以将自定义的单元门应用于电路。它们可以是单量子比特门（如以下源代码所示），也可以是应用于参数qubits定义的多量子比特门。targets

```

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

```

### 示例：扩展现有电路

您可以通过添加指令来扩展现有电路。A Instruction 是一种量子指令，它描述了要在量子器件上执行的量子任务。Instruction运算符Gate仅包括类型的对象。

```

# 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})

```

示例：查看每台设备支持的大门

模拟器支持 Braket SDK 中的所有门，但是 QPU 设备支持的子集较小。你可以在设备属性中找到设备支持的门。以下显示了 ionQ 设备的示例：

```

# 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']

```

支持的门可能需要编译成原生门，然后才能在量子硬件上运行。当您提交电路时，Amazon Braket 会自动执行此编译。

示例：以编程方式检索设备支持的原生门的保真度

您可以在 Braket 控制台的“设备”页面上查看保真度信息。有时，以编程方式访问相同的信息会很有帮助。以下代码显示如何提取 QPU 的两个 qubit 门之间的两门保真度。

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

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

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

## 部分测量

按照前面的示例，我们测量了量子电路中的所有量子比特。但是，可以测量单个量子比特或量子比特的子集。

示例：测量量子比特的子集

在这个例子中，我们通过在电路末端添加一条带有目标量子比特的 `measure` 指令来演示部分测量。

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

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

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

# Get the results
result = task.result()

# Print the circuit and measured qubits
print(circuit)
```

```
print()
print("Measured qubits: ", result.measured_qubits)
```

## 手动qubit分配

当您在量子计算机上运行量子电路时Rigetti，您可以选择使用手动qubit分配来控制qubits哪些用于您的算法。[Amazon Braket 控制台](#)和 [Amazon Braket SDK](#) 可帮助您检查所选量子处理单元 (QPU) 设备的最新校准数据，以便您可以为实验选择最佳校准数据。qubits

手动qubit分配使您能够更准确地运行电路并调查各个qubit特性。研究人员和高级用户可以根据最新的设备校准数据优化其电路设计，从而获得更准确的结果。

以下示例演示了如何qubits显式分配。

```
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)
```

欲了解更多信息，请参阅本笔记本上的[Amazon Braket 示例：在 GitHub QPU 设备上分配量子比特](#)。

### Note

编OQC译器不支持设置disable\_qubit\_rewiring=True。将此标志设置为True会产生以下错误:An error occurred (ValidationException) when calling the CreateQuantumTask operation: Device arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy does not support disabled qubit rewiring.

## 逐字汇编

当您在Rigetti、IonQ或 Oxford Quantum Circuits (OQC) 的量子计算机上运行量子电路时，您可以指示编译器完全按照定义运行你的电路，而无需进行任何修改。使用逐字编译，您可以指定要么精确地保留整个电路（由RigettiIonQ、和支持OQC），要么仅保留其中的特定部分（仅受Rigetti支持）。在为硬件基准测试或错误缓解协议开发算法时，你需要可以选择精确指定在硬件上运行的门和电路布局。Verbatim 编译使您可以通过关闭某些优化步骤来直接控制编译过程，从而确保您的电路完全按照设计运行。

、和 Oxford Quantum Circuits (OQC) 设备目前支持 Verbatim 编译 RigettiIonQ，并且需要使用原生门。使用逐字编译时，建议检查设备的拓扑结构，以确保在连接时调用门，qubits并且电路使用硬件支持的本机门。以下示例说明如何以编程方式访问设备支持的原生门列表。

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

对于 Rigetti，必须通过设置 `disableQubitRewiring=True` 为与逐字编译一起使用来关闭 qubit 重新布线。`disableQubitRewiring=False` 如果在编译中使用逐字记录框时设置，则量子电路验证失败且无法运行。

如果为电路启用了逐字编译并在不支持逐字编译的 QPU 上运行，则会生成错误，表示任务因不支持的操作而导致任务失败。随着越来越多的量子硬件原生支持编译器功能，该功能将扩展到包括这些设备。使用以下代码查询时，支持逐字编译的设备将其列为支持的操作。

```
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
```

使用逐字记录汇编不会产生任何额外费用。对于在 Braket QPU 设备、笔记本实例和按需模拟器上执行的量子任务，您需要继续按照 [Amazon Braket 定价](#) 页面上指定的当前费率付费。有关更多信息，请参阅 [Verbatim 编译](#) 示例笔记本。

### Note

如果您使用 OpenQasm 为 OQC 和 IonQ 设备编写电路，并且希望将电路直接映射到物理量子比特，则需要使用 `#pragma braket verbatim`。OpenQasm 会完全忽略该 `disableQubitRewiring` 标志。

## 噪声模拟

要实例化本地噪声模拟器，你可以按如下方式更改后端。

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

您可以通过两种方式构建噪音电路：

1. 自下而上地建造嘈杂的电路。
2. 采用现有的无噪音电路，并在整个过程中注入噪音。

以下示例显示了使用带有去极化噪声和自定义 Kraus 通道的简单电路的方法。

```
# 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)
```

运行电路的用户体验与以前相同，如以下两个示例所示。

#### 示例 1

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

Or

#### 示例 2

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

有关更多示例，请参阅 [Braket 入门噪声模拟器](#) 示例

## 检查电路

Amazon Braket 中的量子电路有一个名为“伪时间”的概念。Moments 每人 qubit 只能体验一扇大门 Moment。的 Moments 目的是使电路及其大门更易于寻址，并提供时间结构。



**Note**

时刻通常不对应于 QPU 上执行大门的实时时间。

电路的深度由该电路中的总矩数给出。您可以查看调用该方法的电路深度 `circuit.depth`，如以下示例所示。

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

```
T : | 0 | 1 |2|
q0 : -Rx(0.15)-C-----X-
      |
q1 : -Ry(0.2)--|-ZZ(0.15)---
      | |
q2 : -----X-|------
      |
q3 : -----ZZ(0.15)---
T : | 0 | 1 |2|
Total circuit depth: 3
```

上面电路的总电路深度为 3 (显示为力矩 01、和 2)。您可以查看每时每刻的门操作情况。

`Moments` 函数作为键值对的字典。

- 密钥是 `MomentsKey()`，它包含伪时间和信息。qubit
- 该值的分配类型为 `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)]))

```

您也可以为电路添加大门Moments。

```

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

```

```

T : |0|1|2|

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

T : |0|1|2|

```

## 结果类型

Amazon使用ResultType测量电路时，Braket 可以返回不同类型的结果。电路可以返回以下类型的结果。

- `AdjointGradient`返回所提供的可观测值的期望值的梯度（向量导数）。该可观察对象使用伴随微分法根据指定参数对提供的目标起作用。只有在 `shots=0` 时才能使用此方法。
- `Amplitude`返回输出波函数中指定量子态的振幅。它仅在SV1和本地模拟器上可用。
- `Expectation`返回给定可观察对象的期望值，该值可以通过本章后面介绍的`Observable`类来指定。必须指定qubits用于测量可观测值的目标，并且指定目标的数量必须等于可观察对象所针对的qubits数量。如果未指定目标，则可观察对象必须仅在 1 上运行，qubit并行应用qubits于所有目标。
- `Probability`返回测量计算基态的概率。如果未指定目标，则`Probability`返回测量所有基态的概率。如果指定了目标，则仅返回指定qubits基向量的边际概率。
- `Reduced density matrix`返回系统中指定目标qubits子系统的密度矩阵。qubits为了限制此结果类型的大小，Braket 将目标数量限制qubits为最多 8。
- `StateVector`返回完整的状态向量。它可在本地模拟器上使用。
- `Sample`返回指定目标qubit集和可观察对象的测量计数。如果未指定目标，则可观察对象必须仅在 1 上运行，qubit并行应用qubits于所有目标。如果指定了目标，则指定目标的数量必须等于可观察对象作用的数量。qubits
- `Variance`返回指定目标qubit集的方差 ( $\text{mean}([x - \text{mean}(x)]^2)$ )，并作为请求的结果类型进行观察。如果未指定目标，则可观察对象必须仅在 1 上运行，qubit并行应用qubits于所有目标。否则，指定的目标数量必须等于可以应用可观察对象的数量。qubits

不同设备支持的结果类型：

	本地模拟卡	SV1	DM1	TN1	Rigetti	IonQ	OQC
伴随渐变	否	Y	否	否	否	否	否
Amplitude	Y	Y	否	否	否	否	否
期望	Y	Y	Y	Y	Y	Y	Y
Probability	Y	Y	Y	否	是*	Y	Y
低密度矩阵	Y	否	Y	否	否	否	否
状态向量	Y	否	否	否	否	否	否

样本	Y	Y	Y	Y	Y	Y	Y
方差	Y	Y	Y	Y	Y	Y	Y

### Note

\* Rigetti 仅支持最多 40 的概率结果类型 qubits。

您可以通过检查设备属性来检查支持的结果类型，如以下示例所示。

```
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
```

要调用 `aResultType`，请将其附加到电路中，如以下示例所示。

```
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**

例如Rigetti，有些设备将测量结果作为结果提供，而另一些设备则提供概率作为结果（例如IonQ和OQC）。SDK在结果上提供了测量属性，但对于返回概率的设备，该属性是事后计算的。因此，诸如由提供的设备IonQ和OQC其测量结果由概率决定的设备，因为不会返回每次拍摄的测量结果。您可以通过查看结果对象measurements\_copied\_from\_device上的（如[本文件](#)所示）来检查结果是否经过后期计算。

## 可观察对象

AmazonBraket 包括一个Observable类，该类可用于指定要测量的观测值。

您最多可以对每个应用一个可观察到的唯一非身份。qubit如果您将两个或多个不同的非身份可观察对象指定为相同的对象qubit，则会看到错误。为此，张量乘积的每个因子都算作一个单独的可观测值，因此允许有多个张量乘积作用于同一个因子qubit，前提是作用于该因子的因子相同。qubit

您还可以缩放可观测值并添加可观察对象（无论是否缩放）。Sum这将创建可在AdjointGradient结果类型中使用的。

该Observable类包括以下可观察对象。

```
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)))
```

## 参数

电路可能包含自由参数，您可以以“构造一次——运行多次”的方式使用这些参数并计算梯度。自由参数具有字符串编码的名称，您可以使用该名称来指定其值或确定是否对其进行区分。

```
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])
```

对于要区分的参数，请使用其名称（作为字符串）或通过直接引用来指定它们。请注意，使用AdjointGradient结果类型计算梯度是相对于可观测值的期望值完成的。

**注意：**如果您通过将自由参数的值作为参数传递给参数化电路来固定这些值，则运行指定了结果类型和参数的电路将产生错误。AdjointGradient这是因为我们用来区分的参数已不复存在。请参阅以下示例。

```
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
```

## 向 QPU 和模拟器提交量子任务

Amazon Braket 提供对多个可以运行量子任务的设备的访问权限。您可以单独提交量子任务，也可以设置量子任务批处理。

### QPU

您可以随时向 QPU 提交量子任务，但该任务在 Amazon Braket 控制台的“设备”页面上显示的特定可用性窗口内运行。您可以使用量子任务 ID 检索量子任务的结果，下一节将对此进行介绍。

- 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 ( 仅限预订 ) : `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`

### 模拟器

- 密度矩阵模拟器，DM1: `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- 状态向量模拟器，SV1: `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- 张量网络模拟器，TN1: `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- 本地模拟器 : `LocalSimulator()`

#### Note

您可以取消处于该CREATED状态的 QPU 和按需模拟器的量子任务。对于按需模拟器和 QPU，您可以尽最大努力取消该QUEUED州的量子任务。请注意，在 QPU 可用性窗口期间，QPU QUEUED 量子任务不太可能成功取消。

本节内容：

- [Amazon Braket 上的量子任务示例](#)
- [向 QPU 提交量子任务](#)
- [使用本地仿真器运行量子任务](#)
- [量子任务批处理](#)
- [设置 SNS 通知 \( 可选 \)](#)
- [检查编译后的电路](#)

## Amazon Braket 上的量子任务示例

本节介绍运行示例量子任务的各个阶段，从选择设备到查看结果。作为 Amazon Braket 的最佳实践，我们建议您首先在模拟器上运行电路，例如SV1。

本节内容：

- [指定设备](#)
- [提交量子任务示例](#)
- [提交参数化任务](#)
- [指定 shots](#)
- [投票结果](#)
- [查看示例结果](#)

### 指定设备

首先，为您的量子任务选择并指定设备。此示例说明如何选择模拟器SV1。

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

您可以按如下方式查看此设备的某些属性：

```
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)])
```

## 提交量子任务示例

提交要在按需模拟器上运行的量子任务示例。

```
# 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()
```

该`device.run()`命令通过 `CreateQuantumTask` [API](#) 创建量子任务。在短暂的初始化时间后，量子任务将排队，直到有能力在设备上运行量子任务。在本例中，设备是SV1。设备完成计算

后，Amazon Braket 会将结果写入调用中指定的 Amazon S3 位置。除本地模拟器 `s3_location` 之外的所有设备都需要位置参数。

### Note

Braket 量子任务动作的大小限制为 3MB。

## 提交参数化任务

Amazon Braket 按需模拟器和本地模拟器以及 QPU 还支持在提交任务时指定免费参数的值。您可以通过使用 `to` 的 `inputs` 参数来执行此操作 `device.run()`，如以下示例所示。`inputs` 必须是字符串浮点对的字典，其中键是参数名。

参数化编译可以提高在某些 QPU 上执行参数电路的性能。将参数电路作为量子任务提交给支持的 QPU 时，Braket 将编译电路一次，然后缓存结果。对于同一电路的后续参数更新，无需重新编译，从而缩短了使用相同电路的任务的运行时间。在编译电路时，Braket 会自动使用硬件提供商提供的更新的校准数据，以确保获得最高质量的结果。

### Note

所有来自脉冲电平程序的超导、基于栅极的 QPU 都支持参数编译，Rigetti Computing 脉冲电 Oxford Quantum Circuits 程序除外。

```
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})
```

## 指定 shots

shots参数指的是所需的测量次数shots。诸如之类的模拟器SV1支持两种仿真模式。

- 对于 shots = 0，模拟器执行精确模拟，返回所有结果类型的真实值。（不适用于TN1。）
- 对于的非零值shots，仿真器会从输出分布中采样以模拟真实 QPU 的shot噪声。QPU 设备仅允许 shots > 0。

有关每个量子任务的最大射门数的信息，请参阅 [Braket 配额](#)。

## 投票结果

执行时my\_task.result()，SDK 开始使用您在创建量子任务时定义参数轮询结果：

- poll\_timeout\_seconds是在按需模拟器和/或 QPU 设备上运行量子任务时，在量子任务超时之前对其进行轮询的秒数。默认值为 432,000 秒，即 5 天。
- 注意：对于Rigetti和之类的 QPU 设备IonQ，我们建议您留出几天的时间。如果您的轮询超时时间太短，则可能无法在轮询时间内返回结果。例如，当 QPU 不可用时，会返回本地超时错误。
- poll\_interval\_seconds是对量子任务进行轮询的频率。它指定了在按需模拟器和 QPU API 设备上运行量子任务时，你多久调用 Braket 以获取状态。默认值为 1 秒。

这种异步执行便于与并非总是可用的 QPU 设备进行交互。例如，在常规维护时段内，设备可能不可用。

返回的结果包含一系列与量子任务相关的元数据。您可以使用以下命令检查测量结果：

```
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}
```

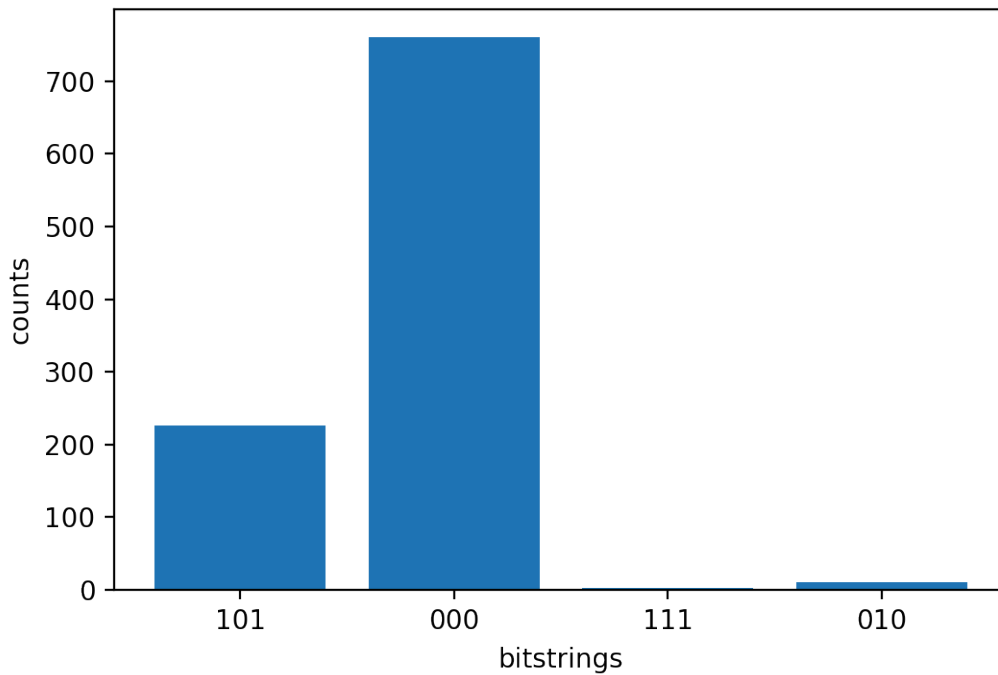
## 查看示例结果

由于您还指定了 `ResultType`，因此您可以查看返回的结果。结果类型按添加到电路中的顺序出现。

```
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]
```



## 向 QPU 提交量子任务

Amazon Braket 允许你在 QPU 设备上运行量子电路。以下示例说明如何向 Rigetti 或 IonQ 设备提交量子任务。

选择 Rigetti Aspen-M-3 设备，然后查看相关的连接图

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

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

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

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

前面的字典connectivityGraph包含有关当前Rigetti设备连接的信息。

### 选择IonQ Harmony设备

对于该IonQ Harmony设备，connectivityGraph为空，如以下示例所示，因为该设备提供全方位连接。因此，connectivityGraph不需要详细说明。

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

如以下示例所示，如果您选择指定默认存储桶以外的其他位置，则可以选择调整 `poll_timeout_seconds` (默认值 = 1000)、`poll_interval_seconds` (默认 = 432000 = 5 天 `s3_location`)、(默认 = 1) 和 S3 存储桶的位置 ( )。shots

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

IonQ和Rigetti设备会自动将提供的电路编译成各自的原生门极集，并将抽象qubit索引映射到相应的QPU qubits 上的物理索引。

### Note

QPU 设备的容量有限。达到容量后，等待时间可能会更长。

Amazon Braket 可以在特定的可用性窗口内运行 QPU 量子任务，但您仍然可以随时 (全天候) 提交量子任务，因为所有相应的数据和元数据都可靠地存储在相应的 S3 存储桶中。如下一节所示，您可以使用 `AwsQuantumTask` 和你唯一的量子任务 ID 来恢复你的量子任务。

## 使用本地仿真器运行量子任务

您可以将量子任务直接发送到本地仿真器，以进行快速原型设计和测试。此模拟器在您的本地环境中运行，因此您无需指定 Amazon S3 位置。结果是在您的会话中直接计算出来的。要在本地仿真器上运行量子任务，您只需指定 `shots` 参数即可。

### Note

本地模拟器可以处理 qubits 的执行速度和最大数量取决于 Amazon Braket 笔记本实例类型或本地硬件规格。

以下命令全部相同，用于实例化状态向量 (无噪音) 本地模拟器。

```
# 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")
```

然后使用以下命令运行量子任务。

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

要实例化局部密度矩阵（噪声）仿真器，客户需要按如下方式更改后端。

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

## 在本地仿真器上测量特定的量子比特

局部状态向量模拟器和局部密度矩阵仿真器支持运行电路，在这些电路中，可以测量电路的量子比特子集，这通常称为部分测量。

例如，在下面的代码中，您可以创建一个双量子比特电路，并且只能通过添加一条带有目标量子比特的measure指令来测量第一个量子比特。

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

# Use the local simulator device
device = LocalSimulator()

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

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

# Get the results
result = task.result()

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



## 量子任务批处理

Quantum 任务批处理可在每台 Amazon Braket 设备上使用，本地模拟器除外。批处理对于在按需模拟器 (TN1 或 SV1) 上运行的量子任务特别有用，因为它们可以并行处理多个量子任务。为了帮助您设置各种量子任务，Amazon Braket 提供了 [示例](#) 笔记本。

批处理允许您并行启动量子任务。例如，如果你想进行需要 10 个量子任务的计算，并且这些量子任务中的电路相互独立，那么使用批处理是个好主意。这样，您就不必等待一项量子任务完成后再开始另一项任务。

以下示例显示如何运行一批量子任务：

```
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
```

有关更多信息，请参阅有关 [Quantum 任务批处理的 Amazon Braket 示例](#)，其中包含有关批处理的更多具体信息。GitHub

### 关于量子任务批处理和成本

关于量子任务批处理和计费成本，请记住一些注意事项：

- 默认情况下，量子任务批处理重试全部超时或量子任务失败 3 次。
- 一批长时间运行的量子任务 (qubits 例如 SV1 34 for) 可能会产生高昂的成本。在开始一批量子 `run_batch` 任务之前，请务必仔细检查分配值。我们不建议 TN1 搭配使用 `run_batch`。
- TN1 排练阶段失败的任务可能会产生费用 (有关更多信息，[请参阅 TN1 说明](#))。自动重试可能会增加成本，因此我们建议在使用时将批处理时的 “`max_retries`” 次数设置为 0 TN1 (参见 [Quantum Task Batching](#)，第 186 行)。

### 量子任务批处理和 PennyLane

在 Braket PennyLane 上使用时，通过设置 `parallel = True` 实例化 Amazon Braket 设备的时间，利用批处理的 Amazon 优势，如以下示例所示。

```
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,)
```

有关批处理的更多信息 PennyLane，请参阅量子电路的[并行优化](#)。

## 任务批处理和参数化电路

在提交包含参数化电路的量子任务批处理时，您可以提供一个inputs字典（用于批次中的所有量子任务），也可以提供一个list输入字典，在这种情况下，i第-th字典与第-th个任务配对，如以下示例所示。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)
```

您还可以为单个参数电路准备输入字典列表，然后将其作为量子任务批量提交。如果列表中有 N 个输入字典，则该批次包含 N 个量子任务。i第 n 个量子任务对应于使用i第-t 个输入字典执行的电路。

```
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)
```

## 设置 SNS 通知 ( 可选 )

您可以通过亚马逊简单通知服务 (SNS) Simple Notification Service 设置通知，以便在 Amazon Braket 量子任务完成时收到提醒。如果您预计等待时间很长，则活动通知很有用；例如，当您提交大型量子任务或在设备可用性窗口之外提交量子任务时。如果您不想等待量子任务完成，则可以设置 SNS 通知。

Amazon Braket 笔记本将引导您完成设置步骤。有关更多信息，请参阅[上的 Amazon Braket 示例 GitHub](#)，特别是[用于设置通知的笔记本示例](#)。

## 检查编译后的电路

当电路在硬件设备上运行时，必须以可接受的格式对其进行编译，例如将电路向下转换到 QPU 支持的本机门。检查实际的编译输出对于调试目的非常有用。您可以使用下面的代码查看这两者的电路 Rigetti 和 OQC 设备。

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

### Note

如今，您无法查看已编译的 IonQ 设备电路。

## 使用 OpenQasm 3.0 运行你的赛道

Amazon Braket 现在支持用于基于门的量子设备和模拟器的 [OpenQasm 3.0](#)。本用户指南提供了有关 Braket 支持的 OpenQasm 3.0 子集的信息。[Braket 客户现在可以选择使用软件开发工具包提交 Braket 电路，也可以使用亚马逊 Braket API 和 Amazon Braket Python SDK 直接向所有基于门禁的设备提供 OpenQasm 3.0 字符串。](#)

本指南中的主题将引导您了解如何完成以下量子任务的各种示例。

- [在不同的 Braket 设备上创建和提交 OpenQasm 量子任务](#)

- [访问支持的操作和结果类型](#)
- [使用 OpenQasm 模拟噪声](#)
- [在 OpenQasm 中使用逐字编译](#)
- [解决 OpenQasm 问题](#)

本指南还介绍了某些特定于硬件的功能，这些功能可以通过 Braket 上的 OpenQasm 3.0 实现，并提供了更多资源的链接。

本节内容：

- [什么是 OpenQasm 3.0 ?](#)
- [何时使用 OpenQasm 3.0](#)
- [OpenQasm 3.0 的工作原理](#)
- [先决条件](#)
- [Braket 支持哪些 OpenQasm 功能？](#)
- [创建并提交示例 OpenQasm 3.0 量子任务](#)
- [在不同的 Braket 设备上支持 OpenQasm](#)
- [使用 OpenQasm 3.0 模拟噪声](#)
- [Qubit使用 OpenQasm 3.0 重新布线](#)
- [使用 OpenQasm 3.0 进行逐字编译](#)
- [Braket 控制台](#)
- [更多资源](#)
- [使用 OpenQasm 3.0 计算梯度](#)
- [使用 OpenQasm 3.0 测量特定的量子比特](#)

## 什么是 OpenQasm 3.0 ?

开放量子汇编语言 (OpenQASM) 是量子指令的[中间表示形式](#)。OpenQasm 是一个开源框架，广泛用于规范基于门的设备的量子程序。使用 OpenQasm，用户可以对构成量子计算基块的量子门和测量操作进行编程。许多量子编程库都使用先前版本的 OpenQasm (2.0) 来描述简单的程序。

新版本的 OpenQasm (3.0) 扩展了之前的版本，增加了更多功能，例如脉冲电平控制、门定时和经典控制流，以弥合最终用户界面和硬件描述语言之间的差距。当前版本 3.0 的详细信息和规格可在 GitHub

[OpenQasm 3.x](#) 实时规格中找到。OpenQasm 的未来发展由 OpenQasm 3.0 [技术指导委员会管理](#)，该委员会与 IBM、微软和因斯布鲁克大学一起 AWS 是该委员会的成员。

## 何时使用 OpenQasm 3.0

OpenQasm 提供了一个富有表现力的框架，可通过非特定架构的低级控件来指定量子程序，因此非常适合作为多个基于门的设备的表示形式。Braket 对 OpenQasm 的支持进一步推动了其作为开发基于门的量子算法的一致方法的采用，从而减少了用户在多个框架中学习和维护库的需求。

如果您在 OpenQasm 3.0 中已有程序库，则可以对其进行调整以使其与 Braket 配合使用，而不必完全重写这些电路。研究人员和开发人员还应受益于越来越多的支持 OpenQasm 算法开发的可用第三方库。

## OpenQasm 3.0 的工作原理

Braket 对 OpenQasm 3.0 的支持提供了与当前中间表示法相同的功能。这意味着，你今天在硬件设备和使用 Braket 的按需模拟器上能做的任何事情，都可以使用 Braket 在 OpenQasm 上做任何事情。API 您可以通过直接向所有基于门的设备提供 OpenQasm 字符串来运行 OpenQasm 3.0 程序，其方式类似于当前向 Braket 上的设备提供电路的方式。Braket 用户还可以集成支持 OpenQasm 3.0 的第三方库。本指南的其余部分详细介绍了如何开发用于 Braket 的 OpenQasm 表示形式。

## 先决条件

[要在 Braket 上使用 OpenQasm 3.0，您必须有 Amazon Amazon Braket Python Schemas 的 1.8.0 版本和 1.17.0 或更高版本的 Amazon Braket Python SDK。](#)

如果您是首次使用 Braket 的用户，则需要启用 Amazon Braket。有关说明，请参阅[启用 Amazon Braket](#)。

## Braket 支持哪些 OpenQasm 功能？

下一节列出了 Braket 支持的 OpenQasm 3.0 数据类型、语句和编译指令。

本节内容：

- [支持的 OpenQasm 数据类型](#)
- [支持的 OpenQasm 语句](#)
- [Braket openQasm 编译指示](#)
- [本地模拟器上对 OpenQasm 的高级功能支持](#)
- [支持的操作和语法 OpenPulse](#)

## 支持的 OpenQasm 数据类型

Braket 支持以下 OpenQasm 数据类型。Amazon

- 非负整数用于 ( 虚拟和物理 ) 量子比特索引 :
  - `cnot q[0], q[1];`
  - `h $0;`
- 浮点数或常数可用于浇口旋转角度 :
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

### Note

pi 是 OpenQasm 中的内置常量，不能用作参数名称。

- 在用于定义一般赫米特观测值的结果类型编译指示和单一编译用中，允许使用复数数组 ( 虚数部分使用 OpenQasm `im` 表示法 ) :
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

## 支持的 OpenQasm 语句

Braket 支持以下 OpenQasm 语句。Amazon

- Header: `OPENQASM 3;`
- 经典位声明 :
  - `bit b1; ( 等同于 , ) creg b1;`
  - `bit[10] b2; ( 等同于 , ) creg b2[10];`
- 量子比特声明 :
  - `qubit b1; ( 等同于 , ) qreg b1;`
  - `qubit[10] b2; ( 等同于 , ) qreg b2[10];`
- 在数组内建立索引 : `q[0]`
- 输入 : `input float alpha;`

- 物理规格qubits : \$0
- 设备上支持的门禁和操作 :
  - h \$0;
  - iswap q[0], q[1];

### Note

设备支持的门可以在 OpenQasm 操作的设备属性中找到；使用这些门不需要任何门定义。

- 逐字记录声明。目前，我们不支持方框持续时间表示法。原生大门和实体qubits门必须放在逐字记录框中。

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

- 在qubits或整个qubit寄存器上进行测量和测量分配。
  - measure \$0;
  - measure q;
  - measure q[0];
  - b = measure q;
  - measure q # b;

### Note

pi 是 OpenQasm 中的内置常量，不能用作参数名称。

## Braket openQasm 编译指示

Braket 支持以下 OpenQasm 编译指示指令。Amazon

- 噪音编译指示

- `#pragma braket noise bit_flip(0.2) q[0]`
- `#pragma braket noise phase_flip(0.1) q[0]`
- `#pragma braket noise pauli_channel`
- Verbatim pragmas
  - `#pragma braket verbatim`
- 结果类型编译指示
  - 基数不变结果类型 :
    - 状态向量 : `#pragma braket result state_vector`
    - 密度矩阵 : `#pragma braket result density_matrix`
  - 梯度计算实用程序 :
    - 伴随渐变 : `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Z 基准结果类型 :
    - 振幅 : `#pragma braket result amplitude "01"`
    - 概率 : `#pragma braket result probability q[0], q[1]`
  - 基础轮换结果类型
    - 期望 : `#pragma braket result expectation x(q[0]) @ y([q1])`
    - 方差 : `#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
    - 示例 : `#pragma braket result sample h($1)`

### Note

OpenQasm 3.0 向后兼容 OpenQasm 2.0，因此使用 2.0 编写的程序可以在 Braket 上运行。但是，Braket 支持的 OpenQasm 3.0 的功能确实存在一些细微的语法差异，例如 `vs` 和 `qreg v creg s`。qubit bit 测量语法也有差异，需要用正确的语法来支持这些语法。

## 本地模拟器上对 OpenQasm 的高级功能支持

LocalSimulator 支持高级 OpenQasm 功能，这些功能不是作为 Braket 的 QPU 或按需模拟器的一部分提供的。以下功能列表仅在中受支持 LocalSimulator :

Braket 支持哪些 OpenQasm 功能？



- 大门修改器
- OpenQasm 内置大门
- 经典变量
- 经典运算
- 定制大门
- 经典控制
- QASM 文件
- 子例程

有关每项高级功能的示例，请参阅此[示例笔记本](#)。有关完整的 OpenQasm 规范，请访问 [OpenQasm](#) 网站。

## 支持的操作和语法 OpenPulse

支持 OpenPulse 的数据类型

校准方块：

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

Defcal 方块：

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

镜架 :

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

波形 :

```
// 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};
```

自定义栅极校准示例 :

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

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

defcal my_cz $1, $0 {
  barrier q0_q1_cz_frame, q0_rf_frame;
  play(q0_q1_cz_frame, wf1);
  delay[300ns] q0_rf_frame
  shift_phase(q0_rf_frame, 4.366186381749424);
  delay[300ns] q0_rf_frame;
  shift_phase(q0_rf_frame.phase, 5.916747563126659);
  barrier q0_q1_cz_frame, q0_rf_frame;
  shift_phase(q0_q1_cz_frame, 2.183093190874712);
}

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

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

任意脉冲示例：

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

## 创建并提交示例 OpenQasm 3.0 量子任务

你可以使用 Amazon Braket Python SDK、Boto3 或 boto3 向 Braket AWS CLI 设备提交 OpenQasm 3.0 量子任务。Amazon

本节内容：

- [一个 OpenQasm 3.0 程序的示例](#)
- [使用 Python SDK 创建 OpenQasm 3.0 量子任务](#)
- [使用 Boto3 创建 OpenQasm 3.0 量子任务](#)
- [使用创建 OpenQasm 3.0 任务 AWS CLI](#)

### 一个 OpenQasm 3.0 程序的示例

要创建 OpenQasm 3.0 任务，你可以从一个简单的 OpenQasm 3.0 程序 (ghz.qasm) 开始，该程序可以准备 [GHZ](#) 状态，如以下示例所示。

```
// 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;
```

## 使用 Python SDK 创建 OpenQasm 3.0 量子任务

您可以使用以下代码使用 [Amazon Braket Python 软件开发工具包](#) 将此程序提交到 Amazon Braket 设备。

```
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,  
)
```

## 使用 Boto3 创建 OpenQasm 3.0 量子任务

你也可以使用适用于 [Braket 的 AWS Python SDK \(Boto3\)](#) 使用 OpenQasm 3.0 字符串创建量子任务，如以下示例所示。以下代码片段引用了 `ghz.qasm`，它准备了 [GHZ](#) 状态，如上所示。

```
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,
)
```

## 使用创建 OpenQasm 3.0 任务 AWS CLI

[AWS Command Line Interface \(CLI\)](#) 还可用于提交 OpenQasm 3.0 程序，如以下示例所示。

```
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)
}'
```

## 在不同的 Braket 设备上支持 OpenQasm

对于支持 OpenQasm 3.0 的设备，该action字段支持通过GetDevice响应执行新操作，如以下 Rigetti和IonQ设备的示例所示。

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

```
    ],  
    ...  
  }  
}  
}
```

对于支持脉冲控制的设备，该pulse字段显示在GetDevice响应中。以下示例显示了Rigetti和OQC设备的此pulse字段。

```
// Rigetti  
{  
  "pulse": {  
    "braketSchemaHeader": {  
      "name": "braket.device_schema.pulse.pulse_device_action_properties",  
      "version": "1"  
    },  
    "supportedQhpTemplateWaveforms": {  
      "constant": {  
        "functionName": "constant",  
        "arguments": [  
          {  
            "name": "length",  
            "type": "float",  
            "optional": false  
          },  
          {  
            "name": "iq",  
            "type": "complex",  
            "optional": false  
          }  
        ]  
      },  
      ...  
    },  
    "ports": {  
      "q0_ff": {  
        "portId": "q0_ff",  
        "direction": "tx",  
        "portType": "ff",  
        "dt": 1e-9,  
        "centerFrequencies": [  
          375000000  
        ]  
      }  
    }  
  }  
}
```

```
    },
    ...
  },
  "supportedFunctions": {
    "shift_phase": {
      "functionName": "shift_phase",
      "arguments": [
        {
          "name": "frame",
          "type": "frame",
          "optional": false
        },
        {
          "name": "phase",
          "type": "float",
          "optional": false
        }
      ]
    },
    ...
  },
  "frames": {
    "q0_q1_cphase_frame": {
      "frameId": "q0_q1_cphase_frame",
      "portId": "q0_ff",
      "frequency": 462475694.24460185,
      "centerFrequency": 375000000,
      "phase": 0,
      "associatedGate": "cphase",
      "qubitMappings": [
        0,
        1
      ]
    },
    ...
  },
  "supportsLocalPulseElements": false,
  "supportsDynamicFrames": false,
  "supportsNonNativeGatesWithPulses": false,
  "validationParameters": {
    "MAX_SCALE": 4,
    "MAX_AMPLITUDE": 1,
    "PERMITTED_FREQUENCY_DIFFERENCE": 400000000
  }
}
```



```
}
}

// OQC

{
  "pulse": {
    "braketSchemaHeader": {
      "name": "braket.device_schema.pulse.pulse_device_action_properties",
      "version": "1"
    },
    "supportedQhpTemplateWaveforms": {
      "gaussian": {
        "functionName": "gaussian",
        "arguments": [
          {
            "name": "length",
            "type": "float",
            "optional": false
          },
          {
            "name": "sigma",
            "type": "float",
            "optional": false
          },
          {
            "name": "amplitude",
            "type": "float",
            "optional": true
          },
          {
            "name": "zero_at_edges",
            "type": "bool",
            "optional": true
          }
        ]
      },
      ...
    },
    "ports": {
      "channel_1": {
        "portId": "channel_1",
        "direction": "tx",
        "portType": "port_type_1",

```

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

```

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

```

前面的字段详细说明了以下内容：

端口：

描述在 QPU 上声明的预制外部 (extern) 设备端口，以及给定端口的相关属性。此结构中列出的所有端口都预先声明为用户提交的 OpenQASM 3.0 程序中的有效标识符。端口的其他属性包括：

- 端口 ID (端口 ID)
  - 在 OpenQasm 3.0 中声明为标识符的端口名称。
- 方向 (方向)
  - 港口的方向。驱动端口传输脉冲 (方向 “tx”)，而测量端口接收脉冲 (方向 “rx”)。
- 端口类型 (端口类型)
  - 此端口负责的操作类型 (例如，驱动、捕获或 ff-fast-flux)。
- Dt (dt)
  - 以秒为单位的时间，表示给定端口上的单个采样时间步长。
- 量子比特映射 (QubitMappings)
  - 与给定端口关联的量子比特。
- 中心频率 (中心频率)
  - 端口上所有预先声明或用户定义的帧的相关中心频率列表。有关更多信息，请参阅框架。
- QHP 特定属性 (q SpecificProperties hp)
  - 一张可选地图，详细介绍有关 QHP 特定端口的现有属性。

框架：

描述在 QPU 上声明的预制外部帧以及与这些帧相关的属性。此结构中列出的所有帧都预先声明为用户提交的 OpenQASM 3.0 程序中的有效标识符。框架的其他属性包括：

- 帧编号 (frameID)
  - 在 OpenQasm 3.0 中声明为标识符的帧名称。
- 端口 ID (端口 ID)
  - 框架的关联硬件端口。
- 频率 (频率)
  - 帧的默认初始频率。
- 中心频率 (中心频率)
  - 帧频率带宽的中心。通常，只能将帧调整到中心频率周围的特定带宽。因此，频率调整应保持在中心频率的给定增量之内。您可以在验证参数中找到带宽值。
- 阶段 (阶段)
  - 帧的默认初始阶段。
- 关联门 (AssociatedGate)
  - 与给定帧关联的大门。
- 量子比特映射 (QubitMappings)
  - 与给定帧相关的量子比特。
- QHP 特定属性 (q SpecificProperties hp)
  - 一张可选地图，详细介绍有关特定于 QHP 的帧的现有属性。

SupportsDynamic 镜架：

描述是否可以在函数中声明帧 `cal` 或通过 `OpenPulse.newframe` 函数 `defcal` 阻塞帧。如果该值为 `false`，则只能在程序中使用框架结构中列出的框架。

SupportedFunctions:

除了给定 `OpenPulse` 函数的关联参数、参数类型和返回类型之外，还描述设备支持的函数。要查看使用这些 `OpenPulse` 函数的示例，请参阅 [OpenPulse 规范](#)。目前，Braket 支持：

- 移相
  - 按指定值将帧的相位移动
- `set_phase`
  - 将帧的相位设置为指定值

- 移位频率
  - 按指定值移动帧的频率
- 设置频率
  - 将帧频设置为指定值
- play
  - 安排波形
- capture\_v0
  - 将捕获帧上的值返回到寄存器

### SupportedQhpTemplateWaveforms:

描述设备上可用的预建波形函数以及相关的参数和类型。默认情况下，Braket Pulse 在所有设备上提供预先构建的波形例程，它们是：

#### 常量

$$\text{Constant}(t, \tau, iq) = iq$$

$\tau$ 是波形的长度， $iq$ 是一个复数。

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

#### 高斯

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

$\tau$ 是波形的长度， $\sigma$ 是高斯的宽度， $A$ 是振幅。如果设置为True，则 $ZaE$ 对高斯进行偏移和重新缩放，使其在波形的开头和结尾处等于零，并达到最大值。 $A$

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

#### DRAG Gaussian

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

$\tau$ 是波形的长度， $\sigma$ 是高斯的宽度， $\beta$ 是自由参数， $A$ 是振幅。如果设置`ZaE`为`True`，则偏移并重新调整绝热门 (DRAG) Gaussian 的导数去除，使其在波形的开头和结尾处等于零，实数部分达到最大值。 $A$ 有关阻力波形的更多信息，请参阅 paper 论文《[消除弱非线性量子比特中泄漏的简单脉冲](#)》。

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

`SupportsLocalPulseElements`:

描述脉冲元素（例如端口、帧和波形）是否可以在`defcal`块中本地定义。如果值为`false`，则必须以`cal`块形式定义元素。

`SupportsNonNativeGatesWithPulses`:

描述我们是否可以将非原生门与脉冲程序结合使用。例如，如果不先为所使用的量子比特定义门通H，我们就不能像程序中的门一样使用非原生门。`defcal`你可以在设备功能下方找到原生门`nativeGateSet`键列表。

`ValidationParameters`:

描述脉冲元件验证边界，包括：

- 波形的最大缩放/最大振幅值（任意和预先构建）
- 所提供中心频率的最大频率带宽（以 Hz 为单位）
- 最小脉冲长度/持续时间（以秒为单位）
- 最大脉冲长度/持续时间（以秒为单位）

## OpenQasm 支持的操作、结果和结果类型

要了解每台设备支持哪些 OpenQasm 3.0 功能，可以参考设备功能输出`action`字段中的`braket.ir.openqasm.program`密钥。例如，以下是 Braket 状态矢量仿真器SV1支持的操作和结果类型。

```
...
  "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
  }
]
}
},
...

```

## 使用 OpenQasm 3.0 模拟噪声

要使用 OpenQasm3 模拟噪声，您可以使用编译指示来添加噪声运算符。例如，要模拟之前提供的 [GHZ 程序](#) 的噪音版本，您可以提交以下 OpenQasm 程序。

```

// 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;
```

以下列表提供了所有支持的 pragma 噪声运算符的规格。

```
#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
```

## 克劳斯操作员

为了生成 Kraus 运算符，您可以遍历矩阵列表，将矩阵的每个元素打印为复杂表达式。

使用 Kraus 运算符时，请记住以下几点：

- 的数量 qubits 不得超过 2。[架构中的当前定义设定](#)了此限制。
- 参数列表的长度必须是 8 的倍数。这意味着它只能由 2x2 矩阵组成。
- 总长度不超过  $2^{2 \times \text{num\_qubits}}$  矩阵。这意味着 1 有 4 个矩阵 qubit 和 16 个矩阵表示 2。qubits
- 所有提供的矩阵均为[完全正迹线保持 \(CPTP\)](#)。
- Kraus 运算符及其转置共轭的乘积需要相加得出一个单位矩阵。

## Qubit 使用 OpenQasm 3.0 重新布线

Amazon Braket 支持 Rigetti 设备上的 OpenQasm 中的物理 qubit 符号 ([要了解更多信息，请参阅此页面](#))。将物理设备 qubits 与[天真的重新布线策略](#)配合使用时，请确保 qubits 它们已连接到所选设备上。或者，如果改用 qubit 寄存器，则默认情况下，Rigetti 设备上会启用 PARTIAL 重新布线策略。

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

```
OPENQASM 3;

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

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

## 使用 OpenQasm 3.0 进行逐字编译

当你在 Rigetti、OQC 和的量子计算机上运行量子电路时 IonQ，你可以指示编译器完全按照定义运行你的电路，而无需进行任何修改。此功能称为逐字编译。使用 Rigetti 设备，您可以精确地指定要保留的内容，要么是整个电路，要么仅是其中的特定部分。要仅保留赛道的特定部分，你需要在保留区域内使用原生大门。目前，IonQ 并且 OQC 仅支持整个电路的逐字编译，因此电路中的每条指令都需要包含在逐字记录框中。

使用 OpenQasm，您可以围绕一盒未被修改且未经过硬件低级编译例程优化的代码框指定逐字编译指示。以下代码示例说明了如何使用 `#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;
```

有关逐字编译的更多信息，请参阅 [Verbatim 编译示例笔记本](#)。

## Braket 控制台

OpenQasm 3.0 任务可用，可以在 Amazon Braket 控制台中进行管理。在控制台上，你在 OpenQasm 3.0 中提交量子任务的体验与提交现有量子任务的体验相同。

## 更多资源

OpenQasm 在所有 Amazon Braket 区域都可用。

[有关在 Braket 上开始使用 OpenQasm 的笔记本示例，请参阅 Amazon Braket 教程。GitHub](#)

## 使用 OpenQasm 3.0 计算梯度

Amazon Braket 支持使用伴随微分法在shots=0 (精确) 模式下在按需模拟器和本地模拟器上计算梯度。您可以提供适当的编译指示来指定要计算的梯度，如以下示例所示。

```
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
```

您不必单独列出所有参数，还可以在编译指示all中指定。这将计算与列出的所有input参数相关的梯度。当参数数量非常多时，这可能很方便。在这种情况下，编译指示将类似于以下示例。

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

支持所有可观测类型，包括单个运算符、张量积、Hermitian 可观测值和。Sum要用来计算梯度的运算符必须封装在expectation()封装器中，并且必须指定每个项作用的量子比特。

## 使用 OpenQasm 3.0 测量特定的量子比特

局部状态向量模拟器和局部密度矩阵仿真器支持提交可以测量电路量子比特子集的OpenQASM程序。这通常被称为部分测量。例如，在以下代码中，您可以创建一个双量子比特电路，并且只测量第一个量子比特。

```
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];
""""
```

有两个量子比特 $q[0]$ ， $q[1]$ 但我们在这里只测量量子比特 0:  $b[0] = \text{measure } q[0]$  现在，在本地状态向量模拟器上运行以下命令。

```
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)
```

您可以通过检查设备动作属性中的`requiresAllQubitsMeasurement`字段来检查设备是否支持部分测量；如果是`False`，则支持部分测量。

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

这里，`requiresAllQubitsMeasurement`是`False`，这表明并非所有量子比特都必须进行测量。

## 使用 QuEra's Aquila 提交模拟节目

本页提供了有关Aquila计算机功能的全面文档QuEra。此处涵盖的详细信息如下：1) 参数化哈密顿仿真Aquila，2) AHS 程序参数，3) AHS 结果内容，4) 能力参数。Aquila我们建议使用 Ctrl+F 文本搜索来查找与您的问题相关的参数。

## Hamiltonian

来自的Aquila机器本机QuEra模拟以下（与时间有关的）哈密顿量程：

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

**Note**

访问本地失谐是一项[实验性功能](#)，可通过 [Braket Direct](#) 申请获得。

其中

- $H_{\text{drive},k}(t) = \left( \frac{1}{2}\Omega(t) e^{i(t) e^{i(t)}} S_{-,k} + \frac{1}{2}\Omega(t) e^{-i(t) S} + (-\Delta_{\text{global}}(t_{+,k}) n \right), k$ 
  - $\Omega(t)$  是随时间变化的全局驱动振幅（又名 Rabi 频率），单位为 (rad/s)
  - $(t)$  是随时间变化的全局相位，以弧度为单位
  - $S_{-,k}$  和  $S_{+,k}$  是原子  $k$  的自旋降低和升高运算符（在基数中  $=|\downarrow\#g, |\uparrow\#r\rangle$ ，它们是  $S = |g\rangle\langle g|$ ， $S = (S_-)^\dagger = |r\rangle\langle r|$ ）
  - $\Delta T_{\text{global}}(t)$  是随时间变化的全局失调
  - $n_k$  是原子  $k$  的里德伯格状态的投影运算符（即  $n = |rr\rangle\langle rr|$ ）
- $H_{\text{local detuning},k}(t) = -\Delta_{\text{local}} A(t) h n_{k,k}$ 
  - $\Delta T_{\text{local}}(t)$  是局部频移的时变因子，单位为 (rad/s)
  - $h_k$  是与地点相关的因子，是一个介于 0.0 和 1.0 之间的无量纲数
- $V_{\text{vdw},k,l} = C_6 / (d_{k,l})^6 n_k n_l$ 
  - $C_6$  是范德华系数，单位为 (rad/s) \* (m)^6
  - $d_{k,l}$  是原子  $k$  和  $l$  之间的欧几里得距离，以米为单位。

用户可以通过 Braket AHS 程序架构控制以下参数。

- 二维原子排列（每个原子  $k$  的  $x_k$  和  $y_k$  坐标，以 um 为单位），它控制成对原子距离  $d$  和  $k, l$ ， $l=1,2, \dots, N$
- $\Omega(t)$ ，随时间变化的全局 Rabi 频率，单位为 (rad/s)
- $(t)$ ，随时间变化的全局相位，单位为 (rad)
- $\Delta T_{\text{global}}(t)$ ，随时间变化的全局失调，单位为 (rad/s)
- $\Delta T_{\text{local}}(t)$ ，局部失谐幅度的随时间变化（全局）因子，单位为 (rad/s)
- $h_k$ ，局部失谐幅度的（静态）与地点相关的因子，介于 0.0 和 1.0 之间的无量纲数字

**Note**

用户无法控制涉及哪些级别（即  $S_-$ 、 $S$ 、 $n$  运算符是固定的），也无法控制 Rydberg-Rydberg 交互系数 ( $C$ ) 的强度。<sup>6</sup>

## 支架 AHS 程序架构

braket.ir.ahs.program\_v1.Program 对象（示例）

```

Program(
  braketSchemaHeader=BraketSchemaHeader(
    name='braket.ir.ahs.program',
    version='1'
  ),
  setup=Setup(
    ahs_register=AtomArrangement(
      sites=[
        [Decimal('0'), Decimal('0')],
        [Decimal('0'), Decimal('4e-6')],
        [Decimal('4e-6'), Decimal('0')],
      ],
      filling=[1, 1, 1]
    )
  ),
  hamiltonian=Hamiltonian(
    drivingFields=[
      DrivingField(
        amplitude=PhysicalField(
          time_series=TimeSeries(
            values=[Decimal('0'), Decimal('15700000.0'),
Decimal('15700000.0'), Decimal('0')],
            times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
          ),
        pattern='uniform'
      ),
      phase=PhysicalField(
        time_series=TimeSeries(
          values=[Decimal('0'), Decimal('0')],
          times=[Decimal('0'), Decimal('0.000001')]
        ),

```

```

        pattern='uniform'
    ),
    detuning=PhysicalField(
        time_series=TimeSeries(
            values=[Decimal('-54000000.0'), Decimal('54000000.0')],
            times=[Decimal('0'), Decimal('0.000001')]
        ),
        pattern='uniform'
    )
)
],
localDetuning=[
    LocalDetuning(
        magnitude=PhysicalField(
            times_series=TimeSeries(
                values=[Decimal('0'), Decimal('25000000.0'),
Decimal('25000000.0'), Decimal('0')],
                times=[Decimal('0'), Decimal('0.000001'), Decimal('0.000002'),
Decimal('0.000003')]
            ),
            pattern=Pattern([Decimal('0.8'), Decimal('1.0'), Decimal('0.9')])
        )
    )
]
)
)
)

```

## JSON ( 示例 )

```

{
  "braketSchemaHeader": {
    "name": "braket.ir.ahs.program",
    "version": "1"
  },
  "setup": {
    "ahs_register": {
      "sites": [
        [0E-7, 0E-7],
        [0E-7, 4E-6],
        [4E-6, 0E-7],
      ],
      "filling": [1, 1, 1]
    }
  }
}

```



```
    },
    "hamiltonian": {
      "drivingFields": [
        {
          "amplitude": {
            "time_series": {
              "values": [0.0, 15700000.0, 15700000.0, 0.0],
              "times": [0E-9, 0.000001000, 0.000002000, 0.000003000]
            },
            "pattern": "uniform"
          },
          "phase": {
            "time_series": {
              "values": [0E-7, 0E-7],
              "times": [0E-9, 0.000001000]
            },
            "pattern": "uniform"
          },
          "detuning": {
            "time_series": {
              "values": [-54000000.0, 54000000.0],
              "times": [0E-9, 0.000001000]
            },
            "pattern": "uniform"
          }
        }
      ],
      "localDetuning": [
        {
          "magnitude": {
            "time_series": {
              "values": [0.0, 25000000.0, 25000000.0, 0.0],
              "times": [0E-9, 0.000001000, 0.000002000, 0.000003000]
            },
            "pattern": [0.8, 1.0, 0.9]
          }
        }
      ]
    }
  }
}
```

## 主要字段

程序字段	type	description
设置.ahs_register.sites	列表 [列表 [十进制]]	镊子捕获原子的二维坐标清单
设置.ahs_register.filling	列表 [int]	用 1 标记占据陷阱位点的原子，用 0 标记占据空位点的原子
Hamiltonian.drivingFields [] .amplitude.time_series.t	列表 [十进制]	驱动振幅的时间点，欧米茄 (t)
Hamiltonian.drivingFields [] .amplitude.time_series.v	列表 [十进制]	驱动振幅值，欧米茄 (t)
Hamiltonian.drivingFields [] .amplitude.pat	str	驱动振幅的空间模式，Omega (t)；必须是“均匀的”
Hamiltonian.drivingFields [] .phase.time_series.tim	列表 [十进制]	驾驶阶段的时间点，phi (t)
Hamiltonian.drivingFields [] .phase.time_series.val	列表 [十进制]	驱动阶段的值，phi (t)
Hamiltonian.drivingFields [] .phase.patter	str	驾驶阶段的空间模式，phi (t)；必须是“均匀的”
Hamiltonian.drivingFields [] .detuning.time_series.times	列表 [十进制]	驾驶失调的时间点，delta_Global (t)
Hamiltonian.drivingFields [] .detuning.time_series.valu	列表 [十进制]	驱动失调的值，delta_Global (t)

程序字段	type	description
Hamiltonian.drivingFields [] .detuning.pattern	str	驾驶失调的空间模式，delta_Global (t)；必须是“均匀的”
Hamiltonian.localdetuning [] .magnitude.time_series.times	列表 [十进制]	局部失谐幅度的时变因子 delta_Local (t) 的时间点
Hamiltonian.localdetuning [] .magnitude.time_series.values	列表 [十进制]	局部失谐幅度的随时间变化的因子值 delta_Local (t)
Hamiltonian.localTuning [] .magnitude.pattern	列表 [十进制]	局部失谐幅度的站点相关因子 h_k (值对应于 setup.ahs_register .sites 中的站点)

## 元数据字段

程序字段	type	description
braket .name SchemaHeader	str	架构的名称；必须是 'braket.ir.ahs.program'
braket .version SchemaHeader	str	架构的版本

## Braket AHS 任务结果架构

braket.tasks.analog\_hamiltonian\_simulation\_quantum\_task\_result  
AnalogHamiltonianSimulationQuantumTaskResult ( 示例 )

```

AnalogHamiltonianSimulationQuantumTaskResult(
    task_metadata=TaskMetadata(
        braketSchemaHeader=BraketSchemaHeader(

```

```

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

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

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

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

```

## JSON ( 示例 )

```

{
  "braketSchemaHeader": {
    "name": "braket.task_result.analog_hamiltonian_simulation_task_result",
    "version": "1"
  },
  "taskMetadata": {
    "braketSchemaHeader": {
      "name": "braket.task_result.task_metadata",
      "version": "1"
    },

```

```
    "id": "arn:aws:braket:us-east-1:123456789012:quantum-task/12345678-90ab-
cdef-1234-567890abcdef",
    "shots": 2,
    "deviceId": "arn:aws:braket:us-east-1::device/qpu/quera/Aquila",

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

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

## 主要字段

任务结果字段	type	description
测量 [] .shortresult.preSequen	列表 [int]	每次射击的预序测量位（每个原子位点一个）：如果位点为空，则为 0，如果位点已填满，则为 1，在运行量子进化的脉冲序列之前测量
measurresult.postSequence []。	列表 [int]	每次射击的序列后测量位：如果原子处于里德伯格状态或位点为空，则为 0；如果原子处于基态，则为 1，在运行量子进化的脉冲序列末尾测量

## 元数据字段

任务结果字段	type	description
braket .nam SchemaHeader e	str	架构的名称；必须是“braket.task_result.analog_hamiltonian_simulation_task_result.result”
支架 .ver SchemaHeader sion	str	架构的版本
taskmetadata.braket.name SchemaHeader	str	架构的名称；必须是“braket.task_metadata”

任务结果字段	type	description
taskmetadata.braket .ver SchemaHeader	str	架构的版本
taskmetadata.id	str	量子任务的 ID。对于 AWS 量子任务，这是量子任务 ARN。
任务元数据.shots	int	量子任务的镜头数量
taskmetadata.shots.deviceID	str	运行量子任务的设备的 ID。对于 AWS 设备，这是设备 ARN。
taskmetadata.shots.createdate	str	创建的时间戳；格式必须采用 ISO-8601/ RFC3339 字符串格式 yyyy-mm-ddthh:mm:ss.sssz。默认为“无”。
taskmetadata.shots.endedat	str	量子任务结束的时间戳；格式必须采用 ISO-8601/ RFC3339 字符串格式 yyyy-mm-ddthh:mm:ss.sssz。默认为“无”。

任务结果字段	type	description
任务元数据.shots.Status	str	量子任务的状态 ( 已创建、已排队、正在运行、已完成、失败 )。默认为“无”。
taskmetadata.shots.failureReason	str	量子任务失败的原因。默认为“无”。
其他元数据.action	braket.ir.ahs.program_v1.Program	( 请参阅 <a href="#">Braket AHS 程序架构</a> 部分 )
其他元数据.action.braket.querame tadata.name SchemaHeader	str	架构的名称；必须是“braket.task_result.querame tadata”
其他元数据.action.braket.querame tadata.version SchemaHeader	str	架构的版本
其他 metadata.action.num Successful IShots	int	完全成功拍摄的次数；必须等于请求的拍摄次数
测量 [] .shotMetadata.shotStatus	int	镜头的状态 ( 成功、部分成功、失败 ) ；必须为“成功”



## QuEra 设备属性架构

braket.device\_schema.quera.quera\_device\_capabilities\_v1。QueraDevice能力 ( 示例 )

```
QueraDeviceCapabilities(  
  service=DeviceServiceProperties(  
    braketSchemaHeader=BraketSchemaHeader(  
      name='braket.device_schema.device_service_properties',  
      version='1'  
    ),  
    executionWindows=[  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.MONDAY: 'Monday'>,  
        windowStartHour=datetime.time(1, 0),  
        windowEndHour=datetime.time(23, 59, 59)  
      ),  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.TUESDAY: 'Tuesday'>,  
        windowStartHour=datetime.time(0, 0),  
        windowEndHour=datetime.time(12, 0)  
      ),  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.WEDNESDAY: 'Wednesday'>,  
        windowStartHour=datetime.time(0, 0),  
        windowEndHour=datetime.time(12, 0)  
      ),  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.FRIDAY: 'Friday'>,  
        windowStartHour=datetime.time(0, 0),  
        windowEndHour=datetime.time(23, 59, 59)  
      ),  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.SATURDAY: 'Saturday'>,  
        windowStartHour=datetime.time(0, 0),  
        windowEndHour=datetime.time(23, 59, 59)  
      ),  
      DeviceExecutionWindow(  
        executionDay=<ExecutionDay.SUNDAY: 'Sunday'>,  
        windowStartHour=datetime.time(0, 0),  
        windowEndHour=datetime.time(12, 0)  
      )  
    ],  
    shotsRange=(1, 1000),
```

```

        deviceCost=DeviceCost(
            price=0.01,
            unit='shot'
        ),
        deviceDocumentation=
            DeviceDocumentation(
                imageUrl='https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png',
                summary='Analog quantum processor based on neutral atom arrays',
                externalDocumentationUrl='https://www.quera.com/aquila'
            ),
            deviceLocation='Boston, USA',
            updatedAt=datetime.datetime(2024, 1, 22, 12, 0,
tzinfo=datetime.timezone.utc),
            getTaskPollIntervalMillis=None
        ),
        action={
            <DeviceActionType.AHS: 'braket.ir.ahs.program': DeviceActionProperties(
                version=['1'],
                actionType=<DeviceActionType.AHS: 'braket.ir.ahs.program'>
            )
        },
        deviceParameters={},
        braketSchemaHeader=BraketSchemaHeader(
            name='braket.device_schema.quera.quera_device_capabilities',
            version='1'
        ),
        paradigm=QueraAhsParadigmProperties(
            ...
            # See https://github.com/amazon-braket/amazon-braket-schemas-python/blob/main/
src/braket/device_schema/quera/quera_ahs_paradigm_properties_v1.py
            ...
        )
    )
)

```

## JSON ( 示例 )

```

{
  "service": {
    "braketSchemaHeader": {
      "name": "braket.device_schema.device_service_properties",
      "version": "1"
    }
  }
}

```

```
    },
    "executionWindows": [
      {
        "executionDay": "Monday",
        "windowStartHour": "01:00:00",
        "windowEndHour": "23:59:59"
      },
      {
        "executionDay": "Tuesday",
        "windowStartHour": "00:00:00",
        "windowEndHour": "12:00:00"
      },
      {
        "executionDay": "Wednesday",
        "windowStartHour": "00:00:00",
        "windowEndHour": "12:00:00"
      },
      {
        "executionDay": "Friday",
        "windowStartHour": "00:00:00",
        "windowEndHour": "23:59:59"
      },
      {
        "executionDay": "Saturday",
        "windowStartHour": "00:00:00",
        "windowEndHour": "23:59:59"
      },
      {
        "executionDay": "Sunday",
        "windowStartHour": "00:00:00",
        "windowEndHour": "12:00:00"
      }
    ],
    "shotsRange": [
      1,
      1000
    ],
    "deviceCost": {
      "price": 0.01,
      "unit": "shot"
    },
    "deviceDocumentation": {
```

```

        "imageUrl": "https://
a.b.cdn.console.awsstatic.com/59534b58c709fc239521ef866db9ea3f1aba73ad3ebcf60c23914ad8c5c5c878/
a6cfc6fca26cf1c2e1c6.png",
        "summary": "Analog quantum processor based on neutral atom arrays",
        "externalDocumentationUrl": "https://www.quera.com/aquila"
    },
    "deviceLocation": "Boston, USA",
    "updatedAt": "2024-01-22T12:00:00+00:00"
},
"action": {
    "braket.ir.ahs.program": {
        "version": [
            "1"
        ],
        "actionType": "braket.ir.ahs.program"
    }
},
"deviceParameters": {},
"braketSchemaHeader": {
    "name": "braket.device_schema.quera.quera_device_capabilities",
    "version": "1"
},
"paradigm": {
    ...
    # See Aquila device page > "Calibration" tab > "JSON" page
    ...
}
}

```

## 服务属性字段

服务属性字段	type	description
Service.executionWindows [].execution	ExecutionDay	执行窗口的天数；必须是“每天”、“工作日”、“周末”、“星期一”、“星期二”、“星期三”、“星期四”、“星期五”、“星期六”或“星期日”
Service.executionWindows [].wind StartHour	datetime.time	执行窗口开始时间的 UTC 24 小时格式

服务属性字段	type	description
Service.executionWindows [].wind EndHour	datetime.time	执行窗口结束时间的 UTC 24 小时格式
service.qpu_Capabilities.service.ShotsRange	元组 [整数, 整数]	设备的最小和最大拍摄次数
service.qpu_Capabilities.service.cost.Price	float	设备价格 (以美元计)
service.qpu_Capabilities.servicecost.unit	str	计费单位, 例如: “分钟”、“小时”、“拍摄”、“任务”

### 元数据字段

元数据字段	type	description
操作 [].version	str	AHS 程序架构的版本
action []. actionTy	ActionType	AHS 程序架构名称; 必须是 “braket.ir.ahs.program”
服务.braket .name SchemaHeader	str	架构的名称; 必须是 “braket.device_schema.device_service_properties”
服务.braket .version SchemaHeader	str	架构的版本
服务.deviceDocumation.imageUrl	str	设备图片的网址
服务。设备文档。摘要	str	设备上的简要描述
服务。设备文档。外部 DocumentationUrl	str	外部文档网址
服务. 设备位置	str	设备的地理位置

元数据字段	type	description
Service.updatedat	datetime	上次更新设备属性的时间

## 使用 Boto3

Boto3 是 Python 的 AWS 软件开发工具包。使用 Boto3，Python 开发者可以创建、配置和管理 AWS 服务，比如 Amazon Braket。Boto3 提供了面向对象的访问权限API以及对 Braket 的低级访问权限。Amazon

按照 [Boto3 快速入门指南](#) 中的说明进行操作，了解如何安装和配置 Boto3。

Boto3 提供了与 Brake Amazon t Python SDK 配合使用的核心功能，可帮助您配置和运行量子任务。Python 客户总是需要安装 Boto3，因为这是核心实现。如果您想使用其他辅助方法，则还需要安装 Amazon Braket SDK。

例如，当你调用CreateQuantumTask时，AmazonBraket SDK 会将请求提交给 Boto3，Boto3 然后调用。AWS API

本节内容：

- [打开 Amazon Braket Boto3 客户端](#)
- [AWS CLI 配置 Boto3 和 Amazon Braket SDK 的配置文件](#)

## 打开 Amazon Braket Boto3 客户端

要将 Boto3 与 B Amazon raket 配合使用，必须导入 Boto3，然后定义用于连接到 Braket 的客户端。Amazon API在以下示例中，命名为 Boto3 客户端。braket

### Note

为了向后兼容的旧版本 BraketSchemas，调用中省略了 OpenQasm 信息。GetDevice API 要获取此信息，用户代理需要提供最新版本的 BraketSchemas（1.8.0 或更高版本）。Braket SDK 会自动为您报告此问题。如果您在使用 Braket SDK 时未在GetDevice响应中看到 OpenQasm 结果，则可能需要设置AWS\_EXECUTION\_ENV环境变量来配置用户代理。有关如何为、Boto3 和 Go、Java 和/SDK 执行此操作 AWS CLI，请参阅[未返回 OpenQasm 结果错误](#)主题中提供的代码示例。GetDevice JavaScript TypeScript

```
import boto3
import botocore

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

现在您已经建立了braket客户端，可以从 Amazon Braket 服务中提出请求和处理响应。您可以在 [API 参考](#) 中获取有关请求和响应数据的更多详细信息。

以下示例说明如何使用设备和量子任务。

- [搜索设备](#)
- [取回设备](#)
- [创建量子任务](#)
- [检索量子任务](#)
- [搜索量子任务](#)
- [取消量子任务](#)

## 搜索设备

- `search_devices(**kwargs)`

使用指定的过滤器搜索设备。

```
# 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'])
```

## 取回设备

- `get_device(deviceArn)`

检索 Amazon Braket 中可用的设备。

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

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

## 创建量子任务

- `create_quantum_task(**kwargs)`

创建量子任务。

```
# 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")
```

## 检索量子任务

- `get_quantum_task(quantumTaskArn)`

检索指定的量子任务。

```
# 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'])
```

## 搜索量子任务

- `search_quantum_tasks(**kwargs)`

搜索与指定过滤值匹配的量子任务。

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

## 取消量子任务

- `cancel_quantum_task(quantumTaskArn)`

取消指定的量子任务。

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

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

## AWS CLI 配置 Boto3 和 Amazon Braket SDK 的配置文件

除非您另行明确指定，否则 Amazon Braket SDK 依赖于默认 AWS CLI 凭证。我们建议您在托管 Amazon Braket 笔记本上运行时保留默认值，因为您必须提供有权启动笔记本实例的 IAM 角色。

或者，如果您在本地运行代码（例如在 Amazon EC2 实例上），则可以建立命名 AWS CLI 配置文件。您可以为每个配置文件指定不同的权限集，而不必定期覆盖默认配置文件。

本节简要说明了如何配置这样的 CLI profile 以及如何将该配置文件合并到 Amazon Braket 中，以便使用该配置文件的权限进行 API 调用。

本节内容：

- [步骤 1：配置本地 AWS CLI profile](#)
- [步骤 2：建立 Boto3 会话对象](#)
- [第 3 步：将 Boto3 会话合并到 Braket 中 AwsSession](#)

### 步骤 1：配置本地 AWS CLI profile

解释如何创建用户和如何配置非默认配置文件超出了本文档的范围。有关这些主题的信息，请参阅：

- [入门](#)
- [将配置 AWS CLI 为使用 AWS IAM Identity Center](#)

要使用 Amazon Braket，您必须向该用户以及相关的 CLI profile 提供必要的 Braket 权限。例如，您可以附加 AmazonBraketFullAccess 策略。

### 步骤 2：建立 Boto3 会话对象

要建立 Boto3 会话对象，请使用以下代码示例。

```
from boto3 import Session
```

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

### Note

如果预期的API呼叫具有基于区域的限制，而这些限制与您的profile默认区域不一致，则可以以为 Boto3 会话指定一个区域，如以下示例所示。

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

对于指定为的参数region，请替换一个与 Amazon Braket 可用的值相对应的值us-east-1，例如us-west-1、等。AWS 区域

## 第 3 步：将 Boto3 会话合并到 Braket 中 AwsSession

以下示例说明如何初始化 Boto3 Braket 会话并在该会话中实例化设备。

```
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)
```

设置完成后，您可以向该实例化的AwsDevice对象提交量子任务（例如，通过调用device.run(...) 命令）。该设备API发出的所有调用均可利用与您之前指定的 CLI 配置文件关联的 IAM 证书profile。

# Amazon Braket 上的脉冲控制

本节介绍如何在 Amazon Braket 中的各种 QPU 上使用脉冲控制。

本节内容：

- [支架脉冲](#)
- [帧和端口的作用](#)
- [你好脉冲](#)
- [使用脉冲访问原生门](#)

## 支架脉冲

脉冲是控制量子计算机中量子比特的模拟信号。使用 Amazon Braket 上的某些设备，您可以访问脉冲控制功能，使用脉冲提交电路。你可以通过 Braket SDK、OpenQasm 3.0 或直接通过 Braket API 访问脉冲控制。首先，让我们介绍一下 Braket 中脉冲控制的一些关键概念。

## 帧

帧是一种软件抽象，既充当量子程序中的时钟，又充当相位。每次使用时钟时间都会递增，并且有状态的载波信号由频率定义。向量子比特传输信号时，帧决定量子比特的载波频率、相位偏移以及波形包络的发射时间。在 Braket Pulse 中，构造帧取决于器件、频率和相位。根据设备的不同，您可以选择预定义的帧，也可以通过提供端口来实例化新帧。

```
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)
```

## 端口

端口是一种软件抽象，代表控制量子比特的任何输入/输出硬件组件。它可以帮助硬件供应商提供一个接口，用户可以通过该界面进行交互以操作和观察量子比特。端口的特征是代表连接器名称的单个字符串。该字符串还显示了最小时间增量，该增量指定了我们可以定义波形的精细程度。

```
from braket.pulse import Port
```

```
Port0 = Port("channel_0", dt=1e-9)
```

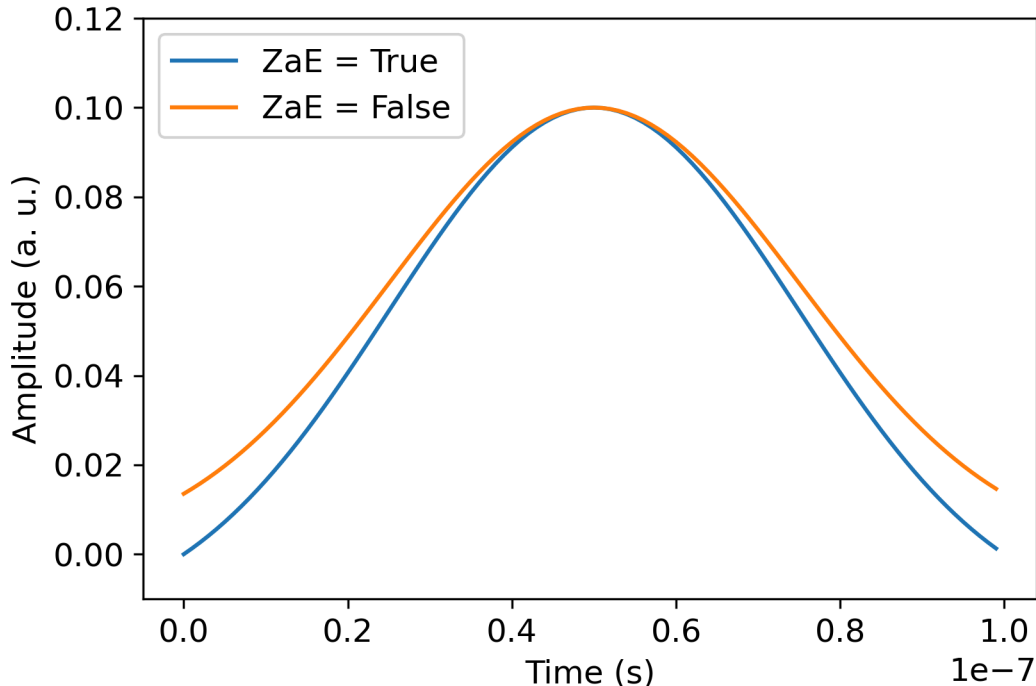
## 波形

波形是一种随时间变化的包络，我们可以用它在输出端口上发射信号或通过输入端口捕获信号。您可以通过复数列表直接指定波形，也可以使用波形模板生成硬件提供商提供的列表。

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

Braket Pulse 提供了一个标准的波形库，包括恒定波形、高斯波形和绝热门导数去除 (DRAG) 波形。您可以通过 `sample` 函数检索波形数据以绘制波形的形状，如以下示例所示。

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



上图描绘了从中创建的高斯波形。GaussianWaveform 我们选择的脉冲长度为 100 ns，宽度为 25 ns，振幅为 0.1（任意单位）。波形在脉冲窗口中居中。GaussianWaveform 接受布尔参

数`zero_at_edges` ( 图例中的 `zaE` )。设置为`True`，此参数会偏移高斯波形，使  $t=0$  和  $t=length$  处的点为零，并重新缩放其振幅，使最大值与参数相对应。`amplitude`

现在我们已经介绍了脉冲级访问的基本概念，接下来我们将看到如何使用门和脉冲来构造电路。

## 帧和端口的作用

本节介绍每台设备可用的预定义帧和端口。我们还将简要讨论在某些帧上播放脉冲时所涉及的机制。

### Rigetti

#### 帧

Rigetti设备支持预定义的帧，这些帧的频率和相位已校准为与相关的量子比特共振。命名约定是 $\{i\}$ 指第一个量子比特数， $q\{i\}[_q\{j\}]_{\{role\}}\_frame\{j\}$ 如果帧用于激活双量子比特交互作用，则指第二个量子比特数，并 $\{role\}$ 指帧的作用。角色如下：

- `rf`是驱动量子比特0-1过渡的帧。脉冲作为先前通过和`shift`函数提供的频率和相位的微波瞬态信号传输。`set`信号的时变幅度由帧上播放的波形给出。该框架插入了单量子比特、非对角线交互作用。欲了解更多信息，请参阅 [Krantz 等人](#)。还有 [Rahamim 等人](#)。。
- `rf_f12`类似于`rf`，其参数以 1-2 过渡为目标。
- `ro_rx`用于通过耦合共面波导实现量子比特的色散读数。读出波形的频率、相位和全套参数均已预先校准。它目前通过使用`capture_v0`，除了帧标识符之外不需要任何参数。
- `ro_tx`用于传输来自谐振器的信号。它目前未使用。
- `cz`是一个经过校准以启用双`cz`量子比特门的框架。与所有与`ff`端口关联的帧一样，它通过调制与其邻居共振的配对的可调量子比特来开启通过磁通线的纠缠相互作用。有关纠缠机制的更多信息，请参阅 [Reagor 等人](#)。，[Caldwell 等人](#)。，以及 [Didier 等人](#)。。
- `cphase`是一个经过校准以启用双量子比特`cphaseshift`门的框架，并且与端口相连。`ff`有关纠缠机制的更多信息，请参阅框架的`cz`描述。
- `xy`是一个经过校准以启用双量子比特  $XY(\theta)$  门的框架，并且与端口相连。`ff`有关纠缠机制以及如何实现  $XY$  门的更多信息，请参阅`cz`框架和 [Abrams 等人](#)的描述。。

当基于`ff`端口的帧偏移可调量子比特的频率时，与量子比特相关的所有其他驱动帧都将偏移与振幅和频移持续时间相关的量。因此，必须通过向相邻量子比特的帧添加相应的相移来补偿这种影响。

#### 端口

这些Rigetti设备提供了一份端口列表，您可以通过设备功能检查这些端口。端口名称遵循惯例 $q\{i\}_{\{type\}}$ 例，其中 $\{i\}$ 指量子比特数并 $\{type\}$ 指端口的类型。请注意，并非所有的量子比特都有一套完整的端口。端口的类型如下：

- `rf`代表驱动单量子比特过渡的主接口。它与`rf`和`rf_f12`帧相关联。它与量子比特电容耦合，允许微波在千兆赫兹范围内驱动。
- `ro_tx`用于将信号传输到电容耦合到量子比特的读出谐振器。读出信号传输通过八角形多路复用八倍。
- `ro_rx`用于接收来自耦合到量子比特的读出谐振器的信号。
- `ff`表示电感耦合到量子比特的快速通量线。我们可以用它来调整 `transmon` 的频率。只有设计为高度可调的量子比特才有端口。`ff`该端口用于激活量子比特与量子比特的相互作用，因为每对相邻的传输之间存在静态电容耦合。

有关架构的更多信息，请参阅 [Valery 等人](#)。。

## OQC

### 帧

OQC设备支持预定义的帧，这些帧的频率和相位已校准为与相关的量子比特共振。这些帧的命名约定如下：

- `driving frame`：`q\{i\}[_q\{j\}]\_{role}`其中 $\{i\}\{j\}$ 指第一个量子比特数，指第二个量子比特数，以防帧用于激活双量子比特交互，并 $\{role\}$ 指帧的作用，如下所述。
- `量子比特读出帧`：`r\{i\}\_{role}`其中 $\{i\}$ 指量子比特数， $\{role\}$ 指帧的作用，如下所述。

我们建议将每个框架用于其设计角色，如下所示：

- `drive`用作驱动量子比特0-1过渡的主框架。脉冲作为先前通过和`shift`函数提供的频率和相位的微波瞬态信号传输。`set`信号的时变幅度由帧上播放的波形给出。该框架插入了单量子比特、非对角线交互作用。欲了解更多信息，请参阅 [Krantz 等人](#)。还有 [Rahamim 等人](#)。。
- `second_state`等同于`drive`帧，但其频率是根据与 1-2 过渡的共振进行调整的。
- `measure`用于读取。读出波形的频率、相位和全套参数均已预先校准。它目前通过使用`capture_v0`，除了帧标识符之外不需要任何参数。
- `acquire`用于捕获来自谐振器的信号。它目前未使用。

- `cross_resonance` 激活量子比特之间的交叉共振相互作用， $i$  和  $j$  以目标量子比特的过渡频率驱动控制量子比特  $i$ 。因此，帧频率是使用目标量子比特的频率设置的。相互作用的速率与该交叉谐振驱动器的振幅成正比。不同类型的串扰会产生不良影响，需要进行校正。见 [Patterson 等人](#)。了解有关与同轴形变量子比特 ('coaxmons') 的交叉共振相互作用的更多信息。
- `cross_resonance_cancellation` 帮助您添加校正以抑制激活交叉共振相互作用时串扰引起的有害影响。初始帧频设置为控制量子比特  $i$  的过渡频率。有关取消方法的更多信息，请参阅 [Patterson 等人](#)。

## 端口

这些 OQC 设备提供了一份端口列表，您可以通过设备功能检查这些端口。前面描述的帧与通过其 `id` 标识的端口相关联，`channel_{N}` 其中  $\{N\}$  为整数。端口是连接同轴器的控制线（方向 `tx`）和读出谐振器（方向 `rx`）的接口。每个量子比特都与一条控制线和一个读出谐振器相关联。传输端口是单量子比特和双量子比特操作的接口。接收端口用于量子比特读取。

## 你好脉冲

在这里，你将学习如何直接用脉冲构造一个简单的 Bell 对，并在 Rigetti 设备上执行这个脉冲程序。Bell 对是一个双量子比特电路，由第一个量子比特上的哈达玛德门，然后是第一个量子比特和第二个量子比特之间的 `cnot` 门。使用脉冲创建纠缠状态需要特定的机制，这些机制取决于硬件类型和设备架构。我们不会使用原生机制来创建 `cnot` 门。取而代之的是，我们将使用特定的波形和帧来原生启用 `cz` 门控。在这个例子中，我们将使用单量子比特原生门创建一个 Hadamard 门，`rxrz` 并使用脉冲来表达门。`cz`

首先，让我们导入必要的库。除了该 `Circuit` 类之外，你现在还需要导入该 `PulseSequence` 类。

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

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

接下来，使用新的 Braket 设备的 Amazon 资源名称 (ARN) 实例化该设备。Rigetti Aspen-M-3 请参阅 Amazon Braket 控制台上的“设备”页面，查看设备的布局。Rigetti Aspen-M-3

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



由于 Hadamard 门不是 Rigetti 设备的原生门，因此不能与脉冲结合使用。因此，你需要将其分解为一系列  $r_x$  和  $r_z$  原生门。

```
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)
    )
```

对于  $c_z$  栅极，我们将使用任意波形，其参数（振幅、上升/下降时间和持续时间）由硬件提供商在校准阶段预先确定。此波形将应用于 `q10_q113_cz_frame` 有关此处使用的任意波形的最新版本，请参阅网站上的 [QCS](#)。Rigetti 您可能需要创建 QCS 账户。

```
a_b_cz_wfm = ArbitraryWaveform([0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.00017888439538396808, 0.00046751103636033026, 0.0011372942989106456,
0.002577059611929697, 0.005443941944632366, 0.010731922770068104, 0.01976701723583167,
0.03406712171899736, 0.05503285980691202, 0.08350670755829034, 0.11932853352131022,
0.16107456696238298, 0.20614055551722368, 0.2512065440720643, 0.292952577513137,
0.328774403476157, 0.3572482512275353, 0.3782139893154499, 0.3925140937986156,
0.40154918826437913, 0.4068371690898149, 0.4097040514225177, 0.41114381673553674,
0.411813599998087, 0.4121022266390633, 0.4122174383870584, 0.41226003881132406,
0.4122746298554775, 0.4122792591252675, 0.4122806196003006, 0.41228098995582513,
0.41228108334474756, 0.4122811051578895, 0.4122811098772742, 0.4122811108230642,
0.4122811109986316, 0.41228111102881937, 0.41228111103362725, 0.4122811110343365,
0.41228111103443343, 0.4122811110344457, 0.4122811110344471, 0.41228111103444737,
0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
0.4122811110344471, 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')

```

这应该返回：

CZ pulse duration: 124 ns

现在我们可以使用刚才定义的波形来构造cz门。回想一下，如果控制量子比特处于状态，则cz门由目标量子比特的相位翻转组成。|1>

```

phase_shift_a=1.1733407221086924
phase_shift_b=6.269846678712192

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

frames = [a_rf_frame, b_rf_frame, a_b_cz_frame]

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

```

a\_b\_cz\_wfm波形在与快速通量端口关联的帧上播放。它的作用是改变量子比特频率以激活量子比特与量子比特的相互作用。有关更多信息，请参阅[帧和端口的作用](#)。随着频率的变化，量子比特帧的旋转速度与保持不变的单量子比特rf帧的旋转速率不同：后者正在脱相。这些相移是事先通过Ramsey序列校

准的，并在此处作为硬编码信息提供，直至`phase_shift_a`和`phase_shift_b`（完整周期）。我们使用帧上的`shift_phaserf`说明来纠正这种去相位现象。请注意，此序列仅在没有XY帧与量子比特相关的程序中起作用，`ab`并且之所以使用，是因为我们无法补偿这些帧上发生的相移。这个单个 Bell 配对程序就是这种情况，它只使用`rf`和`cz`帧。欲了解更多信息，请参阅 [Caldwell 等人](#)。

现在我们已经准备好创建带有脉冲的 Bell 配对了。

```
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)
```

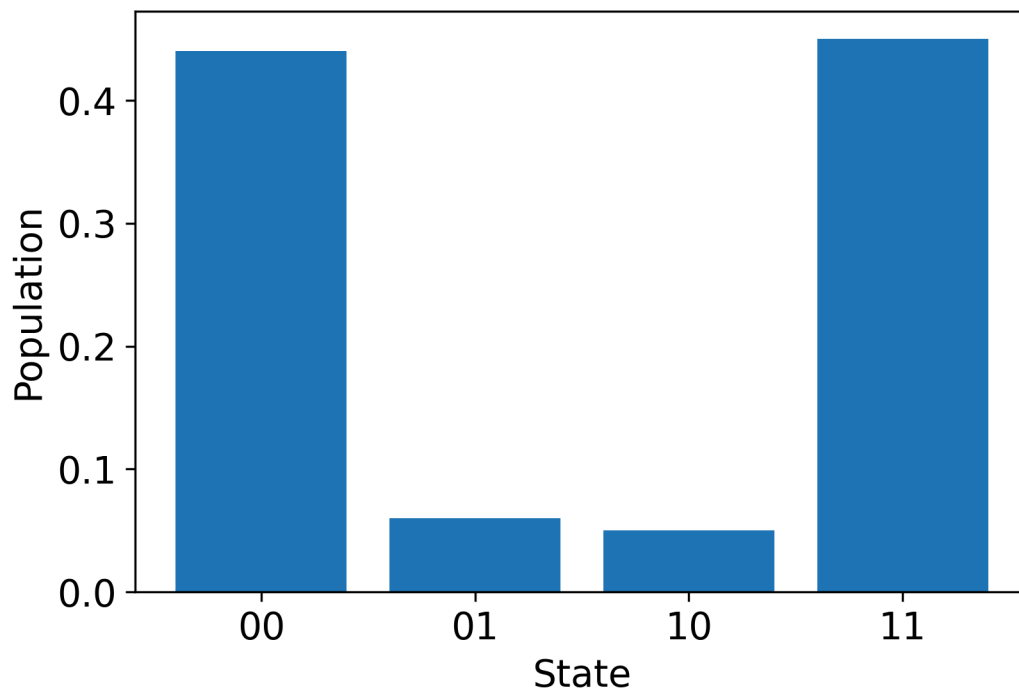
```
T : | 0 | 1 | 2 | 3 |4 | 5 | 6 | 7 | 8 |
q5 : -Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-PG-----
      |
q6 : -Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-PG-Rz(3.14)-Rx(1.57)-Rz(1.57)-Rx(-1.57)-
T : | 0 | 1 | 2 | 3 |4 | 5 | 6 | 7 | 8 |
```

让我们在Rigetti设备上运行这对 Bell。请注意，运行此代码块将产生费用。有关这些费用的更多信息，请参阅 Amazon Braket [定价](#)页面。我们建议您使用少量镜头测试您的电路，以确保它可以在设备上运行，然后再增加射击次数。

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

counts = task.result().measurement_counts

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



## 你好 Pulse 使用 OpenPulse

[OpenPulse](#) 是一种用于指定通用量子器件脉冲电平控制的语言，也是 OpenQasm 3.0 规范的一部分。Amazon Braket 支持 OpenPulse 使用 OpenQasm 3.0 表示形式直接对脉冲进行编程。

Braket OpenPulse 用作在原生指令中表达脉冲的底层中间表示形式。OpenPulse 支持以 (“定义校准”的缩写 `defcal`) 声明形式添加指令校准。通过这些声明，你可以在较低级别的控制语法中指定门指令的实现。

在此示例中，我们将使用 OpenQasm 3.0 构造贝尔电路，并在设备 OpenPulse 上使用可调频率的传输。回想一下，贝尔电路是一个双量子比特电路，它由第一个量子比特上的哈达玛德门和两个量子比特之间的 `cnot` 门组成。由于 `cnot` 和 `cz` 门仅通过基础变换与大门不同，因此我们将改用 Hadamard 和 `cz` 门来定义一对 Bell，因为该设备为本演示提供了一种更简单的方法来创建 `cz` 门。

让我们首先使用设备的原生门来定义 Hadamard 门。

```
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;
```











```

0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.4844321964728096,
0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
0.4753811409710734, 0.46121311095968553, 0.4331554251320285, 0.38631750509562957,
0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
0.0008644760431374626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
}
defcal h $10 {
    rz(pi) $10;
    rx(pi/2) $10;
    rz(pi/2) $10;
    rx(-pi/2) $10;
}
defcal h $113 {
    rz(pi) $113;
    rx(pi/2) $113;
    rz(pi/2) $113;
    rx(-pi/2) $113;
}
defcal cz $10, $113 {
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
    play(q10_q113_cz_frame, q10_q113_cz_wfm);
    delay[124ns] q10_rf_frame;
    shift_phase(q10_rf_frame, 1.1733407221086924);
    delay[124ns] q113_rf_frame;
    shift_phase(q113_rf_frame, 6.269846678712192);
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
}
bit[2] c;
h $10;
h $113;
cz $10, $113;
h $113;

```

```
c[0] = measure $10;  
c[1] = measure $113;
```

现在，您可以使用以下代码使用 Braket SDK 在 Rigetti 设备上执行此 OpenQasm 3.0 程序。

```
# 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,  
)
```

## 使用脉冲访问原生门

研究人员通常需要确切地知道特定 QPU 支持的原生门是如何实现为脉冲的。脉冲序列由硬件提供商仔细校准，但是访问脉冲序列为研究人员提供了设计更好的门或探索缓解错误的协议的机会，例如通过拉伸特定门的脉冲来进行零噪声外推。

Amazon Braket 支持从 Rigetti 以编程方式访问本机门。

```
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

硬件提供商定期校准 QPU，通常每天不止一次。Braket SDK 使您能够获得最新的栅极校准。

```
device.refresh_gate_calibrations()
```

要检索给定的原生门，例如 RX 或 XY 门，您需要传递感兴趣的 Gate 物体和量子比特。例如，您可以检查应用于 0 的 RX ( $\pi/2$ ) 的脉冲实现。qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

您可以使用该 `filter` 函数创建一组经过筛选的校准。您传递一个大门列表或一个列表 `QubitSet`。以下代码创建了两个集合，其中包含 RX ( $\pi/2$ ) 和 0 的所有校准。qubit

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

现在，您可以通过附加自定义校准集来提供或修改原生门的操作。例如，考虑以下电路。

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

您可以通过将 `PulseSequence` 对象字典传递给 `gate_definitions` 关键字参数 `qubit 0` 来运行它，为 rx 门开启自定义门校准。你可以根据 `GateCalibrations` 对象的属性 `pulse_sequences` 构造字典。所有未指定的门都被量子硬件提供商的脉冲校准所取代。

```
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)
```

# 亚马逊 Braket 混合作业用户指南

本节提供有关如何在 Amazon Braket 中设置和管理混合作业的说明。

您可以使用以下方式在 Braket 中访问混合作业：

- [亚马逊 Braket Python 软件开发工具包](#)。
- [亚马逊 Braket 主机](#)。
- AmazonBraket API。

本节内容：

- [什么是混合 Job ?](#)
- [何时使用 Amazon Braket 混合工作](#)
- [将本地代码作为混合作业运行](#)
- [使用 Amazon Braket 混合作业运行混合作业](#)
- [创建你的第一个 Hybrid Job](#)
- [输入、输出、环境变量和辅助函数](#)
- [保存作业结果](#)
- [使用检查点保存并重启混合作业](#)
- [为算法脚本定义环境](#)
- [使用超参数](#)
- [配置混合作业实例以运行算法脚本](#)
- [取消混合 Job](#)
- [使用参数化编译加快混合作业的速度](#)
- [PennyLane 与 Amazon Braket 一起使用](#)
- [使用 Amazon Braket 混合任务并 PennyLane 运行 QAOA 算法](#)
- [使用来自的嵌入式模拟器加速混合工作负载 PennyLane](#)
- [使用本地模式构建和调试混合作业](#)
- [自带集装箱 \(BYOC\)](#)
- [在中配置默认存储桶 AwsSession](#)
- [使用直接与混合作业互动 API](#)

## 什么是混合 Job ？

Amazon Braket Hybrid Jobs 为您提供了一种运行混合量子经典算法的方法，需要经典 AWS 资源和量子处理单元 (QPU)。Hybrid Jobs 旨在启动请求的经典资源，运行您的算法，并在完成后释放实例，因此您只需为实际使用的资源付费。

Hybrid Jobs 非常适合涉及经典资源和量子资源的长期运行迭代算法。您提交算法以供运行，Braket 在可扩展的容器化环境中运行该算法，然后在算法完成后检索结果。

此外，通过混合作业创建的量子任务受益于更高的优先级排队到目标 QPU。这样可以确保您的量子任务在队列中的其他任务之前得到处理和运行。这对于迭代混合算法特别有益，在迭代混合算法中，后续任务取决于先前量子任务的结果。[此类算法的示例包括量子近似优化算法 \(QAOA\)、变分量子特征求解器或量子机器学习](#)。您还可以近乎实时地监控算法进度，从而跟踪成本、预算或自定义指标，例如训练损失或期望值。

## 何时使用 Amazon Braket 混合作业

Amazon Braket Hybrid Jobs 使您能够运行混合量子经典算法，例如变分量子特征求解器 (VQE) 和量子近似优化算法 (QAOA)，它们将经典计算资源与量子计算设备相结合，以优化当今量子系统的性能。Amazon Braket Hybrid Jobs 提供三个主要好处：

1. 性能：与在自己的环境中运行混合算法相比，Amazon Braket 混合任务提供的性能更好。当您的作业运行时，它可以优先访问选定的目标 QPU。您的任务在设备上排队的其他任务之前运行。这使得混合算法的运行时间更短、更具可预测性。Amazon Braket 混合作业还支持参数化编译。您可以使用免费参数提交电路，然后 Braket 只需编译一次电路，无需重新编译即可对同一电路进行后续参数更新，从而实现更快的运行时间。
2. 便利：Amazon Braket Hybrid Jobs 简化了计算环境的设置和管理，并在混合算法运行时保持计算环境的运行。您只需提供算法脚本，然后选择要运行的量子设备（量子处理单元或模拟器）即可。Amazon Braket 等待目标设备可用，启动传统资源，在预先构建的容器环境中运行工作负载，将结果返回到亚马逊简单存储服务 (Amazon S3)，然后释放计算资源。
3. 指标：Amazon Braket Hybrid Job on-the-fly s 提供有关运行算法的见解，并近乎实时地向亚马逊 CloudWatch 和 Amazon Braket 控制台提供可自定义的算法指标，以便您可以跟踪算法的进度。

## 将本地代码作为混合作业运行

Amazon Braket Hybrid Jobs 提供了混合量子经典算法的完全托管编排，将 Amazon EC2 计算资源与 Amazon Braket 量子处理单元 (QPU) 访问权限相结合。在混合作业中创建的量子任务优先于单个量子

任务，因此您的算法不会因量子任务队列的波动而中断。每个 QPU 都维护一个单独的混合作业队列，确保在任何给定时间只能运行一个混合作业。

本节内容：

- [使用本地 Python 代码创建混合作业](#)
- [安装其他 Python 软件包和源代码](#)
- [将数据保存并加载到混合作业实例中](#)
- [混合工作装饰者的最佳实践](#)

## 使用本地 Python 代码创建混合作业

您可以将本地 Python 代码作为 Amazon Braket Hybrid Job 运行。您可以通过使用 `@hybrid_job` 装饰器对代码进行注释来实现此目的，如以下代码示例所示。对于自定义环境，您可以选择 [使用 Amazon 弹性容器注册表 \(ECR\) 中的自定义容器](#)。

### Note

默认情况下，仅支持 Python 3.10。

您可以使用 `@hybrid_job` 装饰器来注释函数。 [Braket 将装饰器内部的代码转换为 Braket 混合作业算法脚本](#)。然后，混合作业在 Amazon EC2 实例上调用装饰器内部的函数。您可以使用 `job.state()` 或使用 Braket 控制台监控作业的进度。以下代码示例显示了如何在上运行由五个状态组成的序列 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}
```

你可以像普通的 Python 函数一样通过调用函数来创建混合作业。但是，装饰器函数返回的是混合任务句柄而不是函数的结果。要在结果完成后检索结果，请使用 `job.result()`。

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

@`hybrid_job`装饰器中的 `device` 参数指定混合作业可以优先访问的设备，在本例中为SV1模拟器。要获得 QPU 优先级，必须确保函数中使用的设备 ARN 与装饰器中指定的设备 ARN 相匹配。为方便起见，您可以使用辅助函数 `get_job_device_arn()` 来捕获中声明的设备 ARN。@`hybrid_job`

### Note

自从在 Amazon EC2 上创建容器化环境以来，每个混合任务至少有一分钟的启动时间。因此，对于非常短的工作负载，例如单个电路或一批电路，使用量子任务可能就足够了。

## 超级参数

该 `run_hybrid_job()` 函数采用参数 `num_tasks` 来控制创建的量子任务的数量。混合作业会自动将其捕获为 [超参数](#)。



**Note**

超参数在 Braket 控制台中显示为字符串，限制为 2500 个字符。

## 指标和日志

在 `run_hybrid_job()` 函数中，使用来记录来自迭代算法的 `log_metrics` 指标。指标会自动绘制在 Braket 控制台页面的“混合作业”选项卡下。在混合作业运行期间，您可以使用 [Braket 成本跟踪器近乎实时地跟踪量子任务成本](#)。上面的示例使用指标名称“概率”来记录 [结果类型的](#) 第一个概率。

## 正在检索结果

混合作业完成后，您可以使用 `job.result()` 检索混合作业的结果。Braket 会自动捕获返回语句中的任何对象。请注意，该函数返回的对象必须是一个元组，每个元素都是可序列化的。例如，以下代码显示了一个工作和一个失败的示例。

```
@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 名称

默认情况下，此混合作业的名称是根据函数名称推断出来的。您也可以指定长度不超过 50 个字符的自定义名称。例如，在以下代码中，作业名称为“my-job-name”。

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

## 本地模式

通过向装饰器添加参数 `local=True` 来创建 [@@ 本地作业](#)。这将在本地计算环境（例如笔记本电脑）的容器化环境中运行混合作业。本地作业没有优先排队等候量子任务。对于诸如多节点或 MPI 之类的高级案例，本地作业可以访问所需的 Braket 环境变量。以下代码使用设备作为 SV1 模拟器创建本地混合作业。

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

支持所有其他混合作业选项。有关选项列表，请参阅 [braket.jobs.quantum\\_job\\_creation](#) 模块。

## 安装其他 Python 软件包和源代码

您可以自定义运行时环境以使用首选的 Python 包。您可以使用 `requirements.txt` 文件、软件包名称列表或 [自带容器 \(BYOC\)](#)。要使用 `requirements.txt` 文件自定义运行时环境，请参阅以下代码示例。

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

例如，该 `requirements.txt` 文件可能包含其他要安装的软件包。

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

或者，您可以按如下方式以 Python 列表的形式提供软件包名称。

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

其他源代码可以指定为模块列表，也可以指定为单个模块，如以下代码示例所示。

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

## 将数据保存并加载到混合作业实例中

### 指定输入训练数据

创建混合任务时，您可以通过指定亚马逊简单存储服务 (Amazon S3) 存储桶来提供输入训练数据集。您也可以指定本地路径，然后 Braket 会自动将数据上传到 Amazon S3，网址为。s3://<default\_bucket\_name>/jobs/<job\_name>/<timestamp>/data/<channel\_name>如果您指定本地路径，则频道名称默认为“input”。以下代码显示了来自本地路径的 numpy 文件。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 ...
```

对于 S3，必须使用get\_input\_data\_dir()辅助函数。

```
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")
```

您可以通过提供通道值和 S3 URI 或本地路径的字典来指定多个输入数据源。

```
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

当输入数据很大 (>1GB) 时，需要等待很长时间才能创建作业。这是由于本地输入数据首次上传到 S3 存储桶，然后将 S3 路径添加到任务请求中。最后，工作请求将提交给 Braket 服务。

## 将结果保存到 S3

要保存未包含在装饰函数的返回语句中的结果，必须将正确的目录附加到所有文件写入操作中。以下示例显示保存一个 numpy 数组和 matplotlib 图。

```
@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 ...
```

所有结果都压缩到名为的文件中`model.tar.gz`。您可以使用 Python 函数下载结果`job.result()`，也可以从 Braket 管理控制台的混合作业页面导航到结果文件夹。

## 保存并从检查点恢复

对于长时间运行的混合作业，建议定期保存算法的中间状态。您可以使用内置的`save_job_checkpoint()`帮助器函数，也可以将文件保存到`AMZN_BRAKET_JOB_RESULTS_DIR`路径中。后者可通过辅助函数获得`get_job_results_dir()`。

以下是使用混合作业装饰器保存和加载检查点的最小工作示例：

```
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()
```

在第一个混合作业中 `save_job_checkpoint()`，使用包含我们要保存的数据的字典调用。默认情况下，每个值都必须可序列化为文本。对于检查点更复杂的 Python 对象，例如 `numpy` 数组，你可以设置 `data_format = PersistedJobDataFormat.PICKLED_V4`。此代码在名为“checkpoints”的子文件夹下的混合作业工件 `<jobname>.json` 中创建并覆盖具有默认名称的检查点文件。

要创建一个新的混合作业以从检查点继续，我们需要传递前一个作业的混合作业 ARN `copy_checkpoints_from_job=job_arn` 在 `job_arn` 哪里。然后我们使用 `load_job_checkpoint(job_name)` 从检查点加载。

## 混合工作装饰者的最佳实践

### 拥抱异步性

使用 `decorator` 注释创建的混合作业是异步的，它们将在经典资源和量子资源可用后运行。您可以使用 `Braket Management Console` 或 `Amazon 监控算法的进度 CloudWatch`。当您提交算法以供运行时，`Braket` 会在可扩展的容器化环境中运行您的算法，并在算法完成后检索结果。

### 运行迭代变分算法

混合作业为你提供了运行迭代量子经典算法的工具。对于纯粹的量子问题，请使用 [量子任务](#) 或 [一批量子任务](#)。优先访问某些 QPU 最有利于长时间运行的变分算法，这些算法需要对 QPU 进行多次迭代调用，中间有经典处理。

### 使用本地模式进行调试

在 QPU 上运行混合作业之前，建议先在模拟器 `SV1` 上运行以确认其按预期运行。对于小规模测试，可以在本地模式下运行，以实现快速迭代和调试。

### 使用 [自带容器](#) (BYOC) 提高可重复性

将您的软件及其依赖项封装在容器化环境中，从而创建可重现的实验。通过将所有代码、依赖项和设置打包到容器中，可以防止潜在的冲突和版本控制问题。

### 多实例分布式仿真器

要运行大量电路，可以考虑使用内置的 `MPI` 支持，在单个混合作业中的多个实例上运行本地模拟器。有关更多信息，请参阅 [嵌入式模拟器](#)。

## 使用参数电路

您通过混合作业提交的参数电路会使用[参数化编译在特定 QPU 上自动编译](#)，以改善算法的运行时间。

### 定期检查点

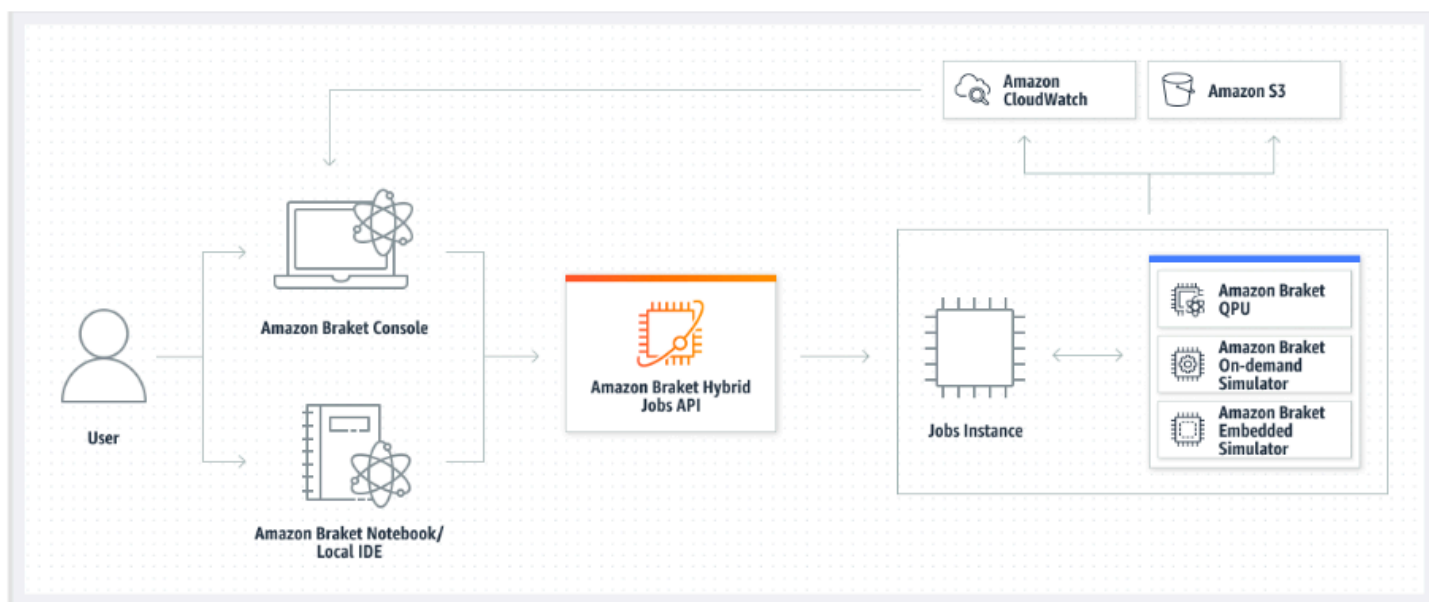
对于长时间运行的混合作业，建议定期保存算法的中间状态。

有关更多示例、用例和最佳实践，请参阅 [Amazon Bra GitHub](#) ket 示例。

## 使用 Amazon Braket 混合作业运行混合作业

要使用 Amazon Braket Hybrid Jobs 运行混合作业，您首先需要定义算法。您可以通过使用 Amazon Braket Python 软件开发工具包编写算法脚本以及其他依赖项文件来定义它，也可以使用 [Amazon Braket Python SDK](#) 或 [PennyLane](#)。如果您想使用其他（开源或专有）库，则可以使用 Docker 定义自己的自定义容器镜像，其中包括这些库。有关更多信息，请参阅[自带容器 \(BYOC\)](#)。

无论哪种情况，接下来都要使用 Amazon Braket 创建混合作业 API，在其中提供算法脚本或容器，选择混合作业要使用的目标量子设备，然后从各种可选设置中进行选择。为这些可选设置提供的默认值适用于大多数用例。对于运行混合作业（Hybrid Job）的目标设备，您可以在 QPU、按需模拟器（例如 SV1、DM1 或 TN1）或经典混合作业实例本身之间进行选择。使用按需模拟器或 QPU，您的混合作业容器可以对远程设备进行 API 调用。使用嵌入式仿真器，仿真器与算法脚本嵌入在同一个容器中。中的[闪电模拟 PennyLane 器](#)嵌入了默认的预建混合作业容器供您使用。如果您使用嵌入式 PennyLane 模拟器或自定义模拟器运行代码，则可以指定实例类型以及要使用的实例数量。有关每种选择的相关费用，请参阅 [Amazon Braket 定价页面](#)。



如果您的目标设备是按需模拟器或嵌入式模拟器，Amazon Braket 会立即开始运行混合任务。它启动混合作业实例（您可以在API调用中自定义实例类型），运行算法，将结果写入 Amazon S3，然后释放您的资源。此版本的资源可确保您只需为实际使用的资源付费。

每个量子处理单元 (QPU) 的并发混合任务总数受到限制。如今，在任何给定时间，QPU 上只能运行一个混合作业。队列用于控制允许运行的混合作业的数量，以免超过允许的限制。如果您的目标设备是 QPU，则混合作业将首先进入所选 QPU 的作业队列。Amazon Braket 启动所需的混合作业实例，并在设备上运行您的混合作业。在算法持续时间内，您的混合作业具有优先访问权限，这意味着混合作业中的量子任务优先于设备上排队的其他 Braket 量子任务，前提是该作业的量子任务每隔几分钟提交给 QPU 一次。混合任务完成后，资源就会被释放，这意味着您只需为实际使用的资源付费。

#### Note

设备是区域性的，您的混合任务与主设备在 AWS 区域同一设备上运行。

在模拟器和 QPU 目标场景中，您可以选择将自定义算法指标（例如哈密顿能量）定义为算法的一部分。这些指标会自动报告给亚马逊，CloudWatch 然后在亚马逊 Braket 控制台中以近乎实时的方式显示。

#### Note

如果您想使用基于 GPU 的实例，请务必使用 Braket 上嵌入式模拟器中提供的基于 GPU 的模拟器（例如 `lightning.gpu`）。如果您选择基于 CPU 的嵌入式模拟器之一（例如 `lightning.qubit`、或 `braket:default-simulator`），则不会使用 GPU，并且可能会产生不必要的成本。

## 创建你的第一个 Hybrid Job

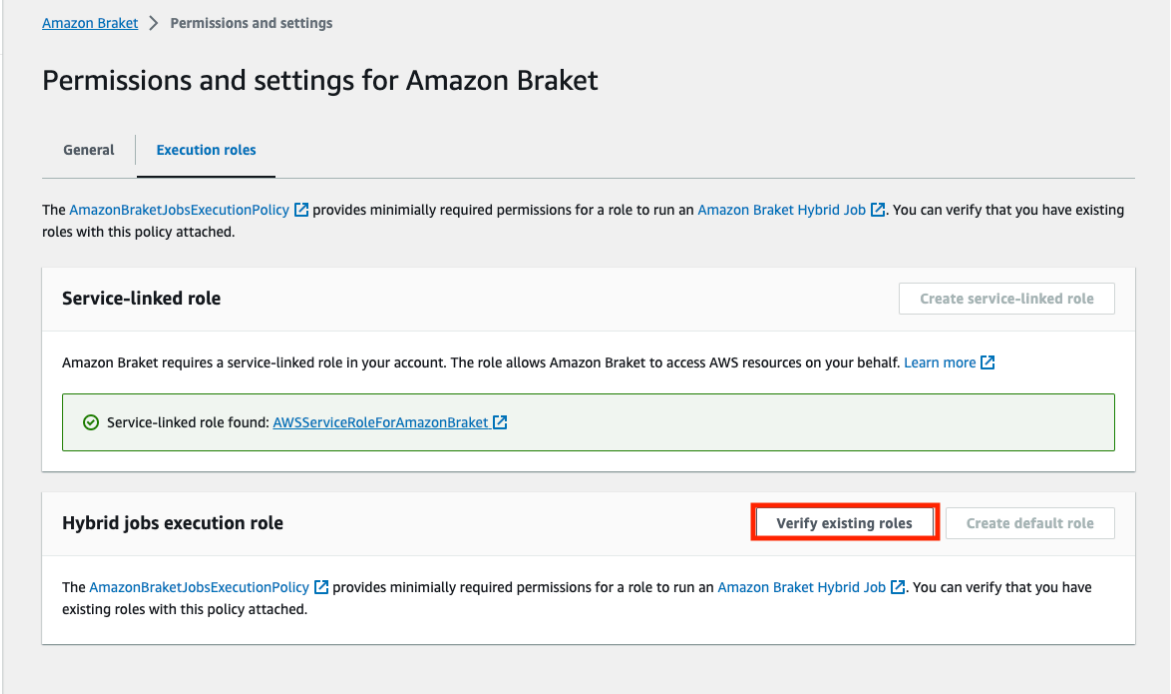
本节介绍如何使用 Python 脚本创建混合 Job。或者，要使用本地 Python 代码（例如您首选的集成开发环境 (IDE) 或 Braket 笔记本）创建混合作业，请参阅[将本地代码作为混合作业运行](#)。

本节内容：

- [设置权限](#)
- [创建并运行](#)
- [监控结果](#)

## 设置权限

在运行第一个混合作业之前，必须确保您有足够的权限来继续执行此任务。要确定您的权限是否正确，请从 Braket 控制台左侧的菜单中选择“权限”。Amazon Braket 的权限管理页面可帮助您验证您的一个现有角色是否具有足以运行混合任务的权限，或者指导您创建可用于运行混合任务的默认角色（如果您还没有此类角色）。



**Amazon Braket** ×

Dashboard  
Devices  
Notebooks  
Hybrid Jobs  
Quantum Tasks

Algorithm library

Announcements 1  
**Permissions and settings**

Amazon Braket > Permissions and settings

### Permissions and settings for Amazon Braket

General | **Execution roles**

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

**Service-linked role** Create service-linked role

Amazon Braket requires a service-linked role in your account. The role allows Amazon Braket to access AWS resources on your behalf. [Learn more](#)

✔ Service-linked role found: [AWSServiceRoleForAmazonBraket](#)

**Hybrid jobs execution role** Verify existing roles Create default role

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

要验证您的角色是否具有足够的权限来运行混合作业，请选择验证现有角色按钮。如果你这样做，你会收到一条消息，告知角色已找到。要查看角色的名称及其角色 ARN，请选择显示角色按钮。



Amazon Braket > Permissions and settings

## Permissions and settings for Amazon Braket

General | Execution roles

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

### Service-linked role

Create service-linked role

Amazon Braket requires a service-linked role in your account. The role allows Amazon Braket to access AWS resources on your behalf. [Learn more](#)

Service-linked role found: [AWSServiceRoleForAmazonBraket](#)

### Hybrid jobs execution role

Verify existing roles | Create default role

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

Roles were found with sufficient permissions to execute hybrid jobs.

Show roles

Role name	Role ARN
<a href="#">AmazonBraketJobsExecutionRole</a>	<a href="#">arn:aws:iam::260818742045:role/service-role/AmazonBraketJobsExecutionRole</a>

如果您的角色没有足够的权限来运行混合作业，则会收到一条消息，提示未找到此类角色。选择“创建默认角色”按钮以获取具有足够权限的角色。

**Amazon Braket** ×

Dashboard  
Devices  
Notebooks  
Hybrid Jobs  
Quantum Tasks

Algorithm library

Announcements **1**  
Permissions and settings

Amazon Braket > Permissions and settings

## Permissions and settings for Amazon Braket

General | **Execution roles**

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

**Service-linked role** Create service-linked role

Amazon Braket requires a service-linked role in your account. The role allows Amazon Braket to access AWS resources on your behalf. [Learn more](#)

✔ Service-linked role found: [AWSServiceRoleForAmazonBraket](#)

**Hybrid jobs execution role** Verify existing roles Create default role

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

❗ No roles found with the [AmazonBraketJobsExecutionPolicy](#) attached and [braket.amazonaws.com](#) as a trusted entity in IAM.

如果角色已成功创建，您将收到一条确认消息。

**Amazon Braket** ×

Dashboard  
Devices  
Notebooks  
Hybrid Jobs  
Quantum Tasks

Algorithm library

Announcements **1**  
Permissions and settings

Amazon Braket > Permissions and settings

## Permissions and settings for Amazon Braket

General | **Execution roles**

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

**Service-linked role** Create service-linked role

Amazon Braket requires a service-linked role in your account. The role allows Amazon Braket to access AWS resources on your behalf. [Learn more](#)

✔ Service-linked role found: [AWSServiceRoleForAmazonBraket](#)

**Hybrid jobs execution role** Verify existing roles Create default role

The [AmazonBraketJobsExecutionPolicy](#) provides minimally required permissions for a role to run an [Amazon Braket Hybrid Job](#). You can verify that you have existing roles with this policy attached.

✔ Created [AmazonBraketJobsExecutionRole](#) successfully.

如果您无权进行此查询，则将被拒绝访问。在这种情况下，请联系您的内部 AWS 管理员。


Amazon Braket > Permissions

## Permissions management for Amazon Braket

When you create a resource, such as an Amazon Braket notebook or job, you have the ability to specify the actions this resource can perform on your behalf by attaching an execution policy to an [IAM Role](#). You can create default roles for different Amazon Braket resources here. To build custom Roles for advanced use cases visit [IAM](#).

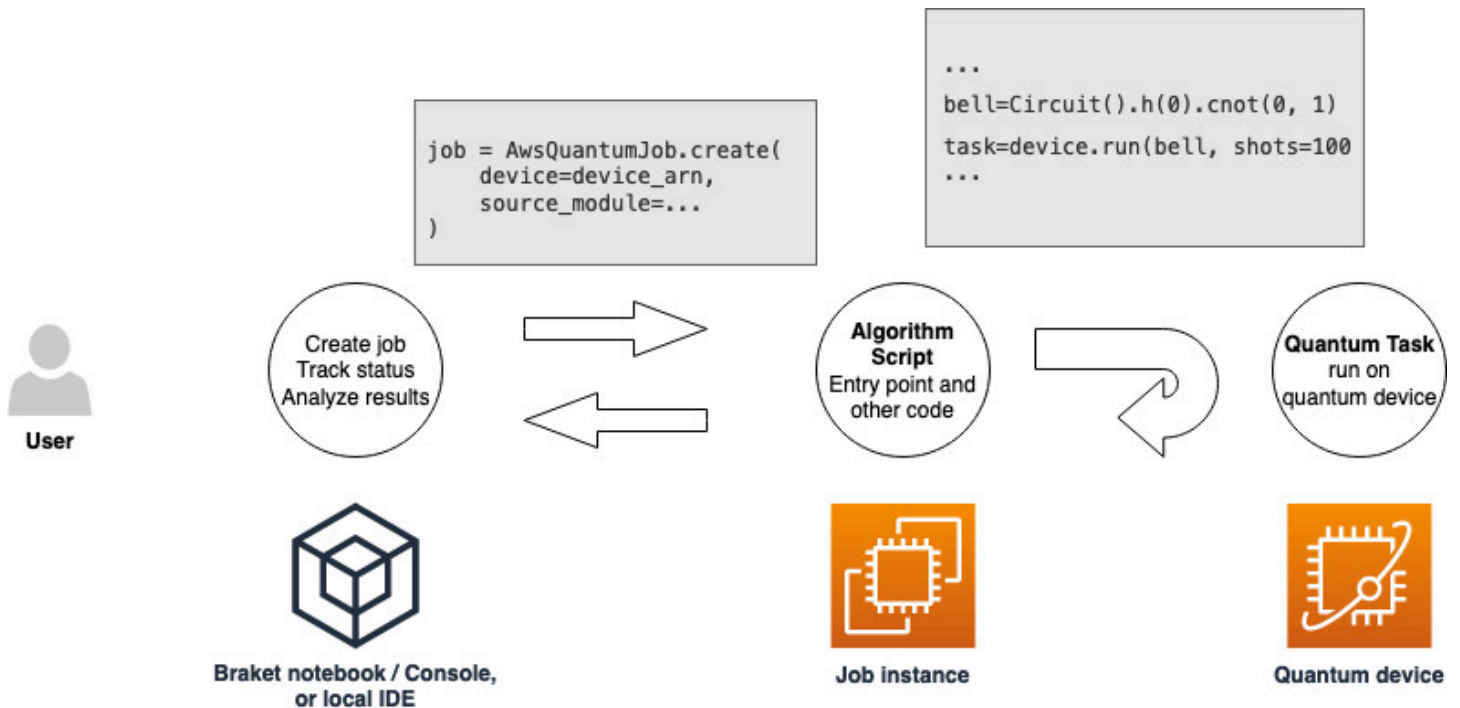
**Jobs** Verify existing roles Create default role

[Amazon Braket jobs](#) require the roles with managed policy [AmazonBraketJobsExecutionPolicy](#) attached, which provides minimally required permissions to an Amazon Braket job.

 **AccessDenied**  
User: arn:aws:sts::012345678912:assumed-role/SampleRoleName/username is not authorized to perform: iam:ListAttachedRolePolicies on resource: role AmazonBraketJobsExecutionRole with an explicit deny

## 创建并运行

拥有运行混合作业权限的角色后，就可以继续操作了。你的第一个 Braket 混合作业的关键部分是算法脚本。它定义了你要运行的算法，并包含作为算法一部分的经典逻辑和量子任务。除了算法脚本之外，您还可以提供其他依赖文件。算法脚本及其依赖关系被称为源模块。入口点定义了混合作业启动时要在源模块中运行的第一个文件或函数。



首先，考虑以下算法脚本的基本示例，该脚本创建了五个钟形状态并打印相应的测量结果。

```
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!")
```

将此文件名为 `algorithm_script.py` 保存在 Braket 笔记本或本地环境的当前工作目录中。`algorithm_script.py` 文件已 `start_here()` 作为计划的入口点。

接下来，在与 `algorithm_script.py` 文件相同的目录下创建 Python 文件或 Python 笔记本。此脚本启动混合作业并处理任何异步处理，例如打印我们感兴趣的状态或关键结果。此脚本至少需要指定您的混合作业脚本和主设备。

### Note

有关如何创建 Braket 笔记本或将文件（例如 `algorithm_script.py` 文件）上传到与笔记本相同的目录中的更多信息，请参阅[使用 Amazon Braket Python SDK 运行你的第一个电路](#)

对于基本的第一种情况，你要瞄准一个模拟器。无论你瞄准的是哪种类型的量子设备，无论是模拟器还是实际的量子处理单元 (QPU)，你在以下脚本 `device` 中指定的设备都用于调度混合作业，并且可以作为环境变量提供给算法脚本。 `AMZN_BRAKET_DEVICE_ARN`

### Note

您只能使用混合作业中 AWS 区域 可用的设备。亚马逊 Braket SDK 会自动选择此选项。AWS 区域例如，`us-east-1` 中的混合作业可以 IonQ 使用 `SV1`、`TN1` 和设备 `DM1`，但不能使用设备 `Rigetti`

如果您选择量子计算机而不是模拟器，Braket 会安排您的混合作业，以优先访问权限运行其所有量子任务。

```
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
)
```

该参数 `wait_until_complete=True` 设置了详细模式，以便您的作业在运行时打印实际作业的输出。您应该会看到类似于以下示例的输出。

```
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

您还可以通过传递其位置（本地目录或文件的路径或 tar.gz 文件的 S3 URI）将自定义模块与 `AwsQuantumjob.Create` 方法一起使用。[有关工作示例，请参阅 Amazon Braket 示例 Github 存储库中混合作业文件夹中的 `parallelize\_Training\_for\_qml.ipynb` 文件。](#)

## 监控结果

或者，您可以访问来自 Amazon 的日志输出 CloudWatch。为此，请转到作业详细信息页面左侧菜单上的日志组选项卡，选择日志组 `aws/braket/jobs`，然后选择包含该作业名称的日志流。在上述示例中，即为 `braket-job-default-1631915042705/algo-1-1631915190`。

CloudWatch > Log groups > /aws/braket/jobs > JobTest-autograd-1636588595/algo-1-1636588740

**Log events**  
You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

View as text  Actions

Filter events  Clear 1m 30m 1h 12h Custom

Timestamp	Message
There are older events to load. <a href="#">Load more</a> .	
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_gates.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_instruction.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_moments.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_noise.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_noise_helpers.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_noises.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_observable.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_observables.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_quantum_operator.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_quantum_operator_helpers.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_qubit.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_qubit_set.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_result_type.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/circuits/test_result_types.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/devices/
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/devices/test_local_simulator.py
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/jobs/
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/jobs/local/
2021-11-10T17:01:01.993-07:00	aws-amazon-braket-sdk-python-staging-3f885a94c09911b104eee053328733f34779fa6/test/unit_tests/braket/jobs/local/test_local_job.py

您还可以在控制台中查看混合作业的状态，方法是选择“混合作业”页面，然后选择“设置”。

Amazon Braket > Hybrid Jobs > braket-job-default-1693508892180

**braket-job-default-1693508892180**

**Summary**

Status ✔ COMPLETED Runtime 00:01:21 Hybrid job logs [View in CloudWatch](#)

**Settings** | Events | Monitor | Quantum Tasks | Tags

**Details**

Hybrid job name braket-job-default-1693508892180	Hybrid job ARN <a href="#">arn:aws:braket:us-west-2:260818742045:job/braket-job-default-1693508892180</a>	Event times
Device arn:aws:braket::device/quantum-simulator/amazon/sv1	Execution role <a href="#">arn:aws:iam::260818742045:role/service-role/AmazonBraketJobsExecutionRole</a>	Created at Aug 31, 2023 19:08 (UTC)
Status reason —		Started at Aug 31, 2023 19:09 (UTC)
		Ended at Aug 31, 2023 19:10 (UTC)

**Source code and instance configuration**

Entry point job_test_script:start_here	Instance type m5.large
---	---------------------------

**Stopping conditions**

Max runtime (seconds) 432000
---------------------------------

您的混合作业在运行时会在 Amazon S3 中生成一些工件。S3 存储桶的默认名称为 `amazon-braket-  
<region>-<accountid>`，内容位于 `jobs/<jobname>/<timestamp>` 目录中。使用 Braket

Python SDK 创建混合作业 `code_location` 时，您可以通过指定其他位置来配置存储这些工件的 S3 位置。

#### Note

此 S3 存储桶必须与您的任务脚本位于同一 AWS 区域 位置。

该 `jobs/<jobname>/<timestamp>` 目录包含一个子文件夹，文件中包含入口点脚本的 `model.tar.gz` 输出。还有一个名为 `script` 的目录，`source.tar.gz` 文件中包含您的算法脚本工件。实际量子任务的结果位于名为 `jobs/<jobname>/tasks` 的目录中。

## 输入、输出、环境变量和辅助函数

除了构成完整算法脚本的一个或多个文件外，您的混合作业还可以有其他输入和输出。当您的混合作业启动时，Amazon Braket 会将创建混合作业时提供的输入复制到运行算法脚本的容器中。混合任务完成后，算法期间定义的所有输出都将复制到指定的 Amazon S3 位置。

#### Note

算法指标是实时报告的，不遵循此输出程序。

Amazon Braket 还提供了多个环境变量和辅助函数，以简化与容器输入和输出的交互。

本节介绍了 Amazon Braket Python SDK 提供的 `AwsQuantumJob.create` 函数的关键概念及其与容器文件结构的映射。

本节内容：

- [输入](#)
- [输出](#)
- [环境变量](#)
- [辅助函数](#)

## 输入

**输入数据：**通过使用 `input_data` 参数指定设置为字典的输入数据文件，可以将输入数据提供给混合算法。用户在 SDK 的 `AwsQuantumJob.create` 函数中定义 `input_data` 参数。这会将输入数据复制



到环境变量所给位置的容器文件系统"AMZN\_BRAKET\_INPUT\_DIR"。有关如何在混合算法中使用输入数据的几个示例，请参阅[带有 Amazon Braket Hybrid Jobs 的 QAOA PennyLane 和 Amazon Braket Hybrid Jobs Jupyter 笔记本中的量子机器学习](#)。

### Note

当输入数据很大 (>1GB) 时，混合作业需要很长时间才能提交。这是因为本地输入数据将首先上传到 S3 存储桶，然后将 S3 路径添加到混合任务请求中，最后，混合任务请求将提交给 Braket 服务。

超参数：如果传入 hyperparameters，则它们在环境变量"AMZN\_BRAKET\_HP\_FILE"下可用。

### Note

[有关如何创建超参数和输入数据然后将这些信息传递给混合作业脚本的更多信息，请参阅使用超参数部分和此 github 页面。](#)

检查点：要指定 job-arn 要在新的混合作业中使用哪个检查点，请使用 `copy_checkpoints_from_job` 命令。此命令将检查点数据复制到新的混合作业中，使其在作业运行时在环境变量 `AMZN_BRAKET_CHECKPOINT_DIR` 给出的路径上可用。 `checkpoint_configs3Uri` 默认值为 `None`，这意味着来自其他混合作业的检查点数据将不会用于新的混合作业。

## 输出

量子任务：量子任务结果存储在 S3 位置 `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks`。

Job 结果：您的算法脚本保存到环境变量给定目录的所有内容 `"AMZN_BRAKET_JOB_RESULTS_DIR"` 都将复制到中指定的 S3 位置 `output_data_config`。如果您未指定此值，则默认为 `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data`。我们提供 SDK 帮助器函数 `save_job_result`，当从算法脚本调用时，您可以使用该函数以字典的形式方便地存储结果。

检查点：如果要使用检查点，可以将其保存在环境变量给出的目录中 `"AMZN_BRAKET_CHECKPOINT_DIR"`。您也可以 `save_job_checkpoint` 改用 SDK 帮助器函数。

算法指标：您可以将算法指标定义为算法脚本的一部分，这些指标将在混合作业运行时发送到 Amazon CloudWatch 并实时显示在 Amazon Braket 控制台中。有关如何使用算法指标的示例，请参阅[使用 Amazon Braket 混合任务运行 QAOA 算法](#)。

## 环境变量

Amazon Braket 提供了多个环境变量来简化与容器输入和输出的交互。以下代码列出了 Braket 使用的环境变量。

```
# 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"]
```

## 辅助函数

Amazon Braket 提供了多个辅助函数，以简化与容器输入和输出的交互。这些辅助函数将从用于运行 Hybrid Job 的算法脚本中调用。以下示例演示了如何使用它们。

```
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
```

## 保存作业结果

您可以保存算法脚本生成的结果，以便可以从混合作业脚本中的混合作业对象以及 Amazon S3 的输出文件夹（名为 model.tar.gz 的 tar 压缩文件中）获得这些结果。

必须使用 JavaScript 对象表示法 (JSON) 格式将输出保存在文件中。如果无法将数据轻松序列化为文本（例如 numpy 数组），则可以传入一个使用腌制数据格式进行序列化的选项。有关更多详细信息，请参阅 [braket.jobs.data\\_persisting](#) 模块。

要保存混合作业的结果，请在算法脚本中添加以下用 #ADD 注释的行。

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

然后，您可以通过附加带有 `#ADD print(job.result())` 注释的行来显示作业脚本中的作业结果。

```
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
```

在此示例中，我们删除了 `wait_until_complete=True` 以抑制详细输出。您可以将其重新添加以进行调试。当您运行这个混合作业时，它会每隔 10 秒输出一行标识符和 `job-arn`，然后是混合作业的状态，然后是混合作业的状态 `COMPLETED`，之后它会向您显示钟形回路的结果。请参阅以下示例。

```
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}]}
```

## 使用检查点保存并重启混合作业

您可以使用检查点保存混合作业的中间迭代。在上一节的算法脚本示例中，您将添加以下用 #ADD 注释的行来创建检查点文件。

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

当你运行混合作业时，它会在检查点目录中的混合作业工件中使用默认路径创建文件-checkpoint-1.json <jobname>。/opt/jobs/checkpoints除非您要更改此默认路径，否则混合作业脚本将保持不变。

如果要从之前的混合作业生成的检查点加载混合作业，则算法脚本会使用from braket.jobs import load\_job\_checkpoint。加载到算法脚本中的逻辑如下所示。

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

加载此检查点后，您可以根据加载到的内容继续执行逻辑checkpoint-1。

### Note

checkpoint\_file\_suffix 必须与之前在创建检查点时指定的后缀匹配。

您的编排脚本需要指定前一个混合作业job-arn中的，该行用 #ADD 注释。

```
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
)
```

## 为算法脚本定义环境

Amazon Braket 支持三种由容器为算法脚本定义的环境：

- 基础容器（如果未指定，则image\_uri为默认容器）
- 一个装有 Tensorflow 的容器和 PennyLane
- 带有 PyTorch 和的容器 PennyLane

下表提供了有关容器及其包含的库的详细信息。

### 亚马逊 Braket 容器

类型	PennyLane 与 TensorFlow	PennyLane 与 PyTorch	Pennylane
Base	292282985366.dkr.ecr.us-east-1.amazonaws.com	292282985366.dkr.ecr.us-west-2.amazonaws.com	292282985366.dkr.ecr.us-west-2.amazonaws.com /

类型	PennyLane 与 TensorFlow	PennyLane 与 PyTorch	Pennylane
	naws.com /amazon-braket-tensorflow-jobs: latest	naws.com /amazon-braket-pytorch-jobs: latest	amazon-braket-base-jobs: latest
继承的库	<ul style="list-style-type: none"> <li>• awscli</li> <li>• numpy</li> <li>• pandas</li> <li>• scipy</li> </ul>	<ul style="list-style-type: none"> <li>• awscli</li> <li>• numpy</li> <li>• pandas</li> <li>• scipy</li> </ul>	
其他库	<ul style="list-style-type: none"> <li>• 亚马逊 braket-默认模拟器</li> <li>• 亚马逊 braket-pennylane-plugin</li> <li>• 亚马逊 braket-schemas</li> <li>• amazon-braket-sdk</li> <li>• ipykernel</li> <li>• keras</li> <li>• matplotlib</li> <li>• 网络x</li> <li>• openbabel</li> <li>• PennyLane</li> <li>• protobuf</li> <li>• psi4</li> <li>• rsa</li> <li>• PennyLane-闪电般的 GPU</li> <li>• cuQuantum</li> </ul>	<ul style="list-style-type: none"> <li>• 亚马逊 braket-默认模拟器</li> <li>• 亚马逊 braket-pennylane-plugin</li> <li>• 亚马逊 braket-schemas</li> <li>• amazon-braket-sdk</li> <li>• ipykernel</li> <li>• keras</li> <li>• matplotlib</li> <li>• 网络x</li> <li>• openbabel</li> <li>• PennyLane</li> <li>• protobuf</li> <li>• psi4</li> <li>• rsa</li> <li>• PennyLane-闪电般的 GPU</li> <li>• cuQuantum</li> </ul>	<ul style="list-style-type: none"> <li>• 亚马逊 braket-默认模拟器</li> <li>• 亚马逊 braket-pennylane-plugin</li> <li>• 亚马逊 braket-schemas</li> <li>• amazon-braket-sdk</li> <li>• awscli</li> <li>• boto3</li> <li>• ipykernel</li> <li>• matplotlib</li> <li>• 网络x</li> <li>• numpy</li> <li>• openbabel</li> <li>• pandas</li> <li>• PennyLane</li> <li>• protobuf</li> <li>• psi4</li> <li>• rsa</li> <li>• scipy</li> </ul>

您可以在 [aws/amazon-braket-containers](#) 上查看和访问开源容器定义。选择最符合您的用例的容器。容器必须位于 AWS 区域 从中调用混合作业的容器中。在创建混合作业时，您可以通过在混合作业脚本中的 `create(...)` 调用中添加以下三个参数之一来指定容器镜像。由于 Amazon Braket 容器具有互联网连接，因此您可以在运行时将其他依赖项安装到您选择的容器中（以启动或运行时为代价）。以下示例适用于 us-west-2 区域。

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

`image-uris` 也可以使用 Amazon Braket SDK 中的 `retrieve_image()` 函数进行检索。以下示例说明如何从 us-west-2 AWS 区域 2 中检索它们。

```
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")
```

## 使用超参数

创建混合作业时，您可以定义算法所需的超参数，例如学习率或步长。超参数值通常用于控制算法的各个方面，并且通常可以对其进行调整以优化算法的性能。要在 Braket 混合作业中使用超参数，您需要将其名称和值明确指定为字典。请注意，这些值必须是字符串数据类型。在搜索最优值集时，您可以指定要测试的超参数值。使用超参数的第一步是将超参数设置并定义为字典，这可以在以下代码中看到：

```
#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
}
```

然后，您将传递上面给出的代码片段中定义的超参数，以便在您选择的算法中使用，其内容如下所示：

```
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

要了解有关输入数据的更多信息，请参阅[输入](#)部分。

然后，将使用以下代码将超参数加载到混合作业脚本中：

```
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**

有关如何将输入数据和设备 arn 等信息传递到混合作业脚本的更多信息，请参阅此 [github 页面](#)。

[QAOA 在 Amazon Braket Hybrid Jobs 和 PennyLane Amazon Braket Hybrid Jobs 教程中](#)提供了几个对学习如何使用超参数非常有用的指南，这些指南对学习如何使用超参数非常有用。

## 配置混合作业实例以运行算法脚本

根据您的算法，您可能有不同的要求。默认情况下，Amazon Braket 在 `m1.m5.large` 实例上运行您的算法脚本。但是，在创建混合作业时，您可以使用以下导入和配置参数自定义此实例类型。

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="m1.p3.8xlarge"), # Use NVIDIA Tesla
    V100 instance with 4 GPUs.
    ...
),
```

如果您正在运行嵌入式仿真并在设备配置中指定了本地设备，则还可以 `InstanceConfig` 通过指定 `InstanceCount` 并将其设置为大于一个来请求多个实例。上限为 5。例如，您可以按如下方式选择 3 个实例。

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="m1.p3.8xlarge", instanceCount=3), #
    Use 3 NVIDIA Tesla V100
    ...
),
```

当您使用多个实例时，请考虑使用数据 `parallel` 功能分配混合作业。有关如何查看 [此 Braket 示例](#) 的更多详细信息，请参阅以下示例笔记本。

以下三个表格列出了标准实例、计算优化型实例和加速计算实例的可用实例类型和规格。

**Note**

要查看混合任务的默认经典计算实例配额，请参阅[此页面](#)。

标准实例	vCPU	内存
ml.m5.large (默认)	2	8 GiB
ml.m5.xlarge	4	16 GiB
ml.m5.2xlarge	8	32 GiB
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB
ml.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

计算优化型实例	vCPU	内存
ml.c4.xlarge	4	7.5 GiB
ml.c4.2xlarge	8	15GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB
ml.c5.xlarge	4	8 GiB

计算优化型实例	vCPU	内存
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
ml.c5n.xlarge	4	10.5 GiB
ml.c5n.2xlarge	8	21 GiB
ml.c5n.4xlarge	16	42 GiB
ml.c5n.9xlarge	36	96 GiB
ml.c5n.18xlarge	72	192 GiB

加速计算实例	vCPU	内存
ml.p2.xlarge	4	61 GiB
ml.p2.8xlarge	32	488 GiB
ml.p2.16xlarge	64	732 GiB
ml.p3.2xlarge	8	61 GiB
ml.p3.8xlarge	32	244 GiB
ml.p3.16xlarge	64	488 GiB
ml.g4dn.xlarge	4	16 GiB
ml.g4dn.2xlarge	8	32 GiB
ml.g4dn.4xlarge	16	64 GiB

加速计算实例	vCPU	内存
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

### Note

p3 实例在 us-west-1 中不可用。如果您的混合任务无法预置请求的 ML 计算容量，请使用其他区域。

每个实例使用 30 GB 的数据存储 (SSD) 的默认配置。但是，您可以按照与配置相同的方式调整存储 `instanceType`。以下示例说明如何将总存储空间增加到 50 GB。

```
from braket.jobs.config import InstanceConfig

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

## 取消混合 Job

您可能需要取消处于非终止状态的混合作业。这可以在控制台中完成，也可以使用代码完成。

要在控制台中取消混合作业，请从“混合作业”页面中选择要取消的混合作业，然后从“操作”下拉菜单中选择“取消混合作业”。

The screenshot shows the Amazon Braket console interface. On the left is a navigation sidebar with options like Dashboard, Devices, Notebooks, Hybrid Jobs, Quantum Tasks, Algorithm library, Announcements, and Permissions and settings. The main content area displays a table of Hybrid Jobs (4). The table has columns for Hybrid job name, Status, Device, and a timestamp. One job, 'braket-job-default-1693600353661', is in a 'QUEUED' state. An 'Actions' menu is open for this job, with 'Cancel hybrid job' highlighted in red. Other actions include 'View hybrid job' and 'Manage tags'.

	Hybrid job name	Status	Device	
<input type="radio"/>	<a href="#">braket-job-default-1693603871840</a>	⊘ CANCELLED	arn:aws:braket:us-east-1::device/gpu/ionq/Aria-2	Sep 01, 2023 21:31 (UTC)
<input checked="" type="radio"/>	<a href="#">braket-job-default-1693600353661</a>	⌚ QUEUED	arn:aws:braket:us-east-1::device/gpu/ionq/Aria-2	Sep 01, 2023 20:32 (UTC)
<input type="radio"/>	<a href="#">test-job-example</a>	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Jun 02, 2022 22:26 (UTC)
<input type="radio"/>	<a href="#">Test-ashlhans</a>	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	May 25, 2022 19:50 (UTC)

要确认取消，请在出现提示时在输入字段中输入“取消”，然后选择“确定”。

The screenshot shows a confirmation dialog box titled 'Cancel Job "JobTest-autograd-1637034526"?'. It contains a warning icon and a list of bullet points:
 

- Cancelling the specified job can't be undone.
- Cancelling will terminate the container immediately and does a best effort to cancel all of the related tasks that are in a non-terminal state.
- Tasks that have already completed will still be charged.
- You can create a new job using your checkpoint data, if you defined it, to rerun your experiments

 Below the list, it says 'To confirm cancellation, enter *cancel* in the text input field.' A text input field contains the word 'cancel'. At the bottom right, there are two buttons: 'Cancel' and 'Ok', with the 'Ok' button highlighted in red.

要使用 Braket Python SDK 中的代码取消混合作业，请使用识别混合作业，然后对其调用cancel命令，如以下代码所示。job\_arn

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

该cancel命令会立即终止经典的混合作业容器，并尽最大努力取消所有仍处于非终端状态的相关量子任务。

## 使用参数化编译加快混合作业的速度

Amazon Braket 支持在某些 QPU 上进行参数化编译。这使您能够通过只编译一次电路，而不是为混合算法中的每次迭代编译一次来减少与计算成本高昂的编译步骤相关的开销。这可以显著缩短 Hybrid Jobs 的运行时间，因为您无需在每一步都重新编译电路。只需将参数化电路作为 Braket Hybrid Job 提交到我们支持的 QPU 即可。对于长时间运行的混合作业，Braket 在编译电路时会自动使用硬件提供商提供的更新的校准数据，以确保获得最高质量的结果。

要创建参数化电路，首先需要在算法脚本中提供参数作为输入。在这个例子中，我们使用了一个小的参数化电路，忽略了每次迭代之间的任何经典处理。对于典型的工作负载，您需要批量提交许多电路并执行经典处理，例如在每次迭代中更新参数。

```
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.")
```

您可以使用以下作业脚本提交算法脚本以混合 Job 的形式运行。在支持参数化编译的 QPU 上运行 Hybrid Job 时，仅在第一次运行时才编译电路。在接下来的运行中，将重复使用编译后的电路，无需任何额外的代码行即可提高 Hybrid Job 的运行性能。

```
from braket.aws import AwsQuantumJob

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

)

**Note**

所有来自脉冲电平程序的超导、基于栅极的 QPU 都支持参数编译，Rigetti Computing 脉冲电 Oxford Quantum Circuits 程序除外。

## PennyLane 与 Amazon Braket 一起使用

混合算法是同时包含经典指令和量子指令的算法。经典指令在经典硬件（EC2 实例或笔记本电脑）上运行，量子指令要么在模拟器上运行，要么在量子计算机上运行。我们建议您使用混合作业功能运行混合算法。有关更多信息，请参阅[何时使用 Amazon Braket 任务](#)。

Amazon Braket 使您能够在 Braket PennyLane 插件的帮助下设置和运行混合量子算法，或者使用 Amazon Braket Python SDK 和示例笔记本存储库。Amazon 基于 SDK 的 Braket 示例笔记本使您无需 PennyLane 插件即可设置和运行某些混合算法。但是，我们建议这样 PennyLane 做，因为它提供了更丰富的体验。

### 关于混合量子算法

混合量子算法对当今的行业很重要，因为当代量子计算设备通常会产生噪声，因此会产生错误。计算中添加的每个量子门都会增加增加噪声的机会；因此，长期运行的算法可能会被噪声所淹没，从而导致计算错误。

诸如肖尔（[量子相位估计示例](#)）或格罗弗（[格罗弗的例子](#)）之类的纯量子算法需要数千或数百万次运算。出于这个原因，它们对于现有的量子器件来说可能是不切实际的，这些器件通常被称为噪声中量子（NISQ）器件。

在混合量子算法中，量子处理单元（QPU）充当经典 CPU 的协处理器，专门用于加快经典算法中的某些计算。电路执行时间大大缩短，触手可及。

## 带有 Amazon Braket PennyLane

Amazon Braket 为 [PennyLane](#) 围绕量子微分编程概念构建的开源软件框架提供支持。你可以使用这个框架来训练量子电路，就像训练神经网络来寻找量子化学、量子机器学习和优化中计算问题的解决方案一样。

该 PennyLane 库为熟悉的机器学习工具（包括 PyTorch 和 TensorFlow）提供了接口，使量子电路训练变得快速而直观。



- PennyLane 库 —— PennyLane 已预先安装在 Amazon Braket 笔记本电脑中。要从中访问 Amazon Braket 设备 PennyLane，请打开笔记本并使用以下命令导入 PennyLane 库。

```
import pennylane as qml
```

教程笔记本可帮助您快速入门。或者，您可以通过自己选择的 IDE PennyLane 在 Amazon Braket 上使用。

- AmazonBraket PennyLane 插件 — 要使用你自己的 IDE，你可以手动安装 Amazon Braket PennyLane 插件。该插件 PennyLane 与 [Amazon Braket Python SDK](#) 连接，因此你可以在 Amazon Braket 设备 PennyLane 上运行电路。要安装该 PennyLane 插件，请使用以下命令。

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

以下示例演示了如何在中 PennyLane 设置对 Amazon Braket 设备的访问权限：

```
# 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)
```

有关教程示例和更多信息 PennyLane，请参阅 [Amazon Braket 示例存储库](#)。

AmazonBraket PennyLane 插件使您只需一行代码即可在 Amazon PennyLane Braket QPU 和嵌入式仿真器设备之间切换。它提供了两个 Amazon Braket 量子设备可供使用 PennyLane：

- `braket.aws.qubit`用于与 Amazon Braket 服务的量子设备一起运行，包括 QPU 和模拟器
- `braket.local.qubit`用于使用 Amazon Braket SDK 的本地模拟器运行

Amazon Braket PennyLane 插件是开源的。你可以从[PennyLane 插件 GitHub 存储库](#)中安装它。

有关的更多信息 PennyLane，请参阅[PennyLane 网站](#)上的文档。

## Amazon Braket 中的混合算法示例笔记本

Amazon Braket 确实提供了各种不依赖 PennyLane 插件来运行混合算法的笔记本示例。你可以开始使用这些说明变分方法的 [Amazon Braket 混合示例笔记本](#)中的任何一个，例如量子近似优化算法 (QAOA) 或变分量子特征求解器 (VQE)。

Amazon Braket 示例笔记本依赖于 [Amazon Braket Python](#) SDK。SDK 提供了一个通过 Amazon Braket 与量子计算硬件设备进行交互的框架。它是一个开源库，旨在帮助您完成混合工作流程的量子部分。

您可以使用我们的[示例笔记本](#)进一步探索 Amazon Braket。

## 带有嵌入式 PennyLane 仿真器的混合算法

Amazon Braket Hybrid Jobs 现在配备了基于 CPU 和 GPU 的高性能嵌入式模拟器。[PennyLane](#)该系列嵌入式模拟器可以直接嵌入到您的混合作业容器中，包括快速状态向量 `lightning.qubit` 模拟器、使用 NVIDIA 的 `cuQuantum` 库加速的 `lightning.gpu` 仿真器等。这些嵌入式仿真器非常适合变分算法，例如量子机器学习，这些算法可以从高级方法（例如伴随微分法）中受益。您可以在一个或多个 CPU 或 GPU 实例上运行这些嵌入式模拟器。

借助 Hybrid Jobs，您现在可以使用经典协处理器和 QPU 的组合、Amazon Braket 按需模拟器（例如）或直接使用中的嵌入式仿真器来运行变分算法代码。SV1 PennyLane

嵌入式模拟器已经在 Hybrid Jobs 容器中可用，你只需要用 `@hybrid_job` 装饰你的主 Python 函数即可。要使用 PennyLane `lightning.gpu` 模拟器，您还需要在中指定 GPU 实例，`InstanceConfig` 如以下代码片段所示：

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

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

要开始使用带有 Hybrid Jobs 的 PennyLane 嵌入式模拟器，请参阅[示例笔记本](#)。

## PennyLane 使用 Amazon Braket 模拟器开启伴随渐变

借助 Amazon Braket 的 PennyLane 插件，在局部状态向量模拟器或 SV1 上运行时，您可以使用伴随微分法计算梯度。

**注意：**要使用伴随微分法，必须在 “” 和 “不是 `diff_method='adjoint'`” `diff_method='device'` 中 `qnode` 指定。请参阅以下示例。

```
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

目前，PennyLane 将计算 QAOA Hamiltonians 的分组指数，并使用它们将哈密顿函数拆分为多个期望值。如果要在从中运行 QAOA 时使用 SV1 的伴随微分能力 PennyLane，则需要通过移除分组指数来重建成本哈密顿模型，如下所示：`cost_h`, `mixer_h = qml.qaoa.max_clique(g, constrained=False)` `cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)`

## 使用 Amazon Braket 混合任务并 PennyLane 运行 QAOA 算法

在本节中，您将使用所学知识与参数化编译 PennyLane 一起编写实际的混合程序。您可以使用算法脚本来解决量子近似优化算法 (QAOA) 问题。该程序创建了一个与经典的 Max Cut 优化问题相对应的成

本函数，指定了参数化的量子电路，并使用简单的梯度下降方法来优化参数，从而使成本函数最小化。在此示例中，为了简单起见，我们在算法脚本中生成问题图，但对于更典型的用例，最佳做法是通过输入数据配置中的专用通道提供问题规范。该标志`parametrize_differentiable`默认为`True`因此您可以通过在支持的 QPU 上进行参数化编译来自动获得提高运行时性能的好处。

```
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

所有来自脉冲电平程序的超导、基于栅极的 QPU 都支持参数编译，Rigetti Computing 脉冲电 Oxford Quantum Circuits 程序除外。

## 使用来自的嵌入式模拟器加速混合工作负载 PennyLane

让我们来看看如何使用 Amazon Braket Hybrid Jobs PennyLane 上的嵌入式模拟器来运行混合工作负载。PennyLane 基于 GPU 的嵌入式仿真器使用 [Nvidia cuQuantum 库](#) 来加速电路仿

真。lightning.gpu嵌入式 GPU 模拟器已在所有 Braket [作业容器中进行了预配置](#)，用户可以开箱即用。在本页中，我们将向您展示如何使用lightning.gpu来加快混合工作负载的速度。

## lightning.gpu用于量子近似优化算法工作负载

考虑本笔记本中的[量子近似优化算法 \(QAOA\) 示例](#)。要选择嵌入式模拟器，请将device参数指定为以下形式的字符串："local:<provider>/<simulator\_name>"。例如，您可以设置"local:pennylane/lightning.gpu"lightning.gpu。启动时提供给 Hybrid Job 的设备字符串将作为环境变量传递给该作业"AMZN\_BRAKET\_DEVICE\_ARN"。

```
device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
prefix, device_name = device_string.split("/")
device = qml.device(simulator_name, wires=n_wires)
```

在本页中，让我们比较两个嵌入式 PennyLane 状态矢量模拟器lightning.qubit ( 基于 CPU ) 和lightning.gpu ( 基于 GPU )。为了计算各种梯度，你需要为模拟器提供一些自定义的门分解。

现在，您可以准备混合作业启动脚本了。您将使用两种实例类型运行 QAOA 算法：m5.2xlarge和。p3.2xlarge和m5.2xlarge实例类型相当于标准的开发者笔记本电脑。p3.2xlarge这是一个加速计算实例，它具有单个 NVIDIA Volta GPU 和 16GB 内存。

你所有的混合工作都是一样的。hyperparameters要尝试不同的实例和模拟器，你所需要做的就是更改两行，如下所示。

```
# 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')
```

或者：

```
# 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**

如果您将指定`instance_config`为使用基于 GPU 的实例，但选择作为基于 CPU 的嵌入式模拟器 (`lightning.qubit`)，则不会使用 GPU。如果你想瞄准 GPU，一定要使用基于 GPU 的嵌入式模拟器！

首先，您可以创建两个混合作业，并在具有 18 个顶点的图表上使用 QAOA 求解 Max-Cut。这意味着一个 18 量子比特的电路——相对较小，在笔记本电脑或实例上快速运行是可行的。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,
)
```

m5.2xlarge实例的平均迭代时间约为 25 秒，而p3.2xlarge实例的平均迭代时间约为 12 秒。对于这个 18 量子比特的工作流程，GPU 实例为我们提供了 2 倍的加速。如果你看一下Amazon Braket Hybrid Jobs的[定价页面](#)，你可以看到一个m5.2xlarge实例的每分钟费用为0.00768美元，而实例的每分钟费用为0.06375美元。p3.2xlarge像你在这里所做的那样，总共运行 5 次迭代，使用 CPU 实例的费用为 0.016 美元，使用 GPU 实例的费用为 0.06375 美元，两者都非常便宜！

现在让我们让问题变得更难，尝试在 24 个顶点图上求解一个 Max-Cut 问题，该问题将转换为 24 个量子比特。在相同的两个实例上再次运行混合作业并比较成本。



**Note**

您将看到，在 CPU 实例上运行此混合作业的时间可能约为五个小时！

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

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

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

m5.2xlarge实例的平均迭代时间约为一小时，而p3.2xlarge实例的平均迭代时间约为两分钟。对于这个更大的问题，GPU实例要快一个数量级！要从这种加速中受益，你所要做的就是更改两行代码，换掉实例类型和使用的本地模拟器。像这里所做的那样，总共运行5次迭代，使用CPU实例的费用约为2.27072美元，使用GPU实例的费用约为0.775625美元。CPU使用率不仅更昂贵，而且运行时间也更长。使用由NVIDIA支持的嵌入式仿真器AWS，使用上可用PennyLane的GPU实例加速这一工作流程CuQuantum，使您能够以更低的总成本和更短的时间运行具有中间量子比特计数（介于20到30之间）的工作流程。这意味着，即使问题太大，无法在笔记本电脑或类似大小的实例上快速运行，您也可以尝试量子计算。

## 量子机器学习和数据并行性

如果您的工作负载类型是基于数据集训练的量子机器学习(QML)，则可以使用数据并行性进一步加快工作负载。在QML中，模型包含一个或多个量子电路。该模型可能还包含也可能不包含经典神经网络

络。使用数据集训练模型时，会更新模型中的参数以最小化损失函数。损失函数通常是针对单个数据点定义的，以及整个数据集的平均损失的总损失。在 QML 中，通常先串行计算损耗，然后再求平均为梯度计算的总损耗。此过程非常耗时，尤其是在有数百个数据点的情况下。

由于一个数据点的损失不依赖于其他数据点，因此可以并行评估损失！可以同时评估与不同数据点相关的损失和梯度。这就是所谓的数据并行性。借助 SageMaker 分布式数据并行库，Amazon Braket Hybrid Jobs 使您可以更轻松地了解利用数据并行性来加速训练。

考虑以下 QML 数据并行工作负载，该工作负载使用知名 UCI 存储库中的 [Sonar 数据集](#) 数据集作为二进制分类的示例。声纳数据集有 208 个数据点，每个数据点有 60 个特征，这些特征是从材料上反弹的声纳信号中收集的。每个数据点要么被标记为“M”（代表地雷），要么标记为“R”（表示岩石）。我们的 QML 模型由输入层、作为隐藏层的量子电路和输出层组成。输入层和输出层是中实现的经典神经网络 PyTorch。量子电路使用的 `qml.q PennyLane nn` 模块与 PyTorch 神经网络集成。有关工作负载的更多详细信息，请参阅我们的 [示例笔记本](#)。就像上面的 QAOA 示例一样，你可以利用基于 GPU 的嵌入式模拟器（比如 `lightning.gpu`）来提高基于 CPU PennyLane 的嵌入式模拟器的性能。

要创建混合作业，您可以通过其关键字参数调用 `AwsQuantumJob.create` 和指定算法脚本、设备和其他配置。

```
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,
    ...
)
```

要使用数据并行性，您需要修改 SageMaker 分布式库算法脚本中的几行代码，以正确并行化训练。首先，您导入 `smdistributed` 软件包，该软件包负责将工作负载分配到多个 GPU 和多个实例。此软件包已在 Braket PyTorch 和 TensorFlow 容器中预先配置。该 `dist` 模块告诉我们的算法脚本用于训练 (`world_size`) 的 GPU 总数，以及 GPU 内核 `local_rank` 的 `rank` 和。 `rank` 是 GPU 在所有实例中

的绝对索引，而`local_rank`是 GPU 在实例中的索引。例如，如果有四个实例，每个实例都为训练分配了八个 GPU，则`rank`范围为 0 到 31，`local_rank`范围为 0 到 7。

```
import smdistributed.dataparallel.torch.distributed as dist

dp_info = {
    "world_size": dist.get_world_size(),
    "rank": dist.get_rank(),
    "local_rank": dist.get_local_rank(),
}
batch_size //= dp_info["world_size"] // 8
batch_size = max(batch_size, 1)
```

接下来，`DistributedSampler`根据定义一个，`world_size``rank`然后将其传递到数据加载器中。此采样器可避免 GPU 访问数据集的同一片段。

```
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,
)
```

接下来，使用该`DistributedDataParallel`类来启用数据并行性。

```
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"])
```

以上是使用数据并行性所需的更改。在 QML 中，您通常希望保存结果并打印训练进度。如果每个 GPU 都运行保存和打印命令，则日志中将充斥重复的信息，结果将相互覆盖。为避免这种情况，您只能使用具有 rank 0 的 GPU 进行保存和打印。

```
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 混合任务支持 SageMaker 分布式数据并行库的 `m1.p3.16xlarge` 实例类型。您可以通过 `Hybrid Job InstanceConfigs` 中的参数配置实例类型。要使 SageMaker 分布式数据并行库知道数据并行性已启用，您需要再添加两个超参数，即 `"sagemaker_distributed_dataparallel_enabled"` 设置为正在使用的实例类型，`"true"` 并 `"sagemaker_instance_type"` 设置为正在使用的实例类型。这两个超参数由 `smdistributed` 软件包使用。您的算法脚本无需明确使用它们。在 Amazon Braket SDK 中，它提供了一个方便的关键字参数。 `distribution="data_parallel"` 在混合任务创建中，Amazon Braket SDK 会自动为您插入两个超参数。如果您使用 Amazon Braket API，则需要包含这两个超参数。

配置好实例和数据并行度后，您现在可以提交混合作业了。一个 `m1.p3.16xlarge` 实例中有 8 个 GPU。设置后 `instanceCount=1`，工作负载将分布在实例中的 8 个 GPU 上。如果设置 `instanceCount` 大于 1，则工作负载将分布在所有实例中可用的 GPU 上。使用多个实例时，每个实例会根据您的使用时间收取费用。例如，当您使用四个实例时，计费时间是每个实例运行时间的四倍，因为有四个实例同时运行您的工作负载。

```
instance_config = InstanceConfig(instanceType='m1.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

在上面的混合作业创建中，`train_dp.py`是修改后的用于使用数据并行性的算法脚本。请记住，只有当你根据上述部分修改算法脚本时，数据并行性才能正常工作。如果在未正确修改算法脚本的情况下启用数据并行度选项，则混合作业可能会引发错误，或者每个 GPU 可能会重复处理相同的数据切片，从而效率低下。

让我们在一个示例中比较运行时间和成本，在该示例中，针对上述二元分类问题，使用26量子比特量子电路训练模型。本示例中使用的`m1.p3.16xlarge`实例费用为每分钟 0.4692 美元。如果没有数据并行性，模拟器需要大约 45 分钟才能在 1 个纪元（即超过 208 个数据点）内训练模型，成本约为 20 美元。在 1 个实例和 4 个实例之间实现数据并行处理时，分别只需 6 分钟和 1.5 分钟，这意味着两者都花费大约 2.8 美元。通过在 4 个实例上使用数据并行性，您不仅可以将运行时间缩短 30 倍，还可以将成本降低一个数量级！

## 使用本地模式构建和调试混合作业

如果您正在构建新的混合算法，则本地模式可以帮助您调试和测试算法脚本。本地模式是一项功能，允许您运行计划在 Amazon Braket Hybrid Jobs 中使用的代码，但不需要 Braket 管理运行混合作业的基础架构。相反，您可以在 Braket Notebook 实例或首选客户端（例如笔记本电脑或台式机）上本地运行混合作业。在本地模式下，您仍然可以将量子任务发送到实际设备，但是在本地模式下针对实际 QPU 运行时，无法获得性能优势。

要使用本地模式，请将其修改`AwsQuantumJob`为出现`LocalQuantumJob`的任何位置。例如，要运行[创建第一个混合作业中的示例](#)，请按如下方式编辑混合作业脚本。

```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

**Note**

Docker 已经预装在 Amazon Braket 笔记本电脑中，需要安装在本地环境中才能使用此功能。可以在[此处](#)找到安装 Docker 的说明。此外，并非所有参数在本地模式下都受支持。

## 自带集装箱 (BYOC)

Amazon Braket Hybrid Jobs 提供了三个预先构建的容器，用于在不同的环境中运行代码。如果其中一个容器支持您的用例，则只需在创建混合作业时提供算法脚本即可。可以从算法脚本或使用 `requirements.txt` 文件中添加少量缺失的依赖关系 `pip`。

如果这些容器都不支持你的用例，或者你想对其进行扩展，Braket Hybrid Jobs 支持使用你自己的自定义 Docker 容器镜像运行混合作业，或者自带容器 (BYOC)。但是在我们深入探讨之前，让我们确保它实际上是适合你的用例的功能。

### 什么时候自带集装箱才是正确的决定？

将自己的容器 (BYOC) 带到 Braket Hybrid Jobs 中，通过将自己的软件安装在打包环境中，可以灵活地使用自己的软件。根据您的具体需求，可能有一些方法可以实现同样的灵活性，而不必经历完整的 BYOC Docker 构建 - Amazon ECR 上传 - 自定义图像 URI 周期。

**Note**

如果您想添加少量公开发布的其他 Python 包（通常少于 10 个），BYOC 可能不是正确的选择。例如，如果你正在使用 PyPi。

在这种情况下，您可以使用其中一个预先构建的 Braket 镜像，然后在作业提交时在源目录中加入一个 `requirements.txt` 文件。该文件会自动读取，`pip` 并将照常安装具有指定版本的软件包。如果您要安装大量软件包，则作业的运行时间可能会大大增加。检查要用来测试软件能否运行的预构建容器的 Python 和 CUDA 版本（如果适用）。

当你在作业脚本中使用非 Python 语言（如 C++ 或 Rust），或者你想使用无法通过 Braket 预建容器提供的 Python 版本时，BYOC 是必需的。在以下情况下，这也是一个不错的选择：

- 您正在使用带有许可证密钥的软件，需要通过许可服务器对该密钥进行身份验证才能运行该软件。使用 BYOC，您可以将许可证密钥嵌入 Docker 图像中，并包含用于对其进行身份验证的代码。

- 你使用的软件不是公开的。例如，该软件托管在私有 GitHub 存储库 GitLab 或存储库上，您需要使用特定 SSH 密钥才能访问该存储库。
- 您需要安装一套未打包在 Braket 提供的容器中的大型软件。BYOC 将允许您消除由于安装软件而导致混合作业容器的启动时间过长。

BYOC 还允许您使用您的软件构建 Docker 容器并将其提供给用户，从而使您的自定义 SDK 或算法可供客户使用。为此，您可以在 Amazon ECR 中设置适当的权限。

### Note

您必须遵守所有适用的软件许可证。

## 自带集装箱的食谱

在本节中，我们提供了 Braket Hybrid Jobs 需要做什么的 step-by-step 指南，包括脚本、文件和将它们组合在一起的步骤，以便启动和运行自定义 Docker 映像。bring your own container (BYOC) 我们提供两种常见案例的食谱：

1. 在 Docker 镜像中安装其他软件，并在作业中仅使用 Python 算法脚本。
2. 使用使用非 Python 语言编写的算法脚本与 Hybrid Jobs 或 x86 之外的 CPU 架构搭配使用。

对于案例 2，定义容器入口脚本更为复杂。

当 Braket 运行您的混合任务时，它会启动所请求的数量和类型的 Amazon EC2 实例，然后运行在 Docker 这些实例上创建任务时输入的图像 URI 所指定的映像。使用 BYOC 功能时，您可以指定托管在您拥有读取权限的 [私有 Amazon ECR 存储库](#) 中的图像 URI。Braket Hybrid Jobs 使用该自定义镜像来运行作业。

构建可用于 Hybrid Jobs 的 Docker 映像所需的特定组件。如果您不熟悉编写和构建 Dockerfiles，我们建议您在阅读这些说明时根据需要参阅 [Dockerfile 文档](#) 和 [Amazon ECR CLI 文档](#)。

以下是你需要什么的概述：

- [你的 Dockerfile 的基础镜像](#)
- [\( 可选 \) 修改后的容器入口脚本](#)
- [安装 Dockerfile 所有必需的软件并包含容器脚本](#)

## 你的 Dockerfile 的基础镜像

如果您使用的是 Python，并且想要在 Braket 提供的容器中提供的内容之上安装软件，那么基础映像的一个选项是托管在我们的[GitHub 存储库](#)和 [Amazon ECR 上的 Braket 容器镜像](#)。您需要向 [Amazon ECR 进行身份验证](#)才能提取映像并在其上进行构建。例如，您的 BYOC Docker 文件的第一行可能是：`FROM [IMAGE_URI_HERE]`

接下来，填写 Dockerfile 要安装的其余部分，然后设置要添加到容器中的软件。预先构建的 Braket 镜像已经包含了相应的容器入口点脚本，因此您无需担心包含该脚本。

如果你想使用非 Python 语言，例如 C++、Rust 或 Julia，或者你想为非 x86 CPU 架构（比如 ARM）构建镜像，则可能需要在基本公共镜像的基础上构建。您可以在 [Amazon Elastic Container Registry 公共图库](#)中找到许多这样的图片。请务必选择适合 CPU 架构的 GPU，必要时还要选择要使用的 GPU。

### ( 可选 ) 修改后的容器入口点脚本

#### Note

如果您只是在预先构建的 Braket 镜像中添加其他软件，则可以跳过本节。

要在混合作业中运行非 Python 代码，您需要修改定义容器入口点的 Python 脚本。例如，[亚马逊 Braket braket\\_container.py Github 上的 python 脚本](#)。这是 Braket 预先构建的图像用来启动算法脚本和设置相应环境变量的脚本。容器入口点脚本本身必须使用 Python，但可以启动非 Python 脚本。在预先构建的示例中，您可以看到 Python 算法脚本要么作为 [Python 子进程](#)启动，要么作为 [全新的进程](#)启动。通过修改此逻辑，您可以启用入口点脚本来启动非 Python 算法脚本。例如，您可以修改 [thekick\\_off\\_customer\\_script\(\)](#) 函数以启动 Rust 进程，具体取决于文件扩展名的结尾。

你也可以选择写一个全新的东西 `braket_container.py`。它应将输入数据、源档案和其他必要文件从 Amazon S3 复制到容器中，并定义相应环境变量。

## 安装 Dockerfile 所有必需的软件并包含容器脚本

#### Note

如果您使用预先构建的 Braket 镜像作为 Docker 基础镜像，则容器脚本已经存在。



如果您在上一步中创建了修改后的容器脚本，则需要将其复制到容器中，然后将环境变量SAGEMAKER\_PROGRAM定义为新容器入口点脚本braket\_container.py，或者定义您为新容器入口点脚本命名的变量。

以下是允许您在 GPU 加速任务实例上使用 Julia 的示例：Dockerfile

```
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
```

此示例下载并运行提供的脚本 AWS ，以确保符合所有相关的开源许可证。例如，通过正确归因任何受控制的已安装代码。MIT license

如果您需要包含非公开代码，例如托管在私有代码 GitHub 或 GitLab 存储库中的代码，请不要在 Docker 映像中嵌入 SSH 密钥来访问它。相反，请在构建 Docker Compose 时使用 Docker，以允许在其构建的主机上访问 SSH。有关更多信息，请参阅《[在 Docker 中安全使用 SSH 密钥访问私有 Github 存储库指南](#)》。

## 创建和上传您的 Docker 图片

有了正确定义的 Dockerfile，您现在就可以按照步骤[创建私有 Amazon ECR 存储库](#)了（如果尚不存在）。您也可以构建、标记容器映像并将其上传到存储库。

您已准备好构建、标记和推送映像。有关选项的完整说明 docker build 和一些示例，请参阅 [Docker 构建文档](#)。

对于上面定义的示例文件，你可以运行：

```
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
```

## 分配相应的 Amazon ECR 权限

Braket Hybrid Jobs Docker 图像必须托管在私有 Amazon ECR 存储库中。默认情况下，私有 Amazon ECR 存储库不向 Braket Hybrid Jobs IAM role 或任何其他想要使用您的图像的用户（例如合作者或学生）提供读取权限。您必须[设置存储库策略](#)才能授予相应的权限。通常，仅向您想要访问图像的特定用户和 IAM 角色授予权限，而不允许任何拥有该用户和角色 image URI 的用户提取图像。

## 在自己的容器中运行 Braket 混合作业

要使用自己的容器创建混合作业，请 `AwsQuantumJob.create()` 使用 `image_uri` 指定的参数调用。你可以使用 QPU、按需模拟器，也可以在 Braket Hybrid Jobs 提供的经典处理器上本地运行代码。我们建议先在 SV1、DM1 或 TN1 等模拟器上测试代码，然后再在真正的 QPU 上运行。

要在经典处理器上运行代码，请通过更新来指定 `instanceType` 和使用 `InstanceConfig`。 `instanceCount` 请注意，如果您指定 `instance_count > 1`，则需要确保您的代码可以在多个主机上运行。您可以选择的实例数量上限为 5。例如：

```
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(instance_type="ml.p3.8xlarge", instance_count=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

### Note

使用设备 ARN 跟踪您用作混合作业元数据的模拟器。可接受的值必须遵循格式 `device = "local:<provider>/<simulator_name>"`。请记住，`<provider>` 并且 `<simulator_name>` 必须仅包含字母、数字、`_`、`-`、和 `.`。该字符串限制为 256 个字符。

如果您计划使用 BYOC，但不使用 Braket SDK 创建量子任务，则应将环境变量的值传递 `AMZN_BRAKET_JOB_TOKEN` 给请求中的 `jobTokenCreateQuantumTask` 参数。如果你不这样做，量子任务就不会获得优先级，而是作为常规的独立量子任务计费。

## 在中配置默认存储桶 `AwsSession`

例如，提供自己的存储桶 `AwsSession` 可以让您在默认存储桶的位置上获得更大的灵活性。默认情况下，`AwsSession` 的存储桶的默认位置为 `f"amazon-braket-{id}-{region}"`。但是您可以在创建时覆盖该默认值 `AwsSession`。用户可以选择将 `AwsQuantumJob.create` 具有参数名称的 `AwsSession` 对象传入到 `aws_session`，如以下代码示例所示。

```
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
)
```

## 使用直接与混合作业互动 API

您可以直接使用 Amazon Braket Hybrid Jobs 访问并与之交互。API 但是，API 直接使用时，默认方法和便捷方法不可用。

### Note

我们强烈建议你使用 Amazon Braket Python SDK 与 Amazon Braket Hybrid Jobs 互动。它提供便捷的默认设置和保护功能，可帮助您的混合作业成功运行。

本主题涵盖了使用的基础知识 API。如果您选择使用 API，请记住这种方法可能更加复杂，并且需要为几次迭代做好准备，以使您的混合作业得以运行。

要使用 API，您的账户应具有 `AmazonBraketFullAccess` 托管策略的角色。

**Note**

有关如何使用AmazonBraketFullAccess托管策略获取角色的更多信息，请参阅[启用 Amazon Braket 页面](#)。

此外，您还需要一个执行角色。此角色将传递给服务。您可以使用 Amazon Braket 控制台创建角色。使用“权限和设置”页面上的“执行角色”选项卡为混合作业创建默认角色。

CreateJobAPI要求您为混合作业指定所有必需的参数。要使用 Python，请将算法脚本文件压缩为 tar 包，例如 input.tar.gz 文件，然后运行以下脚本。更新尖括号 (<>) 内的代码部分，以匹配您的账户信息和指定混合作业开始路径、文件和方法的入口点。

```
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
    },
)
```

创建混合作业后，您可以通过GetJobAPI或控制台访问混合作业的详细信息。要从运行createJob代码的 Python 会话中获取混合作业的详细信息，请使用以下 Python 命令。

```
getJob = client.get_job(jobArn=job["jobArn"])
```

要取消混合作业，请CancelJobAPI使用该任务Amazon Resource Name的 ('JobArn') 调用。

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

您可以使用checkpointConfig参数将检查点指定为的一部分。createJob API

```
checkpointConfig = {
    "localPath" : "/opt/omega/checkpoints",
    "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
},
```

#### Note

的 localPath checkpointConfig 不能以以下任何保留路径开头：/opt/ml、/opt/braket/tmp、或/usr/local/nvidia。



# 错误缓解

量子误差缓解是一套旨在减少量子计算机中错误影响的技术。

量子设备会受到环境噪音的影响，这会降低所执行的计算质量。尽管容错量子计算有望解决这个问题，但当前的量子器件受到量子比特数量和相对较高的错误率的限制。为了在短期内解决这个问题，研究人员正在研究提高噪音量子计算准确性的方法。这种方法被称为量子误差缓解，涉及使用各种技术从噪声测量数据中提取最佳信号。

## 错误缓解功能已开启 IonQ Aria

错误缓解包括运行多个物理电路并将它们的测量结果组合在一起以获得更好的结果。该 IonQ Aria 设备具有一种名为去偏的错误缓解方法。

去偏置将电路映射成多个变体，这些变体作用于不同的量子比特排列或具有不同的栅极分解。通过使用不同的电路实现方式，这可以减少系统误差（例如栅极过旋转或单个错误的量子比特）的影响，否则这些实现可能会使测量结果产生偏差。这是以校准多个量子比特和门的额外开销为代价的。

有关去偏的更多信息，请参阅[通过对称化增强量子计算机性能](#)。

### Note

使用去偏移至少需要 2500 张照片。

您可以使用以下代码在 IonQ Aria 设备上运行带去偏的量子任务：

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

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

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

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

量子任务完成后，您可以看到量子任务的测量概率和任何结果类型。所有变体的测量概率和计数汇总到一个分布中。电路中指定的任何结果类型（例如期望值）均使用汇总测量计数进行计算。

## 锐化

您还可以访问使用不同的后处理策略（称为锐化）计算的测量概率。Sharpening 会比较每个变体的结果并丢弃不一致的镜头，从而有利于各变体之间最有可能的测量结果。有关更多信息，请参阅[通过对称化增强量子计算机性能](#)。

重要的是，锐化假设输出分布的形式是稀疏的，高概率状态很少，零概率状态很多。如果此假设无效，则可能会扭曲概率分布。

您可以在 Braket Python SDK 中访问 `additional_metadata` 字段中经过锐化分布的 `GateModelTaskResult` 概率。请注意，锐化不会返回测量计数，而是返回重新归一化的概率分布。以下代码片段显示了如何在锐化后访问发行版。

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

# Braket Di

借助 Braket Direct，您可以保留对自己选择的不同量子设备的专用访问权限，与量子计算专家联系以获取工作负载指导，并尽早使用下一代功能，例如可用性有限的新量子设备。

本节内容：

- [预留](#)
- [专家建议](#)
- [实验能力](#)

## 预留

通过预订，您可以独家使用自己选择的量子设备。您可以在方便的时候安排预约，这样您就可以确切地知道工作负载何时开始和结束执行。预订以 1 小时为增量提供，最多可提前 48 小时取消预订，无需支付额外费用。您可以选择提前将量子任务和混合任务排队等候即将到来的预订，也可以在预留期间提交工作负载。

无论您在量子处理单元 (QPU) 上运行多少量子任务和混合作业，访问专用设备的费用都取决于您的预留时间。

以下量子计算机可供预订：

- ionQ 的 Aria
- IQM 的石榴石
- QuEra's Aquila
- Rigetti 的 Aspen-M-3

### 何时使用预订

利用带预留功能的专用设备访问权限为您提供了便捷性和可预测性，可以准确地知道量子工作负载何时开始和结束执行。与按需提交任务和混合作业相比，您无需排队等候其他客户任务。由于您在预留期间拥有对设备的独占访问权限，因此在整个预留期间，只有您的工作负载在设备上运行。

我们建议在研究的设计和原型设计阶段使用按需访问权限，从而实现算法的快速且具有成本效益的迭代。准备好得出最终实验结果后，可以考虑在方便的时候安排设备预订，以确保能够在项目或出版的最

后期限之前完成。如果您希望在特定时间执行任务，例如在量子计算机上运行现场演示或研讨会时，我们还建议您使用预留功能。

本节内容：

- [创建预订](#)
- [通过预留来运行您的工作负载](#)
- [取消或重新安排现有预订](#)

## 创建预订

要创建预订，请按照以下步骤联系 Braket 团队：

1. 打开 Amazon Braket 控制台。
2. 在左侧窗格中选择 Braket Direct，然后在“预订”部分中，选择“预留设备”。
3. 选择您要预订的设备。
4. 提供您的联系信息，包括姓名和电子邮件。请务必提供您定期检查的有效电子邮件地址。
5. 在“告诉我们您的工作负载”下，提供有关使用您的预留运行的工作负载的所有详细信息。例如，所需的预留时长、相关的限制条件或所需的时间表。
6. 如果您有兴趣在预订确认后与 Braket 专家联系以进行预订准备会议，可以选择我对预备课程感兴趣。

您也可以按照以下步骤联系我们进行预订：

1. 打开 Amazon Braket 控制台。
2. 在左侧窗格中选择“设备”，然后选择要预订的设备。
3. 在“摘要”部分，选择“保留设备”。
4. 按照前一过程中的步骤 4-6 进行操作。

提交表格后，您将收到一封来自 Braket 团队的电子邮件，其中包含创建预订的后续步骤。预订确认后，您将通过电子邮件收到预订 ARN。

### Note

只有在您收到预订 ARN 后，您的预订才会得到确认。

预订以至少 1 小时为增量提供，某些设备可能有额外的预约时长限制（包括最短和最长预约时长）。在确认预订之前，Braket 团队会与您共享所有相关信息。

如果您表示有兴趣参加预订准备会议，Braket 团队会通过电子邮件与您联系，安排与 Braket 专家进行 30 分钟的会话。

## 通过预留来运行您的工作负载

在预留期间，只有您的工作负载在设备上运行。要指定要在设备预留期间运行的量子任务和混合作业，必须使用有效的预留 ARN。

### Note

预订视 AWS 账户和设备而定。只有创建预留的 AWS 账户才能使用您的预订 ARN。此外，预留 ARN 仅在所选的开始和结束时间在预留设备上有效。

要充分利用您的预留时间，您可以选择在预约之前对任务和作业进行排队。在预留开始之前，这些工作负载将保持 QUEUED 状态。当预留开始时，所有排队的工作负载都将按提交的顺序运行。Job 任务的优先级优先于独立的量子任务。

### Note

由于在预留期间只有您的工作负载运行，因此使用预留 ARN 提交的任务和作业无法查看队列。

为预留创建量子任务的代码示例：

1. 定义一个电路以准备采用 OpenQasm 格式的 GHZ 状态。

```
// 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. 使用您的电路和预留 ARN 创建量子任务。

```
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>"  
)
```

为 Braket Direct 预留创建混合作业的代码示例：

### 1. 定义您的算法脚本。

```
//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. 使用您的算法脚本和预留 ARN 创建混合作业。

```
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. 使用远程装饰器创建混合作业...

```
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
```

## 预订结束后会发生什么

预约结束后，您将不再拥有该设备的专用访问权限。通过此预留排队的所有剩余工作负载都将自动取消。

### Note

任何在预留结束时处于RUNNING状态的任务都将被取消。我们建议您使用[检查点在方便时保存和重新启动作业](#)。

正在进行的预约（例如预约开始后和预约结束之前）无法延长，因为每项预留都代表独立的专用设备访问权限。例如，两个 back-to-back 预留被视为分开的，第一个预留中的任何待处理任务都会自动取消。他们不会在第二个保留中恢复。

### Note

预订代表您 AWS 账户的专用设备访问权限。即使设备处于闲置状态，其他客户也无法使用它。因此，无论使用时间长短，都要按预留时间长度收费。

## 取消或重新安排现有预订

您可以在预定预订开始时间前不少于 48 小时取消预订。要取消，请回复您收到的预订确认电子邮件以及取消申请。



要重新安排，您必须取消现有预订，然后创建新的预订。

## 专家建议

直接在 Braket 管理控制台中与量子计算专家联系，获取有关工作负载的更多指导。

要通过 Braket Direct 浏览专家建议选项，请打开 Braket 控制台，在左侧窗格中选择 Braket Direct，然后导航到专家建议部分。以下专家建议选项可供选择：

- **Braket 办公时间**：Braket 办公时间为 1:1 会议，先到先得，每月举行一次。每个可用的办公时间为 30 分钟，且免费。与 Braket 专家交谈可以探索 use-case-to-device 拟合度，确定最好地利用 Braket 进行算法的选项，并获得有关如何使用某些 Braket 功能的建议，例如 Amazon Braket 混合任务、Braket Pulse 或模拟哈密尔顿模拟，从而帮助您更快地从构思到执行。
- 要注册 Braket 办公时间，请选择注册并填写联系信息、工作量详情和所需的讨论主题。
- 您将通过电子邮件收到下一个可用时段的日历邀请。

### Note

[对于紧急问题或快速故障排除问题，我们建议您联系 Support AWS。](#) 对于非紧急问题，您也可以使用 [re AWS : Post 论坛](#) 或 [Quantum Computing Stack Exchange](#)，在那里您可以浏览之前回答过的问题并提出新的问题。

- **Quantum 硬件提供商产品**：ionQ Oxford Quantum Circuits、QuEra、和 Rigetti 均通过以下方式提供专业服务。AWS Marketplace
  - 要浏览他们的产品，请选择 Connect 并浏览他们的房源。
  - 要详细了解上提供的专业服务 AWS Marketplace，请参阅 [专业服务产品](#)。
- **Amazon 量子解决方案实验室 ( QSL )**：QSL 是一个合作研究和专业服务团队，由量子计算专家组成，可以帮助您有效地探索量子计算并评估该技术的当前性能。
  - 要联系 QSL，请选择 Connect，然后填写联系信息和用例详细信息。
  - QSL 团队将通过电子邮件与您联系，告知后续步骤。

## 实验能力

为了提高您的研究工作量，重要的是要快速获得新的创新功能。借助 Braket Direct，您可以直接在 Braket 控制台中请求访问可用的实验功能，例如可用性有限的新型量子设备。

有些实验功能在标准设备规格之外运行，需要根据您的用例量身定制的实践指导。为确保您的工作负载设置成功，可通过 Braket Direct 请求进行访问。

## 仅限预约访问 ionQ Forte

有了 Braket Direct，你只能通过预订获得 ionQ Forte QPU 的访问权限。由于可用性有限，该设备只能通过 Braket Direct 获得。

要了解更多信息并申请访问 ionQ Forte，请按照以下步骤操作：

1. 打开 Amazon Braket 控制台。
2. 在左侧菜单中选择 Braket Direct，然后在“实验功能”中导航到 ionQ Forte。选择“查看设备”。
3. 在 Forte 设备详情页面的“摘要”中，选择“保留设备”。
4. 提供您的联系信息，包括姓名和电子邮件。提供您定期检查的有效电子邮件地址。
5. 在“告诉我们您的工作负载”下，提供有关使用您的预留运行的工作负载的详细信息，例如所需的预留时长、相关限制或所需的时间表。
6. ( 可选 ) 如果您有兴趣在预订确认后与 Braket 专家联系以进行预订准备会议，请选择“我对预备课程感兴趣”。

表格提交后，Braket 团队将与您联系并告知后续步骤。

### Note

由于设备可用性有限，对 Forte 的访问受到限制。联系我们了解更多。

## 在 Aquila 上 QuEra 访问本地停机功能

使用 Braket Direct，在 QPU 上编程时，您可以请求访问权限以控制本地失调。QuEra Aquila 使用此功能，您可以调整驱动场对每个特定量子比特的影响程度。

要了解更多信息并申请访问此功能，请按照以下步骤操作：

1. 打开 Amazon Braket 控制台。
2. 在左侧菜单中选择 Braket Direct，然后在“实验功能”中导航到 QuEra Aquila-本地失调。选择“获取访问权限”。
3. 提供您的联系信息，包括姓名和电子邮件。提供您定期检查的有效电子邮件地址。

4. 在“告诉我们您的工作量”下，提供有关工作负载以及您计划在何处使用此功能的详细信息。

## 在 Aquila 上 QuEra 访问高大的几何形状

使用 Braket Direct，在 QPU 上编程时，您可以请求访问扩展的 QuEra Aquila 几何形状。借助此功能，您可以进行超出标准设备功能的实验，并通过增加晶格高度来指定几何形状。

要了解更多信息并申请访问此功能，请按照以下步骤操作：

1. 打开 Amazon Braket 控制台。
2. 在左侧菜单中选择 Braket Direct，然后在实验功能中导航到 QuEra Aquila-高大的几何形状。选择“获取访问权限”。
3. 提供您的联系信息，包括姓名和电子邮件。提供您定期检查的有效电子邮件地址。
4. 在“告诉我们您的工作量”下，提供有关工作负载以及您计划在何处使用此功能的详细信息。

## 在 Aquila 上 QuEra 可以看到紧凑的几何形状

使用 Braket Direct，在 QPU 上编程时，您可以请求访问扩展的 QuEra Aquila 几何形状。借助此功能，您可以进行超出标准设备功能的实验，并以更紧的垂直间距排列晶格排。

要了解更多信息并申请访问此功能，请按照以下步骤操作：

1. 打开 Amazon Braket 控制台。
2. 在左侧菜单中选择 Braket Direct，然后在实验功能中导航到 QuEra Aquila-高大的几何形状。选择“获取访问权限”。
3. 提供您的联系信息，包括姓名和电子邮件。提供您定期检查的有效电子邮件地址。
4. 在“告诉我们您的工作量”下，提供有关工作负载以及您计划在何处使用此功能的详细信息。

## 日志记录和监控

提交量子任务后，您可以通过 Amazon Braket SDK 和控制台跟踪其状态。量子任务完成后，Braket 会将结果保存在您指定的 Amazon S3 位置。完成可能需要一些时间，尤其是对于 QPU 设备，具体取决于队列的长度。状态类型包括：

- **CREATED**— 亚马逊 Braket 收到了你的量子任务。
- **QUEUED**— Amazon Braket 处理了你的量子任务，现在它正在等待在设备上运行。
- **RUNNING**— 您的量子任务在 QPU 或按需模拟器上运行。
- **COMPLETED**— 你的量子任务在 QPU 或按需模拟器上运行完毕。
- **FAILED**— 你的量子任务尝试运行但失败了。根据您的量子任务失败的原因，请尝试再次提交量子任务。
- **CANCELLED**— 你取消了量子任务。量子任务没有运行。

本节内容：

- [通过 Amazon Braket SDK 跟踪量子任务](#)
- [通过 Amazon Braket 控制台监控量子任务](#)
- [标记 Amazon Braket 资源](#)
- [Amazon Braket 与亚马逊合作的事件和自动操作 EventBridge](#)
- [使用亚马逊监控 Amazon Braket CloudWatch](#)
- [使用 Amazon Braket API 进行登录 CloudTrail](#)
- [使用创建 Amazon Braket 笔记本实例 AWS CloudFormation](#)
- [高级日志](#)

## 通过 Amazon Braket SDK 跟踪量子任务

该命令 `device.run(...)` 定义具有唯一量子任务 ID 的量子任务。您可以使用查询和跟踪状态，`task.state()` 如以下示例所示。

**注意：** `task = device.run()` 是一种异步操作，这意味着当系统在后台处理量子任务时，你可以继续工作。

检索结果

当你打电话时`task.result()`，SDK 开始轮询 Amazon Braket 以查看量子任务是否完成。SDK 使用您在`run()`中定义的轮询参数。量子任务完成后，SDK 会从 S3 存储桶中检索结果并将其作为`QuantumTaskResult`对象返回。

```
# 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: QUEUED
Status: QUEUED
Status: QUEUED
Status: RUNNING
Status: RUNNING
Status: COMPLETED
```

## 取消量子任务

要取消量子任务，请调用该`cancel()`方法，如以下示例所示。

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

```
Status of task: CANCELLING
```

## 检查元数据

您可以检查已完成的量子任务的元数据，如以下示例所示。

```
# 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
```

## 检索量子任务或结果

如果你的内核在你提交量子任务后死亡，或者你关闭了笔记本或电脑，你可以用其唯一的 ARN (量子任务 ID) 重建 `task` 对象。然后，您可以调用 `task.result()` 以从存储结果的 S3 存储桶中获取结果。

```
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()
```

## 通过 Amazon Braket 控制台监控量子任务

Amazon Braket 提供了一种通过 [Amazon Braket](#) 控制台监控量子任务的便捷方法。所有提交的量子任务都列在“量子任务”字段中，如下图所示。该服务是特定于区域的，这意味着您只能查看在特定区域中创建的量子任务。AWS 区域

The screenshot shows the Amazon Braket console interface for Quantum Tasks. At the top, there is a navigation bar with "Amazon Braket" and "Quantum Tasks". Below this is a warning box stating "QPUs are region specific. Select the correct device region for corresponding quantum tasks. [Learn more](#)". The main content area is titled "Quantum Tasks (10+)" and includes a search bar, a refresh button, an "Actions" dropdown, and a "Show quantum task details" button. A table lists several quantum tasks, all with a status of "COMPLETED".

Quantum Task ID	Status	Device ARN	Created at
<a href="#">d87730f0-414f-4a60-9de2-7fd18c20f7f2</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Sep 05, 2023 19:13 (UTC)
<a href="#">62a5b6f9-2334-4bad-af4f-a5aebbe6032</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
<a href="#">85f05c12-c4d0-42bf-8782-b825775f057a</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Aug 31, 2023 19:11 (UTC)
<a href="#">1fa148a2-aaaa-4948-b7df-808513145a20</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
<a href="#">aee8d2ad-a396-4c11-9f13-9aa62db680b9</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
<a href="#">dfce97af-3aae-4e57-bd64-29d6f9521937</a>	COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Aug 31, 2023 19:11 (UTC)

您可以通过导航栏搜索特定的量子任务。搜索可以基于 Quantum Task ARN (ID)、状态、设备和创建时间。当您选择导航栏时，这些选项会自动出现，如以下示例所示。

This screenshot is similar to the previous one but shows a search dropdown menu open over the search bar. The dropdown lists "Properties" with sub-items: "Status", "Device ARN", "Quantum task ARN", and "Created at". The search results table is partially visible, showing the same list of completed tasks as in the previous screenshot.

下图显示了根据量子任务的唯一量子任务 ID 搜索量子任务的示例，该任务可以通过调用获得 `task.id`。

Amazon Braket > Quantum Tasks

QPU's are region specific. Select the correct device region for corresponding quantum tasks. [Learn more](#)

Quantum Tasks (1) Refresh Actions Show quantum task details

Search  (1) matches

Quantum task ARN = `arn:aws:braket:us-west-2:260818742045:quantum-task/4cd1a31e-61c0-469c-a9cf-a2fbe7b4e358` Clear filters

Quantum Task ID	Status	Device ARN	Created at
<a href="#">4cd1a31e-61c0-469c-a9cf-a2fbe7b4e358</a>	COMPLETE	arn:aws:braket::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:10 (UTC)

此外，如下图所示，可以在量子任务QUEUED处于状态时对其状态进行监控。点击量子任务 ID 会显示详细信息页面。此页面显示量子任务相对于要处理的设备的动态队列位置。

Amazon Braket > Quantum Tasks > 3d11c509-454d-4fe2-b3b9-fad6d8eab83b

3d11c509-454d-4fe2-b3b9-fad6d8eab83b

Quantum task details Actions

Quantum task ARN	Status	Queue position <a href="#">info</a>
<code>arn:aws:braket:us-east-1:1984631112496:quantum-task/3d11c509-454d-4fe2-b3b9-fad6d8eab83b</code>	QUEUED	3 (Normal)
Device ARN	Created	Ended
<code>arn:aws:braket:us-east-1:device/gpu/fong/Aria-2</code>	Sep 08, 2023 19:22 (UTC)	—
Shots	Results	Status reason
100	—	—

作为混合作业的一部分提交的 Quantum 任务在排队时将具有优先级。在混合作业之外提交的 Quantum 任务将具有正常的排队优先级。

想要查询 Braket SDK 的客户可以通过编程方式获取其量子任务和混合作业队列位置。有关更多信息，请参阅“[我的任务何时运行](#)”页面。

## 标记 Amazon Braket 资源

标签是您分配或分配给 AWS 资源的自定义属性标签。AWS 标签是元数据，可以详细介绍您的资源。每个标签均包含一个键 和一个值。这些被统称为键/值对。对于您分配的标签，需要定义键和值。

在 Amazon Braket 控制台中，您可以导航到量子任务或笔记本并查看与之相关的标签列表。您可以添加标签、移除标签或修改标签。可以在创建量子任务或笔记本时对其进行标记，然后通过控制台 AWS CLI、或管理关联的标签API。



## 使用标签

标签可以将您的资源组织成对您有用的类别。例如，您可以分配“部门”标签来指定拥有此资源的部门。

每个标签具有两个部分：

- 标签密钥（例如 CostCenter，“环境”或“项目”）。标签键区分大小写。
- 一个称为标签值的可选字段（例如，111122223333 或 Production）。省略标签值与使用空字符串效果相同。与标签键一样，标签值区分大小写。

标签可以帮助你做以下事情：

- 识别和整理您的 AWS 资源。许多 AWS 服务支持标记，因此您可以为来自不同服务的资源分配相同的标签，以表明这些资源是相关的。
- 追踪您的 AWS 成本。您可以在 AWS Billing and Cost Management 控制面板上激活这些标签。AWS 使用标签对您的成本进行分类，并向您提供每月成本分配报告。有关更多信息，请参阅 [AWS Billing and Cost Management 用户指南](#) 中的 [使用成本分配标签](#)。
- 控制对 AWS 资源的访问权限。有关更多信息，请参阅 [使用标签控制访问权限](#)。

## 更多关于 AWS 和标签

- 有关标记的一般信息，包括命名和使用惯例，请参阅《AWS 通用参考》中的 [标记 AWS 资源](#)。
- 有关标签限制的信息，请参阅《AWS 一般参考》中的 [标签命名限制和要求](#)。
- 有关最佳做法和标签策略，请参阅 [标记最佳做法](#) 和 [AWS 标记策略](#)。
- 有关支持使用标签的服务的列表，请参阅 [《Resource Groups 标记 API 参考》](#)。

以下各节提供了有关 Amazon Braket 标签的更多具体信息。

## Amazon Braket 中支持的资源

Amazon Braket 中的以下资源类型支持标记：

- [quantum-task](#) 资源
- 资源名称：AWS::Service::Braket

- ARN 正则表达式：`arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

注意：您可以在 Amazon Braket 控制台中应用和管理您的 B Amazon raket 笔记本的标签，方法是使用控制台导航到笔记本资源，尽管这些笔记本实际上是 Ama SageMaker zon 资源。有关更多信息，请参阅 SageMaker 文档中的[笔记本实例元数据](#)。

## 标签限制

以下基本限制适用于 Amazon Braket 资源上的标签：

- 您可以分配给资源的最大标签数量：50
- 最大密钥长度：128 个 Unicode 字符
- 最大值长度：256 个 Unicode 字符
- 键和值的有效字符：`a-z`，`A-Z`，`0-9`，`space`，以及以下字符：`_`，`.`，`:`，`/`，`=`，`+`，`-`和 `@`
- 键和值区分大小写。
- 请勿 `aws` 用作密钥的前缀；它是保留供 AWS 使用的。

## 在 Amazon Braket 中管理标签

您可以将标签设置为资源的属性。您可以通过 Braket 控制台、Amazon Braket 或，查看、添加、修改、列出 Amazon 和删除标签。API AWS CLI 有关更多信息，请参阅 [Amazon Braket API](#) 参考。

### 添加标签

您可以在以下时间为可标记资源添加标签：

- 创建资源时：使用控制台，或者在 [AWS API](#) 中将 `Tags` 参数包含在 `Create` 操作中。
- 创建资源后：使用控制台导航到量子任务或笔记本资源，或者在 [AWS API](#) 中调用 `TagResource` 操作。

要在创建资源时为其添加标签，还需要创建指定类型的资源的权限。

### 查看标签

您可以使用控制台导航到任务或笔记本资源，或者通过调用操作来查看 Amazon Braket 中任何可标记资源的标签。AWS `ListTagsForResource` API

您可以使用以下 AWS API 命令查看资源上的标签：

- AWS API: `ListTagsForResource`

## 编辑标签

您可以使用控制台导航到量子任务或笔记本资源来编辑标签，也可以使用以下命令修改附加到可标记资源的标签的值。当您指定已存在的标签密钥时，该密钥的值将被覆盖：

- AWS API: `TagResource`

## 删除标签

您可以通过指定要删除的密钥、使用控制台导航到量子任务或笔记本资源，或者在调用 `UntagResource` 操作时从资源中删除标签。

- AWS API: `UntagResource`

## Amazon Braket 中的 CLI 标签示例

如果你使用的是 AWS CLI，这里有一个示例命令，显示如何创建适用于你使用 Rigetti QPU 参数设置为其创建SV1的量子任务的标签。请注意，标签是在示例命令的末尾指定的。在这种情况下，Key 被赋予值 `state`，Value 被赋予值 `Washington`。

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

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

## 使用 Amazon Braket API 进行标记

- 如果您使用 Amazon Braket API 在资源上设置标签，请致电。 [TagResourceAPI](#)

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

- 要从资源中移除标签，请调用 [UntagResourceAPI](#)。

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- 要列出附加到特定资源的所有标签，请调用 [ListTagsForResourceAPI](#)。

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

## Amazon Braket 与亚马逊合作的事件和自动操作 EventBridge

Amazon EventBridge 监控 Amazon Braket 量子任务中的状态变化事件。来自 Amazon Braket 的事件几乎是实时的。EventBridge 您可以编写简单规则来指示所关注的事件，包括要在事件匹配规则时执行的自动化操作。可以触发的自动操作包括：

- 调用函数 AWS Lambda
- 激活 AWS Step Functions 状态机
- 向 Amazon SNS 主题发送通知

EventBridge 监视以下 Amazon Braket 状态更改事件：

- quantum 任务的状态发生了变化

Amazon Braket 保证交付量子任务状态变化事件。这些事件至少传送一次，但可能出现故障。

有关更多信息，请参阅 [中的事件和事件模式 EventBridge](#)。

本节内容：

- [使用监控量子任务状态 EventBridge](#)
- [亚马逊 Braket 活动 EventBridge 示例](#)

## 使用监控量子任务状态 EventBridge

借 EventBridge 助，您可以创建规则，定义在 Amazon Braket 发送有关 Braket 量子任务的状态更改通知时要采取的操作。例如，您可以创建一条规则，规定每次量子任务的状态发生变化时都会向您发送一封电子邮件。

1. AWS 使用有权使用 EventBridge 和 Amazon Braket 的账户登录。
2. 打开亚马逊 EventBridge 控制台，[网址为 https://console.aws.amazon.com/events/](https://console.aws.amazon.com/events/)。
3. 使用以下值创建 EventBridge 规则：
  - 对于规则类型，选择具有事件模式的规则。
  - 对于事件源，选择其他。
  - 在事件模式部分，选择自定义模式(JSON 编辑器)，然后将以下事件模式粘贴到文本区域：

```
{
  "source": [
    "aws.braket"
  ],
  "detail-type": [
    "Braket Task State Change"
  ]
}
```

要从 Amazon Braket 中捕获所有事件，请排除该 detail-type 部分，如以下代码所示：

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

- 对于目标类型，选择 AWS 服务，在选择目标中，选择目标，例如 Amazon SNS 主题或 AWS Lambda 函数。当收到来自 Amazon Braket 的量子任务状态变化事件时，就会触发目标。

例如，使用 Amazon Simple Notification Service (SNS) 主题在事件发生时发送电子邮件或短信。为此，请先使用亚马逊 SNS 控制台创建亚马逊 SNS 主题。要了解更多信息，请参阅[使用 Amazon SNS 发送用户通知](#)。

有关创建规则的详细信息，请参阅[创建对事件做出反应的 Amazon EventBridge 规则](#)。

## 亚马逊 Braket 活动 EventBridge 示例

有关 Amazon Braket Quantum 任务状态更改事件字段的信息，请参阅[中的 EventBridge 事件和事件模式](#)。

以下属性显示在 JSON 的“详细信息”字段中。

- **quantumTaskArn**(str)：生成此事件的量子任务。
- **status** ( 可选 [str] )：量子任务过渡到的状态。
- **deviceArn**(str)：为其创建此量子任务的用户指定的设备。
- **shots**(int)：用户shots请求的数量。
- **outputS3Bucket**(str)：用户指定的输出存储桶。
- **outputS3Directory**(str)：用户指定的输出密钥前缀。
- **createdAt**(str)：以 ISO-8601 字符串表示的量子任务创建时间。
- **endedAt** ( 可选 [str] )：量子任务达到终止状态的时间。只有当量子任务过渡到终端状态时，该字段才会出现。

以下 JSON 代码显示了 Amazon Braket Quantum 任务状态更改事件的示例。

```
{
  "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"
    }
  }
}

```

## 使用亚马逊监控 Amazon Braket CloudWatch

您可以使用亚马逊监控 Amazon Braket CloudWatch，亚马逊会收集原始数据并将其处理为可读的近乎实时的指标。您可以在亚马逊 CloudWatch 控制台中查看 15 个月前生成的历史信息或最近 2 周内更新的搜索指标，以便更好地了解 Amazon Braket 的表现。要了解更多信息，请参阅[使用 CloudWatch 指标](#)。

### Amazon Braket 的指标和维度

指标是中的基本概念 CloudWatch。指标表示发布到的一组按时间顺序排列的数据点。CloudWatch 每个指标都以一组维度为特征。要详细了解中的指标维度 CloudWatch，请参阅[CloudWatch 维度](#)。

Amazon Braket 将以下特定于 Amazon Braket 的指标数据发送到亚马逊指标中：CloudWatch

#### 量子任务指标

如果存在量子任务，则可以使用指标。它们显示在控制台的 AWS/Braket/By Device 下。CloudWatch

指标	描述
计数	量子任务数。
延迟	该指标是在量子任务完成时发出的。它表示从量子任务初始化到完成的总时间。

#### 量子任务指标的维度

量子任务指标以基于deviceArn参数的维度发布，其格式为 `arn:aws:braket:::device/xxx`。

## 支持的设备

有关支持的设备和设备 ARN 的列表，请参阅 [Braket 设备](#)。

### Note

您可以通过导航到亚马逊控制台上的笔记本详情页面来查看 Amazon Braket 笔记本的 CloudWatch 日志流。SageMaker [其他 Amazon Braket 笔记本设置可通过控制台获得](#)。SageMaker

## 使用 Amazon Braket API 进行登录 CloudTrail

Amazon Braket 与 AWS CloudTrail 一项服务集成，该服务提供用户、角色或用户 AWS 服务在 Amazon Braket 中执行的操作的记录。CloudTrail 将所有对 Amazon Braket 的 API 呼叫捕获为事件。捕获的呼叫包括来自 Amazon Braket 控制台的调用和对 Braket Amazon Braket 操作的代码调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括 Amazon Braket 的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的“事件历史记录”中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 Amazon Braket 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅 [AWS CloudTrail 用户指南](#)。

## Amazon Braket 中的信息 CloudTrail

CloudTrail 在您创建账户 AWS 账户时已在您的账户上启用。当 Amazon Braket 中发生活动时，该活动会与其他 CloudTrail 事件一起记录在 AWS 服务事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅 [使用事件历史记录查看 CloudTrail 事件](#)。

要持续记录您的事件 AWS 账户，包括 Amazon Braket 的事件，请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- [创建跟踪概述](#)
- [CloudTrail 支持的服务和集成](#)



- [配置 Amazon SNS 通知 CloudTrail](#)
- [接收来自多个区域的 CloudTrail 日志文件](#)和[接收来自多个账户的 CloudTrail 日志文件](#)

所有 Amazon Braket 操作都由记录。CloudTrail例如，对GetQuantumTask或GetDevice操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅[CloudTrail 用户身份元素](#)。

## 了解 Amazon Braket 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。CloudTrail 日志文件不是公共API调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例是GetQuantumTask操作的日志条目，它获取了量子任务的详细信息。

```
{
  "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 boto3/1.17.33",
"requestParameters": {
    "quantumTaskArn": "foobar"
},
"responseElements": null,
"requestID": "20e8000c-29b8-4137-9cbc-af77d1dd12f7",
"eventID": "4a2fdb22-a73d-414a-b30f-c0797c088f7c",
"readOnly": true,
"eventType": "AwsApiCall",
"recipientAccountId": "foobar"
}

```

下面显示了该GetDevice操作的日志条目，该条目返回了设备事件的详细信息。

```

{
  "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"
}
```

## 使用创建 Amazon Braket 笔记本实例 AWS CloudFormation

您可以使用 AWS CloudFormation 来管理您的 Amazon Braket 笔记本实例。Braket 笔记本实例是在 Amazon SageMaker 上构建的。使用 CloudFormation，您可以使用描述预期配置的模板文件来配置笔记本实例。模板文件以 JSON 或 YAML 格式编写。您可以以有序且可重复的方式创建、更新和删除实例。当您管理多个 Braket 笔记本实例时，你可能会发现这很有用。AWS 账户

为 Braket 笔记本创建 CloudFormation 模板后，您可以使用 AWS CloudFormation 来部署资源。有关更多信息，请参阅 AWS CloudFormation 用户指南中的 [在 AWS CloudFormation 控制台上创建堆栈](#)。

要使用创建 Braket 笔记本实例 CloudFormation，请执行以下三个步骤：

1. 创建 Amazon SageMaker 生命周期配置脚本。
2. 创建要由代入的 AWS Identity and Access Management (IAM) 角色 SageMaker。
3. 使用前缀创建 SageMaker 笔记本实例 **amazon-braket-**

您可以对自己创建的所有 Braket 笔记本重复使用生命周期配置。您还可以为分配相同执行权限的 Braket 笔记本重复使用 IAM 角色。

## 步骤 1：创建 Amazon SageMaker 生命周期配置脚本

使用以下模板创建 [SageMaker 生命周期配置脚本](#)。该脚本为 Braket 自定义 SageMaker 笔记本实例。有关生命周期 CloudFormation 资源的配置选项，请参阅 AWS CloudFormation 用户指南 [AWS::SageMaker::NotebookInstanceLifecycleConfig](#) 中的。

```
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
```

## 步骤 2：创建由亚马逊担任的 IAM 角色 SageMaker

当您使用 Braket 笔记本实例时，SageMaker 会代表您执行操作。例如，假设您在支持的设备上使用电路运行 Braket 笔记本电脑。在笔记本实例中，为您在 Braket 上 SageMaker 运行操作。笔记本执行角色定义了允许代表您执行的确切操作。SageMaker 有关更多信息，请参阅 Amazon SageMaker 开发者指南中的 [SageMaker 角色](#)。

使用以下示例创建具有所需权限的 Braket 笔记本执行角色。您可以根据需要修改策略。

**Note**

确保该角色有权对前缀为的 Amazon S3 存储桶s3:GetObject执行s3:ListBucket和操作。braketnotebookcdk-生命周期配置脚本需要这些权限才能复制 Braket 笔记本安装脚本。

```
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: "*"

```

## 步骤 3：使用前缀创建一个 Amazon SageMaker 笔记本实例 **amazon-braket-**

使用 SageMaker 生命周期脚本和步骤 1 和步骤 2 中创建的 IAM 角色创建 SageMaker 笔记本实例。笔记本实例是为 Braket 定制的，可以通过 Amazon Braket 控制台进行访问。有关此 CloudFormation 资源配置选项的更多信息，请参阅AWS CloudFormation 用户指南[AWS::SageMaker::NotebookInstance](#)中的。

```

BraketNotebook:
  Type: AWS::SageMaker::NotebookInstance
  Properties:
    InstanceType: ml.t3.medium
    NotebookInstanceName: !Sub amazon-braket-notebook-${AWS::StackName}
    RoleArn: !GetAtt ExecutionRole.Arn
    VolumeSizeInGB: 30
    LifecycleConfigName: !GetAtt
      BraketNotebookInstanceLifecycleConfig.NotebookInstanceLifecycleConfigName

```

## 高级日志

您可以使用记录器记录整个任务处理过程。这些高级日志记录技术允许您查看后台轮询并创建记录以供日后调试。

要使用记录器，我们建议更改`poll_timeout_seconds`和`poll_interval_seconds`参数，以便量子任务可以长时间运行，持续记录量子任务状态，并将结果保存到文件中。您可以将此代码传输到 Python 脚本而不是 Jupyter 笔记本，这样该脚本就可以在后台作为进程运行。

### 配置记录器

首先，配置记录器，使所有日志自动写入文本文件，如以下示例行所示。

```
# 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
```

## 创建并运行电路

现在，你可以创建一个电路，将其提交给设备运行，然后看看会发生什么，如本例所示。

```
# 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
)
```

## 检查日志文件

您可以通过输入以下命令来检查文件中写入的内容。

```
# 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})
```

## 从日志文件中获取 ARN

从返回的日志文件输出中（如前面的示例所示），您可以获取 ARN 信息。使用 ARN ID，您可以检索已完成的量子任务的结果。

```
# 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()
```



# 亚马逊 Braket 中的安全

本章帮助您了解在使用 Amazon Braket 时如何应用分担责任模型。它向您展示了如何配置 Amazon Braket 以实现您的安全和合规目标。您还将学习如何使用其他方法来帮助您监控和保护您AWS 服务的 Amazon Braket 资源。

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。您应对其他因素负责，包括数据的敏感性、贵公司的要求以及适用的法律和法规。

## 共同承担安全责任

安全性是 AWS 和您的共同责任。[责任共担模式](#) 将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。作为 [AWS 合规性计划](#) 的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用于 Amazon Braket 的合规计划，请参阅 [合规计划范围内的 AWS 服务](#)。
- 云端安全 — 您负责保持对托管在此 AWS 基础架构上的内容的控制。此内容包括您所使用的 AWS 服务的安全配置和管理任务。

## 数据保护

AWS [分担责任模型](#) 适用于 Amazon Braket 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务的安全配置和管理任务。有关数据隐私的更多信息，请参阅 [数据隐私常见问题](#)。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的博客文章 [AWS Shared Responsibility Model and GDPR](#)。

出于数据保护目的，我们建议您保护 AWS 账户凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。

- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅 [《美国联邦信息处理标准 \(FIPS\) 第 140-2 版》](#)。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括当您 AWS 服务使用控制台、API 或软件开发工具包与 Amazon Braket 或其他人合作 AWS CLI 时。AWS 在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，我们强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

## 数据留存

90 天后，Amazon Braket 会自动删除与您的量子任务关联的所有量子任务 ID 和其他元数据。由于该数据保留政策，尽管这些任务和结果仍存储在您的 S3 存储桶中，但无法再通过从 Amazon Braket 控制台进行搜索来检索。

如果您需要访问在 S3 存储桶中存储超过 90 天的历史量子任务和结果，则必须单独记录任务 ID 以及与该数据关联的其他元数据。请务必在 90 天之前保存信息。您可以使用保存的信息来检索历史数据。

## 管理对 Amazon Braket 的访问权限

本章介绍运行 Amazon Braket 或限制特定用户和角色访问所需的权限。您可以向账户中的任何用户或角色授予（或拒绝）所需的权限。为此，请将相应的 Amazon Braket 策略附加到您账户中的该用户或角色，如以下各节所述。

作为先决条件，您必须 [启用 Amazon Braket](#)。要启用 Braket，请务必以拥有 (1) 管理员权限或 (2) 已分配 AmazonBraketFullAccess 策略并有权创建亚马逊简单存储服务 (Amazon S3) 存储桶的用户或角色登录。

本节内容：

- [亚马逊 Braket 资源](#)
- [笔记本和角色](#)
- [关于 AmazonBraketFullAccess 政策](#)
- [关于 AmazonBraketJobsExecutionPolicy 政策](#)
- [限制用户访问某些设备](#)

- [Amazon Braket 更新了托管 AWS 政策](#)
- [限制用户访问某些笔记本实例](#)
- [限制用户对某些 S3 存储桶的访问权限](#)

## 亚马逊 Braket 资源

Braket 创建了一种资源：量子任务资源。此资源类型的亚马逊资源名称 (ARN) 如下所示：

- 资源名称::: 服务AWS:: Braket
- ARN Regex : `arn : $ {Partition}: b raket : $ {Region}: $ {Region}: $ {Account}: quantum-task/$ {RandomId}`

## 笔记本和角色

您可以在 Braket 中使用笔记本资源类型。笔记本是 Braket 能够共享的 Amazon SageMaker 资源。要将笔记本与 Braket 配合使用，必须指定名称以开头的 IAM 角色。

要创建笔记本，必须使用具有管理员权限或附加了以下内联策略的角色。

```
{
  "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*"
      ]
    }
  }
}
```

要创建该角色，请按照“[创建笔记本](#)”页面中给出的步骤进行操作，或者让管理员为您创建该角色。确保已附上该AmazonBraketFullAccess政策。

创建角色后，您可以将该角色重复用于将来启动的所有笔记本电脑。

## 关于 AmazonBraketFullAccess政策

该AmazonBraketFullAccess政策授予 Amazon Braket 操作权限，包括执行以下任务的权限：

- 从 Amazon Elastic Container Registry 下载容器 — 读取和下载用于 Amazon Braket Hybrid Jobs 功能的容器镜像。容器必须符合“arn:aws:ecr::repository/amazon-braket”的格式。
- 保留 AWS CloudTrail 日志-除了启动和停止查询、测试指标筛选器和筛选日志事件外，还适用于所有描述、获取和列出操作。该 AWS CloudTrail 日志文件包含您的账户中发生的所有 Amazon Braket API 活动的记录。
- 利用角色控制资源-在您的账户中创建服务相关角色。服务相关角色可以代表您访问 AWS 资源。它只能由 Amazon Braket 服务使用。此外，还可以将 IAM 角色传递给 Amazon Braket CreateJobAPI，创建角色并将范围限定为的策略附加 AmazonBraketFullAccess 到该角色。
- 创建日志组、日志事件和查询日志组，以维护您账户的使用情况日志文件 — 创建、存储和查看账户中有关 Amazon Braket 使用情况的日志信息。查询混合作业日志组的指标。包含正确的 Braket 路径并允许放置日志数据。将指标数据放入 CloudWatch。

- 在 Amazon S3 存储桶中创建和存储数据，并列出所有存储桶 — 要创建 S3 存储桶，请列出您账户中的 S3 存储桶，然后将对象放入账户中名称以 amazon-braket-开头的任何存储桶并从中获取对象。Braket 需要这些权限才能将包含已处理量子任务结果的文件放入存储桶并从存储桶中检索。
- 传递 IAM 角色 — 将 IAM 角色传递给 CreateJobAPI。
- 亚马逊 SageMaker 笔记本 — 创建和管理范围为 “arn: aws: sagemaker:: notebook-instance/ amazon-braket-” 中资源的 SageMaker 笔记本实例。
- 验证服务配额 — 要创建 SageMaker 笔记本和 Amazon Braket Hybrid 任务，您的资源数量不能超过账户的[配额](#)。

## 政策内容

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

```

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

```

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

```

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



```

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

```

## 关于 AmazonBraketJobsExecutionPolicy 政策

该 AmazonBraketJobsExecutionPolicy 策略授予在 Amazon Braket 混合任务中使用的执行角色的权限，如下所示：

- 从 Amazon Elastic Container Registry 下载容器-读取和下载用于 Amazon Braket Hybrid Jobs 功能的容器镜像的权限。容器必须符合 “arn:aws:ecr:\*:\*:repository/amazon-braket\*” 的格式。
- 创建日志组、日志事件和查询日志组，以便维护您账户的使用情况日志文件 — 在您的账户中创建、存储和查看有关 Amazon Braket 使用情况的日志信息。查询混合作业日志组的指标。包含正确的 Braket 路径并允许放置日志数据。将指标数据放入 CloudWatch。
- 将 @@ 数据存储在 Amazon S3 存储桶中 — 列出您账户中的 S3 存储桶，将对象放入账户中名称中以 amazon-braket-开头的任何存储桶并从中获取对象。Braket 需要这些权限才能将包含已处理量子任务结果的文件放入存储桶并从存储桶中检索。
- 传递 IAM 角色 — 将 IAM 角色传递给 CreateJob API。角色必须符合 arn:aws:iam::\*:role/service-role/AmazonBraketJobsExecutionRole 的格式。

```

"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"
  }
}
]
```

## 限制用户访问某些设备

要限制某些用户访问某些 Braket 设备，您可以为特定IAM角色添加拒绝权限策略。

使用此类权限可以限制以下操作：

- CreateQuantumTask-拒绝在指定设备上创建量子任务。
- CreateJob-拒绝在指定设备上创建混合作业。
- GetDevice-拒绝获取指定设备的详细信息。

以下示例限制了对所有 QPU 的访问权限。AWS 账户 123456789012

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "braket:CreateQuantumTask",
        "braket:CreateJob",
        "braket:GetDevice"
      ],
      "Resource": [
        "arn:aws:braket:*:*:device/qpu/*"
      ]
    }
  ]
}
```

要改编此代码，请用受限设备的Amazon资源号 (ARN) 代替上一个示例中显示的字符串。此字符串提供资源值。在 Braket 中，设备代表一个 QPU 或模拟器，你可以调用它来运行量子任务。可用设备列在 [“设备” 页面](#)上。有两个架构用于指定对这些设备的访问权限：

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

以下是各种设备访问类型的示例

- 要选择所有区域的所有 QPU，请执行以下操作：`arn:aws:braket:*:*:device/qpu/*`
- 要仅选择 us-west-2 区域的所有 QPU，请执行以下操作：`arn:aws:braket:us-west-2:123456789012:device/qpu/*`
- 同样，要仅选择 us-west-2 区域的所有 QPU（因为设备是一种服务资源，而不是客户资源），请执行以下操作：`arn:aws:braket:us-west-2:* :device/qpu/*`
- 要限制对所有按需模拟器设备的访问，请执行以下操作：`arn:aws:braket:* :123456789012:device/quantum-simulator/*`
- 要限制在 us-east-1 区域访问该 IonQ Harmony 设备，请执行以下操作：`arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony`
- 要限制特定提供商对设备的访问（例如 Rigetti QPU 设备），请执行以下操作：`arn:aws:braket:* :123456789012:device/qpu/rigetti/*`
- 要限制对 TN1 设备的访问，请执行以下操作：`arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1`

## Amazon Braket 更新了托管 AWS 政策

下表提供了自该服务开始跟踪这些更改以来 Braket AWS 托管策略更新的详细信息。

更改	描述	日期
<a href="#">AmazonBraketFullAccess</a> -Braket 的完全访问政策	添加了要包含在策略中的 <code>servicequotas: GetServiceQuota</code> 和 <code>cloudwatch: GetMetricData</code> 操作。AmazonBraketFullAccess	2023 年 3 月 24 日

更改	描述	日期
<a href="#">AmazonBraketFullAccess</a> -Braket 的完全访问政策	Braket 调整了 ia PassRole m: 包含 service-role/ 路径的权限。 AmazonBraketFullAccess	2021 年 11 月 29 日
<a href="#">AmazonBraketJobsExecutionPolicy</a> -Amazon Braket Hybrid Jobs 的混合作业执行政策	Braket 更新了混合作业执行角色 ARN，使其包含 service-role/ 了路径。	2021 年 11 月 29 日
Braket 开始追踪更改	Braket 开始跟踪其 AWS 托管策略的变更。	2021 年 11 月 29 日

## 限制用户访问某些笔记本实例

要限制某些用户访问特定 Braket 笔记本实例，您可以向特定角色、用户或组添加拒绝权限策略。

以下示例使用[策略变量](#)有效地限制启动、停止和访问中特定笔记本实例的权限 AWS 账户 123456789012，该实例根据应具有访问权限的用户命名（例如，用户 Alice 将有权访问名为的笔记本实例 amazon-braket-Alice）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:CreateNotebookInstance",
        "sagemaker>DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker>DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": [
        "sagemaker:DescribeNotebookInstance",
```

```

        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance",
    ],
    "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
        ${aws:username}"
    ]
},
{
    "Effect": "Deny",
    "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
    ],
    "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
        ${aws:username}*"
    ]
}
]
}

```

## 限制用户对某些 S3 存储桶的访问权限

要限制某些用户访问特定 Amazon S3 存储桶，您可以向特定角色、用户或组添加拒绝策略。

以下示例限制了检索对象并将其放入特定 S3 存储桶 (arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice) 的权限，还限制了这些对象的列表。

```

{
    "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/*"
  ]
}
]
```

要限制某个 notebook 实例对存储桶的访问权限，您可以将上述策略添加到笔记本执行角色中。

## 亚马逊 Braket 服务相关角色

当您启用 Amazon Braket 时，将在您的账户中创建服务相关角色。

服务相关角色是一种独特的 IAM 角色，在本例中，它直接链接到 Amazon Braket。Amazon Braket 服务相关角色已预先定义为包含 Braket 在代表您致电他人时所需的所有权限。AWS 服务

服务相关角色可以更轻松地设置 Amazon Braket，因为您不必手动添加必要的权限。亚马逊 Braket 定义了其服务相关角色的权限。除非您更改这些定义，否则只有亚马逊 Braket 才能扮演其角色。定义的权限包括信任策略和权限策略。不能将该权限策略附加到任何其他 IAM 实体。

Amazon Braket 设置的服务相关角色是 AWS Identity and Access Management (IAM) [服务相关角色](#) 功能的一部分。有关其他支持服务相关角色的信息 AWS 服务，请参阅与 [IAM 配合使用的 AWS 服务](#)，并在“服务相关角色”列中查找具有“是”的服务。请选择 Yes 与查看该服务的服务相关角色文档的链接。

## 亚马逊 Braket 的服务相关角色权限

亚马逊 Braket 使用 `AWSServiceRoleForAmazonBraket` 服务相关角色，该角色信任 `braket.amazonaws.com` 实体担任该角色。

您必须配置权限以允许 IAM 实体（例如群组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅 [服务相关角色权限](#)。

默认情况下，Amazon Braket 中的服务相关角色被授予以下权限：

- Amazon S3 — 列出您账户中的存储桶，以及将对象放入您的账户中任何名称以 `amazon-braket-` 开头的存储桶并从中获取对象的权限。
- Amazon CloudWatch Logs — 列出和创建日志组、创建关联日志流以及将事件放入为 Amazon Braket 创建的日志组的权限。



以下策略附加到**AWSServiceRoleForAmazonBraket**服务相关角色：

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

## 亚马逊 Braket 的韧性

AWS全球基础设施是围绕AWS 区域可用区构建的。

每个区域提供多个物理分离和隔离的可用区。这些可用区 (AZ) 通过低延迟、高吞吐量和高度冗余的网络连接。因此，与传统的单或多数据中心基础架构相比，可用区具有更高的可用性、容错性和可扩展性。

您可以设计和运行在可用区之间自动故障切换的应用程序和数据库，而不会中断。

有关AWS 区域和可用区的更多信息，请参阅[AWS全球基础架构](#)。

## 亚马逊 Braket 的合规性验证

第三方审计师定期评估 Amazon Braket 的安全性和合规性以及我们与第三方硬件提供商的集成。有关 Braket 的合规性信息up-to-date列表，请参阅[AWS 服务按合规计划划分的范围](#)。有关一般信息，请参阅[AWS合规性](#)。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅[在中下载报告AWS Artifact](#)。

### Note

AWS合规性报告不涵盖来自第三方硬件提供商的 QPU，他们可以选择接受自己的独立审计。

在使用 Amazon Braket 时，您的合规责任由您的数据的敏感性、贵公司的合规目标以及适用的法律和法规决定。AWS提供以下资源以帮助实现合规性：

- [安全性与合规性快速入门指南](#) – 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署基于安全性和合规性的基准环境的步骤。
- [AWS 合规性资源](#) – 此业务手册和指南集合可能适用于您的行业和位置。

## 亚马逊 Braket 中的基础设施安全

作为一项托管服务，Amazon Braket 受AWS全球网络安全程序的保护，如[安全流程概述AWS白皮书](#)所述。

要通过网络访问亚马逊 Braket，您需要调用已发布的 AWS API。客户端必须支持传输层安全性 (TLS) 1.2 或更高版本。客户还必须支持具有完全前向保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 主体关联的秘密访问密钥来对请求进行签名。或者，您可以使用 [AWS Security Token Service \(AWS STS\)](#) 生成临时安全凭证来对请求进行签名。

## 亚马逊 Braket 硬件提供商的安全

Amazon Braket 上的 QPU 由第三方硬件提供商托管。当您在 QPU 上运行量子任务时，Amazon Braket 在将电路发送到指定的 QPU 进行处理时使用 DeviceEarn 作为标识符。

如果您使用 Amazon Braket 访问由第三方硬件提供商运营的量子计算硬件，则您的电路及其相关数据将由其运营设施之外的硬件提供商处理。物理和地理信息和 AWS 每个 QPU 都可用的区域可以在 [中找到设备详情](#) 亚马逊 Braket 控制台的部分。

您的内容是匿名的。只有处理电路所需的内容才会发送给第三方。AWS 账户信息不会传输给第三方。

静态和传输中的数据始终加密。数据解密仅用于处理。Amazon Braket 第三方提供商不得将您的内容存储或用于处理您的电路以外的其他用途。回路完成后，结果将返回到 Amazon Braket 并存储在您的 S3 存储桶中。

Amazon Braket 第三方量子硬件提供商的安全会定期接受审计，以确保符合网络安全、访问控制、数据保护和物理安全标准。

## 适用于 Amazon Braket 的亚马逊 VPC 终端节点

您可以通过创建接口 VPC 终端节点在您的 VPC 和 Amazon Braket 之间建立私有连接。接口端点由一种技术提供支持 [AWS PrivateLink](#)，该技术无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接即可访问 Braket API。您的 VPC 中的实例不需要公有 IP 地址即可与 Braket API 通信。

每个接口端点均由子网中的一个或多个 [弹性网络接口](#) 表示。

使用 PrivateLink，您的 VPC 和 Braket 之间的流量不会离开 Amazon 网络，这可以提高您与基于云的应用程序共享的数据的安全性，因为它可以减少您的数据暴露在公共互联网上的风险。有关更多信息，请参阅 Amazon [VPC 用户指南中的接口 VPC 终端节点 \(AWS PrivateLink\)](#)。

## Amazon Braket VPC 终端节点的注意事项

在为 Braket 设置接口 VPC 终端节点之前，请务必查看 Amazon VPC 用户指南中的 [接口终端节点属性和限制](#)。

Braket 支持从您的 VPC 调用其所有 [API 操作](#)。

默认情况下，允许通过 VPC 终端节点对 Braket 进行完全访问。如果您指定 VPC 终端节点策略，则可以控制访问权限。有关更多信息，请参阅《Amazon VPC 用户指南》中的[使用 VPC 端点控制对服务的访问权限](#)。

## 设置 Braket 然后 PrivateLink

要 AWS PrivateLink 与 Amazon Braket 一起使用，您必须创建一个亚马逊虚拟私有云（亚马逊 VPC）终端节点作为接口，然后通过 Amazon Braket API 服务连接到该终端节点。

以下是此过程的一般步骤，将在后面的章节中详细介绍。

- 配置并启动 Amazon VPC 来托管您的 AWS 资源。如果您已有 VPC，请跳过此步骤。
- 为 Braket 创建亚马逊 VPC 终端节点
- 通过您的端点连接并运行 Braket 量子任务

### 第 1 步：如果需要，启动亚马逊 VPC

请记住，如果您的账户已有 VPC 在运行，则可以跳过此步骤。

VPC 控制您的网络设置，例如 IP 地址范围、子网、路由表和网络网关。本质上，您是在自定义虚拟网络中启动 AWS 资源。有关 VPC 的更多信息，请参阅[Amazon VPC User Guide](#)。

打开[Amazon VPC 控制台](#)，创建一个包含子网、安全组和网络网关的新 VPC。

### 步骤 2：为 Braket 创建接口 VPC 终端节点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 Braket 服务创建 VPC 终端节点。有关更多信息，请参阅《Amazon VPC 用户指南》中的[创建接口端点](#)。

要在控制台中创建 VPC 终端节点，请打开[Amazon VPC 控制台](#)，打开终端节点页面，然后继续创建新的终端节点。记下端点 ID 以供日后参考。当你向 Braket API 拨打某些电话时，它必须作为 `-endpoint-url` 旗帜的一部分。

使用以下服务名称为 Braket 创建 VPC 终端节点：

- `com.amazonaws.substitute_your_region.braket`

注意：如果您为终端节点启用私有 DNS，则可以使用该区域的默认 DNS 名称向 Braket API 发出请求，`braket.us-east-1.amazonaws.com` 例如。

有关更多信息，请参阅《Amazon VPC 用户指南》中的[通过接口端点访问服务](#)。

### 第 3 步：通过终端连接并运行 Braket 量子任务

创建 VPC 终端节点后，您可以运行包含 `endpoint-url` 参数的 CLI 命令来指定 API 或运行时的接口终端节点，例如以下示例：

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

如果您为 VPC 终端节点启用私有 DNS 主机名，则无需在 CLI 命令中将终端节点指定为 URL。相反，CLI 和 Amazon Braket API SDK 默认使用的 Braket DNS 主机名会解析到您的 VPC 终端节点。它的形式如以下示例所示：

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

这篇名为“[使用 AWS PrivateLink 终端节点从 Amazon VPC 直接访问亚马逊 SageMaker 笔记本电脑](#)”的博客文章提供了一个示例，说明如何设置终端节点以与 SageMaker 笔记本建立安全连接，这与 Amazon Braket 笔记本类似。

如果您正在按照博客文章中的步骤操作，请记得用 Amazon Braket 这个名字代替 Amazon SageMaker。如果您的地区不是 `us-east-1`，请在服务 AWS 区域名称中输入您的正确名称 `com.amazonaws.us-east-1.braket` 或将您的正确名称替换到该字符串中。

## 有关创建终端节点的更多信息

- 有关如何创建带有私有子网的 VPC 的信息，请参阅[创建带有私有子网的 VPC](#)
- 有关使用 Amazon VPC 控制台或 AWS CLI 创建和配置端点的信息，请参阅 Amazon VPC 用户指南中的[创建接口端点](#)。
- 有关使用创建和配置终端节点的信息 AWS CloudFormation，请参阅《用户指南》中的[AWS::EC2::vpceEndpoint 资源](#)。AWS CloudFormation

## 使用 Amazon VPC 终端节点策略控制访问权限

要控制对 Amazon Braket 的连接访问，您可以将 AWS Identity and Access Management (IAM) 终端节点策略附加到您的亚马逊 VPC 终端节点。该策略指定以下信息：

- 可以执行操作的委托人（用户或角色）。

- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的[使用 VPC 端点控制对服务的访问](#)。

示例：Braket 操作的 VPC 终端节点策略

以下示例显示了 Braket 的终端节点策略。当连接到终端节点时，此策略允许所有委托人访问所有资源上列出的 Braket 操作。

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

您可以通过附加多个端点策略来创建复杂的 IAM 规则。有关更多信息和示例，请参阅：

- [Step Functions 的亚马逊虚拟私有云端点策略](#)
- [为非管理员用户创建精细的 IAM 权限](#)
- [使用 VPC 端点控制对服务的访问](#)

# 对 Amazon Braket 进行故障排除

使用本节中的疑难解答信息和解决方案来帮助解决与 Amazon Braket 有关的问题。

本节内容：

- [AccessDeniedException](#)
- [调用 CreateQuantumTask 操作时出错 \(ValidationException\)](#)
- [某个 SDK 功能不起作用](#)
- [由于以下原因，混合作业失败 ServiceQuotaExceededException](#)
- [组件在笔记本实例中停止工作](#)
- [亚马逊 Braket 配额](#)
- [对 OpenQASM 进行故障](#)

## AccessDeniedException

如果您 AccessDeniedException 在启用或使用 Braket 时收到 Braket，则您可能正在尝试在您的受限角色无权访问的区域启用或使用 Braket。

在这种情况下，您应联系内部 AWS 管理员，了解以下哪些条件适用：

- 是否存在角色限制，无法访问某个区域。
- 如果您尝试使用的角色被允许使用 Braket。

如果您的角色在使用 Braket 时无法访问给定区域，则您将无法使用该特定区域的设备。

## 调用 CreateQuantumTask 操作时出错 (ValidationException)

如果您收到类似于以下内容的错误：请 An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-braket-... 检查您指的是现有的 s3\_folder。Braket 不会自动为您创建新的 Amazon S3 存储桶和前缀。

如果您 API 直接访问并收到类似于以下内容的错误：请 Failed to create quantum task: Caller doesn't have access to s3://MY\_BUCKET 检查您是否未包含 s3:// 在 Amazon S3 存储桶路径中。

## 某个 SDK 功能不起作用

您的 Python 版本必须是 3.9 或更高版本。对于 Amazon Braket 混合任务，我们推荐 Python 3.10。

验证您的 SDK 和架构是否正确。up-to-date要通过笔记本或 python 编辑器更新 SDK，请运行以下命令：

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

要更新架构，请运行以下命令：

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

如果您通过自己的客户访问亚马逊 Braket，请确认您的[AWS 地区](#)是否已设置为 Amazon Braket 支持的区域。

## 由于以下原因，混合作业失败 ServiceQuotaExceededException

如果您超过目标模拟器设备的并发量子任务限制，则可能无法创建针对 Amazon Braket 模拟器运行量子任务的混合作业。有关服务限制的更多信息，请参阅[配额](#)主题。

如果您在账户中的多个混合作业中对模拟器设备运行并发任务，则可能会遇到此错误。

要查看针对特定仿真器设备的并发量子任务数，请使用 search-quantum-tasks API，如以下代码示例所示。

```
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
```

您还可以使用亚马逊 CloudWatch 指标查看针对设备创建的量子任务：Braket > B y Device。



为避免遇到这些错误，请执行以下操作：

1. 申请增加模拟器设备的并发量子任务数量的服务配额。这仅适用于该SV1设备。
2. 处理代码中的ServiceQuotaExceeded异常并重试。

## 组件在笔记本实例中停止工作

如果笔记本电脑的某些组件无法正常工作，请尝试以下方法：

1. 将您创建或修改的所有笔记本下载到本地驱动器。
2. 停止您的笔记本实例。
3. 删除您的笔记本实例。
4. 使用不同的名称创建新的笔记本实例。
5. 将笔记本上传到新实例。

## 亚马逊 Braket 配额

下表列出了 Amazon Braket 的服务配额。服务限额（也称为限制）是 AWS 账户使用的服务资源或操作的最大数量。

有些配额可以增加。有关更多信息，请参阅 [AWS 服务 配额](#)。

- 无法提高突发速率配额。
- 可调配额（无法调整的突发速率除外）的最大速率增加是指定默认速率限制的 2 倍。例如，可以将 60 的默认配额调整为最大 120。
- 并发 SV1 (DM1) 量子任务的可调整配额允许每个任务最多 60 个 AWS 区域。
- 混合作业允许的最大计算实例数为 5，并且配额是可调整的。

资源	描述	限制	可调整
API 请求速率	在当前区域内的此账户中，您每秒可以发送的请求的最大数量。	140	是

资源	描述	限制	可调整
API请求的突发率	在当前区域内的此账户中，您每秒可以在一次突增中发送的额外请求的最大数量 (RPS)。	600	否
CreateQuantumTask 请求速率	每个区域您每秒可以在该账户中发送的最大CreateQuantumTask 请求数。	20	是
CreateQuantumTask 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外CreateQuantumTask 请求数 (RPS)。	40	否
SearchQuantumTasks 请求速率	每个区域您每秒可以在该账户中发送的最大SearchQuantumTasks 请求数。	5	是
SearchQuantumTasks 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外SearchQuantumTasks 请求数 (RPS)。	50	不可以
GetQuantumTask 请求速率	每个区域您每秒可以在该账户中发送的最大GetQuantumTask 请求数。	100	是

资源	描述	限制	可调整
GetQuantumTask 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外GetQuantumTask 请求数 (RPS)。	500	否
CancelQuantumTask 请求速率	每个区域您每秒可以在该账户中发送的最大CancelQuantumTask 请求数。	2	可以
CancelQuantumTask 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外CancelQuantumTask 请求数 (RPS)。	20	否
GetDevice 请求速率	每个区域您每秒可以在该账户中发送的最大GetDevice 请求数。	5	是
GetDevice 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外GetDevice 请求数 (RPS)。	50	不可以
SearchDevices 请求速率	每个区域您每秒可以在该账户中发送的最大SearchDevices 请求数。	5	是

资源	描述	限制	可调整
SearchDevices 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外SearchDevices 请求数 (RPS)。	50	不可以
CreateJob 请求速率	每个区域您每秒可以在该账户中发送的最大CreateJob 请求数。	1	是
CreateJob 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外CreateJob 请求数 (RPS)。	5	否
SearchJob 请求速率	每个区域您每秒可以在该账户中发送的最大SearchJob 请求数。	5	是
SearchJob 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外SearchJob 请求数 (RPS)。	50	不可以
GetJob 请求速率	每个区域您每秒可以在该账户中发送的最大GetJob请求数。	5	是

资源	描述	限制	可调整
GetJob请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外GetJob请求数 (RPS)。	25	否
CancelJob 请求速率	每个区域您每秒可以在该账户中发送的最大CancelJob 请求数。	2	可以
CancelJob 请求的突发率	您可在当前区域的此账户中一次性发送的最大每秒额外CancelJob 请求数 (RPS)。	5	否
并发量SV1子任务数	在当前区域的状态向量模拟器 (SV1) 上运行的最大并发量子任务数。	100 us-east-1, 50 us-west-1, 100 us-west-2 , 50 eu-west-2	否
并发量DM1子任务数	在当前区域的密度矩阵模拟器 (DM1) 上运行的最大并发量子任务数。	100 us-east-1, 50 us-west-1, 100 us-west-2 , 50 eu-west-2	否
并发量TN1子任务数	在当前区域的张量网络模拟器 (TN1) 上运行的最大并发量子任务数。	10 us-east-1, 10 us-west-2 , 5 eu-west-2 ,	是

资源	描述	限制	可调整
并发混合作业数量	当前区域中并发混合作业的最大数量。	5	是
混合作业运行时限制	混合作业可以运行的最大时间（以天为单位）。	5	否

以下是混合任务的默认经典计算实例配额。要提高这些配额，请联系 AWS Support。此外，还会为每个实例指定可用区域。

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c4.xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.c4.xlarge 类型实例数。	5	支持	是	是	是	是	不支持
混合作业的 ml.c4.2xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid	5	支持	是	是	是	是	不支持

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
	Jobs , 允许的最大 ml.c4.2xlarge 类型实例数。							
混合作业的最大 ml.c4.4xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c4.4xlarge 类型实例数。	5	支持	是	是	是	是	不支持

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c4.8xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c4.8xlarge 类型实例数。	5	支持	是	是	是	否	否
混合作业的 ml.c5.xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5.xlarge 类型实例数。	5	支持	是	是	是	是	是



资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c5.2xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5.2xlarge 类型实例数。	5	支持	是	是	是	是	是
混合作业的 ml.c5.4xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5.4xlarge 类型实例数。	1	是	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c5.9xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5.9xlarge 类型实例数。	1	是	是	是	是	是	是
混合作业的最大 ml.c5.18xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5.18xlarge 类型实例数。	0	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c5n.xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.c5n.xlarge 类型实例数。	0	支持	是	是	是	否	否
混合作业的 ml.c5n.2xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.c5n.2xlarge 类型实例数。	0	支持	是	是	是	否	否

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c5n.4xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.c5n.4xlarge 类型实例数。	0	支持	是	是	是	否	否
混合作业的 ml.c5n.9xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.c5n.9xlarge 类型实例数。	0	支持	是	是	是	否	否

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.c5n.18xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.c5n.18xlarge 类型实例数。	0	支持	是	是	是	否	否
混合作业的 ml.g4dn.xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.g4dn.xlarge 类型实例数。	0	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.g4dn.2xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.g4dn.2xlarge 类型实例数。	0	支持	是	是	是	是	是
混合作业的 ml.g4dn.4xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.g4dn.4xlarge 类型实例数。	0	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 ml.g4dn.8xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.g4dn.8xlarge 类型实例数。	0	支持	是	是	是	是	是
混合作业的 ml.g4dn.12xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.g4dn.12xlarge 类型实例数。	0	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大实例数 ml.g4dn.16xlarge	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.g4dn.16xlarge 类型实例数。	0	支持	是	是	是	是	是
混合作业的最大 ml.m4.xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m4.xlarge 类型实例数。	5	支持	是	是	是	是	不支持



资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.m4.2xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m4.2xlarge 类型实例数。	5	支持	是	是	是	是	不支持
混合作业的最大 ml.m4.4xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m4.4xlarge 类型实例数。	2	是	是	是	是	是	不支持

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的 最大 ml.m4.10xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.m4.10xlarge 类型实例数。	0	支持	是	是	是	是	不支持
混合作业的 最大实例数 ml.m4.16xlarge	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.m4.16xlarge 类型实例数。	0	支持	是	是	是	是	不支持

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.m5.large 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.m5.large 类型实例数。	5	支持	是	是	是	是	是
混合作业的 ml.m5.xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.m5.xlarge 类型实例数。	5	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.m5.2xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m5.2xlarge 类型实例数。	5	支持	是	是	是	是	是
混合作业的最大 ml.m5.4xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m5.4xlarge 类型实例数。	5	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.m5.12xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m5.12xlarge 类型实例数。	0	支持	是	是	是	是	是
混合作业的最大 ml.m5.24xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.m5.24xlarge 类型实例数。	0	支持	是	是	是	是	是

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.p2.xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p2.xlarge 类型实例数。	0	支持	是	否	是	否	否
混合作业的最大 ml.p2.8xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p2.8xlarge 类型实例数。	0	支持	是	否	是	否	否

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.p2.16xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p2.16xlarge 类型实例数。	0	支持	是	否	是	否	否
混合作业的最大 ml.p3.2xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p3.2xlarge 类型实例数。	0	支持	是	否	是	否	否

资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.p4d.24xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.p4d.24xlarge 类型实例数。	0	支持	是	否	是	否	否
混合作业的 ml.p3dn.24xlarge 实例的最大数量	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs , 允许的最大 ml.p3dn.24xlarge 类型实例数。	0	支持	是	否	是	否	否



资源	描述	限制	可调整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合作业的最大 ml.p3.8xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p3.8xlarge 类型实例数。	0	支持	是	否	是	是	不支持
混合作业的最大 ml.p3.16xlarge 实例数	对于此账户和区域中的所有 Amazon Braket Hybrid Jobs，允许的最大 ml.p3.16xlarge 类型实例数。	0	支持	是	否	是	是	不支持

### 请求限额更新

如果您收到实例类型的 `ServiceQuotaExceeded` 例外情况，但没有足够的可用实例，则可以从 AWS 控制台的 [“服务配额” 页面请求提高限制](#)，然后在“服务”下 AWS 搜索 Amazon Braket。

**Note**

如果您的混合作业无法预置请求的 ML 计算容量，请使用其他区域。此外，如果您在表中看不到实例，则该实例不可用于混合作业。

## 其他配额和限制

- Amazon Braket 量子任务操作的大小限制在 3MB 以内。
- SV1、DM1和Rigetti设备允许的每个任务的最大镜头数为 100,000。
- 每项任务允许的最大镜头数TN1为 1000。
- 对于 Ari IonQ a-1 和 Aria-2 设备，每项任务的最大射门数为 5,000 次。IonQ对于 Harmony 和 Forte 设备以及OQC设备，最大值为 10,000。
- 对于QuEra，每项任务允许的最大射门数为 1000。
- 对于TN1和QPU设备，每项任务的镜头数必须大于 0。

## 对 OpenQASM 进行故障

本节提供了在使用 OpenQasm 3.0 时遇到错误时可能有用的疑难解答指针。

本节内容：

- [包含语句错误](#)
- [非连续错误 qubits](#)
- [将物理错误qubits与虚拟qubits错误混为一谈](#)
- [请求结果类型并在同一个程序错误qubits中进行测量](#)
- [经典限值和qubit寄存器限值超出误差](#)
- [方框前面没有逐字编译指示错误](#)
- [逐字记录框缺少本机门错误](#)
- [逐字记录框缺少物理错误 qubits](#)
- [逐字编译指示缺少“braket”错误](#)
- [qubits无法为单曲编制索引错误](#)
- [双qubit门qubits中的物理未连接错误](#)
- [GetDevice 不返回 OpenQasm 结果错误](#)

- [本地模拟器支持警告](#)

## 包含语句错误

Braket 目前没有标准的门库文件可以包含在 OpenQasm 程序中。例如，以下示例引发了解析器错误。

```
OPENQASM 3;
include "standardlib.inc";
```

以下代码生成错误消息：No terminal matches ''' in the current parser context, at line 2 col 17.

## 非连续错误 qubits

在设备功能true中设置为的requiresContiguousQubitIndices设备qubits上使用非连续会导致错误。

在模拟器和上运行量子任务时IonQ，以下程序会触发错误。

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

以下代码生成错误消息：Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].

## 将物理错误qubits与虚拟qubits错误混为一谈

不允许在同一个程序qubits中混合物理qubits和虚拟，这会导致错误。以下代码生成错误。

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

以下代码生成错误消息：[line 4] mixes physical qubits and qubits registers.

## 请求结果类型并在同一个程序错误qubits中进行测量

请求在同一程序qubits中明确测量的结果类型会导致错误。以下代码生成错误。

```
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])
```

以下代码生成错误消息：Qubits should not be explicitly measured when result types are requested.

## 经典限值和qubit寄存器限值超出误差

只允许使用一个经典寄存器和一个qubit寄存器。以下代码生成错误。

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

以下代码生成错误消息：[line 4] cannot declare a qubit register. Only 1 qubit register is supported.

## 方框前面没有逐字编译指示错误

所有方框前面都必须有逐字编译指示。以下代码生成错误。

```
box{
  rx(0.5) $0;
}
```

以下代码生成错误消息：In verbatim boxes, native gates are required. x is not a device native gate.

## 逐字记录框缺少本机门错误

逐字记录箱应该有原生大门和实体门。qubits以下代码生成原生门错误。

```
#pragma braket verbatim
box{
x $0;
}
```

以下代码生成错误消息：In verbatim boxes, native gates are required. x is not a device native gate.

## 逐字记录框缺少物理错误 qubits

逐字记录框必须有实物。qubits以下代码生成缺失的物理qubits错误。

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

以下代码生成错误消息：Physical qubits are required in verbatim box.

## 逐字编译指示缺少“braket”错误

您必须在逐字记录编译指示中包含“braket”。以下代码生成错误。

```
#pragma braket verbatim // Correct
#pragma verbatim // wrong
```

以下代码生成错误消息：You must include “braket” in the verbatim pragma

## qubits无法为单曲编制索引错误

qubits无法为单曲编制索引。以下代码生成错误。

```
OPENQASM 3;
```

```
qubit q;
h q[0];
```

以下代码会生成错误：`[line 4] single qubit cannot be indexed.`

但是，可以按如下方式对单个qubit数组进行索引：

```
OPENQASM 3;

qubit[1] q;
h q[0]; // This is valid
```

## 双qubit门qubits中的物理未连接错误

要使用物理设备qubits，请首先qubits通过检查来确认设备使用物理设备，`device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits`然后通过选中`device.properties.paradigm.connectivity.connectivityGraph`或来验证连接图`device.properties.paradigm.connectivity.fullyConnected`。

```
OPENQASM 3;

cnot $0, $14;
```

以下代码生成错误消息：`[line 3] has disconnected qubits 0 and 14`

## GetDevice 不返回 OpenQasm 结果错误

如果您在使用 Braket SDK 时在 GetDevice 响应中看不到 OpenQASM 结果，则可能需要设置 `AWS_EXECUTION_ENV` 环境变量来配置用户代理。有关如何为 Go 和 Java SDK 执行此操作，请参阅下面提供的代码示例。

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用时配置用户代理，请执行以下操作：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]
```

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用 Boto3 时配置用户代理，请执行以下操作：

```
import boto3
```

```
import boto3

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

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用 / (SDK v2) 时配置用户代理，请执行以下操作：  
JavaScriptTypeScript

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

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用 / 时配置用户代理 (SDK v3)，请执行以下操作：  
JavaScriptTypeScript

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

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用 Go SDK 时配置用户代理，请执行以下操作：

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

要将 `AWS_EXECUTION_ENV` 环境变量设置为在使用 Java 开发工具包时配置用户代理，请执行以下操作：

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

## 本地模拟器支持警告

LocalSimulator 支持 OpenQasm 中的高级功能，这些功能在 QPU 或按需模拟器上可能不可用。如果您的程序仅包含特定于的 LocalSimulator 语言功能（如以下示例所示），您将收到一条警告。

```
qasm_string = ""
```

```
qubit[2] q;  
  
h q[0];  
ctrl @ x q[0], q[1];  
""  
qasm_program = Program(source=qasm_string)
```

此代码生成警告：`此程序使用仅支持的 OpenQasm 语言功能。 LocalSimulatorQPU 或按需模拟器可能不支持其中一些功能。

有关支持的 OpenQasm 功能的更多信息，[请单击此处](#)。



# 亚马逊 Braket 的 API 和 SDK 参考指南

Amazon Braket 提供 API、SDK 和命令行界面，您可以使用它们来创建和管理笔记本实例以及训练和部署模型。

- [亚马逊 Braket Python SDK \( 推](#)
- [亚马逊 Braket API 参考](#)
- [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](#)

您还可以从 Amazon Braket 教程GitHub存储库获取代码示例。

- [支架教程 GitHub](#)

# 文档历史记录

下表描述了此版本的 Amazon Braket 的文档。

- API版本：2022 年 4 月 28 日
- 最新API参考更新：2023 年 9 月 25 日
- 最新文档更新：2024 年 5 月 22 日

更改	描述	日期
新设备IQM Garnet和区域 Europe North 1	增加了对 <a href="#">IQM Garnet 设备</a> 的支持。一种采用方格拓扑结构的 20 量子比特器件。将 Braket <a href="#">支持的地区</a> 扩展到欧洲北部 1 ( 斯德哥尔摩 )。	2024年5月22日
已发布本地停机功能	<a href="#">实验功能</a> 现在包括 Aquila QPU QuEra 的本地失谐功能。	2024 年 4 月 11 日
笔记本非活动管理器发布	<a href="#">创建笔记本实例</a> 时，启用不活动管理器并设置空闲持续时间以自动重置 Braket 笔记本实例。	2024 年 3 月 27 日
目录重做	重组了 Amazon Braket 目录，以遵守 AWS 风格指南要求并改善内容流以提高客户体验。	2023 年 12 月 12 日
发布了 <a href="#">Braket Di</a>	增加了对 Braket 直接功能的支持，包括： <ul style="list-style-type: none"> <li>• <a href="#">预留</a></li> <li>• <a href="#">专家建议</a></li> <li>• <a href="#">实验能力</a></li> </ul>	2023 年 11 月 27 日

更新了 <a href="#">创建 Amazon Braket 笔记本实例</a>	更新了文档，添加了为新老的 Amazon Braket 客户创建笔记本实例的信息。	2023 年 11 月 27 日
更新了 <a href="#">自带集装箱 (BYOC)</a>	更新了文档，添加了有关何时加入 BYOC、BYOC 配方以及在容器上运行 Braket Hybrid Jobs 的信息。	2023 年 10 月 18 日
混合工作装饰器发布	<p>已添加<a href="#">将本地代码作为混合作业运行</a>页面。包含示例：</p> <ul style="list-style-type: none"> <li>• 使用本地 Python 代码创建混合作业</li> <li>• 安装其他 Python 软件包和源代码</li> <li>• 将数据保存并加载到混合作业实例中</li> <li>• 混合工作装饰者的最佳实践</li> </ul>	2023 年 10 月 16 日
增加了 <a href="#">队列可见性</a>	<p>更新了《开发者指南》文档，使其包含queue depth和queue position。</p> <p>更新了 API 文档，以反映队列可见性的新 API 更改。</p>	2023 年 9 月 25 日
标准化文档中的命名	更新了文档，将“作业”的任何实例更改为“混合作业”，将“任务”更改为“量子任务”	2023 年 9 月 11 日
新设备 IonQ Aria 2	增加了对该IonQ Aria 2设备的支持	2023 年 9 月 8 日
更新了 <a href="#">原生之门</a>	更新了文档，添加了有关 Rigetti 对本机门的编程访问的信息。	2023 年 8 月 16 日

Xanadu离开	更新了文档以移除所有Xanadu设备	2023年6月2日
新设备 IonQ Aria	增加了对该IonQ Aria设备的支持	2023年5月16日
已停用的Rigetti设备	已停止对的支持 Rigetti Aspen-M-2	2023年5月2日
更新的AmazonBraketFullAccess政策信息	更新了定义AmazonBraketFullAccess策略内容的脚本，使其包括 <code>servicequotas: GetServiceQuota</code> 和 <code>cloudwatch: GetMetricData</code> 操作以及有关配额限制的信息。	2023年4月19日
“导游之旅”发布	更改了文档，以反映更新、更简化的 Braket 入职方法。	2023年4月5日
新设备 Rigetti Aspen-M-3	增加了对该Rigetti Aspen-M-3设备的支持	2023年1月17日
新的伴随渐变功能	添加了有关提供的伴随渐变功能的信息 SV1	2022年12月7日
新的算法库功能	添加了有关 Braket 算法库的信息，该库提供了预先构建的量子算法目录	2022年11月28日
D-Wave离开	更新了文档以适应所有D-Wave设备的移除	2022年11月17日
新设备 QuEra Aquila	增加了对该QuEra Aquila设备的支持	2022年10月31日
对 Braket Pulse 的支持	增加了对 Braket Pulse 的支持，允许在设备上Rigetti使用脉冲控制 OQC	2022年10月20日

支持 ionQ 原生门	增加了对 ionQ 设备提供的本机门设置的支持	2022 年 9 月 13 日
新实例配额	更新了与混合任务关联的默认经典计算实例配额	2022 年 8 月 22 日
新的服务仪表盘	更新了控制台屏幕截图以包含服务控制面板	2022 年 8 月 17 日
新设备 Rigetti Aspen-M-2	增加了对该Rigetti Aspen-M-2设备的支持	2022 年 8 月 12 日
OpenQasm 的新功能	添加了 OpenQasm 功能对本地模拟器 ( braket_sv 和 braket_dm ) 的支持	2022 年 8 月 4 日
新的成本追踪程序	增加了如何近乎实时地估算模拟器和硬件工作负载的最大成本	2022 年 7 月 18 日
新Xanadu Borealis设备	增加了对该Xanadu Borealis设备的支持	2022 年 6 月 2 日
新的入职简化程序	添加了有关新的简化入职程序如何运作的信息	2022 年 5 月 16 日
新设备 D-Wave Advantage_system6.1	增加了对该D-WaveAdvantage_system6.1设备的支持	2022 年 5 月 12 日
Support 对嵌入式模拟器的支持	添加了如何使用混合作业运行嵌入式仿真以及如何使用 PennyLane 闪电模拟器	2022 年 5 月 4 日
AmazonBraketFullAccess -亚马逊 Braket 的完整访问政策	新增 s3 : 允许用户查看和检查为 Amazon Braket 创建和使用的存储桶的ListAllMyBuckets权限	2022 年 3 月 31 日

对 OpenQASM 的支持	为基于门的量子设备和模拟器添加了 OpenQasm 3.0 支持	2022 年 3 月 7 日
新的量子硬件提供商Oxford Quantum Circuits和新的区域 eu-west-2	增加了对 OQC 和 eu-west-2 的支持	2022 年 2 月 28 日
新Rigetti设备	增加了对 Rigetti Aspen M-1 的支持	2022 年 2 月 15 日
新的资源限制	将DM1并发SV1任务的最大数量从 55 增加到 100	2022 年 1 月 5 日
新Rigetti设备	增加了对 Rigetti Aspen-11 的支持	2021 年 12 月 20 日
已停用的Rigetti设备	已停止对Rigetti Aspen-10设备的支持	2021 年 12 月 20 日
新的结果类型	局部密度矩阵仿真器和DM1设备支持的低密度矩阵结果类型	2021 年 12 月 20 日
更新了政策描述	亚马逊 Braket 更新了角色 ARN 以包含路径。servicerole/ 有关政策更新的信息，请参阅 <a href="#">Amazon Braket AWS 托管策略更新表</a> 。	2021 年 11 月 29 日
亚马逊 Braket 职位	Amazon Braket 混合作业用户指南并已添加 API	2021 年 11 月 29 日
新Rigetti设备	增加了对 Rigetti Aspen-10 的支持	2021 年 11 月 20 日
已停用的D-Wave设备	已停止对 D-Wave QPU 的支持，Advantage_system1	2021 年 11 月 4 日

新D-Wave设备	增加了对额外 D-Wave QPU 的支持， Advantage_system4	2021 年 10 月 5 日
新的噪音模拟器	增加了对密度矩阵模拟器 (DM1) qubits 和局部噪声模拟器 bra ket_dm 的支持，该模拟器最多可以模拟 17 个电路	2021 年 5 月 25 日
PennyLane 支持	在 Amazon Braket PennyLane 上增加了对的支持	2020 年 12 月 8 日
全新模拟器	增加了对张量网络模拟器 (TN1) 的支持，它允许更大的电路	2020 年 12 月 8 日
任务批处理	Braket 支持客户任务批处理	2020 年 11 月 24 日
手动qubit分配	Braket 支持在设备上手动 qubit分配 Rigetti	2020 年 11 月 24 日
可调整的配额	Braket 支持任务资源的自助服务可调整配额	2020 年 10 月 30 日
Support PrivateLink	您可以为 Braket 任务设置私有 VPC 终端节点	2020 年 10 月 30 日
支持标签	Braket 支持API基于量子任务资源的标签	2020 年 10 月 30 日
新D-Wave设备	增加了对额外 D-Wave QPU 的支持， Advantage_system1	2020 年 9 月 29 日
初始版本	亚马逊 Braket 文档的首次发布	2020 年 8 月 12 日

# AWS 术语表

有关最新AWS术语，请参阅[AWS词汇表](#)参考中的AWS词汇表。



本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。