aws

開發人員指南

Amazon Braket

Amazon Braket: 開發人員指南

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 Amazon Braket？

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

AmazonBraket 是一個完全託管， AWS 服務 可幫助研究人員，科學家和開發人員開始使用量子計算。量子計算具有解決傳統計算機無法觸及的計算問題的潛力，因為它利用量子力學定律以新的方式處理信息。

獲得量子計算硬件的訪問可能是昂貴和不方便的。有限的存取使得執行演算法、最佳化設計、評估技術的目前狀態，以及規劃何時投資資源以獲得最大的利益。布拉克特可以幫助您克服這些挑戰。

Braket 提供了對各種量子計算技術的單點訪問。使用布拉克特，您可以：

- 探索和設計量子演算法和混合演算法。
- 在不同的量子電路模擬器上測試算法。
- 在不同類型的量子計算機上運行算法。
- 建立概念證明應用程式。

定義量子問題和編程量子計算機來解決它們需要一套新的技能。為了幫助您獲得這些技能，Braket 提供了不同的環境來模擬和運行您的量子算法。您可以找到最適合您需求的方法，並透過一組稱為筆記本的範例環境快速開始使用。

Braket 開發有三個階段-構建，測試和運行：

建置- Braket 提供完全受控的 Jupyter 筆記型電腦環境，讓您輕鬆上手。Braket 筆記型電腦已預先安裝範例演算法、資源和開發人員工具，包括 Amazon Braket SDK。借助 Amazon Braket SDK，您可以構建量子算法，然後通過更改一行代碼在不同的量子計算機和模擬器上測試和運行它們。

測試- Braket 可讓您存取完全管理的高效能量子電路模擬器。您可以測試和驗證您的電路。Braket 可處理所有基礎軟體元件和 Amazon Elastic Compute Cloud (Amazon EC2) 叢集，消除在傳統高效能運算 (HPC) 基礎設施上模擬量子電路的負擔。

運行- Braket 提供對不同類型的量子計算機的安全，按需訪問。您可以從IonQ、OQC和Rigetti訪問基於門的量子計算機以及從. QuEra 您也沒有前期承諾，也不需要透過個別供應商取得存取權。

關於量子計算和布拉克

量子計算處於早期發展階段。重要的是要了解，目前沒有通用的，容錯的量子計算機存在。因此，某些類型的量子硬體更適合各種使用案例，而且存取各種運算硬體至關重要。Braket 通過第三方提供商提供各種硬件。

現有的量子硬件由於噪聲而受到限制，從而引入了錯誤。該行業正處於嘈雜的中間尺度量子（NISQ）時代。在 NISQ 時代，量子計算設備過於嘈雜，無法維持純量子算法，例如 Shor 算法或格羅弗算法。在獲得更好的量子糾錯之前，最實用的量子計算需要將傳統（傳統）計算資源與量子計算機相結合才能創建混合算法。Braket 可幫助您使用混合量子算法。

在混合量子算法中，量子處理單元（QPU）被用作 CPU 的協同處理器，從而加快了傳統算法中的特定計算速度。這些算法利用迭代處理，其中計算在傳統和量子計算機之間移動。例如，目前量子計算在化學、最佳化和機器學習中的應用是以變量子演算法為基礎，而變量子演算法是一種混合量子演算法。在變量子演算法中，經典的最佳化常式會反覆調整參數化量子電路的參數，與基於機器學習訓練集中的錯誤反覆調整神經網路權重的方式相同。Braket 提供 PennyLane 開放原始碼軟體庫的存取權，協助您使用變量子演算法。

量子計算正在獲得四個主要領域的計算牽引力：

- 數論-包括因子和密碼學（例如，Shor 算法是數論計算的主要量子方法）
- 最佳化 — 包括約束滿意度、解決線性系統和機器學習
- 口頭計算 — 包括搜尋、隱藏的子群組和訂單尋找 (例如，Grover 演算法是用於口腔計算的主要量子方法)
- 模擬 — 包括直接模擬、結不變量和量子近似最佳化演算法 (QAOA) 應用

這些類別的計算應用程序可以在金融服務，生物技術，製造業和製藥中找到，僅舉幾例。Braket 提供功能和範例筆記型電腦，除了某些實際問題之外，這些筆記型電腦已經可以應用於許多概念驗證問題。

Amazon Braket 術語和概念

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

以下術語和概念在 Braket 中使用：

模擬哈密頓模擬

模擬哈密頓模擬（AHS）是一種獨特的量子計算範式，用於直接模擬多體系統的時間依賴量子動力學。在 AHS 中，用戶直接指定時間依賴的哈密頓計算機，並且量子計算機以直接模擬此哈密頓計算機下的連續時間演變進行了調整。AHS 設備通常是特殊用途設備，而不是諸如基於門的設備之類的通用量子計算機。它們僅限於他們可以模擬的哈密頓人類。但是，由於這些哈密頓人是在設備上自然實現的，因此 AHS 不會遭受將算法計算為電路和實現門操作所需的開銷。

布拉基

我們將 Braket 服務命名為 bra-ket 符號，這是量子力學中的標準符號。它是由保羅·狄拉克在 1939 年推出來描述量子系統的狀態，它也被稱為狄拉克符號。

制動混合工作

AmazonBraket 具有一項稱為 Amazon Braket 混合作業的功能，可提供混合算法的完全託管執行。布拉基混合作業由三個部分組成：

1. 算法的定義，可以作為腳本，Python 模塊或碼頭容器提供。
2. 以 Amazon EC2 為基礎的混合式任務執行個體，在其上執行演算法。預設值是一個毫升的執行個體。
3. 在其上運行量子任務的量子設備，這些任務是算法的一部分。單個混合作業通常包含許多量子任務的集合。

Device

在 Amazon Braket 中，設備是可以運行量子任務的後端。裝置可以是 QPU 或量子電路模擬器。若要進一步了解，請參閱 Amazon Braket 支援的裝置。

基于门的量子计算

在基於門的量子計算（QC）中，也稱為基於電路的 QC，計算被分解為基本操作（門）。某些門是通用的，這意味著每個計算都可以表示為這些門的有限序列。門是量子電路的構建模塊，類似於古典數字電路的邏輯門。

哈密頓

物理系統的量子動力學是由它的哈密頓軸決定的，它編碼了有關系統成分之間的相互作用和外源性驅動力的影響的所有信息。N 量子比特系統的哈密頓軸通常表示為 $2^{N 乘 2^{N}}$ 複數矩陣在古典機器上。透過在量子裝置上執行類比哈密頓模擬，您可以避免這些指數資源需求。

脈衝

脈衝是傳輸到量子位的瞬態物理信號。它由一幀中播放的波形描述，該波形用作載波信號的支持，並綁定到硬件通道或端口。客戶可以通過提供調製高頻正弦載波信號的類比包絡來設計自己的脈

衝。該幀由頻率和相位進行唯一描述，該相位通常被選擇為與量子比特的 |0和 |1的能量水平之間的能量分離進行共振。因此，門被制定為具有預定形狀和校準參數（例如其幅度，頻率和持續時間）的脈衝。範本波形未涵蓋的使用案例將透過自訂波形啟用，這些波形將透過提供以固定實體週期時間分隔的值清單，以單一樣本解析度指定。

量子电路

量子電路是定義基於門的量子計算機上計算的指令集。量子電路是量子閘的序列，它們是qubit寄存器上的可逆轉換，以及測量指令。

量子电路模拟器

量子電路模擬器是一種在傳統計算機上運行並計算量子電路的測量結果的計算機程序。對於一般電路，量子模擬的資源需求隨著要模擬的qubits數量呈指數增長。Braket 提供對託管（通過 Braket 訪問API）和本地（AmazonBraket SDK 的一部分）量子電路模擬器的訪問。

量子計算機

量子計算機是一種物理設備，它使用量子機械現象（例如疊加和糾纏）來執行計算。量子計算（QC）有不同的範例，例如基於門的 QC。

量子處理單元

QPU 是可以在量子任務上運行的物理量子計算設備。QPU 可以基於不同的 QC 範例，例如基於門的 QC。若要進一步了解，請參閱 Amazon Braket 支援的裝置。

QPU 原生大门

QPU 原生閘可透過 QPU 控制系統直接對應至控制脈衝。本機門可以在 QPU 設備上運行，而無需進一步編譯。QPU 支持門的子集。您可以在 Amazon Braket 主控台的「裝置」頁面或透過 Braket SDK 找到裝置的原生閘道。

QPU 支援的閘極

QPU 支援的閘門是 QPU 裝置所接受的閘門。這些閘門可能無法直接在 QPU 上執行，這表示它們可能需要分解為原生閘門。您可以在 Amazon Braket 主控台的「裝置」頁面或透過 Braket SDK 找到裝置支援的閘門。Amazon

量子任務

在 Braket 中，量子任務是對設備的原子請求。對於基於門的 QC 設備，這包括量子電路（包括測量指令和數量shots）以及其他請求元數據。您可以通過 Amazon Braket SDK 或直接使用CreateQuantumTaskAPI操作來創建量子任務。創建量子任務後，它將被排入佇列，直到所請求的設備可用為止。您可以在 Amazon Braket 控制台的量子任務頁面上或使用GetQuantumTask或SearchQuantumTasksAPI操作來查看您的量子任務。

Qubit

量子計算機中的信息的基本單位被稱為qubit（量子位），很像在傳統計算中有點。A qubit 是一種可以通過不同的物理實現來實現的兩級量子系統，例如超導電路或單個離子和原子。其他qubit類型是基於光子，電子或核旋轉，或更奇特的量子系統。

Queue depth

Queue depth指特定裝置排入佇列的量子工作和混合式工作的數量。裝置的量子工作和混合式工作佇列計數可透過Braket Software Development Kit (SDK)或存取Amazon Braket Management Console。

1. 任務隊列深度是指當前以正常優先級運行的量子任務總數。
2. 優先級任務隊列深度是指等待運行的提交量子任務的總數Amazon Braket Hybrid Jobs。一旦混合任務開始，這些任務將優先於獨立任務。
3. 混合式工作佇列深度是指目前在裝置上排入佇列的混合式作業總數。 Quantum tasks作為混合式工作的一部分提交具有優先順序，且會在中彙總Priority Task Queue。

Queue position

Queue position指量子任務或混合任務在各自設備隊列中的當前位置。它可以通過或獲得量子任務或混合作業Amazon Braket Management Console。Braket Software Development Kit (SDK)

Shots

由於量子計算本質上是概率性的，因此任何電路都需要進行多次評估以獲得準確的結果。單個電路執行和測量稱為射擊。根據所需的結果精度來選擇電路的射擊次數（重複執行）。

AWS Amazon Braket 的術語和技巧

IAM 政策

IAM 政策是允許或拒絕 AWS 服務 和資源許可的文件。IAM 政策可讓您自訂使用者對資源的存取層級。例如，您可以允許使用者存取您的所有 Amazon S3 儲存貯體 AWS 帳戶，或僅存取特定儲存貯體中的所有 Amazon S3 儲存貯體。

- 最佳做法：在授予權限時遵循最小權限的安全性原則。遵循此原則，您可以防止使用者或角色擁有執行量子工作所需的更多權限。例如，如果員工只需要存取特定值區，請在 IAM 政策中指定值區，而不是授與員工存取您的所有值區 AWS 帳戶。

IAM 角色

IAM 角色是您可以假設暫時存取權限的身分。使用者、應用程式或服務才能擔任 IAM 角色，他們必須先獲得切換至角色的權限。當某人擔任 IAM 角色時，他們會放棄先前角色下擁有的所有先前許可，並承擔新角色的許可。

- 最佳做法：IAM 角色非常適合需要暫時授予服務或資源存取權限而非長期授予的情況。

Amazon S3 桶

亞馬遜簡單存儲服務（Amazon S3）是一種可 AWS 服務 讓您將數據作為對象存儲在存儲桶中的一種。Amazon S3 儲存貯體提供無限的儲存空間。Amazon S3 儲存貯體中物件的大小上限為 5 TB。您可以將任何類型的檔案資料上傳到 Amazon S3 儲存貯體，例如影像、影片、文字檔、備份檔案、網站的媒體檔案、存檔文件和 Braket 量子任務結果。

- 最佳做法：您可以設定權限來控制 S3 儲存貯體的存取權。如需詳細資訊，請參閱 Amazon S3 文件中的儲存貯體政策和使用者政策。

Amazon Braket 定價

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

使用 Amazon Braket，您可以隨需存取量子運算資源，而無需前期承諾。您僅需按實際用量付費。要了解有關定價的更多信息，請訪問我們的定價頁面。

近乎即時的成本追蹤

Braket SDK 可讓您選擇將近乎即時的成本追蹤功能新增至量子工作負載。我們的每個範例筆記本都包含成本追蹤程式碼，可為您提供 Braket 量子處理單元 (QPU) 和隨選模擬器的最高成本估算。最高預估費用將以美元顯示，不包括任何積分或折扣。

> **Note**
>
> 顯示的費用是根據您的 Amazon Braket 模擬器和量子處理單元 (QPU) 任務使用量估算而定。顯示的估計費用可能與您的實際費用有所不同。預估費用不計入任何折扣或積分，您可能會因使用 Amazon Elastic Compute Cloud (Amazon EC2) 等其他服務而產生額外費用。

SV1 的成本追蹤

為了演示如何使用成本跟踪功能，我們將構建一個 Bell State 電路並在我們的 SV1 模擬器上運行它。首先導入 Braket SDK 模塊，定義貝爾狀態並將該`Tracker()`功能添加到我們的電路中：

```
#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)
```

當您執行筆記本時，您可以預期下列輸出可用於您的貝爾狀態模擬。跟踪器功能將顯示發送的拍攝次數，完成的量子任務，執行持續時間，計費的執行持續時間以及您的最高成本（美元）。每個模擬的執行時間可能會有所不同。

```
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
```

使用成本追蹤器設定最高成本

您可以使用成本跟踪器來設置程序的最高成本。您可能有要花多少給定程序的最大閾值。如此一來，您就可以使用成本追蹤器，在執行程式碼中建立成本控制邏輯。下列範例會在 Rigetti QPU 上採用相同的電路，並將成本限制為 1 USD。在我們的代碼中運行電路的一次迭代的成本為 0.37 美元。我們已經設置了重複迭代的邏輯，直到總成本超過 1 美元; 因此，代碼片段將運行三次，直到下一次迭代超過 1 美元。通常，程序將繼續迭代，直到達到所需的最大成本，在這種情況下-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**
>
> 成本跟踪器不會跟踪失敗的TN1量子任務的持續時間。在TN1模擬過程中，如果您的排練完成，但收縮步驟失敗，您的排練費用將不會顯示在成本追蹤器中。

節省成本的最佳做法

請考慮下列使用 Amazon Braket 網頁的最佳實務。節省時間，將成本降至最低，並避免常見錯誤。

使用模擬器驗證

- 在 QPU 上執行模擬器之前，請先使用模擬器驗證電路，以便您可以微調電路，而不會產生 QPU 使用費用。
- 雖然在模擬器上運行電路的結果可能與在 QPU 上運行電路的結果不相同，但是您可以使用模擬器識別編碼錯誤或配置問題。

限制使用者存取特定裝置

- 您可以設定限制，防止未經授權的使用者在特定裝置上提交量子工作。限制存取的建議方法是使用 AWS IAM。有關如何執行此操作的詳細資訊，請參閱限制存取權。
- 我們建議您不要使用管理員帳戶來授予或限制使用者對 Amazon Braket 裝置的存取權。

設定帳單警示

- 您可以設定帳單警示，以便在帳單達到預設限制時通知您。建議設定鬧鐘的方法是透過 AWS Budgets。您可以設定自訂預算，並在成本或用量可能超過預算金額時接收警示。有關資訊，請瀏覽AWS Budgets。

以低射擊計數測試TN1量子任務

- 模擬器的成本低於 QHP，但是如果量子任務以高射擊計數運行，則某些模擬器可能會很昂貴。我們建議您以低shot數量測試TN1任務。 Shot計數不影響SV1和本地模擬器任務的成本。

檢查所有區域是否有量子任務

- 控制台僅顯示您當前的量子任務 AWS 區域。尋找已提交的可計費量子任務時，請務必檢查所有區域。
- 您可以在支援的裝置說明文件頁面上檢視裝置及其關聯區域的清單。

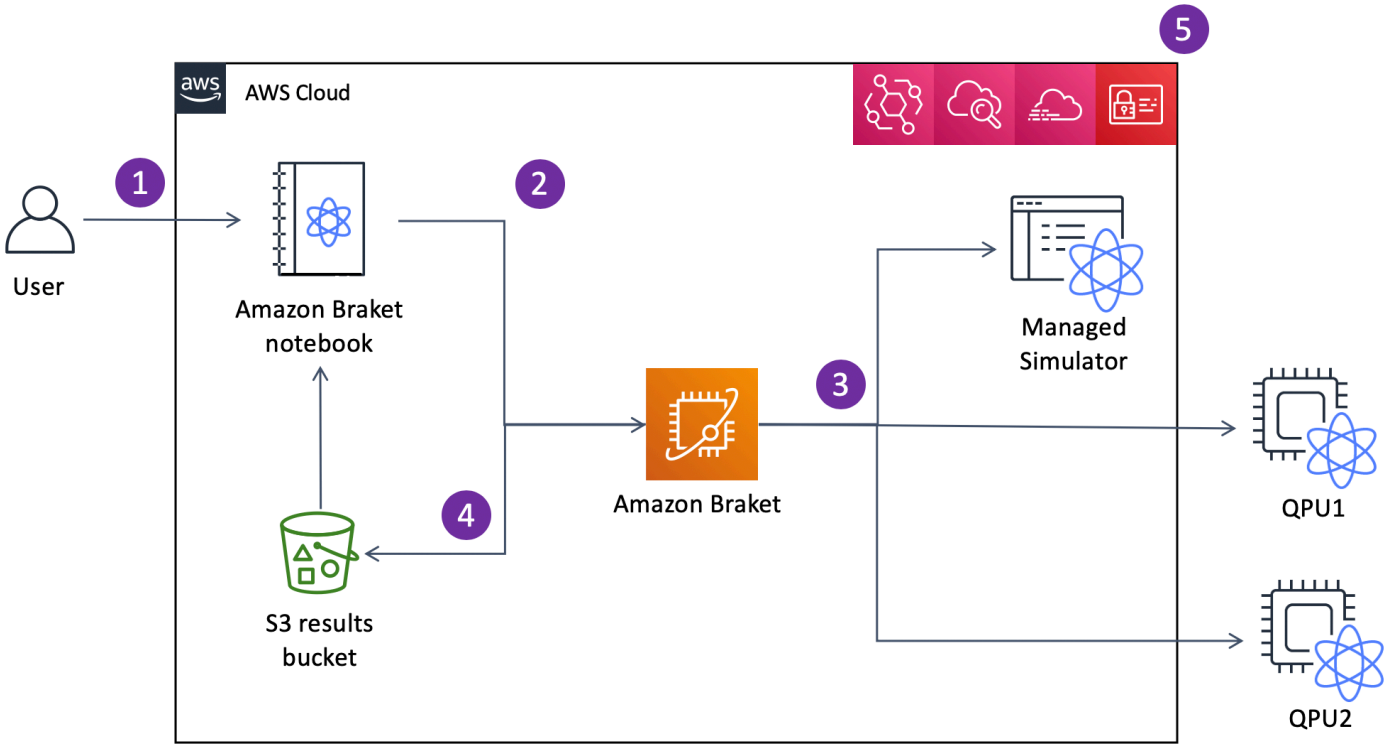
Amazon Braket 如何工作

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

Amazon Braket 提供量子運算裝置的按需存取，包括隨選電路模擬器和不同類型的 QPU。在 Amazon Braket 中，對設備的原子請求是一項量子任務。對於基於門的 QC 設備，此請求包括量子電路（包括測量指令和鏡頭數量）和其他請求元數據。對於模擬哈密頓模擬器，量子任務包含量子寄存器的物理佈局以及操縱字段的時間和空間依賴性。

在本節中，我們將了解在 Amazon Braket 上運行量子任務的高級流程。

Amazon Braket 量子任務流程



使用Jupyter筆記型電腦，您可以從 Amazon Braket 主控台或使用 Amazon Braket 開發套件方便地定義、提交和監控量子任務。您可以直接在 SDK 中構建量子電路。但是，對於類比哈密頓模擬器，您可

以定義暫存器配置和控制欄位。定義量子任務後，您可以選擇要在其上執行量子任務的裝置，並將其提交至 Amazon Braket API (2)。根據您選擇的設備，量子任務會排隊，直到設備變為可用，並將任務發送到 QPU 或模擬器進行實施（3）。Amazon Braket 可讓您存取不同類型的 QPU (IonQ、、、Rigetti) Oxford Quantum Circuits (OQC) QuEra、三個隨選模擬器 (、、) SV1、DM1兩個本機模擬器，以及一個內嵌模擬器。TN1若要進一步了解，請參閱 Amazon Braket 支援的裝置。

處理量子任務後，AmazonBraket 會將結果傳回 Amazon S3 儲存貯體，其中資料存放在您的 AWS 帳戶 (4) 中。同時，SDK 會在背景中輪詢結果，並在量子任務完成時將其載入 Jupyter 筆記本中。您還可以在 Braket 控制台的量子任務頁面上查看和管理您的量子任務，或者通過使用 Amazon Braket 的`GetQuantumTask`操作來查看和管理您的Amazon量子任務。API

AmazonBraket 與 AWS Identity and Access Management （IAM），Amazon AWS CloudTrail 和 Amazon CloudWatch 集成，用 EventBridge 於用戶訪問管理，監控和日誌記錄以及基於事件的處理（5）。

第三方資料處理

提交至 QPU 裝置的量子任務會在位於第三方供應商運營的設施中的量子電腦上進行處理。若要進一步了解 Amazon Braket 中的安全性和第三方處理，請參閱 Amazon Braket 硬體供應商的安全性。

布拉克特的核心存儲庫和插件

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

核心儲存庫

以下顯示核心儲存庫清單，其中包含用於 Braket 的金鑰套件：

- 開發套件-使用開發套件 Python 程式設計語言，在Jupyter筆記本上設定您的程式碼。Jupyter筆記型電腦設定完成後，您可以在 Braket 裝置和模擬器上執行程式碼
- 布拉克特架構-布拉克 SDK 和布拉克特服務之間的合同。
- Braket 默認模擬器-我們所有用於 Braket 的本地量子模擬器（狀態向量和密度矩陣）。

外掛程式

然後是與各種設備和編程工具一起使用的各種插件。這些措施包括 Braket 支持的插件以及由第三方支持的插件，如下圖所示。

Amazon Braket 支持：

- Amazon Braket 算法庫-用 Python 編寫的預構建量子算法的目錄。按原樣運行它們，或者使用它們作為起點來構建更複雜的算法。
- 布拉克特PennyLane 插件- 用PennyLane作布拉克特上的 QML 框架。

第三方（布拉克團隊監控和貢獻）：

- Qiskit 編碼器提供者-使用 Qiskit SDK 存取 Braket 資源。
- 剎車-朱莉婭 SDK-（實驗性）布拉克特 SDK 的朱莉婭本機版本

支持 Amazon Braket 設備

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

在 Amazon Braket 中，裝置代表您可以呼叫以執行量子任務的 QPU 或模擬器。Amazon Braket 提供三個隨選模擬器IonQIQM、Oxford Quantum Circuits三個本機模擬器和一個嵌入式模擬器Rigetti，以及一個內嵌模擬器存取 QPU 裝置。QuEra對於所有裝置，您可以在 Amazon Braket 主控台的「裝置」索引標籤上或透過 `GetDevice` API 找到其他裝置屬性，例如裝置拓撲、校準資料和原生閘門集。使用模擬器建構電路時，Amazon Braket 目前要求您使用連續的量子位元或索引。如果您使用 Amazon Braket 開發套件，則可以存取裝置屬性，如下列程式碼範例所示。

```
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
```

支援量子硬體供應商

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

支援的模擬器

- 本地狀態向量模擬器（braket_sv）（'默認模擬器'）
- 局部密度矩陣模擬器 (braket_dm)
- 本地 AHS 模擬器
- 狀態向量模擬器 (SV1)
- 密度矩陣模擬器 (DM1)
- 張量網絡模擬器（ ）TN1
- PennyLane的閃電模擬器

為您的量子任務選擇最佳模擬器

- 比較模擬器

> **Note**
>
> 若要檢視每個裝置 AWS 區域 的可用項目，請向右捲動下表。

Amazon Braket 設備

供應商	裝置名稱	范式	Type	設備 ARN	區域
IonQ	Aria 1	基於門的	QPU	arn: aws: 胸罩:美國東部 -1:: 設備/QPU/離子 / 詠嘆調 -1	us-east-1
IonQ	Aria 2	基於門的	QPU	ARN: aws: 胸罩:美國東部 -1:: 設備/QPU/離子/ 詠嘆調 -2	us-east-1
IonQ	Forte 1	基於門的	QPU (僅限預約)	arn: aws: 胸罩:美國東部 -1:: 設備/QPU /離子/ 堡壘 -1	us-east-1

供應商	裝置名稱	范式	Type	設備 ARN	區域
IonQ	Harmony	基於門的	QPU	arn: aws: 胸罩:美國東部 -1:: 設備/QPU/離子 / 和諧	us-east-1
IQM	Garnet	基於門的	QPU	手臂：AWS：胸罩：歐盟北部 -1：設備/QPU/IQM/ 石榴石	eu-north-1
Oxford Quantum Circuits	Lucy	基於門的	QPU	手臂:AWS: 胸罩:歐盟西部 -2:: 設備/QPU / OQC/ 露西	eu-west-2
QuEra	Aquila	模擬哈密頓模擬	QPU	ARN: aws: 胸罩:美國東部 -1:: 設備/QPU/查詢/拉拉	us-east-1
Rigetti	Aspen M-3	基於門的	QPU	arn: aws: 胸罩:美國西部 -1:: 設備/QPU /成/米-3	us-west-1
AWS	braket_sv	基於門的	本地模擬器	不適用（本地模擬器 SDK 中的本地模擬器）	N/A
AWS	braket_dm	基於門的	本地模擬器	不適用（本地模擬器 SDK 中的本地模擬器）	N/A
AWS	SV1	基於門的	按需模擬器	arn：awk：口罩：：設備/量子模擬器/亞馬遜/sv1	提供 Amazon 布拉克特的所有區域。

供應商	裝置名稱	范式	Type	設備 ARN	區域
AWS	DM1	基於門的	按需模擬器	arn：awk：口罩：：設備/量子模擬器/亞馬遜/dm1	提供 Amazon 布拉克特的所有區域。
AWS	TN1	基於門的	按需模擬器	arn：awk：口頭：：設備/量子模擬器/亞馬遜/TN1	us-west-2，us-east-1 和 eu-west-2

> **Note**
>
> 某些 QPU 只能透過 Braket Direct 進行預約，請參閱預約。

若要檢視可與 Braket 搭配使用之 QPU 的其他詳細資料，請參閱 Amazon Amazon Braket 硬體供應商。

IonQ

IonQ提供基於離子陷阱技術的閘門式 QPU。 IonQ's被困離子 QPU 建立在被困的 171Yb + 離子鏈上，這些離子通過真空室內的微製表面電極陷阱在空間上受到限制。

IonQ設備支持以下量子門。

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

透過逐字編譯，IonQQPU 支援下列原生閘道。

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

如果僅在使用本機 MS 門時指定兩個相位參數，則會執行完全糾纏的 MS 門。完全糾纏的 MS 門總是執行 /2 旋轉。若要指定不同的角度並執行部分糾纏的 MS 灌嘴，您可以透過加入第三個參數來指定所需的角度。如需詳細資訊，請參閱煞車模組。

這些本地門只能與逐字編譯一起使用。要了解有關逐字編譯的更多信息，請參閱逐字編譯。

IQM

IQM量子處理器是基於超導跨量子比特的通用和門模型設備。該IQM Garnet裝置是具有方形晶格拓撲的 20 量子位元裝置。

這些IQM裝置支援以下量子閘。

```
"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"
```

通過逐字編譯，IQM設備支持以下本機門。

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

Rigetti

Rigetti量子處理器是基於所有可調超導的通用門模型機器。qubits79 量子位元Aspen-M-3件利用其專有的多晶片技術，並由 2 個 40 量子位元處理器組裝而成。

該Rigetti設備支持以下量子門。

```
'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'
```

通過逐字編譯，Rigettti 設備支持以下本機門。

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

Rigetti超導量子處理器只能在 ±/2 或 ±π 的角度下運行「rx」門。

設備上可以使用脈衝級控制，該設Rigetti備支持以下類型的一組預定義幀：

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

如需這些框架的詳細資訊，請參閱框架和連接埠的角色。

Oxford Quantum Circuits (OQC)

OQC量子處理器是通用的門模型機器，使用可擴展的 Coaxmon 技術構建。該OQC Lucy系統是一種8-qubit具有環的拓撲，其中每個環qubit被連接到其兩個最近的鄰居的設備。

該Lucy設備支持以下量子門。

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

通過逐字編譯，OQC 設備支持以下本地門。

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

設備上可以OQC使用脈衝級別控制。這些OQC裝置支援下列類型的一組預先定義框架：

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

OQC設備支持幀的動態聲明，前提是您提供了一個有效的端口標識符。如需這些框架和連接埠的詳細資訊，請參閱框架和連接埠的角色。

> **Note**
>
> 搭配使用脈衝控制時OQC，程式的長度不能超過 90 微秒。單量子位門的最大持續時間約為 50 納秒，雙量子位門的最大持續時間約為 1 微秒。這些數字可能會因使用的量子位元、裝置的電流校正以及電路編譯而有所不同。

QuEra

QuEra提供基於中原子的設備，可以運行模擬哈密頓仿真（AHS）量子任務。這些特殊用途設備忠實地重現了數百個同時交互量子位的時間依賴的量子動態。

人們可以通過規定量子比特寄存器的佈局以及操縱字段的時間和空間依賴性來編程這些設備在模擬哈密頓仿真的範例中。Amazon Braket 提供了通過 python SDK 的 AHS 模塊構建此類程序的實用程序。`braket.ahs`

如需詳細資訊，請參閱類比哈密頓模擬範例筆記本或使用的 Aquila 提交類比程式 QuEra頁面。

本地狀態向量模擬器 (`braket_sv`)

本機狀態向量模擬器 (`braket_sv`) 是在您環境中本機執行的 Amazon Braket SDK 的一部分。根據 Braket 筆記型電腦執行個體或本機環境的硬體規格，它非常適合在小型電路上快速進行原型製作 (最多 25 個qubits)。

本地模擬器支持 Amazon Braket SDK 中的所有門，但 QPU 設備支持較小的子集。您可以在設備屬性中找到設備的支持門。

> **Note**
>
> 本機模擬器支援進階 OpenQASM 功能，QPU 裝置或其他模擬器可能不支援這些功能。如需支援功能的詳細資訊，請參閱 OpenQASM 本機模擬器筆記本中提供的範例。

如需有關如何使用模擬器的詳細資訊，請參閱 Amazon Braket 範例。

局部密度矩陣模擬器 (`braket_dm`)

本機密度矩陣模擬器 (`braket_dm`) 是在您環境中本機執行的 Amazon Braket SDK 的一部分。根據 Braket 筆記型電腦執行個體或本機環境的硬體規格，它非常適合在具有雜訊 (最多 12 個qubits) 的小型電路上進行快速原型製作。

您可以使用閘極雜訊操作 (例如位元翻轉和去極化誤差)，從頭開始建立常見的雜訊電路。您也可以將雜訊操作套用至既有電路的特定電路qubits和閘極，這些電路旨在同時執行與無雜訊。

本`braket_dm`地模擬器可以提供以下結果，給定的指定數量shots：

- 降低密度矩陣：Shots= 0

> **Note**
>
> 本機模擬器支援進階 OpenQASM 功能，QPU 裝置或其他模擬器可能不支援此功能。如需有關支援功能的詳細資訊，請參閱 OpenQASM 本機模擬器筆記本中提供的範例。

要了解有關局部密度矩陣模擬器的更多信息，請參閱 Braket 介紹噪聲模擬器示例。

本地 AHS 模擬器（ ）braket_ahs

本機 AHS (類比哈密頓模擬) 模擬器 (braket_ahs) 是 Amazon Braket SDK 的一部分，可在您的環境中本機執行。它可以用來模擬來自 AHS 程序的結果。它非常適合在小型寄存器（最多 10-12 個原子）上進行原型製作，具體取決於 Braket 筆記本實例或本地環境的硬件規格。

本地模擬器支持具有一個統一駕駛場，一個（非均勻）移位場和任意原子排列的 AHS 程序。有關詳細信息，請參閱布拉科特 AHS 類和布拉克特 A HS 程序模式。

要了解有關本地 AHS 模擬器的更多信息，請參閱 Hello AHS：運行您的第一個模擬哈密頓模擬頁面和模擬哈密頓仿真示例筆記本。

狀態向量模擬器 (SV1)

SV1是一個按需，高性能，通用狀態矢量模擬器。它可以模擬多達 34 的電路qubits。根據所使用的34-qubit灌嘴類型和其他因素，您可以預期密集和方形電路（電路深度 = 34）大約需要 1-2 小時才能完成。帶有 all-to-all 閘極的電路非常適合SV1。它會以完整狀態向量或振幅陣列等形式傳回結果。

SV1最長執行時間為 6 小時。它具有 35 個並發量子任務的默認值，最多 100 個（美西 -1 和 eu-west-2 中為 50）並發量子任務。

SV1結果

SV1可以提供以下結果，給定的指定數目shots：

- 樣本：Shots> 0
- 期望值：Shots> = 0
- 差異數：Shots> = 0
- 機率：Shots> 0
- 振幅：Shots= 0

- 伴隨漸變：Shots= 0

如需結果的詳細資訊，請參閱結果類型。

SV1它始終可用，它可以按需運行電路，並且可以並行運 parallel 多個電路。執行階段會隨作業數目線性縮放，並以的數目呈指數級。qubits的數目shots對執行階段的影響很小。要了解更多信息，請訪問比較模擬器。

模擬器支持 Braket SDK 中的所有門，但 QPU 設備支持較小的子集。您可以在設備屬性中找到設備的支持門。

密度矩陣模擬器 (DM1)

DM1是一個按需，高性能，密度矩陣模擬器。它可以模擬多達 17 的電路qubits。

DM1最長執行時間為 6 小時，預設值為 35 個並行量子任務，最多 50 個並行量子任務。

DM1結果

DM1可以提供以下結果，給定的指定數目shots：

- 樣本：Shots> 0
- 期望值：Shots> = 0
- 差異數：Shots> = 0
- 機率：Shots> 0
- 減少的密度矩陣：Shots= 0，最多 8 qubits

如需有關結果的詳細資訊，請參閱結果類型。

DM1它始終可用，它可以按需運行電路，並且可以並行運 parallel 多個電路。執行階段會隨作業數目線性縮放，並以的數目呈指數級。qubits的數目shots對執行階段的影響很小。若要深入瞭解，請參閱比較模擬器。

噪音閘門和限制

```
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]
```

張量網絡模擬器（ ）TN1

TN1是一個按需，高性能，張量網絡模擬器。 TN1可以模擬特定電路類型，qubits其電路深度為 1,000 或更小。 TN1對於稀疏電路，具有局部閘極的電路以及其他具有特殊結構的電路（例如量子傅立葉變換（QFT）電路尤其強大。 TN1分兩個階段運作。首先，排練階段嘗試識別電路的有效計算路徑，因此TN1可以估計下一階段的運行時間，這稱為收縮階段。如果預估的收縮時間超過TN1模擬執行時間限制，則TN1不會嘗試收縮。

TN1執行時間限制為 6 小時。它最多限制為 10（eu-west-2 中的 5 個）並發量子任務。

TN1結果

收縮階段由一系列矩陣乘法組成。一系列乘法會繼續進行，直到達到結果或判斷無法達到結果為止。

注意：Shots必須 > 0。

結果類型包括：

- 樣本
- 期望
- 變異數

如需結果的詳細資訊，請參閱結果類型。

TN1它始終可用，它可以按需運行電路，並且可以並行運 parallel 多個電路。若要深入瞭解，請參閱比較模擬器。

模擬器支持 Braket SDK 中的所有門，但 QPU 設備支持較小的子集。您可以在設備屬性中找到設備的支持門。

請造訪 Amazon Braket GitHub 儲存庫以取得 TN1 範例筆記本，以協助您開始使用。TN1

使用的最佳做法 TN1

- 避免 all-to-all 電路。
- 用少量的測試新電路或類電路shots，以了解電路的「硬度」TN1。
- 將大型shot模擬分為多個量子任務。

嵌入式模擬器

嵌入式模擬器的工作原理是將演算法程式碼嵌入到同一個容器中，並直接在混合作業實例上執行模擬。這對於消除與模擬與遠程設備進行通信相關的瓶頸非常有用。這可能會大幅降低記憶體使用量、減少電路執行次數以達到預期結果，以及提升十倍以上的效能。如需有關內嵌模擬器的詳細資訊，請參閱使用 Amazon Braket 混合任務執行混合任務頁面。

PennyLane的閃電模擬器

您可以將閃 PennyLane電模擬器用作 Braket 上的嵌入式模擬器。使用 PennyLane的閃電模擬器，您可以利用先進的漸變計算方法，例如伴隨差異化，更快地評估漸變。閃電量子模擬器可通過 Braket NBI 作為設備使用，並作為嵌入式模擬器使用，而閃電 .gpu 模擬器需要作為帶有 GPU 實例的嵌入式模擬器運行。有關使用 lightning.gpu 的示例，請參閱 Braket 混合式工作筆記本中的嵌入式模擬器。

比較模擬器

本節通過描述一些概念，限制和用例來幫助您選擇最適合您的量子任務的 Amazon Braket 模擬器。

在本機模擬器和隨選模擬器之間進行選擇 (SV1、、TN1) DM1

本機模擬器的效能取決於主控本機環境的硬體，例如用來執行模擬器的 Braket 筆記本執行個體。隨選模擬器在 AWS 雲端中執行，其設計目的是超越典型的本機環境。按需模擬器針對較大的電路進行了優

化，但是每個量子任務或批次量子任務會增加一些延遲開銷。如果涉及許多量子任務，這可能意味著取捨。鑑於這些一般性能特性，以下指南可以幫助您選擇如何運行模擬，包括具有噪音的模擬。

對於模擬：

- 使用少於 18 時qubits，請使用本地模擬器。
- 使用 18—24 時qubits，請根據工作負載選擇模擬器。
- 當僱用超過 24 個時qubits，請使用按需模擬器。

對於噪音模擬：

- 使用少於 9 時qubits，請使用本地模擬器。
- 使用 9—12 時qubits，請根據工作負載選擇模擬器。
- 當僱用超過 12qubits, 使用DM1.

什麼是狀態向量模擬器？

SV1是一個通用狀態向量模擬器。它存儲量子狀態的全波函數，並依次將門操作應用於狀態。它存儲了所有可能性，即使是極不可能的可能性。SV1模擬器的量子任務運行時間隨著電路中的門數而線性增加。

什麼是密度矩陣模擬器？

DM1用噪聲模擬量子電路。它儲存系統的完整密度矩陣，並依序套用電路的閘極和雜訊操作。最終密度矩陣包含有關電路運行後量子狀態的完整信息。執行階段通常會隨著作業數目線性縮放，並以指數的數目進行縮放。qubits

什麼是張量網絡模擬器？

TN1將量子電路編碼成結構化圖形。

- 圖形的節點由量子門組成，或qubits。
- 圖形的邊緣代表灌嘴之間的連接。

由於這種結構，TN1可以找到相對較大和複雜的量子電路的模擬解決方案。

TN1需要兩個階段

通常情況下，TN1在兩相的方法來模擬量子計算工作。

- 排練階段：在此階段，提出了一種以TN1有效的方式遍歷圖形的方法，其中涉及訪問每個節點，以便您可以獲得所需的測量結果。作為客戶，您看不到此階段，因為您會一起TN1執行這兩個階段。它完成了第一階段，並確定是否根據實際約束自己執行第二階段。模擬開始後，您對該決定沒有任何輸入。
- 收縮階段：此階段類似於傳統計算機中計算的執行階段。該階段由一系列矩陣乘法組成。這些乘法的順序對計算難度有很大的影響。因此，排練階段首先完成，以便在圖中找到最有效的計算路徑。在排練階段找到收縮路徑後，將電路的門收TN1合在一起，以產生模擬結果。

TN1圖表類似於地圖

從比喻上講，您可以將底層TN1圖表與城市的街道進行比較。在具有計劃網格的城市中，使用地圖很容易找到到達目的地的路線。在沒有規劃的街道，街道名稱重複等的城市中，通過查看地圖可能很難找到到達目的地的路線。

如果TN1沒有進行排練階段，那就像在城市的街道上漫步以找到您的目的地，而不是先看地圖。它真的可以在步行時間方面得到回報，花更多的時間看地圖。同樣，排練階段提供了有價值的信息。

您可能會說，對其所遍歷的基礎電路的結構TN1具有一定的「認知」。它在排練階段獲得了這種意識。

最適合於每種類型的模擬器的問題類型

SV1非常適合主要依賴於具有特定數量qubits和閘門的任何類別的問題。通常，所需的時間隨著門的數量呈線性增長，而不取決於的shots數量。 SV1通常比 28 歲以下TN1的電路快qubits。

SV1對於較高的qubit數字，可能會變慢，因為它實際上模擬了所有可能性，即使是極不可能的可能性。它沒有辦法確定哪些結果是可能的。因此，對於30-qubit評估，SV1必須計算 2^30 配置。由於內存和存儲限制，AmazonBraket SV1 模擬器的 34 qubits 限制是一個實際限制。您可以這樣想：每次添加qubit到時SV1，問題都會變得困難的兩倍。

對於許多類別的問題，TN1可以在現實時間內評估比SV1利用圖形結構的優勢TN1要大得多的電路。它基本上從起始位置跟踪解決方案的演變，並且只保留有助於高效遍歷的配置。換句話說，它保存了配置以創建矩陣乘法的順序，從而導致更簡單的評估過程。

對於TN1，門的數量qubits和門很重要，但圖形的結構更重要。例如，非TN1常擅長評估電路（圖形），其中門是短距離的（即，每個qubit門僅通過門連接到最近的鄰居）和電路（圖形qubits），其中連接（或門）具有相似範圍的電路（圖形）。一個典型的範圍TN1是每次qubit通話只能與其qubits他人 5 qubits 離開。如果大部分結構都可以分解為更簡單的關係，例如這些關係，這些關係可以用更多，更小或更均勻的矩陣來表示，則很容易TN1執行評估。

的局限性 TN1

TN1可能會比SV1取決於圖形的結構複雜性慢。針對特定圖形，由TN1於下列兩個原因之一，會在排練階段之後終止模擬FAILED，並顯示狀態：

- 找不到路徑-如果圖形太複雜，則很難找到一個好的遍歷路徑，並且模擬器放棄了計算。 TN1不能執行收縮。您可能會看到類似下列的錯誤訊息：`No viable contraction path found.`
- 收縮階段太困難了 — 在某些圖表中，TN1可以找到遍歷路徑，但是評估非常長且非常耗時。在這種情況下，收縮是如此昂貴，以至於成本會令人望而卻步，而是在排練階段之後TN1退出。您可能會看到類似下列的錯誤訊息：`Predicted runtime based on best contraction path found exceeds TN1 limit.`

> **Note**
>
> 即使未執行收縮且您看到狀態，您TN1仍需支付的排練階段的費用FAILED。

預測的運行時也取決於shot計數。在最壞情況下，TN1收縮時間線性取決於計數。shot該電路可以用更少shots的收縮。例如，您可以提交 100 的量子任務shots，該任務TN1決定是不可收縮的，但是如果您僅重新提交 10，則收縮將繼續。在這種情況下，要獲得 100 個樣本，您可以shots為同一電路提交 10 個量子任務，並最終合併結果。

作為最佳實踐，我們建議您在繼續進行較大數量的電路之前，始終使用幾個shots（例如 10）測試電路或電路等級TN1，以了解電路的硬度shots。

> **Note**
>
> 形成收縮階段的一系列乘法從小的 nxN 矩陣開始。例如，2-qubit閘門需要 4x4 矩陣。在被判定為太困難的收縮期間所需的中間矩陣是巨大的。這樣的計算需要幾天才能完成。這就是為什麼 Amazon Braket 不會嘗試極其複雜的收縮。

並行數量

所有 Braket 模擬器使您能夠同時運行多個電路。並發限制因模擬器和區域而異。如需並行限制的詳細資訊，請參閱配額頁面。

Amazon Braket 區域和端點

Amazon Braket 提供以下產品： AWS 區域

Amazon Braket 區域的可用性

區域名稱	區域	布拉基端點	QPU
美國東部 (維吉尼亞北部)	us-east-1	布拉克. 美國東部-亞馬遜	IonQ
美國東部 (維吉尼亞北部)	us-east-1	布拉克. 美國東部-亞馬遜	QuEra
美國西部 (加利佛尼亞北部)	us-west-1	布拉克. 美國西部-亞馬遜	里格蒂
歐洲北部 1 (斯德哥爾摩)	eu-north-1	布拉克. 歐盟北部 1. 亞馬遜	IQM
歐洲西部 2 (倫敦)	eu-west-2	布拉克. 歐洲西部-亞馬遜	OQC

您可以從任何可用的區域執行 Amazon Braket，但每個 QPU 僅在單一區域提供。您可以在該裝置區域的 Amazon Braket 主控台中檢視在 QPU 裝置上執行的量子任務。如果您使用的是 Amazon Braket 開發套件，您可以將量子任務提交到任何 QPU 裝置，而不論您所在的區域為何。SDK 會自動為指定的 QPU 建立「區域」的工作階段。

如需有關如何使 AWS 用區域和端點的一般資訊，請參閱AWS 一般參考中的AWS 服務 端點。

我的量子任務何時運行？

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

當您提交電路時，Amazon Braket 會將電路傳送到您指定的裝置。量子處理單元（QPU）和按需模擬器量子任務排隊並按照它們接收的順序進行處理。提交量子任務後，處理量子任務所需的時間會因其他 Amazon Braket 客戶提交的任務數量和複雜性以及所選 QPU 的可用性而有所不同。

電子郵件或簡訊中的狀態變更通知

EventBridge 當 QPU 的可用性發生變化或量子任務的狀態發生變化時，Amazon Braket 會將事件傳送給 Amazon。請依照下列步驟透過電子郵件或 SMS 訊息接收裝置和量子任務狀態變更通知：

1. 建立 Amazon SNS 主題以及電子郵件或簡訊的訂閱。電子郵件或簡訊的可用性取決於您所在地區。如需詳細資訊，請參閱開始使用 Amazon SNS 和傳送簡訊。
2. 在中建立 EventBridge 可觸發 SNS 主題通知的規則。有關更多信息，請參閱使用 Amazon 監控亞馬遜布拉基特。 EventBridge

量子任務完成警報

您可以透過 Amazon Simple Notification Service (SNS) 設定通知，以便在 Amazon Braket 量子任務完成時收到警示。如果您預期等待較長的時間 (例如，當您提交大型工作或在裝置的可用性視窗外提交工作)，作用中通知就很有用。如果您不想等待工作完成，您可以設定 SNS 通知。

Amazon Braket 筆記型電腦會引導您完成設定步驟。如需詳細資訊，請參閱 Amazon Braket 範例筆記本以設定通知。

QPU 可用性視窗與狀態

QPU 可用性因裝置而異。

在 Amazon Braket 主控台的「裝置」頁面中，您可以查看目前和即將推出的可用性視窗和裝置狀態。此外，每個裝置頁面都會顯示量子任務和混合任務的個別佇列深度。

如果客戶無法使用裝置，則無論可用性時段為何，裝置都會被視為離線。例如，由於排程的維護、升級或操作問題，它可能處於離線狀態。

佇列可見度

在提交量子任務或混合任務之前，您可以通過檢查設備隊列深度來查看面前有多少量子任務或混合任務。

佇列深度

Queue depth指特定裝置排入佇列的量子工作和混合式工作的數量。裝置的量子工作和混合式工作佇列計數可透過Braket Software Development Kit (SDK)或存取Amazon Braket Management Console。

1. 任務隊列深度是指當前以正常優先級運行的量子任務總數。

2. 優先級任務隊列深度是指等待運行的提交量子任務的總數Amazon Braket Hybrid Jobs。這些工作會在獨立工作之前執行。
3. 混合式工作佇列深度是指目前在裝置上排入佇列的混合式作業總數。 Quantum tasks作為混合式工作的一部分提交具有優先順序，且會在中彙總Priority Task Queue。

希望透過檢視佇列深度的客戶Braket SDK可以修改下列程式碼片段，以取得其量子工作或混合工作的佇列位置：

```
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'
```

將量子任務或混合任務提交到 QPU 可能會導致工作負載處於某個QUEUED狀態。Amazon Braket 可讓客戶瞭解其量子任務和混合任務佇列位置。

佇列位置

Queue position指量子任務或混合任務在各自設備隊列中的當前位置。它可以通過或獲得量子任務或混合作業Amazon Braket Management Console。Braket Software Development Kit (SDK)

希望透過檢視佇列位置的客戶Braket SDK可以修改下列程式碼片段，以取得其量子工作或混合工作的佇列位置：

```
# 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
```

開始使用 Amazon Braket

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

按照「啟用 Amazon Braket」中的說明進行操作後，您就可以開始使用 Amazon Braket。

開始使用的步驟包括：

- 啟用 Amazon Braket
- 創建一個亞馬遜電腦筆記本實例
- 使用 Amazon Braket Python SDK 運行您的第一個電路
- 運行您的第一個量子算法

啟用 Amazon Braket

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

您可以透過AWS 主控台在您的帳戶中啟用 Amazon Braket。

必要條件

若要啟用和執行 Amazon Braket，您必須擁有具有啟動 Amazon Braket 動作權限的使用者或角色。這些權限包含在 AmazonBraketFullAccess IAM 政策中 (ARN: aw:iam:: aws: 策略/)。AmazonBraketFullAccess

> **Note**
>
> 如果您是管理員：

若要讓其他使用者存取 Amazon Braket，請透過附加AmazonBraketFullAccess政策或附加您建立的自訂政策來授予使用者許可。若要進一步了解使用 Amazon Braket 所需的許可，請參閱管理 Amazon Braket 的存取權限。

啟用 Amazon Braket 的步驟

1. 使 Amazon Braket 的 AWS 帳戶.
2. 打開 Amazon Braket 控制台。
3. 在 Braket 登陸頁面中，按一下「開始使用」以前往「服務儀表板」頁面。服務儀表板頂端的警示會引導您完成以下三個步驟：
   a. 建立服務連結角色 (SLR)
   b. 啟用對第三方量子電腦的存取
   c. 建立新的 Jupyter 筆記本執行個體

為了使用第三方量子設備，您需要同意您自己和這些設備之間有關數據傳輸的某些條件。 AWS本協議的條款和條件在 Amazon Braket 主控台的「許可和設定」頁面的「一般」索引標籤上提供。

> **Note**
>
> 不涉及任何第三方的量子設備，例如 Braket 本地模擬器或按需模擬器，可以在不同意啟用第三方設備協議的情況下使用。
> 如果您正在訪問第三方硬件，則每個帳戶只需要接受這些條款以使用第三方設備一次。

創建一個亞馬遜電腦筆記本實例

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

Amazon Braket 提供全受管的 Jupyter 筆記本，協助您開始使用。Amazon Braket 筆記本執行個體以 Amazon SageMaker 筆記本執行個體為基礎。下列指示概述為新客戶和現有客戶建立新筆記本執行個體所需的步驟。

Amazon Braket 新客戶

1. 開啟 Amazon Braket 主控台，然後導覽至左窗格中的「儀表板」頁面。
2. 在儀表板頁面中央的「歡迎使用 Amazon Braket」模式中按一下「開始使用」，以提供筆記本名稱。這將創建一個默認的 Jupyter 筆記本。
3. 建立您的筆記本可能需要幾分鐘的時間。您的筆記本會列在 [記事本] 頁面上，狀態為 [擱置中]。當您的筆記本執行個體可供使用時，狀態會變更為InService。您可能需要重新整理頁面，才能顯示筆記本的更新狀態。

亞馬遜現有客戶

1. 開啟 Amazon Braket 主控台，在左窗格中選取 [筆記本]，選擇 [建立筆記本執行個體]。如果您沒有記事本，請選取 [標準] 設定以建立預設 Jupyter 記事本，並輸入僅使用英數字元和連字號字元的 Notebook 執行個體名稱，然後選取您偏好的視覺模式。然後，啟用或禁用筆記本的不活動管理器。
   a. 如果啟用，請在重設筆記本之前選取所需的閒置持續時間。重設筆記型電腦時，運算費用將不再產生，但儲存費用仍會繼續。
   b. 若要檢視筆記本執行個體中剩餘的閒置時間，請瀏覽至指令列並選取「Braket」標籤，然後選取「無活動管理員」標籤。

> **Note**
>
> 為了避免您的工作丟失，請考慮將您的 SageMaker 筆記本實例與 git 存儲庫集成在一起。或者，將您的工作移到`/Braket Algorithms`和`/Braket Examples`資料夾之外，可防止重新啟動筆記本執行個體而覆寫檔案。

2. (選擇性) 使用進階設定，您可以建立具有存取權限、其他組態和網路存取設定的筆記本：
   a. 在筆記本配置中選擇您的實例類型。依預設，會選擇標準且符合成本效益的執行個體類型 ml.t3.medium。若要進一步了解執行個體定價，請參閱 Amazon SageMaker 定價。如果您想要將公共 Github 儲存庫與筆記本執行個體建立關聯，請按一下「Git 儲存庫」下拉式清單，然後從「儲存庫」下拉式選單中選取「從 url 複製公用 git 儲存庫」。在 Git 儲存庫網址文字列中輸入存放庫的網址。
   b. 在許可中，設定任何選用的 IAM 角色、根存取權和加密金鑰。
   c. 在 [網路] 中，為您的執行個體設定自訂網路和存取設Jupyter Notebook定。
3. 檢閱您的設定、設定任何標籤以識別您的筆記本執行個體，然後按一下啟動。

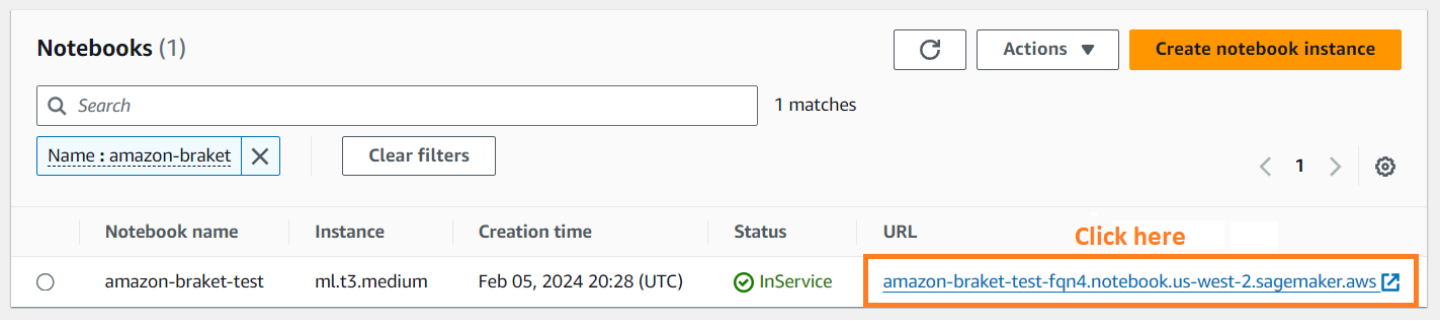
> **Note**
>
> 您可以在 Amazon Braket 和亞馬遜控制台中查看和管理您的 Amazon Braket 筆記本執行個體。 SageMaker 您可透過SageMaker 主控台取得其他 Amazon Braket 筆記型電腦設定。

如果您在 Amazon Braket 開發套件中的 Amazon Braket 主控台 AWS 中工作，且外掛程式已預先載入您建立的筆記本中。如果您想要在自己的電腦上執行，您可以在執行指令`pip install amazon-braket-sdk`或執行指令搭配外掛程式使用時，安裝 SDK 和 PennyLane 外掛程式。`pip install amazon-braket-pennylane-plugin`

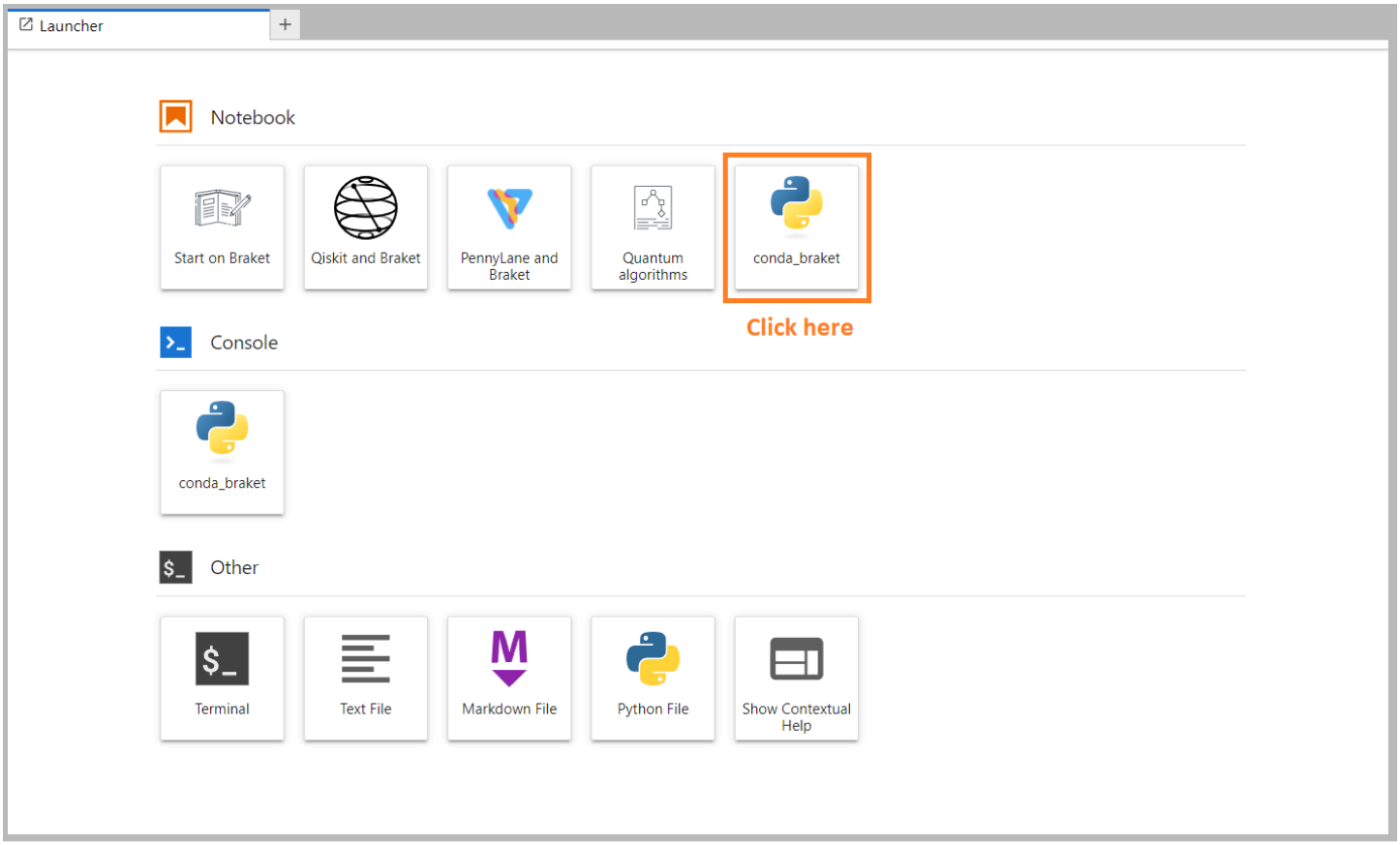
使用 Amazon Braket Python SDK 運行您的第一個電路

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

在您的筆記本執行個體啟動之後，請選擇您剛建立的筆記本，以標準 Jupyter 介面開啟執行個體。



AmazonBraket 筆記型電腦執行個體已預先安裝在 Amazon Braket SDK 及其所有相依性中。首先使用`conda_braket`內核創建一個新的筆記本。



你可以從一個簡單的「你好，世界！」 例子。首先，構建一個準備貝爾狀態的電路，然後在不同的設備上運行該電路以獲得結果。

首先導入 Amazon Braket SDK 模塊並定義一個簡單的貝爾狀態電路。

```
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)
```

您可以使用以下指令將電路視覺化：

```
print(bell)
```

在本地模擬器上運行電路

接下來，選擇要在其上運行電路的量子裝置。AmazonBraket SDK 隨附本地模擬器，用於快速原型製作和測試。我們建議將本地模擬器用於較小的電路，最多可達 25 個qubits（取決於您的本地硬件）。

以下是實例化本地模擬器的方法：

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

並運行電路：

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

你應該看到這樣的結果：

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

您準備的特定貝爾狀態是 |00> 和 |11> 的相等疊加，您會發現大致相等（最多shot噪音）分佈 00 和 11 作為測量結果，如預期。

在按需模擬器上運行電路

AmazonBraket 還可以訪問按需，高性能模擬器SV1，用於運行更大的電路。 SV1是一個按需狀態向量模擬器，允許模擬多達 34 個量子電路。qubits您可以在「支援的SV1裝置」區段和 AWS 主控台中找到更多資訊。在 SV1 (TN1或任何 QPU) 上執行量子任務時，量子任務的結果會存放在帳戶的 S3 儲存貯體中。如果您沒有指定值區，Braket SDK 會`amazon-braket-{region}-{accountID}`為您建立預設值區。若要進一步了解，請參閱管理 Amazon Braket 的存取權限。

> **Note**
>
> 填入您實際的現有值區名稱，其中下列範例顯示`example-bucket`為您的值區名稱。AmazonBraket 的值區名稱始終以您新增的`amazon-braket-`其他識別字元開頭。如果您需要有關如何設定 S3 儲存貯體的資訊，請參閱開始使用 Amazon S3。

```
# get the account ID
aws_account_id = boto3.client("sts").get_caller_identity()["Account"]
```

```
# the name of the bucket
my_bucket = "example-bucket"
# the name of the folder in the bucket
my_prefix = "simulation-output"
s3_folder = (my_bucket, my_prefix)
```

若要執行電路SV1，您必須提供先前選取的 S3 儲存貯體作為.run()呼叫中的位置引數的位置。

```
# 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)
```

AmazonBraket 控制台提供有關您的量子任務的更多信息。導航到控制台中的「量子任務」選項卡，您的量子任務應該位於列表的頂部。或者，您可以使用唯一的量子任務 ID 或其他條件搜索量子任務。

> **Note**
>
> 在 90 天后，AmazonBraket 會自動刪除與您的量子任務相關的所有量子任務 ID 和其他元數據。如需詳細資訊，請參閱資料保留。

在 QPU 上執行

使用 Amazon Braket，只要變更一行程式碼，就可以在實體量子電腦上執行先前的量子電路範例。Amazon Braket 提供對IonQ、、Oxford Quantum Circuits和QPURigetti裝置的存取。QuEra您可以在「支援的裝置」區段和 AWS 主控台的「裝置」索引標籤下，找到有關不同裝置和可用性視窗的資訊。下面的例子演示了如何實例化一個Rigetti設備。

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

選擇具有此代碼的IonQ設備：

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

選取裝置之後，在執行工作負載之前，您可以使用下列程式碼查詢裝置佇列深度，以判斷量子工作或混合作業的數量。此外，客戶可以在的 [裝置] 頁面上檢視裝置特定的佇列深度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
```

當您執行任務時，Amazon Braket 開發套件會輪詢結果 (預設逾時為 5 天)。您可以透過修改`.run()`指令中的`poll_timeout_seconds`參數來變更此預設值，如下列範例所示。請記住，如果輪詢逾時太短，則可能不會在輪詢時間內傳回結果，例如當 QPU 無法使用且傳回本機逾時錯誤時。您可以透過呼叫`task.result()`函式來重新啟動輪詢。

```
# 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)
```

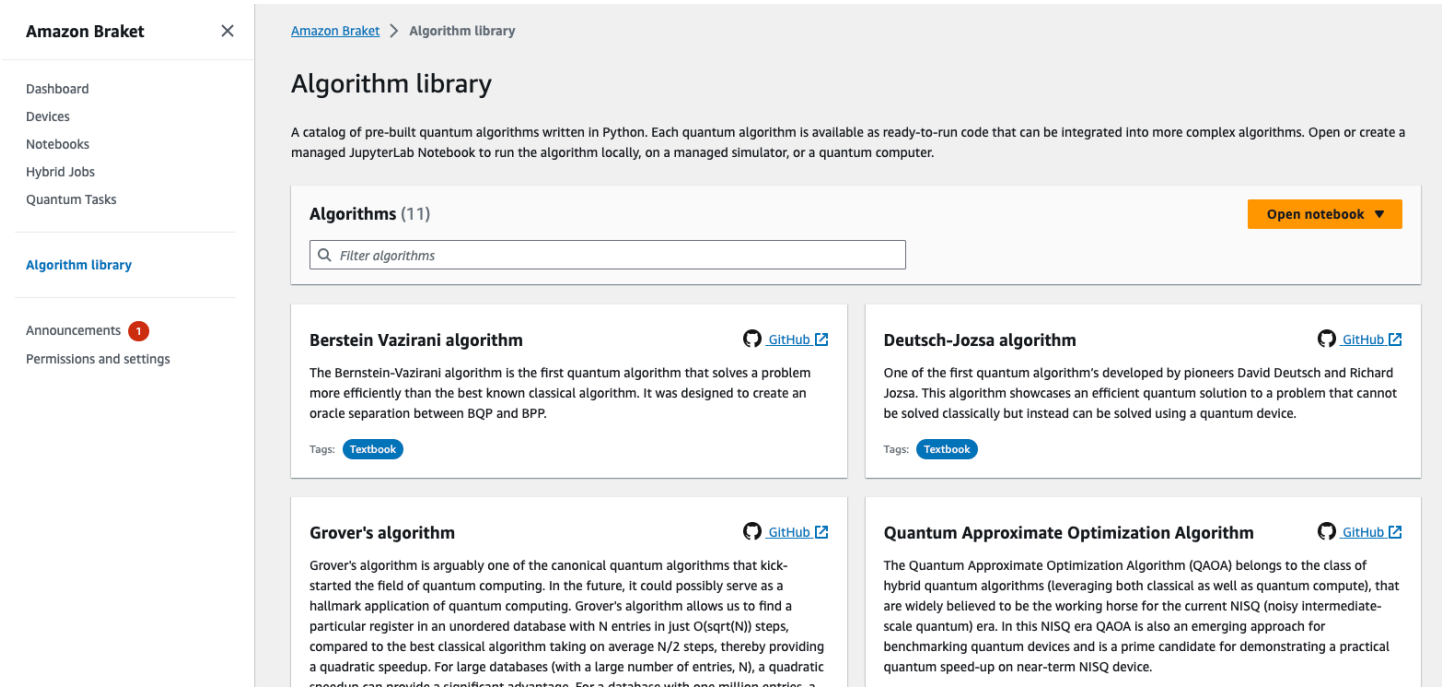
此外，提交量子任務或混合任務後，您可以調用該`queue_position()`函數來檢查隊列位置。

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

運行您的第一個量子算法

> **Tip**
>
> 了解量子計算的基礎 AWS！註冊 Amazon Braket 數位學習計劃，並在完成一系列學習課程和數位評估後獲得自己的數位徽章。

Amazon Braket 算法庫是一個用 Python 編寫的預構建量子算法的目錄。您可以直接執行這些演算法，也可以使用這些演算法做為起點來建置更複雜的演算法。您可以從 Braket 控制台訪問算法庫。您也可以訪問布拉克特算法庫在 Github 上:https://github.com/aws-samples/ amazon-braket-algorithm-library.



Braket 控制台提供演算法庫中每個可用演算法的描述。選擇 GitHub 連結以查看每個演算法的詳細資訊，或選擇 [開啟記事本] 開啟或建立包含所有可用演算法的記事本。如果您選擇筆記本選項，則可以在筆記本的根資料夾中找到 Braket 演算法庫。

與 Amazon Braket 工作

本節說明如何使用 Amazon Braket SDK 設計量子電路、將這些問題作為量子任務提交到裝置，以及如何監控量子任務。

以下是在 Amazon Braket 上與資源互動的主要方法。

- Amazon Braket 主控台提供裝置資訊和狀態，協助您建立、管理和監控資源和量子任務。
- 透過 Amazon Braket Python 開發套件以及透過主控台提交和執行量子任務。SDK 可透過預先設定的筆記型電Amazon腦存取。
- Amazon Braket API 可以通過 Python 開發套件和筆記本訪問。AmazonAPI如果您要建置以程式設計方式搭配量子運算的應用程式，則可以直接呼叫。

本節中的範例將示範如何API直接使用 Amazon Braket Python SDK 以及用於Amazon胸針的 Python SDK (Boto3) 來使用編輯器。AWS

有關 Python SDK 的Amazon更多信息

若要使用Amazon編輯器 Python SDK，請先安裝 AWS Python SDK，以便您可以與. AWS API 您可以將 Amazon Braket Python SDK 視為適用於量子客戶的 Boto3 周圍的便捷包裝器。

- Boto3 包含您需要進入的接口。 AWS API（請注意，Boto3 是一個大型的 Python SDK，可以與. AWS API 大多數 AWS 服務 支持 Boto3 接口。 ）
- AmazonBraket Python SDK 包含用於電路，門，設備，結果類型和量子任務其他部分的軟件模塊。每次創建程序時，都會導入該量子任務所需的模塊。
- AmazonBraket Python SDK 可以通過筆記本訪問，筆記本電腦預先加載了運行量子任務所需的所有模塊和依賴項。
- 如果您不希望使用筆記Amazon本，您可以將模塊從 Python SDK 導入到任何 Python 腳本中。

安裝 Boto3 之後，透過 Amazon Braket Python SDK 建立量子任務的步驟概觀如下所示：

1. (選擇性) 開啟您的筆記本。
2. 匯入電路所需的 SDK 模組。
3. 指定 QPU 或模擬器。
4. 實例化電路。

5. 運行電路。

6. 收集結果。

本節中的範例顯示每個步驟的詳細資訊。

有關更多示例，請參閱上的 Amazon Braket 示例存儲庫。 GitHub

在本節中：

- 您好 AHS：運行您的第一個模擬哈密頓模擬
- 在 SDK 中建構電路
- 將量子任務提交到 QPU 和模擬器
- 使用開啟品質 3.0 執行您的電路
- 使用 QuEra的天鷹提交模擬程序
- 使用肉毒桿菌 3

您好 AHS：運行您的第一個模擬哈密頓模擬

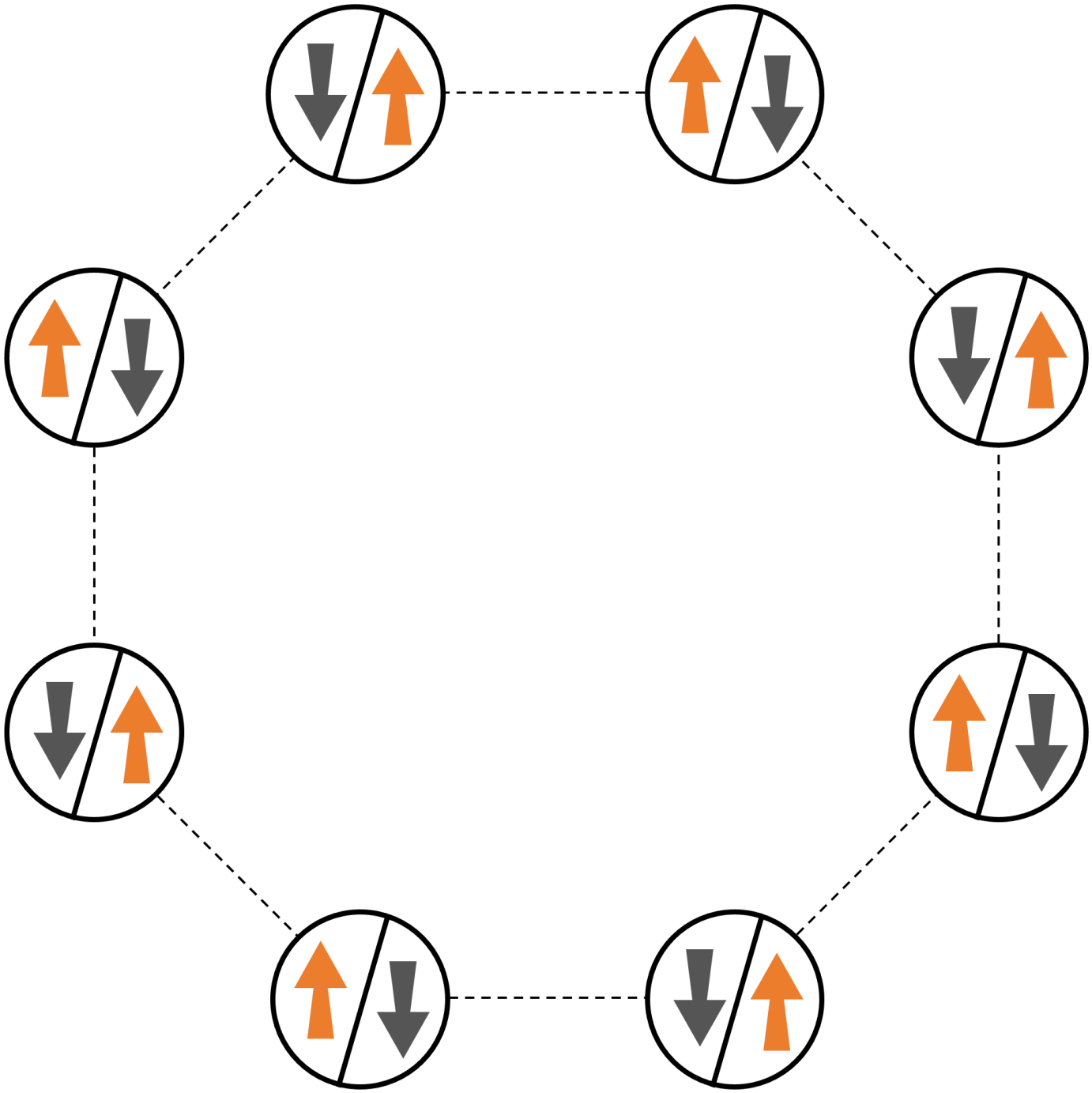
AHS

模擬哈密頓模擬（AHS）是量子計算的一種範式，與量子電路不同：而不是一系列門，每個門一次僅作用於幾個量子比特，而是由所討論的哈密頓時間和空間相關參數來定義。一個系統的哈密頓計算了它的能量水平和外力的影響，共同控制了其狀態的時間演變。對於 N 量子比特系統，哈密爾頓可以由複數的 $2^{N} X 2^{N}$ 平方矩陣來表示。

能夠執行 AHS 的量子設備將調整其參數（例如相干驅動場的振幅和除諧），以密切近似定制哈密頓定制下量子系統的時間演變。AHS 範式適用於模擬許多交互作用粒子的量子系統的靜態和動態特性。依據特定目的建置的 QPU，例如 Aquila 裝置，QuEra可以模擬系統的時間變化，而這些尺寸在傳統硬體上是不可行的。

互動式旋轉鏈

對於許多相互作用粒子系統的標準示例，讓我們考慮一圈八次旋轉（每個旋轉都可以在「向上」| ↑和「向下」| ↓狀態）。雖然很小，這個模型系統已經展現出了一些天然存在的磁性材料的有趣現象。在此示例中，我們將展示如何準備所謂的反鐵磁順序，其中連續旋轉指向相反的方向。



安排

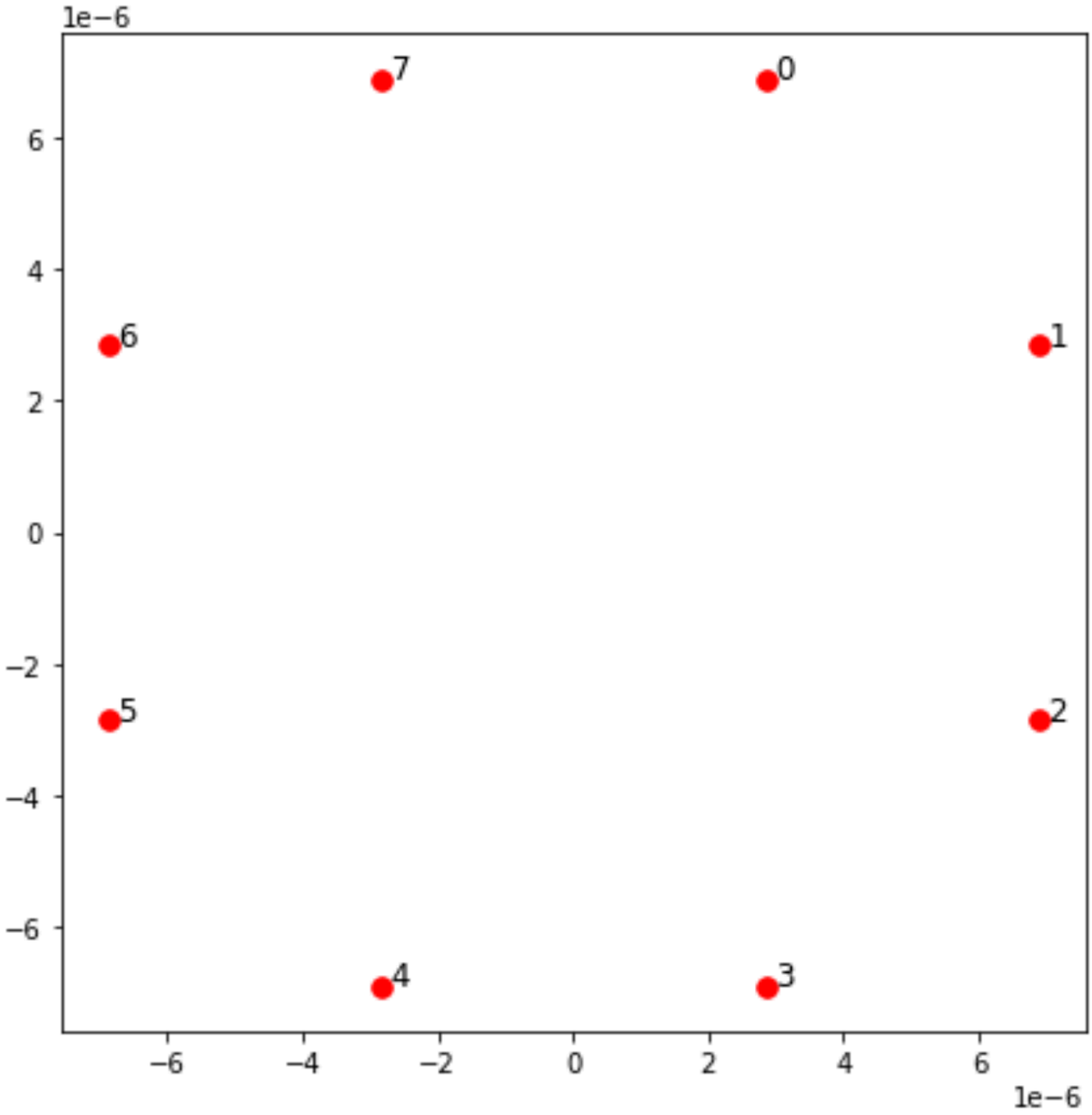
我們將使用一個中性原子來代表每次旋轉，「向上」和「向下」旋轉狀態將分別編碼為 Rydberg 狀態和原子的接地狀態。首先，我們創建 2-d 排列。我們可以使用以下代碼對上述旋轉進行編程。

先決條件：您需要點安裝布拉基特 SDK。（如果您使用的是 Braket 託管的筆記本實例，則此 SDK 已預先安裝在筆記本中。）要重現繪圖，您還需要使用 shell 命令單獨安裝 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)
```

我們也可以繪製

```
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
```



互動

為了準備防鐵磁相，我們需要誘導相鄰旋轉之間的相互作用。我們為此使用范德華互動，該交互作用是由中性原子設備（例如來QuEra自的設備）本身實現的Aquila。使用旋轉表示法，此相互作用的哈密頓術語可以表示為所有自旋對（j，k）的總和。

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

在這裡，nj=| ↑ $_j$ ↑ $_j$ | 是一個運算符，只有當旋轉 j 處於「向上」狀態時才接受值 1，否則為 0。強度為 V $_{j,k}$ = C_6/（ $d_{j,k}$ ）^{6}，其中 C $_6$ 是固定係數，d $_{j,k}$ 是旋轉 j 和 k 之間的歐幾里得距離。這個相互作用術語的直接效果是，旋轉 j 和旋轉 k 都「向上」的任何狀態都會提高能量（由量 $V_{j,k}$ ）。通過仔細設計 AHS 程序的其餘部分，這種交互將防止相鄰旋轉都處於「向上」狀態，這種效果通常稱為「Rydberg 封鎖」。

駕駛領域

在 AHS 程序的開始時，所有旋轉（默認情況下）都以「向下」狀態開始，它們處於所謂的鐵磁相。密切關注我們準備防鐵磁相的目標，我們指定了一個依賴於時間的相干驅動場，該領域可以將旋轉從此狀態平穩地轉換為多體狀態，其中「向上」狀態是首選的。相應的哈密頓軸可以寫成

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

其中 Ω（t），（t），Δ（t）是均勻影響所有旋轉的時間依賴性，全球幅度（又名拉比頻率），相位和解調。這裡 S $_{-,k}$ =| ↓ $_k$ ↑ | $_k$ 和 S $_{+,k}$ = ($S_{-,k}$) † =| ↑ $_k$ ↓ $_k$ | 分別是旋轉 k 的降低和提高運算子，n $_k$ =| ↑ ↑ | 和以前相同的操作員。$_k$ $_k$驅動場的 Ω 部分連貫地結合所有旋轉的「向下」和「上」狀態，而 Δ 部分則控制「向上」狀態的能量獎勵。

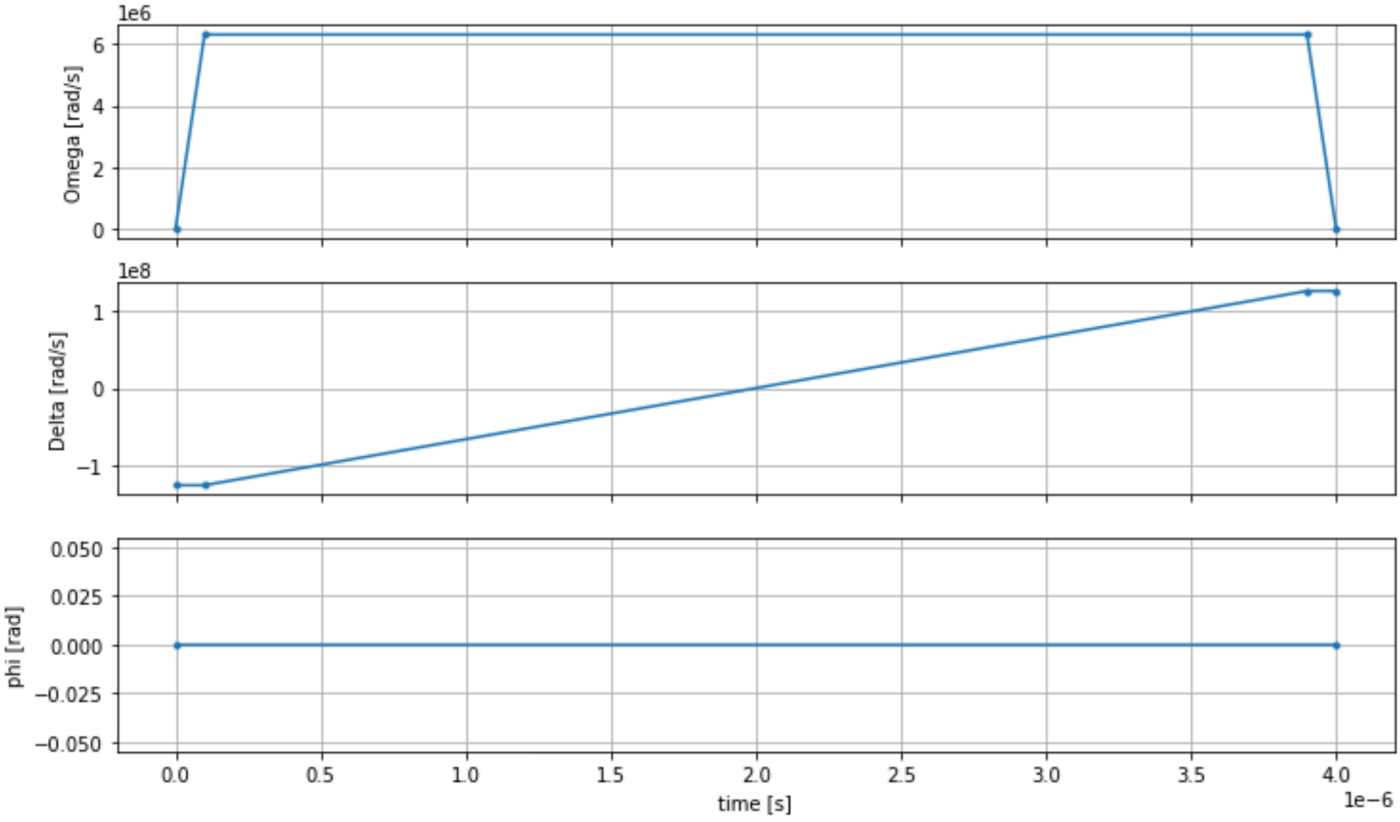
為了編程從鐵磁相到抗鐵磁相的平滑過渡，我們使用以下代碼指定驅動場。

```
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
)
```

我們可以使用以下腳本可視化驅動場的時間序列。

```
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 計劃

寄存器，驅動場（以及隱含的范德華交互作用）構成了模擬哈密頓仿真程序。ahs_program

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

在本地模擬器上運行

由於此示例很小（小於 15 次旋轉），因此在與 AHS 兼容的 QPU 上運行之前，我們可以在 Braket SDK 附帶的本地 AHS 模擬器上運行它。由於 Braket SDK 可免費使用本地模擬器，因此這是確保我們的代碼可以正確執行的最佳實踐。

在這裡，我們可以將拍攝數量設置為高值（例如 100 萬），因為本地模擬器跟踪量子狀態的時間演變並從最終狀態中抽取樣本；因此，增加拍攝數量，同時僅略微增加總運行時間。

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

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

分析模擬器結果

我們可以使用以下函數聚合拍攝結果，該函數推斷每次旋轉的狀態（可能是「d」表示「向下」，「u」表示「向上」，或「e」表示空站點），並計算每個配置在鏡頭中發生的次數。

```
from collections import Counter

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

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

    Args:
       result
 (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQua

    Returns
        dict: number of times each state configuration is measured

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

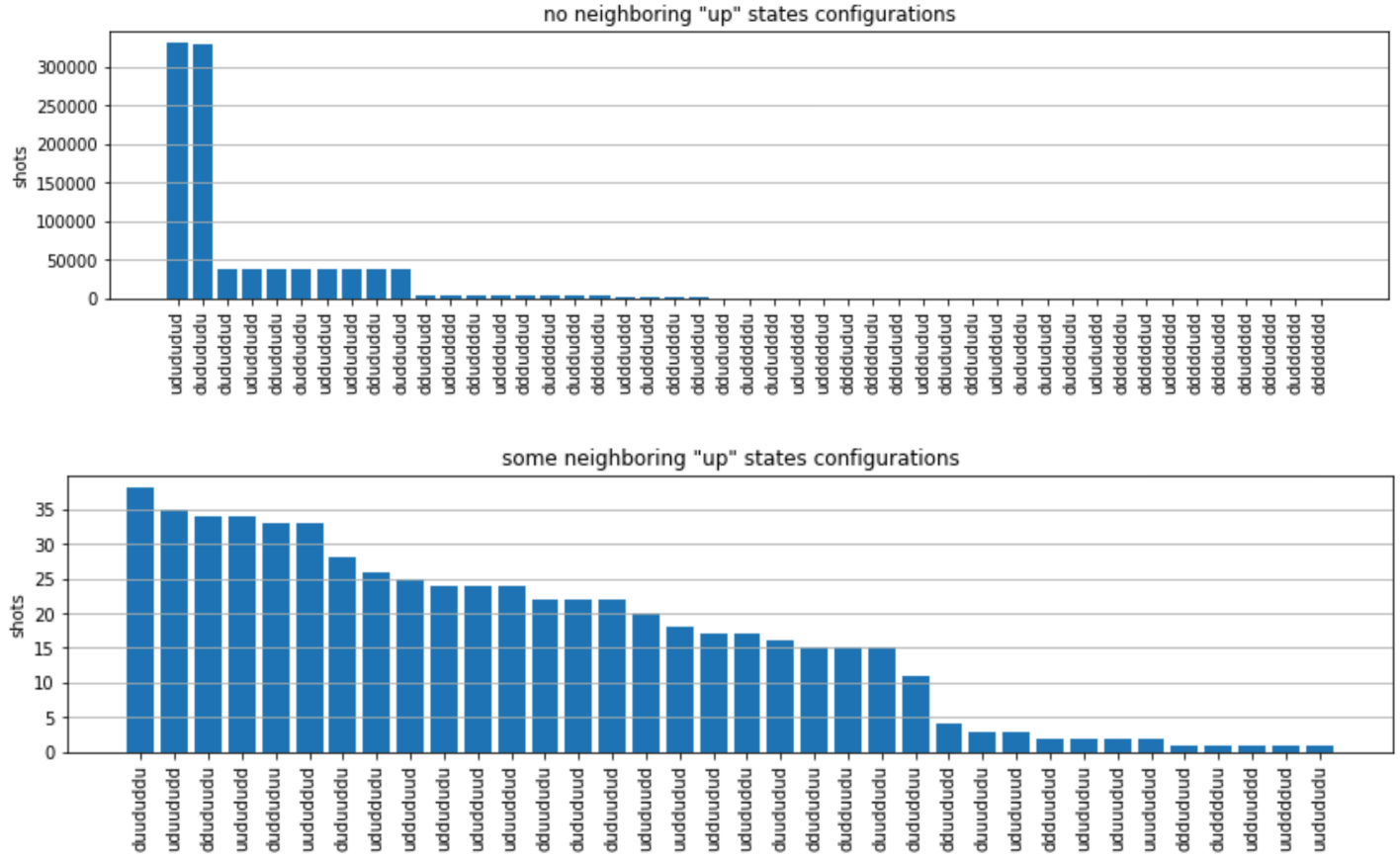
```
counts_simulator = get_counts(result_simulator)  # takes about 5 seconds
print(counts_simulator)
```

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

這counts是一個字典，用於計算每個狀態配置在鏡頭中觀察到的次數。我們也可以用下面的代碼可視化它們。

```
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)
```



從圖中，我們可以閱讀以下觀察結果驗證，我們成功地準備了抗鐵磁相。

1. 通常，非阻塞狀態（其中沒有兩個相鄰旋轉處於「向上」狀態）比至少一對相鄰旋轉都處於「向上」狀態的州更為常見。
2. 通常，除非配置被阻止，否則有更多「向上」激振的狀態會受到青睞。
3. 最常見的狀態確實是完美的抗鐵磁狀態和. "dudududu" "udududud"
4. 第二個最常見的狀態是那些只有 3 個「向上」激振，連續分離為 1，2，2。這表明凡德華互動也會對下一個最近的鄰居產生影響（儘管要小得多）。

跑上 QuEra的阿奎拉 QPU

先決條件：除了安裝 Braket SDK 的 pip 外，如果您是 Amazon Braket 的新手，請確認您已完成必要的入門步驟。

> **Note**
>
> 如果您使用的是 Braket 託管的筆記本執行個體，Braket SDK 會預先安裝在執行個體中。

安裝所有依賴關係後，我們可以連接到 Aquila QPU。

```
from braket.aws import AwsDevice

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

為了使我們的 AHS 程序適用於QuEra機器，我們需要對所有值進行四捨五入以符合 Aquila QPU 允許的精度級別。（這些要求由名稱中帶有「解析度」的設備參數控制。 我們可以通過在筆記本`aquila_qpu.properties.dict()`中執行看到它們。 有關 Aquila 功能和要求的更多詳細信息，請參閱 Aquila 筆記本簡介。）我們可以通過調用`discretize`方法來做到這一點。

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

現在，我們可以在 Aquila QPU 上運行該程序（現在只運行 100 鏡頭）。

> **Note**
>
> 在Aquila處理器上執行此程式將產生費用。Amazon Braket 開發套件包含成本追蹤器，可讓客戶設定成本限制，並以近乎即時的方式追蹤其成本。

```
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
```

由於量子任務可能需要多長時間的差異很大（取決於可用性窗口和 QPU 利用率），記下量子任務 ARN 是一個好主意，因此我們可以在稍後使用下面的代碼片段檢查其狀態。

```
# 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
```

狀態完成後 (也可以從 Amazon Braket 主控台的量子任務頁面檢查)，我們可以使用以下方式查詢結果：

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

分析 QPU 結果

使用與以前相同的`get_counts`函數，我們可以計算計數：

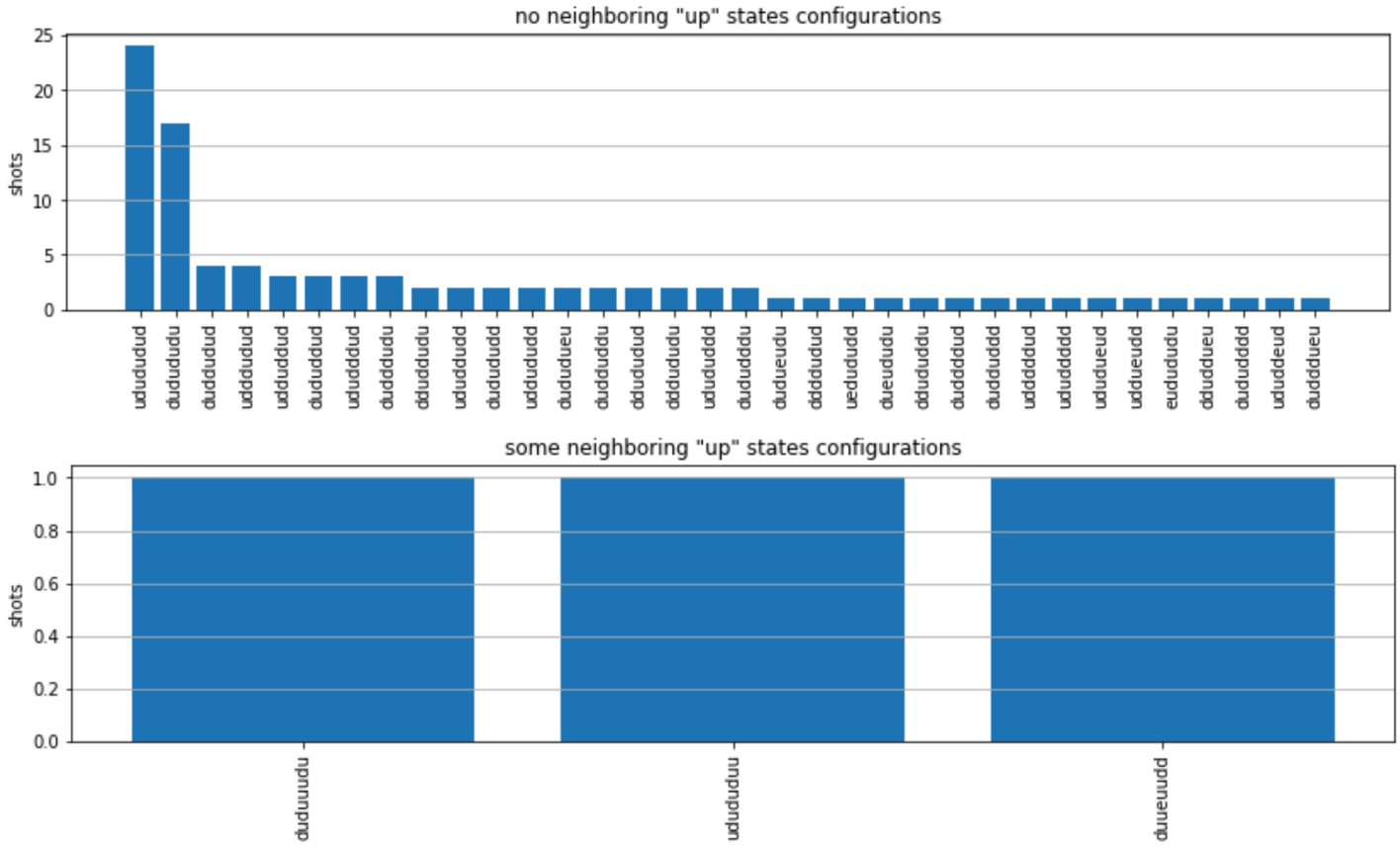
```
counts_aquila = get_counts(result_aquila)
```

```
print(counts_aquila)
```

```
*[Output]*
{'ududuudud': 24, 'duduududu': 17, 'dududdud': 3, ...}
```

並用以下方式繪製它們`plot_counts`：

```
plot_counts(counts_aquila)
```



請注意，一小部分鏡頭有空的網站（標有「e」）。這是由於 1—2% 每個原子製劑缺陷的 QPU. Aquila 除此之外，由於拍攝次數少，結果與預期的統計波動範圍內的模擬匹配。

下一頁

恭喜您，您現在已經使用本地 AHS 模擬器和 QPU 在 Amazon Braket 上運行了第一個 AHS 工作負載。Aquila

要了解有關 Rydberg 物理，模擬哈密頓仿真和Aquila設備的更多信息，請參閱我們的示例筆記本。

在 SDK 中建構電路

本節提供定義電路、檢視可用閘極、延伸電路以及檢視每個裝置支援的閘極的範例。它還包含有關如何手動分配qubits，指示編譯器完全按照定義運行電路，以及使用噪聲模擬器構建嘈雜的電路的說明。

您還可以在 Braket 的脈衝級別上工作，用於具有某些 QPU 的各種門。如需詳細資訊，請參閱 Amazon Braket 上的脈衝控制。

在本節中：

- 門和電路
- 局部測量
- 手動qubit配置
- 逐字编译
- 噪音模擬
- 檢查電路
- 結果類型

門和電路

量子閘和電路是在Amazon布拉克特 Python SDK 的`braket.circuits`類別中定義的。從 SDK 中，您可以通過調`Circuit()`用實例化新的電路對象。

範例：定義電路

此範例從定義四個樣本電路開始 qubits (標籤為、、和q3) `q0` `q1`q2，該電路由標準的單量子位元 Hadamard 閘和雙量子位元 CNT 閘極組成。您可以透過呼叫`print`函式來視覺化此電路，如下列範例所示。

```
# 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 |
```

範例：定義參數化電路

在此示例中，我們定義了一個具有依賴於自由參數的門的電路。我們可以指定這些參數的值來創建一個新的電路，或者在提交電路時，在某些設備上作為量子任務運行。

```
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)
```

您可以通過提供單個float（這是所有空閒參數將採用的值）或將每個參數值指定給電路的關鍵字參數來創建一個新的非參數化電路，如下所示。

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

請注意，my_circuit未經修改，因此您可以使用它來實例化許多具有固定參數值的新電路。

範例：修改電路中的灌嘴

以下範例定義了具有使用控制和功率修改器的閘極的電路。您可以使用這些修改來建立新灌嘴，例如受控灌Ry嘴。

```
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)
```

門修飾符僅在本地模擬器上受支持。

範例：查看所有可用的灌嘴

下面的例子顯示了如何查看 Amazon Braket 中所有可用的灌嘴。

```
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)
```

此代碼的輸出列出了所有門。

```
['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']
```

這些閘門中的任何一個都可以通過調用該類型電路的方法來附加到電路中。例如，您可以打電話給`circ.h(0)`第一個 Hadamard 門添加。qubit

> **Note**
>
> 將現地附加灌嘴，下面的範例將上一個範例中列示的所有灌嘴加入至同一電路。

```
circ = Circuit()
# toffoli gate with q0, q1 the control qubits and q2 the target.
circ.ccnot(0, 1, 2)
# cnot gate
circ.cnot(0, 1)
# controlled-phase gate that phases the |11> state, cphaseshift(phi) =
 diag((1,1,1,exp(1j*phi))), where phi=0.15 in the examples below
circ.cphaseshift(0, 1, 0.15)
```

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

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

除了預先定義的灌嘴組之外，您還可以將自定義的單一灌嘴套用至電路。這些可以是單量子位門（如下面的源代碼所示）或應用於參數qubits定義的多量子位門。targets

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

範例：延伸既有電路

您可以透過加入指示來延伸既有電路。A Instruction 是一個量子指令，描述量子任務在量子設備上執行。 Instruction運算子Gate僅包括類型的物件。

```
# 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})
```

範例：檢視每個裝置支援的閘門

模擬器支持 Braket SDK 中的所有門，但 QPU 設備支持較小的子集。您可以在設備屬性中找到設備的支持門。以下是 iONQ 裝置的範例：

```
# 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']
```

支援的閘可能需要先編譯成原生閘，才能在量子硬體上執行。當您提交電路時，AmazonBraket 會自動執行此編譯。

範例：以程式設計方式擷取裝置支援的原生閘門的擬真度

您可以在 Braket 主控台的「裝置」頁面上檢視擬真度資訊。有時，以編程方式訪問相同的信息是有幫助的。下面的代碼演示了如何提取一個 QPU 的兩個qubit門之間的兩個門保真度。

```
# 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"])
```

局部測量

按照前面的例子，我們已經測量了量子電路中的所有量子比特。但是，可以測量單個量子位或量子位的子集。

範例：測量量子位元的子集

在此示例中，我們通過在電路末端添加帶有目標量子位的`measure`指令來演示局部測量。

```
# 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)
```

手動qubit配置

當您從量子電腦上執行量子電路時Rigetti，您可以選擇性地使用手動qubit配置來控制用於演算法的哪qubits些電路。Amazon Braket 主控台和 Amazon Braket 開發套件可協助您檢查所選量子處理單元 (QPU) 裝置的最新校準資料，以便您選擇最qubits適合實驗的項目。

手動qubit配置可讓您更精確地執行電路，並調查個別qubit屬性。研究人員和高級用戶根據最新的設備校準數據優化其電路設計，並可以獲得更準確的結果。

下面的例子演示了如何qubits明確分配。

```
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)
```

如需詳細資訊，請參閱本筆記本上 GitHub或更具體的 Amazon Braket 範例：在 QPU 裝置上配置量子位元。

> **Note**
>
> 編OQC譯器不支援設定disable_qubit_rewiring=True。將此旗標設定為會True產生下列錯誤：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。

逐字编译

當您從Rigetti、、或 Oxford Quantum Circuits (OQC) 在量子電腦上執行量子電路時IonQ，您可以指示編譯器完全按照定義執行電路，而無需進行任何修改。使用逐字編譯，您可以指定按照指定的方式精確保留整個電路（由RigettiIonQ、和支持OQC），或者僅保留其中的特定部分（Rigetti僅受支持）。在開發硬體基準測試或錯誤緩解通訊協定的演算法時，您需要選擇精確指定您在硬體上執行的閘道和電路配置。逐字彙編可讓您直接控制編譯程序，方法是關閉某些最佳化步驟，從而確保您的電路完全按照設計執行。

、和 Oxford Quantum Circuits (OQC) 裝置目前支援逐字編譯 RigettiIonQ，且需要使用原生閘門。使用逐字編譯時，建議檢查設備的拓撲結構，以確保在連接時調用門，qubits並且電路使用硬件上支持的本機門。下列範例會示範如何以程式設計方式存取裝置支援的原生閘道清單。

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

對於Rigetti，必須通過設置與逐字編譯一起使disableQubitRewiring=True用來關閉qubit重新佈線。如果disableQubitRewiring=False在編譯中使用逐字框時設置，則量子電路驗證失敗並且不運行。

如果針對電路啟用逐字編譯，並在不支援該電路的 QPU 上執行，則會產生錯誤，指出不受支援的作業造成工作失敗。隨著越來越多的量子硬體本身支援編譯器函數，因此此功能將擴充以包含這些裝置。支持逐字編譯的設備在使用以下代碼查詢時將其包含為支持的操作。

```
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
```

使用逐字編譯沒有相關的額外費用。依據 Amazon Braket 定價頁面上指定的目前費率，繼續向您收取在 Braket QPU 裝置、筆記型電腦執行個體和隨需模擬器上執行的量子任務費用。如需詳細資訊，請參閱逐字編譯範例筆記本。

> **Note**
>
> 如果您使用 OpenQASM 為OQC和IonQ設備編寫電路，並且希望將電路直接映射到物理量子位，則需要使用，#pragma braket verbatim因為 OpenQASM 完全忽略了該disableQubitRewiring標誌。

噪音模擬

要實例化本地噪聲模擬器，您可以按如下方式更改後端。

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

您可以通過兩種方式構建嘈雜的電路：

1. 從下往上構建嘈雜的電路。
2. 採取現有的無噪音電路，並在整個過程中注入噪音。

下面的例子顯示了使用具有去極化噪聲和自定義 Kraus 通道的簡單電路的方法。

```
# 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)
```

執行電路與以前相同的使用者體驗，如下列兩個範例所示。

範例 1

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

或

範例 2

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

有關更多示例，請參閱 Braket 入門噪聲模擬器示例

檢查電路

在Amazon布拉克特量子電路有一個偽時間概念稱為. Moments 每個人都qubit可以體驗一個門Moment。的目的Moments是使電路及其門更容易解決並提供時間結構。

> **Note**
>
> 力矩通常不對應於在 QPU 上執行閘門的即時時間。

電路的深度由該電路中力矩的總數給出。您可以檢視呼叫方法`circuit.depth`的電路深度，如下列範例所示。

```
# 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
```

上述電路的總電路深度為 3 (顯示為力矩`0`1、和2)。您可以檢查每個時刻的澆口操作。

`Moments`功能作為鍵值對的字典。

- 關鍵是`MomentsKey()`，其中包含偽時間和qubit信息。
- 會以的類型指派值`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)]))
```

您也可以透過將灌嘴加入至電路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|
```

結果類型

Amazon使用ResultType測量電路時，Braket 可以返回不同類型的結果。電路可以傳回以下類型的結果。

- `AdjointGradient`返回提供的可觀測值的期望值的梯度（向量導數）。該可觀察到的作用於相對於使用伴隨微分法指定參數的所提供目標上。您只能在拍攝 = 0 時使用此方法。
- `Amplitude`傳回輸出波函數中指定量子狀態的振幅。它僅適用於SV1和本機模擬器。
- `Expectation`返回給定的可觀察的，它可以與本章後面介紹的`Observable`類來指定的期望值。必須指定qubits用於測量可觀察到的目標，並且指定目標的數量必須等於可觀察到qubits的作用的數量。如果沒有指定目標，可觀察到的必須僅在 1 操作，qubit並且它被應用到所有 qubits parallel。
- `Probability`返回測量計算基準狀態的概率。如果未指定目標，則`Probability`傳回測量所有基準狀態的可能性。如果指定了目標，則僅返回指定qubits的基向量的邊際概率。
- `Reduced density matrix`從系統傳回指定目標子系統qubits的密度矩陣qubits。為了限制此結果類型的大小，Braket qubits 將目標數量限制為最多 8 個。
- `StateVector`傳回完整狀態向量。它可在本地模擬器上使用。
- `Sample`返回指定目標qubit集合和可觀察的測量計數。如果沒有指定目標，可觀察到的必須僅在 1 操作，qubit並且它被應用到所有 qubits parallel。如果指定的目標，指定的目標的數量必須等於其qubits上的可觀察到的行為的數量。
- `Variance`返回指定目標qubit集合的變異數（`mean([x-mean(x)]`^{2}`)`），並作為請求的結果類型觀察。如果沒有指定目標，可觀察到的必須僅在 1 操作，qubit並且它被應用到所有 qubits parallel。否則，指定的目標的數目必須等於其中可觀察qubits到的可被應用的數量。

不同設備支持的結果類型：

	本地 SIM 卡	SV1	DM1	TN1	Rigetti	IonQ	OQC
伴隨漸變	N	Y	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N
期望	Y	Y	Y	Y	Y	Y	Y
Probability (可能性)	Y	Y	Y	N	Y*	Y	Y
密度矩陣減少	Y	N	Y	N	N	N	N

狀態向量	Y	N	N	N	N	N	N
樣本	Y	Y	Y	Y	Y	Y	Y
變異數	Y	Y	Y	Y	Y	Y	Y

> **Note**
>
> * Rigetti 僅支持最多 40 的概率結果類型qubits。

您可以檢查裝置內容來檢查支援的結果類型，如下列範例所示。

```
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
```

若要呼叫 aResultType，請將其附加至電路，如以下範例所示。

```
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**
>
> 一些設備提供測量結果（例如Rigetti），其他設備提供概率作為結果（例如IonQ和OQC）。SDK 會針對結果提供測量屬性，但是對於傳回概率的裝置，則會進行後期計算。因此，由提供的裝置IonQ和OQC具有由概率決定的量測結果，因為不會傳回每次射出測量。您可以透過檢視結果物件`measurements_copied_from_device`上的結果，如此檔案所示，以檢查結果是否已經過後計算。

可观测量

AmazonBraket 包括一個`Observable`類，它可以被用來指定一個可觀察到的測量。

您最多可以應用一個唯一的非身份可觀察到的每個。qubit如果將兩個或多個不同的非身份觀察值指定為相同qubit，則會看到一個錯誤。為了這個目的，張量產品的每個因子都算作為一個單獨的可觀察到的，因此允許有多個張量產品作用於相同的張量產品qubit，前提是作用於該產品的因子是相同的。qubit

您還可以縮放可觀察值並添加可觀測值（或不縮放）。這將創建一`Sum`個可以在`AdjointGradient`結果類型中使用的。

該`Observable`類包括以下觀察值。

```
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)))
```

參數

電路可以包括自由參數，你可以在一個「構造一次-運行多次」的方式使用和計算梯度。Free 參數具有字串編碼的名稱，可用來指定其值或決定是否要相對於它們進行區分。

```
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]
```

對於要區分的參數，請使用其名稱（作為字符串）或直接引用來指定它們。請注意，使用AdjointGradient結果類型計算梯度是相對於可觀察到的期望值完成的。

注意：如果通過將自由參數作為參數傳遞給參數化電路來固定空閒參數的值，則AdjointGradient以結果類型和指定的參數運行電路將產生錯誤。這是因為我們用來區分的參數不再存在。請參閱以下範例。

```
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
```

將量子任務提交到 QPU 和模擬器

AmazonBraket 提供對可以運行量子任務的多個設備的訪問。您可以單獨提交量子任務，也可以設置量子任務批處理。

QPU

您可以隨時向 QPU 提交量子任務，但該任務在 Amazon Braket 控制台的「設備」頁面上顯示的某些可用性窗口中運行。您可以使用量子任務 ID 來檢索量子任務的結果，該任務將在下一節中介紹。

- 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（僅限預訂）：`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`

模擬器

- 密度矩陣模擬器，DM1：`arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- 狀態向量模擬器，SV1：`arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- 張量網絡模擬器，TN1：`arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- 本地模擬器：`LocalSimulator()`

> **Note**
>
> 您可以取消CREATED處於 QPU 和隨選模擬器狀態的量子任務。您可以針對隨需模擬器和 QPU，盡力取消QUEUED狀態中的量子任務。請注意，QPU QUEUED 量子任務不太可能在 QPU 可用性窗口期間成功取消。

在本節中：

亞馬遜網站上的量子任務示例

本節將逐步介紹執行範例量子任務的各個階段，從選擇裝置到檢視結果。作為 Amazon Braket 的最佳實踐，我們建議您從模擬器上運行電路開始，例如SV1。

在本節中：

指定裝置

首先，選擇並指定量子任務的設備。這個例子演示了如何選擇模擬器，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")
```

您可以檢視此裝置的某些屬性，如下所示：

```
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)])
```

提交量子任務示例

提交量子任務示例以在隨選模擬器上運行。

```
# 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()
```

該device.run()命令通過 CreateQuantumTask API 創建量子任務。在短暫的初始化時間之後，量子任務將排隊，直到存在以在設備上運行量子任務的容量。在這種情況下，設備是SV1。裝置完成計算

後，AmazonBraket 會將結果寫入呼叫中指定的 Amazon S3 位置。除了本地模擬器之外的所有設備都需要位置參數`s3_location`。

> **Note**
>
> 布拉克特量子任務動作的大小限制為 3MB。

提交參數化工作

Amazon Braket 隨需和本機模擬器和 QPU 也支援在任務提交時指定可用參數值。您可以使用`inputs`引數來執行此操作`device.run()`，如下列範例所示。`inputs`必須是字符串-float 對的字典，其中鍵是參數名稱。

參數編譯可以改善在某些 QPU 上執行參數化電路的效能。將參數電路作為量子任務提交給支持的 QPU 時，Braket 將編譯電路一次，並緩存結果。對於相同電路的後續參數更新，不會對使用相同電路的任務進行重新編譯，因此執行時間更快。編譯電路時，Braket 會自動使用硬體供應商提供的更新校準資料，以確保獲得最高品質的結果。

> **Note**
>
> 除了脈衝級別程式之外，所有超導、閘極式 QPU Oxford Quantum Circuits 均支援參數編譯。Rigetti Computing

```
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

該shots引數是指所需的量測次數shots。模擬器，例如SV1支持兩種模擬模式。

- 對於 shots = 0，模擬器執行精確的模擬，返回所有結果類型的真值。（不適用於TN1。）
- 對於的非零值shots，模擬器會從輸出分佈中取樣，以模擬真實 QPU 的shot雜訊。QPU 裝置僅允許超shots過 0。

有關每個量子任務的最大射擊次數的信息，請參閱 Braket 配額。

投票結果

執行時`my_task.result()`，SDK 會使用您在量子任務建立時定義的參數開始輪詢結果：

- `poll_timeout_seconds`是在隨選模擬器和/或 QPU 設備上運行量子任務超時之前輪詢量子任務的秒數。預設值為 432 萬秒，也就是 5 天。
- 注意：對於 QPU 設備（例如Rigetti和）IonQ，我們建議您等待幾天。如果輪詢逾時太短，則輪詢時間內可能不會傳回結果。例如，當 QPU 無法使用時，就會傳回本機逾時錯誤。
- `poll_interval_seconds`是量子任務進行輪詢的頻率。它指定當量子任務在隨選模擬器和 QPU 設備上運行時，您調用 Braket API 以獲取狀態的頻率。預設值為 1 秒。

這種非同步執行有助於與並非總是可用的 QPU 裝置的互動。例如，在定期維護時段期間，裝置可能無法使用。

傳回的結果包含與量子工作相關聯的一系列中繼資料。您可以使用下列指令來檢查量測結果：

```
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]]
```

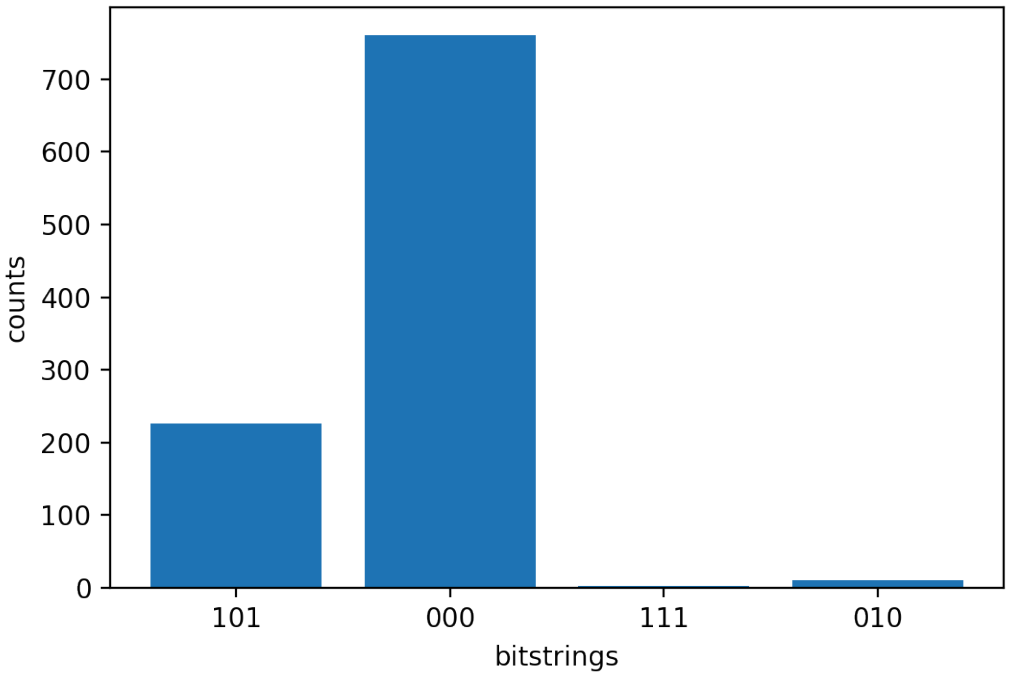
```
Counts for collapsed states:
 Counter({'000': 761, '101': 226, '010': 10, '111': 3})
Probabilities for collapsed states:
 {'101': 0.226, '000': 0.761, '111': 0.003, '010': 0.01}
```

檢視範例結果

因為您也已指定ResultType，因此您可以檢視傳回的結果。結果類型會依其加入至電路的順序顯示。

```
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]
```



將量子任務提交到 QPU

AmazonBraket 允許您在 QPU 設備上運行量子電路。下列範例顯示如何將量子任務提交至Rigetti或IonQ裝置。

選擇裝Rigetti Aspen-M-3置，然後查看關聯的連線圖

```
# 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']}}
```

上述字典connectivityGraph包含目前Rigetti裝置連線的相關資訊。

選擇設IonQ Harmony備

對於IonQ Harmony裝置而言，connectivityGraph是空的，如下列範例所示，因為裝置提供all-to-all連線能力。因此，不需要詳細信息connectivityGraph。

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

如下列範例所示，如果您選擇指定shots預設值區以外的poll_timeout_seconds位置，您可以選擇調整 poll_interval_seconds (預設值 = 1000)、(預設值 = 1)、(預設值 = 1s3_location) 以及儲存結果的位置 ()。

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

IonQ和Rigetti裝置會自動將所提供的電路編譯成各自的原生閘極組，並將抽象qubit索引對應至各自 QPU qubits 上的物理位置。

> **Note**
>
> QPU 裝置的容量有限。達到容量時，您可以預期更長的等待時間。

AmazonBraket 可以在某些可用性時段內執行 QPU 量子任務，但您仍然可以隨時 (全年無休) 提交量子任務，因為所有對應的資料和中繼資料都可靠地存放在適當的 S3 儲存貯體中。如下一節所示，您可以使用AwsQuantumTask和您唯一的量子任務 ID 恢復量子任務。

使用本地模擬器運行量子任務

您可以將量子任務直接發送到本地模擬器以進行快速原型製作和測試。此模擬器在您的本機環境中執行，因此您不需要指定 Amazon S3 位置。結果會直接在您的工作階段中計算。若要在本機模擬器上執行量子工作，您只必須指定shots參數。

> **Note**
>
> 本機模擬器所能處理的qubits執行速度和最大數目取決於 Amazon Braket 筆記型電腦執行個體類型或您本機的硬體規格。

以下命令都是相同的，並實例化狀態向量（無噪聲）本地模擬器。

```
# 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")
```

然後使用以下命令運行量子任務。

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

要實例化局部密度矩陣（噪聲）模擬器，客戶更改後端，如下所示。

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

在本地模擬器上測量特定的量子位

局部狀態向量模擬器和局部密度矩陣模擬器支持運行電路，其中可以測量電路的量子位的子集，這通常稱為局部測量。

例如，在下面的代碼中，您可以創建一個雙量子比特電路，僅通過在電路末端添加帶有目標量子位的measure指令來測量第一個量子比特。

```
# 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)
```

量子任務批處理

除了本地模擬器外，每個 Amazon Braket 設備都可以使用量子任務批處理。批次處理對於在隨選模擬器 (TN1或SV1) 上執行的量子任務特別有用，因為它們可以 parallel 處理多個量子任務。為了協助您設定各種量子任務，Amazon Braket 提供範例筆記本電腦。

批次處理可讓您 parallel 啟動量子任務。例如，如果您希望進行需要 10 個量子任務的計算，而這些量子任務中的電路彼此獨立，則最好使用批次處理。這樣，您就不必等待一個量子任務在另一個任務開始之前完成。

下面的例子演示了如何運行一批量子任務：

```
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
```

如需詳細資訊，請參閱 Amazon Braket GitHub 或 Quantum 任務批次處理的範例，其中包含有關批次處理的更具體資訊。

關於量子任務批處理和成本

有關量子任務批處理和計費成本，請牢記一些注意事項：

- 根據預設，量子任務批次處理會重試所有逾時或失敗量子任務 3 次。
- 一批長時間執行的量子任務 (例如 34 qubits forSV1) 可能會產生很大的成本。在開始一批量子任務之前，請務必仔細檢查`run_batch`分配值。我們不建議TN1搭配使用`run_batch`。
- TN1排練階段失敗的工作可能會產生成本 (如需詳細資訊，請參閱 TN1 說明)。自動重試可能會增加成本，因此我們建議在使用時將批次處理時的「max_retries」數量設置為 0TN1（請參閱量子任務批次處理，第 186 行）。

量子任務批處理和 PennyLane

在 Braket PennyLane 上使用時進行批次處理，方法是在實例化 Amazon Amazon Braket 裝置`parallel = True`時進行設定，如下列範例所示。

```
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,)
```

如需有關批次處理的詳細資訊 PennyLane，請參閱量子電路的平行化最佳化。

工作批次處理和參數化電路

提交包含參數化電路的量子任務批次時，您可以提供用於批次中所有量子任務的字inputs典，也可以提供輸入字典，在這種情況下，i第-th 字典與第 i-th 任務配對，如下列list範例所示。

```
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)
```

您還可以為單個參數電路準備輸入字典清單，並將其作為量子任務批次提交。如果清單中有 N 個輸入字典，則該批次包含 N 個量子工作。第 i-th 量子任務對應於使用 i-th 輸入字典執行的電路。

```
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)
```

設定 SNS 通知 (選用)

您可以透過 Amazon Simple Notification Service (SNS) 設定通知，以便在 Amazon Braket 量子任務完成時收到警示。如果您預期等待較長的時間；例如，當您提交大量量子任務或在設備的可用性窗口之外提交量子任務時，活動通知非常有用。如果您不想等待量子任務完成，則可以設置 SNS 通知。

AmazonBraket 筆記型電腦會引導您完成設定步驟。如需詳細資訊，請參閱上的 Amazon Braket 範例 GitHub，特別是設定通知的範例筆記本。

檢查編譯電路

當電路在硬件設備上運行時，必須以可接受的格式進行編譯，例如將電路轉換到 QPU 支持的本地門。檢查實際編譯的輸出對於調試目的非常有用。您可以使用以下代碼查看Rigetti和OQC設備的此電路。

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

> **Note**
>
> 今天，您無法查看IonQ設備的編譯電路。

使用開啟品質 3.0 執行您的電路

Amazon布拉科特現在支援 OpenQASM 3.0，適用於閘門式量子裝置和模擬器。本使用者指南提供有關支援 OpenQASM 3.0 子集的相關資訊。Braket 客戶現在可以選擇使用 SDK 提交 Braket 電路，或透過使用亞馬遜開發套件 API 和 Amazon Braket 開發套件將 OpenQASM 3.0 字串直接提供給所有閘道式裝置。

本指南中的主題將引導您逐步瞭解如何完成下列量子任務的各種範例。

- 在不同的 Braket 裝置上建立並提交 OpenQASM 量子任務
- 存取支援的作業和結果類型
- 使用開啟品質模擬雜訊

- 使用逐字編譯與 OpenQASM
- 疑難排解開啟品質問題

本指南還介紹了可以在 Braket 上使用 OpenQASM 3.0 實現的特定硬件功能以及更多資源的鏈接。

在本節中：

- 什麼是開放式管理系統 3.0？
- 何時使用
- 如何開啟 3.0 的運作方式
- 必要條件
- 布拉克支援哪些 OpenQASM 功能？
- 創建並提交一個示例 OpenQASM 3.0 量子任務
- Support 不同的胸罩裝置上的 OpenQASM
- 使用開啟品質模擬雜訊
- Qubit使用開啟品質 3.0 重新佈線
- 逐字編譯與開放式 QASM 3.0
- 布拉克特控制台
- 其他 資源
- 使用 OpenQASM 3.0 計算漸層
- 使用 OpenQASM 3.0 測量特定量子位元

什麼是開放式管理系統 3.0？

開放量子彙編語言 (OpenQASM) 是量子指令的中間表示法。OpenQASM 是一個開源框架，被廣泛用於基於門的設備量子程序的規範。使用 OpenQasm，使用者可以對構成量子計算建構基塊的量子閘和量測操作進行編程。許多量子程式設計程式庫使用舊版 OpenQASM (2.0) 來描述簡單的程式。

新版 OpenQASM (3.0) 擴充了舊版，包含更多功能，例如脈衝層級控制、閘門計時和傳統控制流程，以彌合使用者介面與硬體描述語言之間的差距。有關現行版本 3.0 的詳細資料和規格，請參閱 GitHub OpenQASM 3.x 即時規格。OpenQASM 的 future 發展由 OpenQASM 3.0 技術指導委員會管理，其中與 IBM，Microsoft 和 AWS 因斯布魯克大學一起成員。

何時使用

OpenQASM 提供了一個富有表現力的框架，通過不是特定於體系結構的低級控件來指定量子程序，使其非常適合作為跨多個基於門的設備的表示。Braket 對 OpenQASM 的支援更進一步採用其作為開發閘門式量子演算法的一致方法，從而減少使用者在多個架構中學習和維護程式庫的需求。

如果您在 OpenQASM 3.0 中有現有的程式庫，您可以調整它們以配合 Braket 使用，而不是完全重寫這些電路。研究人員和開發人員還應該受益於越來越多的可用第三方庫，並支持 OpenQASM 中的算法開發。

如何開啟 3.0 的運作方式

Support Braket 的 OpenQASM 3.0 可提供與目前中間表示的功能奇偶校驗。這意味著您今天可以在硬件設備和按需模擬器上使用 Braket 做任何事情，您都可以使用 Braket 使用 OpenQASM 進行操作。API您可以執行 OpenQASM 3.0 程式，方式與目前電路供應給 Braket 裝置的方式類似，直接將 OpenQASM 字串提供給所有閘極裝置。使用者也可以整合支援 OpenQASM 3.0 的第三方程式庫。本指南的其餘部分詳細介紹了如何開發 OpenQASM 表示以與 Braket 一起使用。

必要條件

要在布拉克特上使用 OpenQASM 3.0，您必須擁有亞馬遜Amazon布拉克 Python 模式的 v1.8.0 版和 v1.17.0 或更高版本的 Amazon Braket Python 開發套件。

如果您是Amazon布拉克特的第一次使用者，則需要啟用Amazon布拉克特。如需指示，請參閱啟用 Amazon Braket。

布拉克支援哪些 OpenQASM 功能？

以下段落列出 Braket 支援的 OpenQASM 3.0 資料類型、敘述句和編譯指示。

在本節中：

- 支援的資料類型
- 支援的陳述式
- 布拉克特开放 QASM 编译法
- 本機模擬器上 OpenQASM 的進階功能支援
- 支持的操作和語法 OpenPulse

支援的資料類型

下列 OpenQASM 資料類型受到啟動程式的支Amazon援。

- 非負整數用於（虛擬和物理）量子比特索引：
  - `cnot q[0], q[1];`
  - `h $0;`
- 澆口旋轉角度可以使用浮點數或常數：
  - `rx(-0.314) $0;`
  - `rx(pi/4) $0;`

> **Note**
>
> pi 是 OpenQASM 中的內建常數，無法當做參數名稱使用。

- 在結果類型編譯指令中允許複數數組（使用 OpenQASM `im` 表示法用於虛數部分），用於定義一般的赫米特可觀察值和單一編譯：
  - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
  - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

支援的陳述式

下列 OpenQASM 陳述式是由布拉克特所支援的Amazon。

- `Header: OPENQASM 3;`
- 經典位聲明：
  - `bit b1;`(等同地，`creg b1;`)
  - `bit[10] b2;`(等同地，`creg b2[10];`)
- 量子比特聲明：
  - `qubit b1;`(等同地，`qreg b1;`)
  - `qubit[10] b2;`(等同地，`qreg b2[10];`)
- 在數組中索引：`q[0]`
- 輸入:`input float alpha;`

- 物理規範qubits：`$0`
- 設備上支持的門和操作：
  - `h $0;`
  - `iswap q[0], q[1];`

> **Note**
>
> 您可以在 OpenQASM 動作的裝置屬性中找到裝置支援的閘門；使用這些閘門不需要閘門定義。

- 逐字框語句。目前，我們不支持框持續時間符號。本機門和物理qubits是必需的逐字框。

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

- qubits或整個qubit寄存器上的測量和測量分配。
  - `measure $0;`
  - `measure q;`
  - `measure q[0];`
  - `b = measure q;`
  - `measure q # b;`

> **Note**
>
> pi 是 OpenQASM 中的內建常數，無法當做參數名稱使用。

布拉克特开放 QASM 编译法

下列 OpenQASM 編譯指令是由布拉克特所支援。Amazon

- 噪音編譯

  - `#pragma braket noise bit_flip(0.2) q[0]`
  - `#pragma braket noise phase_flip(0.1) q[0]`
  - `#pragma braket noise pauli_channel`
- 逐字譯文
  - `#pragma braket verbatim`
- 結果類型編譯
  - 基礎不變結果類型：
    - 狀態向量：`#pragma braket result state_vector`
    - 密度矩陣：`#pragma braket result density_matrix`
  - 梯度計算編譯：
    - 伴隨漸變：`#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
  - Z 基礎結果類型：
    - 振幅：`#pragma braket result amplitude "01"`
    - 概率：`#pragma braket result probability q[0], q[1]`
  - 基礎旋轉結果類型
    - 期望:`#pragma braket result expectation x(q[0]) @ y([q1])`
    - 差異：`#pragma braket result variance hermitian([[0, -1im], [1im, 0]]) $0`
    - 示例：`#pragma braket result sample h($1)`

> **Note**
>
> OpenQASM 3.0 與 OpenQASM 2.0 向下相容，因此使用 2.0 編寫的程式可以在開頭架上執行。但是，Braket 支持的 OpenQASM 3.0 的功能確實存在一些小的語法差異，例如 `qreg` vs 和 vs `creg`。`qubit` `bit`測量語法也有所不同，而且這些語法必須使用正確的語法來支援。

本機模擬器上 OpenQASM 的進階功能支援

`LocalSimulator`支援先進的 OpenQASM 功能，這些功能不是作為布拉克特 QPU 或按需模擬器的一部分提供的。下列功能清單僅在中受支援`LocalSimulator`：

- 門修飾符
- 內建閘門
- 经典变星
- 经典作业
- 自訂閘門
- 古典控制
- 品質管理檔案
- 子程序

如需每項進階功能的範例，請參閱此範例筆記本。如需完整的 OpenQASM 規格，請參閱 Open QASM 網站。

支持的操作和語法 OpenPulse

支援的 OpenPulse 資料類型

卡路里塊：

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

除錯塊：

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

框架:

```
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};
```

自訂閘門校正範例：

```
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;
```

任意脈衝示例：

```
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);
}
```

創建並提交一個示例 OpenQASM 3.0 量子任務

您可以使用開發 Python Amazon 件、Bto3 或將 OpenQASM 3.0 量子任務提交 AWS CLI 到一個開發裝置。Amazon

在本節中：

- 開啟品質 3.0 程式的範例
- 使用 Python 開發套件建立量子任務
- 使用肉毒桿菌素 3 創建 3.0 量子任務
- 使用 AWS CLI 來建立開啟品質 3.0 工作

開啟品質 3.0 程式的範例

要創建一個 OpenQASM 3.0 任務，你可以用一個簡單的 OpenQASM 3.0 程序（ghz.qasm）開始準備一個 G HZ 狀態，如下面的例子。

```
// 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 開發套件建立量子任務

您可以使用 Amazon Braket 開發 Python 件，使用以下代碼將此程序提交到 Amazon Braket 設備。

```
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,
)
```

使用肉毒桿菌素 3 創建 3.0 量子任務

您也可以使用 AWS Python SDK 開發套件 (Boto3)，使用 OpenQASM 3.0 字串來建立量子工作，如下列範例所示。下面的代碼片段引用 ghz.qasm 準備一個 G HZ 狀態，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 AWS CLI 來建立開啟品質 3.0 工作

AWS Command Line Interface (CLI) 也可以用來提交 OpenQASM 3.0 程式，如下列範例所示。

```
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)
    }'
```

Support 不同的胸罩裝置上的 OpenQASM

對於支援 OpenQASM 3.0 的裝置，此`action`欄位支援透過`GetDevice`回應執行新動作，如下列範例中的Rigetti和IonQ裝置所示。

```
//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"
            ],
```

```
            ….
        }
    }
}
```

對於支持脈衝控制的設備，該pulse字段顯示在GetDevice響應中。下列範例顯示Rigetti和OQC裝置的此pulse欄位。

```
// 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
    }
  }
}
```

上述欄位詳細說明下列內容：

連接埠：

描述除了指定連接埠的相關內容之外，在 QPU 上宣告的預製外部 (extern) 裝置連接埠。此結構中列出的所有端口都預先聲明為用戶提交的OpenQASM 3.0程序中的有效標識符。連接埠的其他屬性包括：

- 連接埠識別碼 (端口)
  - 在 OpenQASM 3.0 中宣告為識別碼的連接埠名稱。
- 方向 (方向)
  - 連接埠的方向。驅動端口傳輸脈衝（方向「tx」），而測量端口接收脈衝（方向「rx」）。
- 連接埠類型 (連接埠類型)
  - 此端口負責的動作類型（例如，驅動器，捕獲或 FF-快通量）。
- DT (DT)
  - 指定連接埠上單一取樣時間步長的時間 (以秒為單位)。
- 量子位映射（量子位映射）
  - 與給定端口相關聯的量子位。
- 中心頻率 (中心頻率)
  - 連接埠上所有預先宣告或使用者定義框架的關聯中心頻率清單。如需詳細資訊，請參閱框架。
- QHP 特定屬性 () qhpSpecificProperties
  - 可選地圖，詳細說明有關 QHP 特定端口的現有屬性。

框架:

描述在 QPU 上宣告的預製外部框架以及有關框架的相關屬性。此結構中列出的所有框架都預先聲明為用戶提交的OpenQASM 3.0程序中的有效標識符。框架的其他屬性包括：

- 影格識別碼 (影格 ID)
  - 在 OpenQASM 3.0 中宣告為識別碼的影格名稱。
- 連接埠識別碼 (端口)
  - 框架的相關硬體連接埠。
- 頻率 (頻率)
  - 幀的默認初始頻率。
- 中心頻率 (中心頻率)
  - 該幀的頻率帶寬的中心。通常情況下，幀只能調整到中心頻率周圍的一定帶寬。因此，頻率調整應保持在中心頻率的給定增量範圍內。您可以在驗證參數中找到頻寬值。
- 階段 (階段)
  - 影格的預設初始階段。
- 相關的期限（關聯之門）
  - 與給定幀相關聯的門。
- 量子位映射（量子位映射）
  - 與給定幀相關聯的量子位。
- QHP 特定屬性 () qhpSpecificProperties
  - 可選地圖，詳細說明有關 QHP 特定框架的現有性質。

SupportsDynamicFrames:

描述框架是否可以通過OpenPulsenewframe函數聲明cal或defcal塊。如果這是假的，只有在幀結構中列出的幀可以在程序中使用。

SupportedFunctions:

描述除了給定OpenPulse函數的關聯引數、引數類型和傳回類型之外，裝置支援的函數。若要查看使用OpenPulse函數的範例，請參閱OpenPulse規格。此時，布拉克特支持：

- 移位相
  - 以指定值移動影格的相位
- 設定相位
  - 將幀的相位設置為指定值

- 移位頻率
  - 通過指定值移動幀的頻率
- 設定頻率
  - 將幀的頻率設置為指定值
- 播放
  - 排程波形
- 捕獲 _v0
  - 返回捕獲幀上的值到位寄存器

SupportedQhpTemplateWaveforms:

描述裝置上可用的預先建置波形函數，以及相關的引數和類型。默認情況下，Braket Pulse 在所有設備上提供預構建的波形常式，這些設備包括：

常數

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

τ是波形的長度，iq是複數。

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

高斯

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

τ是波形的長度，σ是高斯的寬度，A是振幅。如果設定ZaE為True，則高斯會偏移並重新縮放，使其在波形的開始和結束處等於零，並達到A最大值。

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

拖動高斯

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

τ是波形的長度，σ是高斯的寬度，β是一個自由參數，A是振幅。如果設定ZaE為True，則透過絕熱門移除導數 (DRAG) 高斯進行偏移並重新縮放，使其在波形的開始和結束處等於零，並且實數部分達到最大值。A有關 DRAG 波形的更多信息，請參閱用於消除弱非線性量子位中洩漏的簡單脈衝。

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

SupportsLocalPulseElements:

描述脈衝元件 (例如連接埠、訊框和波形) 是否可以在defcal區塊中本機定義。如果值為false，則必須在cal圖塊中定義元素。

SupportsNonNativeGatesWithPulses:

描述我們是否可以與脈衝程序結合使用非本機門。例如，我們不能在程序中使用非本地H門，而不必先定義所使用量子位defcal的門。您可以在設備功能下找到本機門nativeGateSet密鑰的列表。

ValidationParameters:

描述脈衝元素驗證邊界，包括：

- 波形的最大刻度/最大振幅值（任意和預建）
- 來自提供的中心頻率的最大頻寬，單位為 Hz
- 最小脈衝長度/持續時間 (秒)
- 最大脈衝長度/持續時間 (秒)

使用 OpenQASM 支援的作業、結果和結果類型

若要瞭解每個裝置支援哪些 OpenQASM 3.0 功能，您可以參考裝置功能action輸出欄位中的braket.ir.openqasm.program金鑰。例如，以下是可用於 Braket 狀態向量模擬器SV1的支援作業和結果類型。

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

使用開啟品質模擬雜訊

若要使用 OpenQASM3 模擬雜訊，您可以使用編譯指令來新增雜訊運算子。例如，若要模擬先前提供的 GHZ 程式的雜訊版本，您可以提交下列 OpenQASM 程式。

```
// 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;
```

下列清單提供了所有支援的編譯雜訊操作員的規格。

```
#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
```

克勞斯算子

若要產生 Kraus 運算子，您可以逐一查看矩陣清單，將矩陣的每個元素列印為複雜運算式。

使用 Kraus 運算子時，請記住下列事項：

- 的數目不qubits得超過 2。結構描述中的目前定義會設定此限制。
- 引數清單的長度必須是 8 的倍數。這意味著它必須僅由 2x2 矩陣組成。
- 總長度不超過 2 個 ^{2*} 數量子位矩陣。這意味著 4 矩陣為 1 qubit 和 16 對於 2。qubits
- 所有提供的矩陣都是完全正的跟踪保留（CP TP）。
- Kraus 運算子與其轉置共軛的產品需要加起來一個單位矩陣。

Qubit使用開啟品質 3.0 重新佈線

AmazonBraket 支援Rigetti裝置上 OpenQASM 內的實體qubit記號 (若要進一步了解，請參閱此頁面)。當將物理qubits與天真的重新佈線策略一起使用時，請確保在qubits所選設備上連接。或者，如果使用qubit暫存器，則預設會在Rigetti裝置上啟用 PARTIAL 重新佈線策略。

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

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

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

逐字編譯與開放式 QASM 3.0

當您在、和的量子電腦上執行量子電路時 Rigetti OQCIonQ，您可以指示編譯器完全按照定義的方式執行電路，而無需進行任何修改。此功能稱為逐字編譯。使用 Rigettti 設備，您可以準確指定保留的內容-無論是整個電路還是僅特定部分。若要僅保留電路的特定部分，您需要在保留的區域內使用原生閘門。目前，IonQ並且OQC僅支持整個電路的逐字編譯，因此電路中的每條指令都需要在逐字框中包含。

使用 OpenQasm，您可以在一個程式碼方塊周圍指定逐字編譯指令，這些程式碼未受硬體的低階編譯常式進行最佳化。下列程式碼範例會示範如何使用`#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;
```

如需逐字編譯的詳細資訊，請參閱逐字編譯範例筆記本。

布拉克特控制台

OpenQASM 3.0 任務可用，並且可以在管理控Amazon制台中進行管理。在主控台上，您擁有在 OpenQASM 3.0 中提交量子任務的經驗，就像您提交現有的量子任務一樣。

其他 資源

OpenQASM 在所有Amazon布拉克特區域都可使用。

如需開始使用 Amazon Braket 上 OpenQASM 的範例筆記本，請參閱開頭教學課程。 GitHub

使用 OpenQASM 3.0 計算漸層

Amazon Braket 使用隨需差異化方法，以 shots=0 (精確) 模式支援隨選和本機模擬器上的運算漸層。您可以提供適當的編譯指定要計算的梯度，如下面的例子。

```
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
```

而不是單獨列出所有的參數，你也可以all在編譯指定。這會根據列出的所有input參數計算漸層。當參數數量非常大時，這可能很方便。在這種情況下，編譯指數看起來像下面的例子。

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

支持所有可觀察的類型，包括單個操作員，張量產品，埃爾米蒂亞觀察值和. Sum 您要用來計算梯度的運算符必須包裝在expectation()封裝器中，並且必須指定每個項作用的量子比特。

使用 OpenQASM 3.0 測量特定量子位元

局部狀態向量模擬器和局部密度矩陣模擬器支持提交OpenQASM程序，其中可以測量電路的量子位的子集。這通常稱為局部測量。例如，在下面的代碼中，您可以創建一個雙量子比特電路，並僅測量第一個量子比特。

```
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];
"""
```

有兩個量子比特q[0]，q[1]但我們在這裡只測量量子位 0:. b[0] = measure q[0] 現在，在本地狀態向量模擬器上運行以下命令。

```
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)
```

您可以檢查裝置的動作屬性中的requiresAllQubitsMeasurement欄位，以檢查裝置是否支援局部量測；如果是False，則支援局部量測。

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

這裡，requiresAllQubitsMeasurement是False，這表明不是所有的量子位必須測量。

使用 QuEra的天鷹提交模擬程序

本頁提供有關Aquila機器功能的完整文件QuEra。這裡介紹的詳細信息如下：1）模擬的參數化哈密頓 Aquila，2）AHS 程序參數，3）AHS 結果內容，4）功能參數。Aquila我們建議使用 Ctrl+F 文本搜索來查找與您的問題相關的參數。

哈密頓

從Aquila機器本地QuEra模擬以下（依賴時間）哈密頓：

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

> **Note**
>
> 訪問本地解調是一種實驗性功能，可通過 Braket Direct 提供請求。

where

- $H_{drive,k}$（噸）=（$^{1}_{2}$ 歐姆（噸）電$^{子（噸）}$ $S_{-,k}$ + $^{1}/_{2}\Omega$（噸）電子（T）$S^{)+(-\Delta_{global}(t}_{+,k}$）n），k
  - Ω (t) 是時間相依的全球驅動振幅 (又稱拉比頻率)，單位為 (rad/s)
  - (t) 是時間依賴的全局相位，以弧度測量
  - $S_{-,k}$ 和 $S_{+,k}$ 是原子 k 的自旋降低和提高運算符（在基礎上 =|g，|↓#|↑=| R，它們是 S = | gr |，S =（S_{-}）† = | Rg|）+-
  - Δ_{global}（t）是時間依賴的，全局解調
  - n_k 是對原子 k 的雷德伯格狀態的投影運算符（即 n = | rr | ）
- $H_{local\ detuning,k}$（噸）=-Δ_{local}（噸）小時 $N_{k\ k}$
  - Δ_{local} (t) 是局部頻率偏移的時間依賴因子，單位為 (rad/s)
  - h_k 是網站相依係數，介於 0.0 和 1.0 之間的無維度數字
- $V_{vdw,k,l} = C_6/(d_{k,l})^6 N_k N_l$，
  - C_6 是凡德瓦係數，以 (rad /s) * (m) ^6 為單位
  - $d_{k,l}$ 是原子 k 和 l 之間的歐幾里得距離，以米為單位。

用戶可以通過 Braket AHS 程序模式控制以下參數。

- 2-d 原子排列（每個原子 k 的 x_k 和 y_k 坐標，以 um 為單位），$_{k,l}$用 k 控制成對原子距離 d，l = 1,2，... N
- Ω (t)，時間依賴的全球拉比頻率，單位為 (rad/s)
- (t)，依賴於時間的全局階段，以（rad）為單位
- Δ_{global} (t)，依賴於時間的全局解調，單位為（rad/s）
- Δ_{local} (t)，局部解調的大小的時間依賴（全局）係數，單位為（rad /s）
- h_k，局部解調量值的（靜態）站點相關係數，0.0 和 1.0 之間的無維度數

> **Note**
>
> 用戶無法控制涉及哪些級別（即 S_{-}，S，n 運算符是固定的）$_{+}$，也不能控制 Rydberg 格-里德伯格相互作用係數（C）的強度。$_{6}$

布拉基特 AHS 程序模式

程序 V1. 程序對象（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 (範例)

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

主要領域

程式欄位	type	description
安裝 .ahs_ 註冊. 網站	列表 [列表 [十進制]	鑷子陷阱原子的 2-d 坐標列表
安裝 .ahs_ 註冊. 填充	列表 [詮釋]	標記佔用陷阱地點的原子與 1, 和空的網站 0

程式欄位	type	description
哈密爾頓. 驅動場 []. 振幅. 時間 _ 系列.	清單 [十進位]	驅動幅度的時間點，歐米茄（t）
驅動字段 []. 振幅. 時間 _ 系列.	清單 [十進位]	驅動幅度值，歐米茄（t）
哈密頓. 驅動場 []. 振幅. 模式	str	驅動幅度的空間模式，歐米茄（t）；必須是「統一」
哈密爾托尼亞驅動字段 []. 相位. 時間 _ 系列.	清單 [十進位]	駕駛階段的時間點，phi (t)
哈密爾頓驅動字段 []. 相位. 時間 _ 系列.	清單 [十進位]	驅動相位值，phi (t)
哈密爾托尼亞. 驅動字段 []. 相位模式	str	駕駛階段的空間模式，phi（t）；必須是「統一」
驅動器字段 []. 減少. 時間 _ 系列。	清單 [十進位]	驅動解調的時間點，增量 _ 全球 (t)
驅動器字段 []. 減少. 時間 _ 系列.	清單 [十進位]	驅動解調的值，增量 _ 全局 (t)
哈密爾托尼亞。驅動字段 []. 解調模式	str	驅動解調的空間模式，三角洲 _ 全局（t）；必須是「統一」
本地解調 []. 時間系列.	清單 [十進位]	局部解調幅度的時間依賴係數的時間點，delta_Local (t)
本地解調 []. 時間系列.	清單 [十進位]	局部解調幅度的時間相依係數的值，增量 _ 局部 (t)

程式欄位	type	description
本地解調 []. 大小模式	清單 [十進位]	局部解調幅度的站點依賴因子，h_k (值對應於設置 .ahs_ 註冊網站)

元數據字段

程式欄位	type	description
braketSchemaHeader. 名稱	str	綱要的名稱；必須是「編輯程式」
braketSchemaHeader. 版本	str	結構描述的版本

規劃 AHS 任務結果模式

制動。任務。類似於哈密爾托尼 _ 量子 _ 任務 _ 結果。AnalogHamiltonianSimulationQuantumTaskResult(範例)

```
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 (範例)

```
{
    "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
        }
    }
}
```

主要領域

作業結果欄位	type	description
測量 []. 射擊結果. 前序	列表 [詮釋]	每次射擊的前序量測位元（每個原子部位各一個）：0 如果位點為空，如果網站已填滿，則在執行量子演化的脈衝序列之前測量
測量 []. 射擊結果. 後序列	列表 [詮釋]	每次射擊的序列後測量位：如果原子處於 Rydberg 狀態或位點為空，則為 0，如果原子處於接地狀態，則在運行量子演化的脈衝序列結束時測量為 1

元數據字段

作業結果欄位	type	description
braketSchemaHeader. 名稱	str	結構描述的名稱；必須是「編輯結果」
braketSchemaHeader. 版本	str	結構描述的版本
工作中繼資料。 braketSchemaHeader. 名稱	str	結構描述的名稱；必須是「編輯結果」
工作中繼資料。 braketSchemaHeader. 版本	str	結構描述的版本
任務中繼資料 ID	str	量子任務的 ID。對於 AWS 量子任務，這是量子任務 ARN。
任務元數據. 快照	int	量子任務的射擊次數
工作中繼資料. 快照. 裝置 ID	str	執行量子工作的裝置識別碼。對於 AWS 設備，這是設備 ARN。
工作中繼資料. 快照. 建立	str	建立的時間戳記；格式必須是 ISO-8601/RFC3339 字串格式的年-月-日:公釐:SSSZ

作業結果欄位	type	description
		。預設值為「無」。
任務元數據. 截圖.	str	量子任務結束時的時間戳記；格式必須是 ISO-8601/RFC3339 字符串格式年-月-日:mm:SSSZ。預設值為「無」。
工作中繼資料. 快照. 狀態	str	量子工作的狀態 (已建立、排入佇列、執行中、已完成、失敗)。預設值為「無」。
工作中繼資料. 快照. 失敗原因	str	量子任務的失敗原因。預設值為「無」。
附加元數據. 行動	电脑软件计划 V1 程序	（請參閱布拉克 AHS 程序模式部分）
附加元數據。行動。 braketSchemaHeader .Querametadata 名稱	str	結構描述的名稱；必須是「編輯結果」
附加元數據。行動。 braketSchemaHeader . 查拉米塔塔. 版本	str	結構描述的版本

作業結果欄位	type	description
附加元數據。行動。 numSuccessfulShots	int	完全成功拍攝的次數；必須等於要求的拍攝張數
測量 []. 獵食元數據。	int	拍攝狀態，（成功，部分成功，失敗）；必須是「成功」

QuEra 設備屬性模式

制動 .device_schema.quera_device_ 權限 _v1. QueraDeviceCapabilities(範例)

```
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 (範例)

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

服務屬性欄位

服務屬性欄位	type	description
服務. 執行視窗 []. 執行日	ExecutionDay	執行窗口的天數; 必須是「每天」，「平日」，「週末」，「星期一」，「星期二」，「星期三」，星期四，「星期五」，「星期六」或「星期日」
服務. 執行視窗 []. windowStartHour	日期時間. 時間	執行視窗開始時的 24 小時 UTC 格式
服務. 執行視窗 []. windowEndHour	日期時間. 時間	執行視窗結束時間的 24 小時 UTC 格式
服務能力. 服務. 拍攝範圍	元組 [詮釋，詮釋]	裝置的最小和最大拍攝張數
服務一般。設備環境優越。	float	在美元方面的設備價格
服務能力. 服務. 設備.	str	收取價格的單位，例如：「分鐘」，「小時」，「射擊」，「任務」

元數據字段

元數據字段	type	description
動作 []. 版本	str	AHS 程序模式的版本
動作 []. actionType	ActionType	AHS 程序模式名稱；必須是「編輯器」
服務。 braketSchemaHeader. 名稱	str	結構描述的名稱；必須是「編輯」
服務。 braketSchemaHeader. 版本	str	結構描述的版本

元數據字段	type	description
服務. 設備文檔. 圖片網址	str	設備圖像的 URL
服務. 裝置文件. 摘要	str	設備上的簡要說明
服務。設備文檔。 externalDocumentationUrl	str	外部文檔網址
服務. 設備位置	str	裝置的地理位置
服務. 更新	datetime	上次更新裝置內容的時間

使用肉毒桿菌 3

博托 3 是 Python 的 AWS SDK。使用 Boto3，Python 開發人員可以創建，配置和管理 AWS 服務，例如Amazon布拉克特。Boto3 提供了一個面向對象的API，以及低級別的訪問布拉克特。Amazon

按照 Boto3 快速入門指南中的說明學習如何安裝和配置 Boto3。

Boto3 提供了與 Amazon Braket Python SDK 一起使用的核心功能，以幫助您配置和運行量子任務。Python 客戶總是需要安裝 Boto3，因為這是核心實現。如果你想使用額外的輔助方法，你還需要安裝 Amazon Braket SDK。

例如，當您呼叫時`CreateQuantumTask`，AmazonBraket SDK 會將請求提交給 Boto3，然後呼叫. AWS API

在本節中：

- 打開 Amazon Braket 博托 3 客戶端
- 設 AWS CLI 定 Boto3 和亞馬遜開發套件的設定檔

打開 Amazon Braket 博托 3 客戶端

要將 Boto3 與 Amazon Braket 一起使用，您必須導入 Boto3，然後定義一個用於連接到 Braket 的客戶端。Amazon API在下列範例中，會命名為 Boto3 用戶端。`braket`

> **Note**
>
> 為了向下相容於舊版的 BraketSchemas，OpenQASM 資訊會在呼叫中`GetDeviceAPI`省略。若要取得此資訊，使用者代理程式必須呈現最新版本的 BraketSchemas (1.8.0 或更新版本)。Braket SDK 會自動為您報告此問題。如果您在使用 Braket SDK 時沒有看到 OpenQASM 產生`GetDevice`回應的結果，您可能需要設定`AWS_EXECUTION_ENV`環境變數來設定使用者代理程式。請參閱GetDevice 不傳回 OpenQASM 結果錯誤主題中提供的程式碼範例，瞭解如何針對 AWS CLI、Boto3 和 Go、Java 和/SDK 執行此動作。 JavaScript TypeScript

```
import boto3
import botocore

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

現在您已經建立了一個`braket`客戶，您可以從 Amazon Braket 服務發出請求並處理響應。您可以在API 參考中獲得有關請求和響應數據的更多詳細信息。

下列範例說明如何使用裝置和量子工作。

- 搜尋裝置
- 擷取裝置
- 建立量子任務
- 擷取量子任務
- 搜尋量子任務
- 取消量子任務

搜尋裝置

- `search_devices(**kwargs)`

使用指定的篩選器搜尋裝置。

```
# 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'])
```

擷取裝置

- `get_device(deviceArn)`

檢索Amazon布拉克特中可用的設備。

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

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

建立量子任務

- `create_quantum_task(**kwargs)`

創建一個量子任務。

```
# 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")
```

擷取量子任務

- `get_quantum_task(quantumTaskArn)`

擷取指定的量子任務。

```
# 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'])
```

搜尋量子任務

- `search_quantum_tasks(**kwargs)`

搜尋符合指定篩選器值的量子工作。

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

取消量子任務

- `cancel_quantum_task(quantumTaskArn)`

取消指定的量子任務。

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

設 AWS CLI 定 Boto3 和亞馬遜開發套件的設定檔

除非您另有明確指定，否則 Amazon Braket SDK 依賴於預設 AWS CLI 憑證。我們建議您在受管理的 Amazon Braket 筆記本上執行時保留預設值，因為您必須提供具有啟動筆記本執行個體權限的 IAM 角色。

或者，如果您在本機執行程式碼 (例如，在 Amazon EC2 執行個體上)，您可以建立具名 AWS CLI 設定檔。您可以為每個設定檔提供不同的權限集，而不必定期覆寫預設設定檔。

本節提供如何設定此類 CLI 的簡短說明，以`profile`及如何將該設定檔合併到 Amazon Braket 中，以便以該設定檔的權限進行API呼叫。

在本節中：

- 步驟 1：設定本機 AWS CLIprofile
- 第 2 步：建立一個 Boto3 會話對象
- 步驟 3：將 Boto3 會話合併到胸針 AwsSession

步驟 1：設定本機 AWS CLI**`profile`**

說明如何建立使用者以及如何設定非預設設定檔，已超出本文件的範圍。如需這些主題的相關資訊，請參閱：

- 入門
- 設定 AWS CLI 要使用的 AWS IAM Identity Center

若要使用 Amazon Braket，您必須向此使用者及相關的 CLI `profile` 提供必要的 Braket 權限。例如，您可以附加AmazonBraketFullAccess原則。

第 2 步：建立一個 Boto3 會話對象

為了建立一個 Boto3 會話對象，使用下面的代碼示例。

```
from boto3 import Session

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

> **Note**
>
> 如果預期的API呼叫具有與預`profile`設區域不一致的區域型限制，您可以指定 Boto3 工作階段的 Region，如下列範例所示。

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

針對指定為的引數`region`，取代對應於 Amazon Braket 可使用的其 AWS 區域 中一個值`us-east-1`，例如`us-west-1`、等。

步驟 3：將 Boto3 會話合併到胸針 AwsSession

下面的示例演示了如何初始化 Boto3 Braket 會話並在該會話中實例化設備。

```
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)
```

完成此設置後，您可以將量子任務提交到該實例化AwsDevice對象（例如調用device.run(…) 命令）。該裝置發出的所有API呼叫都可以利用與您先前指定為的 CLI 設定檔相關聯的 IAM 登入資料profile。

Amazon Braket 上的脈衝控制

本節說明如何在 Amazon Braket 中的各種 QPU 上使用脈衝控制。

在本節中：

剎車脈衝

脈衝是控制量子計算機中量子比特的模擬信號。對於 Amazon Braket 上的某些裝置，您可以存取脈衝控制功能以使用脈衝提交電路。您可以透過開發套件、使用 OpenQASM 3.0 或直接透過啟動器 API 存取脈衝控制。首先，讓我們介紹 Braket 中脈衝控制的一些關鍵概念。

影格

幀是一種軟件抽象，充當量子程序中的時鐘和相位。時鐘時間會在每次使用時遞增，並且由頻率定義的狀態載波訊號。將信號傳輸到量子比特時，幀確定量子比特的載波頻率，相位偏移以及波形包絡發射的時間。在 Braket 脈衝，構造幀取決於設備，頻率和相位。根據設備的不同，您可以選擇預定義的框架或通過提供端口實例化新框架。

```
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)
```

連接埠

端口是代表任何輸入/輸出硬件組件控制量子比特的軟件抽象。它可以幫助硬件供應商提供與用戶可以交互操縱和觀察量子位的接口。連接埠的特徵是以代表連接器名稱的單一字串。該字符串還暴露了一個最小時間增量，該增量指定了我們可以定義波形的精細程度。

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

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

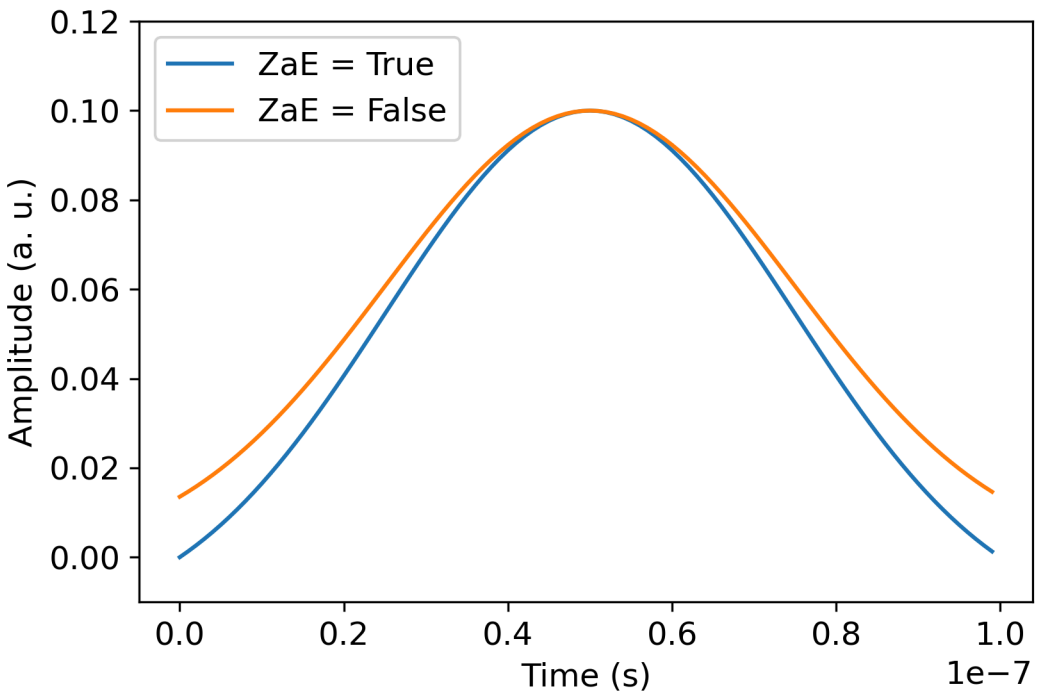
波形

波形是一個時間依賴的包絡，我們可以使用它在輸出端口上發出信號或通過輸入端口捕獲信號。您可以透過複數清單直接指定波形，也可以使用波形範本從硬體提供者產生清單來指定波形。

```
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 提供標準波形庫，包括常數波形、高斯波形以及絕熱門（DRAG）波形的導數去除。您可以透過`sample`函數擷取波形資料，以繪製波形的形狀，如下列範例所示。

```
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))
```



前面的影像描繪了從建立的高斯波形。`GaussianWaveform`我們選擇了 100 ns 的脈衝長度，寬度為 25 ns，振幅為 0.1（任意單位）。波形在脈衝視窗中居中。 `GaussianWaveform`接受布林引

數 zero_at_edges (圖例中的 ZAE)。當設定為時True，此引數會偏移高斯波形，使得 t=0 和 t= length 處的點為零，並重新調整其振幅，使得最大值對應於引數。amplitude

現在我們已經介紹了脈衝級訪問的基本概念，接下來我們將看到如何使用門和脈衝構建電路。

框架和端口的角色

本節說明每個裝置可用的預先定義框架和連接埠。我們還將簡要討論在某些幀上播放脈衝時所涉及的機制。

里格蒂

Frames (影格)

Rigetti設備支持預定義的幀，這些幀的頻率和相位經過校準以與相關的量子比特共振。命名約定是{i}指第一個量子位數的q{i}[_q{j}]_{role}_frame地方，{j}指的是第二個量子位數，以防幀用於激活兩個量子比特交互，並{role}指幀的作用。角色如下：

- rf是驅動量子比特 0-1 轉換的幀。脈衝被作為頻率和相位的微波瞬態信號傳輸先前通過set和shift功能提供。信號的時間依賴振幅由幀上播放的波形給出。該框架可插入單量子位元、離對角線的互動。如需詳細資訊，請參閱 Krantz 等。 和拉哈米姆等. 。
- rf_f12與其參數相似rf，其參數針對 1-2 轉換。
- ro_rx用於通過耦合共面波導實現量子比特的分散讀出。已預先校準讀出波形的頻率、相位和完整參數集。它目前通過使用capture_v0，除了幀標識符之外不需要任何參數。
- ro_tx用於從諧振器傳輸信號。它目前未使用。
- cz是經過校準以啟用雙cz量子位門的框架。與所有與ff端口相關聯的幀一樣，它通過與其鄰居在共振上調製對的可調量子比特來通過磁通線打開糾纏交互作用。如需有關糾纏機制的詳細資訊，請參閱 Reag or et al。 ，考德威爾等. ，和迪迪埃等. 。
- cphase是經過校準以啟用雙量子位元cphaseshift閘極的框架，並連接到端口。ff如需有關糾纏機制的詳細資訊，請參閱框架的cz說明。
- xy是經過校準以啟用雙量子位元 XY (θ) 閘的框架，並連結至連接埠。ff有關糾纏機制以及如何實現 XY 門的更多信息，請參閱cz框架的說明和 Abrams 等。 。

當基於ff端口的幀移動可調量子比特的頻率時，與量子比特相關的所有其他驅動幀將通過與頻率偏移的幅度和持續時間相關的量去除。因此，您必須在鄰近量子位元的影格中加入對應的相位移來補償此效果。

連接埠

Rigetti裝置提供可透過裝置功能檢查的連接埠清單。端口名稱遵循約定，q{i}_{type}其中{i}指的是量子位數，並{type}指的是端口的類型。請注意，並非所有的量子位都有一組完整的端口。連接埠的類型如下：

- rf代表驅動單量子位轉換的主界面。它與rf和rf_f12框架相關聯。它電容耦合到量子位元，允許微波在十億赫茲範圍內驅動。
- ro_tx用於將信號發送到讀出諧振器電容耦合到量子位。讀出信號傳遞由八角形多路復用八倍。
- ro_rx用於接收來自耦合到量子位的讀出諧振器的信號。
- ff表示感應耦合到量子比特的快通量線。我們可以用它來調整變形器的頻率。只有設計為高度可調節的量子位才有一個ff端口。該端口用於激活量子比特-量子比特交互作用，因為每對相鄰傳輸之間存在靜態電容耦合。

如需有關架構的詳細資訊，請參閱瓦列裡等。 。

OQC

Frames (影格)

OQC設備支持預定義的幀，這些幀的頻率和相位經過校準以與相關的量子比特共振。這些框架的命名慣例如下：

- 驅動幀：q{i}[_q{j}]_{role}其中{i}指的是第一量子位數，{j}是指幀用於激活兩個量子比特交互的情況下的第二量子位數，並{role}指幀的作用，如下所述。
- 量子位讀出幀：r{i}_{role}其中{i}指的是量子位數，並{role}指幀的作用，如下所述。

我們建議將每個框架用於其設計作用，如下所示：

- drive用作驅動量子比特 0-1 轉換的主幀。脈衝被作為頻率和相位的微波瞬態信號傳輸先前通過set和shift功能提供。信號的時間依賴振幅由幀上播放的波形給出。該框架可插入單量子位元、離對角線的互動。如需詳細資訊，請參閱 Krantz 等。 和拉哈米姆等. 。
- second_state相當於drive幀，但它的頻率調諧諧振與 1-2 過渡。
- measure是用於讀出。已預先校準讀出波形的頻率、相位和完整參數集。它目前通過使用capture_v0，除了幀標識符之外不需要任何參數。
- acquire用於從諧振器捕獲信號。它目前未使用。

- cross_resonance激活量子比特之間的交叉共振相互作用，i並j通過以目標量子比特的過渡頻率驅動控制量i子比特。j因此，使用目標量子位元的頻率來設定訊框頻率。相互作用發生在與此交叉諧振驅動器的振幅成比例的速率。不同類型的串擾會誘導不必要的效果，這需要更正。見帕特森等。有關與同軸形跨子量子比特（'coaxmons'）的交叉共振相互作用的更多信息。
- cross_resonance_cancellation幫助您添加校正以抑制在交叉共振相互作用激活時由串擾引起的有害效果。初始幀頻率被設置為控制量子比特i的過渡頻率。有關取消方法的更多信息，請參閱帕特森等人。 。

連接埠

OQC裝置提供可透過裝置功能檢查的連接埠清單。先前描述的幀與由其 ID channel_{N} 標識的端口相關聯，其中{N}是一個整數。端口是用於控制連接到同軸的線路（方向tx）和讀出諧振器（方向rx）的接口。每個量子比特與一個控制線和一個讀出諧振器相關聯。傳輸端口是用於單量子位和雙量子位操作的接口。接收端口用於量子位讀出。

你好脈衝

在這裡，您將學習如何直接用脈衝構建一個簡單的 Bell 對，並在Rigetti設備上執行此脈衝程序。貝爾對是一個雙量子比特電路，由第一量子比特上的 Hadamard 門組成，然後是第一和第二量子比特之間的cnot門。使用脈衝建立糾纏狀態需要依賴於硬體類型和裝置架構的特定機制。我們不會使用本機機制來創建cnot門。相反，我們將使用特定的波形和幀來本地啟用cz門。在這個例子中，我們將使用單量子比特本地門創建一個 Hadamard 門，rxrz並使用脈衝表達門。cz

首先，讓我們導入必要的庫。除了Circuit類別之外，您現在還需要匯入PulseSequence類別。

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

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

接下來，使用設備的 Amazon 資源名稱（ARN）實例化一個新的 Braket 設備。Rigetti Aspen-M-3請參閱 Amazon Braket 主控台上的「裝置」頁面，以檢視裝置的配Rigetti Aspen-M-3置。

```
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")
```

由於 Hadamard 門不是Rigetti設備的本地門，因此不能與脈衝結合使用。因此，您需要將其分解為rx和rz本機門的序列。

```
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)
    )
```

對於cz門，我們將使用帶有參數（振幅，上升/下降時間和持續時間）的任意波形，這些參數已由硬件提供商在校準階段預定。此波形將應用於q10_q113_cz_frame. 有關此處使用的任意波形的最新版本，請參閱Rigetti網站上的 QCS。您可能需要建立 QCS 帳號。

```
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, 0.00543941944632366, 0.010731922770068104, 0.01976701723583167,
 0.03406712171899736, 0.05503285980691202, 0.08350670755829034, 0.11932853352131022,
 0.16107456696238298, 0.20614055551722368, 0.2512065440720643, 0.292952577513137,
 0.328774403476157, 0.3572482512275353, 0.3782139893154499, 0.3925140937986156,
 0.40154918826437913, 0.4068371690898149, 0.4097040514225177, 0.41114381673553674,
 0.411813599998087, 0.4121022266390633, 0.4122174383870584, 0.41226003881132406,
 0.4122746298554775, 0.4122792591252675, 0.4122806196003006, 0.41228098995582513,
 0.41228108334474756, 0.4122811051578895, 0.4122811098772742, 0.4122811108230642,
 0.4122811109986316, 0.41228111102881937, 0.41228111103362725, 0.4122811110343365,
 0.41228111103443343, 0.4122811110344457, 0.4122811110344471, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 0.41228111103444737, 0.41228111103444737, 0.41228111103444737,
 0.41228111103444737, 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, 0.0004675110363603 3026, 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')
```

這應該返回：

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

現在，我們可以使用剛剛定義的波形構造cz門。回想一下，如果控制量子比特處於狀態，則cz門由目標量子比特的相位翻轉組成|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波形在與快通量端口關聯的幀上播放。它的作用是移動量子比特頻率以激活量子比特交互。如需詳細資訊，請參閱框架和連接埠的角色。隨著頻率的變化，量子比特幀的旋轉速率與保持不變的單量子比特rf幀不同：後者越來越淡化。這些相移已事先通過Ramsey序列進行校準，並在此處

作為通過phase_shift_a和phase_shift_b（完整週期）的硬編碼信息提供。我們通過在rf幀上使用shift_phase指令糾正這種去相。請注意，此序列僅適用於沒有a與b量子位相關的XY幀的程序，因為我們不補償這些幀上發生的相移。這是這個單一的貝爾對程序，它只使用rf和cz幀的情況。如需詳細資訊，請參閱考德威爾等。 。

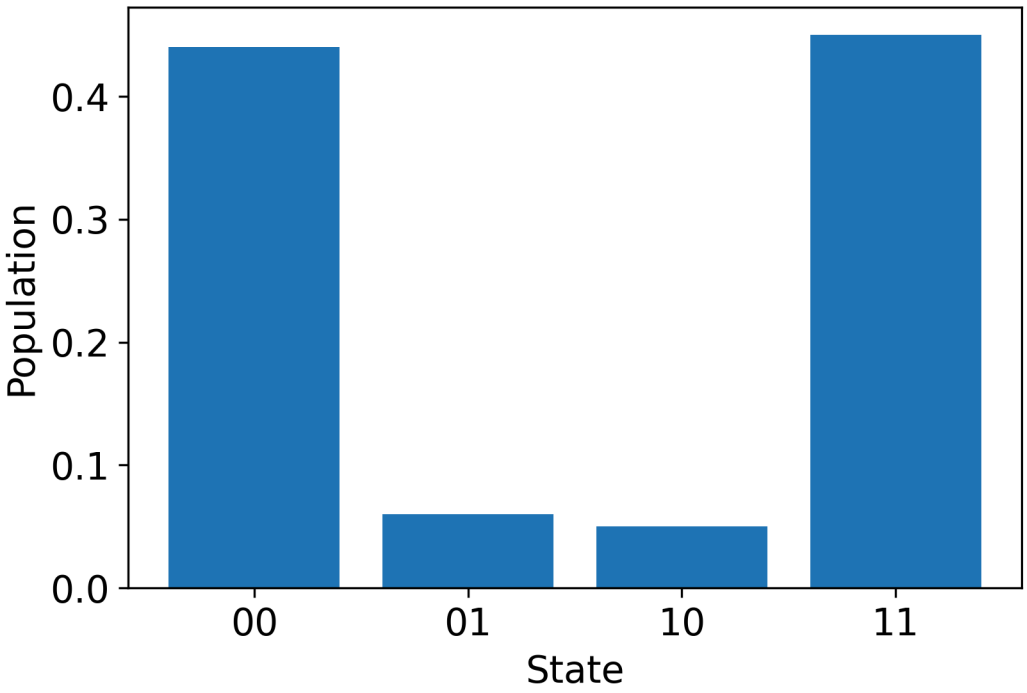
現在，我們已經準備好創建一個帶脈衝的 Bell 對。

```
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    |
```

讓我們在Rigetti設備上運行此貝爾對。請注意，執行此程式碼區塊會產生費用。如需這些費用的詳細資訊，請參閱 Amazon Braket 定價頁面。我們建議您使用少量鏡頭測試電路，以確保在增加射擊次數之前可以在設備上運行。

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

counts = task.result().measurement_counts

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



你好脈衝使用 OpenPulse

OpenPulse是一種用來指定一般量子裝置的脈衝層級控制的語言，也是 OpenQASM 3.0 規範的一部分。Amazon Braket 支持使OpenPulse用 OpenQASM 3.0 表示直接編程脈衝。

Braket 用OpenPulse作在本機指令中表達脈衝的底層中間表示。 OpenPulse支持以`defcal`（「定義校準」）聲明的形式添加指令校準。使用這些聲明，您可以在較低級別的控制語法中指定 gate 指令的實現。

在這個例子中，我們將使用 OpenQASM 3.0 和使用可頻率調諧傳輸的設備OpenPulse上構建一個貝爾電路。回想一下，貝爾電路是一個雙量子比特電路，其中包括在第一量子比特上的 Hadamard 門，然後是兩個量子比特之間的`cnot`門。由於`cnot`閘門與`cz`閘門不同，因此我們將使用 Hadamard 和 `cz` gate 來定義 Bell 對，因為該設備提供了一種更簡單的方法來為本演示創建`cz`門。

讓我們開始使用設備的本機門定義 Hadamard 門。

```
client = boto3.client('braket', region_name='us-west-1')
defcal h $10 {
    rz(pi) $10;
    rx(pi/2) $10;
    rz(pi/2) $10;
    rx(-pi/2) $10;
```

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

對於 CZ 門，我們將使用具有事先確定的參數（振幅，上升/下降時間和持續時間）的任意波形。此波形將應用於`q10_q113_cz_frame`.

```
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.4331554251320, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.484432196472809,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.48443981200754405, 0.4844398120075284,
 0.48443981200718716, 0.4844398120009317, 0.48443981190433844, 0.48443981064783953,
 0.48443979687791755, 0.4844396697373718, 0.4844386806183817, 0.4844321964728096,
 0.4843963766398558, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
 0.4753811409710734, 0.46121311095968553, 0.4331554251320285, 0.38631750509562957,
 0.32040677258513167, 0.24221990600377236, 0.16403303942240913, 0.0981223069119151,
 0.0512843868755143, 0.023226701047858084, 0.009058671036471328, 0.0030281044668842563,
 0.0008644760431374626, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
}
defcal cz $10, $113 {
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
```

```
    play(q10_q113_cz_frame, q10_q113_cz_wfm);
    delay[124ns] q10_rf_frame;
    shift_phase(q10_rf_frame, 1.1733407221086924);
    delay[124ns] q113_rf_frame;
    shift_phase(q113_rf_frame, 6.269846678712192);
    barrier q10_rf_frame, q113_rf_frame, q10_q113_cz_frame;
}
```

q10_q113_cz_wfm波形的持續時間為 124 個樣本，其對應於 124 納秒，因為最小時間增量dt為 1 ns。

q10_q113_cz_wfm波形在綁定到快通量端口的幀上播放。它的作用是移動量子比特頻率以激活量子比特交互。如需詳細資訊，請參閱框架和連接埠的角色。隨著頻率的變化，與保持不變的單量子位幀相比，量子比特rf幀以不同的速率旋轉：後者越來越淡化。這種去相位可以在校準階段使用Ramsey序列進行測量，並用上的shift_phase指令和幀進行補償。rf xy如需詳細資訊，請參閱考德威爾等。 。

現在，我們可以執行貝爾對電路，我們使用一對夫婦的 Hadamard 和cnot門分解門。cz

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

使用本機門和脈衝的組合構造的貝爾電路的完整 OpenQasm 3.0 表示如下。

```
// bell_pair_with_pulse.qasm
OPENQASM 3.0;
cal {
    waveform q10_q113_cz_wfm = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
 0.00021019328936380065, 0.0008644760431374357, 0.003028104466884364,
 0.009058671036471382, 0.02322670104785881, 0.05128438687551476, 0.09812230691191462,
 0.16403303942241076, 0.24221990600377236, 0.32040677258513395, 0.38631750509563006,
 0.43315542513203, 0.4612131109596859, 0.4753811409710734, 0.4814117075406603,
 0.48357533596440727, 0.48422961871818093, 0.4843963766398558, 0.4844321964728096,
 0.4844386806183817, 0.4844396697373718, 0.48443979687791755, 0.48443981064783953,
 0.48443981190433844, 0.4844398120009317, 0.48443981200718716, 0.4844398120075284,
 0.48443981200754405, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
  0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
```

```
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 0.4844398120075447,
 0.4844398120075447, 0.4844398120075447, 0.4844398120075447, 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, 0.484432196472809 6,
 0.484396376639855 8, 0.48422961871818093, 0.48357533596440727, 0.4814117075406603,
 0.475381140971073 4, 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;
```

您現在可以使用下列程式碼，使用開發套件在Rigetti裝置上執行這個 OpenQASM 3.0 程式。

```
# 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,
)
```

使用脈衝訪問本地門

研究人員通常需要確切地知道特定 QPU 支持的本機門是如何實現為脈衝的。脈衝序列由硬體供應商仔細校準，但是存取它們可讓研究人員有機會設計更好的閘門或探索錯誤緩解的協定，例如透過拉伸特定閘的脈衝來進行零雜訊外推法。

Amazon Braket 支持從里格蒂對本機門進行程序化訪問。

```
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**
>
> 硬體供應商會定期校準 QPU，通常每天不止一次。Braket SDK 可讓您取得最新的閘門校正。

```
device.refresh_gate_calibrations()
```

要檢索給定的本地門，例如 RX 或 XY 門，您需要傳遞Gate對象和感興趣的量子位。例如，您可以檢查套用於 0 的 RX (/2) 脈衝實作。qubit

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

您可以使用此filter功能建立一組已篩選的校準。您傳遞閘門的清單或清單QubitSet。下列程式碼會建立兩組，其中包含 RX (/2) 和 0 的所有校正。qubit

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

現在，您可以透過附加自訂校正集來提供或修改原生灌嘴的動作。例如，請考慮下列電路。

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

您可以通過將PulseSequence對象字典傳遞給gate_definitions關鍵字參數qubit 0來使用自定義rx門校準來運行它。您可以從GateCalibrations物件pulse_sequences的屬性建構字典。所有未指定的閘都會取代為量子硬體供應商的脈衝校準。

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

Amazon Braket 混合任務用戶指南

本節提供有關如何在 Amazon Braket 中設置和管理混合作業的說明。

您可以使用以下命令在 Braket 中存取混合式工作：

- Amazon Braket 蛇 SDK。
- Amazon Braket 控制台。
- Amazon布拉基特API。

在本節中：

什麼是混合式 Job？

Amazon Braket 混合任務可讓您執行需要傳統 AWS 資源和量子處理單元 (QPU) 的混合量子經典演算法。Hybrid Jobs 旨在增加請求的傳統資源，運行算法並在完成後釋放實例，因此您只需按使用量付費。

Hybrid Jobs 是涉及古典和量子資源的長時間運行迭代算法的理想選擇。您提交要執行的演算法，Braket 會在可擴充的容器化環境中執行，然後在演算法完成時擷取結果。

此外，透過混合式工作建立的量子任務，可受惠於較高優先順序佇列到目標 QPU。這樣可以確保您的量子任務得到處理，並在隊列中比其他任務領先。這對於迭代混合算法特別有益，其中後續任務取決於先前的量子任務的結果。此類演算法的範例包括量子近似最佳化演算法 (QAOA)、變分量子特徵求解析器或量子機器學習。您也可以近乎即時地監控演算法進度，讓您追蹤成本、預算或自訂指標，例如訓練損失或期望值。

何時使用 Amazon Braket 混合任務

Amazon Braket 混合任務可讓您執行混合量子經典演算法，例如變分量子特徵解析器 (VQE) 和量子近似最佳化演算法 (QAOA)，將傳統運算資源與量子運算裝置結合在一起，以最佳化現今量子系統的效能。Amazon Braket 混合任務提供三個主要好處：

1. 效能：Amazon Braket 混合式任務比從您自己的環境執行混合演算法提供更好的效能。當您的工作正在執行時，它具有優先權存取選取的目標 QPU。工作中的工作會在裝置上排入佇列的其他工作之前執行。這會導致混合演算法的執行時間縮短且更可預測。Amazon Braket 混合任務也支援參數化編譯。您可以使用自由參數提交電路，Braket 編譯電路一次，而無需重新編譯以進行相同電路的後續參數更新，從而導致更快的運行時間。
2. 便利性：Amazon Braket 混合式任務可簡化運算環境的設定和管理，並在混合演算法執行時保持運算環境的執行程序。您只需提供算法腳本並選擇要在其上運行的量子設備（量子處理單元或模擬器）即可。Amazon Braket 會等待目標裝置可用、增加傳統資源、在預先建置的容器環境中執行工作負載、將結果傳回 Amazon Simple Storage Service (Amazon S3)，然後釋放運算資源。
3. 指標：Amazon Braket 混合任務提供執行演算法的 on-the-fly 深入解析，並以近乎即時的方式將可自訂的演算法指標交付給 Amazon CloudWatch 和 Amazon Braket 主控台，讓您追蹤演算法的進度。

以混合工作的形式執行您的本機程式碼

Amazon Braket 混合任務提供混合量子經典演算法的全受管協調流程，將 Amazon EC2 運算資源與 Amazon Braket 量子處理單元 (QPU) 存取相結合。在混合任務中創建的量子任務優先於單個量子任務，因此您的算法不會被量子任務隊列中的波動打斷。每個 QPU 都會維護個別的混合式作業佇列，以確保在任何指定時間只能執行一個混合式作業。

在本節中：

- 從本地 Python 代碼創建一個混合任務
- 安裝額外的 Python 軟件包和源代碼
- 將資料儲存並載入混合式作業執行個體
- 混合工作裝飾器的最佳實踐

從本地 Python 代碼創建一個混合任務

您可以將本地 Python 代碼 Job 為亞馬遜編碼混合任務運行。您可以使用@hybrid_job裝飾器為程式碼加上註解，如下列程式碼範例所示。對於自訂環境，您可以選擇使用 Amazon 彈性容器登錄 (ECR) 提供的自訂容器。

> **Note**
>
> 默認情況下，只支持 Python 3.10。

您可以使用@hybrid_job裝飾器來註釋函數。Braket 將裝飾器內的代碼轉換為 Braket 混合作業算法腳本。然後，混合任務會在 Amazon EC2 執行個體上叫用裝飾器內部的函數。您可以使用job.state()或使用 Braket 主控台監控工作進度。下列程式碼範例示範如何在上執行五種狀態序列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}
```

您可以像普通 Python 函數一樣調用函數來創建混合作業。但是，裝飾器函數返回混合作業句柄，而不是函數的結果。若要在完成後擷取結果，請使用`job.result()`。

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

`@hybrid_job`裝飾器中的設備參數指定了混合作業優先訪問的設備-在本例中為SV1模擬器。要獲得 QPU 優先級，您必須確保函數中使用的設備 ARN 與裝飾器中指定的設備匹配。為了方便起見，您可以使用輔助函數捕`get_job_device_arn()`獲在中`@hybrid_job`聲明的設備 ARN。

> **Note**
>
> 每個混合任務至少有一分鐘的啟動時間，因為它會在 Amazon EC2 上建立容器化環境。因此，對於非常短的工作負載，例如單一電路或一批電路，您可以使用量子任務。

超參數

該run_hybrid_job()函數採用參數num_tasks來控制創建的量子任務的數量。混合式工作會自動將其擷取為超參數。

> **Note**
>
> 超參數會以字串的形式顯示在 Braket 主控台中，長度限制為 2500 個字元。

指標和記錄

在run_hybrid_job()函數中，來自迭代算法的度量記錄。log_metrics指標會自動繪製在「混合工作」標籤下的 Braket 主控台頁面中。使用 Braket 成本追蹤器，您可以使用指標在混合任務執行期間近乎即時地追蹤量子任務成本。上面的示例使用度量名稱「概率」，該名稱記錄結果類型的第一個概率。

擷取結果

混合式工作完成後，您可job.result()以用來擷取混合式工作結果。return 語句中的任何對象都會被 Braket 自動捕獲。請注意，函數返回的對象必須是一個元組，每個元素都是可序列化的。例如，下面的代碼顯示了一個工作和一個失敗的例子。

```
@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 名稱

根據預設，這個混合工作的名稱是從函數名稱推斷出來的。您也可以指定最多 50 個字元的自訂名稱。例如，在下列程式碼中，工作名稱為 "my-job-name"。

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

本機模式

通過將參數添加到裝飾器`local=True`來創建本地作業。這會在您的本機運算環境 (例如筆記型電腦) 的容器化環境中執行混合式工作。本機作業沒有量子工作的優先順序佇列。對於多節點或 MPI 等進階案例，本機作業可以存取必要的 Braket 環境變數。下面的代碼創建一個本地混合作業，該設備作為 SV1 模擬器。

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

支援所有其他混合式工作選項。有關選項的列表，請參閱文件。

安裝額外的 Python 軟件包和源代碼

您可以自訂執行階段環境，以使用偏好的 Python 套件。您可以使用`requirements.txt`檔案、套件名稱清單，或使用自己的容器 (BYOC)。若要使用`requirements.txt`檔案自訂執行階段環境，請參閱下列程式碼範例。

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

例如，`requirements.txt`檔案可能包含要安裝的其他套件。

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

或者，您可以提供包名稱作為 Python 列表，如下所示。

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

其他原始程式碼可以指定為模組清單，也可以指定為單一模組，如下列程式碼範例所示。

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

```
    return ...
```

將資料儲存並載入混合式作業執行個體

指定輸入訓練資料

建立混合任務時，您可以透過指定 Amazon Simple Storage Service (Amazon S3) 儲存貯體來提供輸入訓練資料集。您也可以指定本機路徑，然後 Braket 會自動將資料上傳到 Amazon S3。`s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>`如果您指定本機路徑，通道名稱預設為「input」。下面的代碼顯示了來自本地路徑的 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 ...
```

對於 S3，您必須使用`get_input_data_dir()`輔助功能。

```
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")
```

您可以透過提供通道值和 S3 URI 或本機路徑的字典來指定多個輸入資料來源。

```
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**
>
> 當輸入資料很大 (>1GB) 時，在建立工作之前需要很長的等待時間。這是由於本機輸入資料第一次上傳到 S3 儲存貯體，然後 S3 路徑會新增至任務請求。最後，工作請求被提交到 Braket 服務。

將結果儲存到 S3

要保存未包含在裝飾函數的 return 語句中的結果，您必須將正確的目錄附加到所有文件寫入操作中。下面的例子中，顯示了保存一個 numpy 數組和一個矩陣圖。

```
@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 ...
```

所有結果都會壓縮到名為的檔案中`model.tar.gz`。您可以使用 Python 函數下載結果`job.result()`，或從 Braket 管理主控台的混合工作頁面導覽至結果資料夾。

儲存並從檢查點恢復

對於長時間執行的混合式工作，建議定期儲存演算法的中繼狀態。您可以使用內置的`save_job_checkpoint()`輔助功能，或將文件保存到`AMZN_BRAKET_JOB_RESULTS_DIR`路徑中。後者可與輔助函數一起使用`get_job_results_dir()`。

以下是使用混合作業裝飾器保存和加載檢查點的最小工作示例：

```
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()
```

在第一個混合作業中save_job_checkpoint()，使用包含我們要保存的數據的字典調用。默認情況下，每個值必須是可序列化為文本。對於檢查點更複雜的 Python 對象，如 numpy 數組，你可以設置。data_format = PersistedJobDataFormat.PICKLED_V4此程式碼會<jobname>.json在名為「檢查點」的子資料夾下，以預設名稱建立並覆寫混合工作成品中的檢查點檔案。

要創建一個新的混合作業從檢查點繼續，我們需要通過copy_checkpoints_from_job=job_arn哪裡job_arn是以前的作業的混合作業 ARN。然後，我們load_job_checkpoint(job_name)使用從檢查點加載。

混合工作裝飾器的最佳實踐

擁抱異步

使用裝飾器註釋創建的混合作業是異步的-一旦傳統和量子資源可用，它們就會運行。您可以使用 Braket Management Console或 Amazon 監視算法的進度 CloudWatch。當您提交要執行的演算法時，Braket 會在可擴充的容器化環境中執行您的演算法，並在演算法完成時擷取結果。

執行反覆運算變數演算法

混合任務為您提供了運行迭代量子古典算法的工具。對於純粹的量子問題，使用量子任務或一批量子任務。對於需要在兩者之間進行多次迭代調用的 QPU 的長時間運行變性算法，優先訪問某些 QPU 最有利。

使用本機模式偵錯

在您在 QPU 上運行混合作業之前，建議首先在模擬器 SV1 上運行，以確認它按預期運行。對於小規模測試，您可以使用本地模式運行以進行快速迭代和調試。

使用自攜容器 (BYOC) 提高再現性

透過將軟體及其相依性封裝在容器化環境中，建立可重現的實驗。透過將所有程式碼、相依性和設定封裝在容器中，即可防止潛在的衝突和版本控制問題。

多開分散式模擬器

若要執行大量電路，請考慮使用內建 MPI 支援，在單一混合式工作中的多個執行個體上執行本機模擬器。如需詳細資訊，請參閱內嵌模擬器。

使用參數式電路

您從混合任務提交的參數電路會使用參數編譯在某些 QPU 上自動編譯，以改善演算法的執行時間。

定期檢查點

對於長時間執行的混合式工作，建議定期儲存演算法的中繼狀態。

如需進一步的範例、使用案例和最佳實務，請參閱 Amazon Braket 範例。 GitHub

使用 Amazon Braket 混合式任務執行混合任務

若要使用 Amazon Braket 混合式工作執行混合作業，您首先需要定義演算法。您可以透過撰寫演算法指令碼和選擇性地使用 Amazon Braket Python 開發套件或PennyLane其他相依性檔案來定義它。如果您想使用其他（開源或專有）庫，則可以使用 Docker（包括這些庫）定義自己的自定義容器映像。如需詳細資訊，請參閱攜帶您自己的容器 (BYOC)。

在任何一種情況下，接下來都要使用 Amazon Braket API 建立混合任務 (提供演算法指令碼或容器)，選取混合任務要使用的目標量子裝置，然後從各種選用設定中進行選擇。為這些可選設定提供的預設值適用於大多數使用案例。若要讓目標裝置執行您的混合式 Job，您可以選擇 QPU、隨選模擬器 (例如SV1、DM1或TN1)，或傳統的混合式作業執行個體本身。使用隨選模擬器或 QPU，您的混合式任務容器會對遠端裝置進行 API 呼叫。使用嵌入式模擬器，模擬器嵌入到與算法腳本相同的容器中。來自 PennyLane 的閃電模擬器嵌入了默認的預構建混合作業容器供您使用。如果您使用內嵌 PennyLane 模擬器或自訂模擬器執行程式碼，您可以指定執行個體類型以及要使用的執行個體數目。請參閱 Amazon 網頁定價頁面，瞭解與每個選項相關的費用。

如果您的目標裝置是隨選或嵌入式模擬器，Amazon Braket 會立即開始執行混合任務。它會啟動混合任務執行個體 (您可以在API呼叫中自訂執行個體類型)、執行演算法、將結果寫入 Amazon S3，然後釋放資源。此版本的資源可確保您只需按使用量付費。

每個量子處理單元 (QPU) 的並行混合式作業總數會受到限制。現在，只有一個混合式工作可以在任何給定的時間在 QPU 上執行。佇列是用來控制允許執行的混合式作業數目，以免超過允許的限制。如果您的目標裝置是 QPU，則混合任務會先進入所選 QPU 的作業佇列。Amazon Braket 會啟動所需的混合式任務執行個體，並在裝置上執行您的混合任務。在演算法期間，您的混合任務具有優先權存取，這意味著您混合任務中的量子任務會在裝置上排入佇列的其他 Braket 量子任務之前執行，前提是工作量子任務每隔幾分鐘提交到 QPU 一次。一旦您的混合工作完成，資源就會釋放，這意味著您只需按使用量付費。

> **Note**
>
> 裝置是區域性的，而您的混合任務會在與主要裝置 AWS 區域 相同的位置執行。

在模擬器和 QPU 目標案例中，您可以選擇定義自訂演算法度量，例如哈密頓的能量，做為演算法的一部分。這些指標會自動報告給亞馬遜， CloudWatch 並從那裡以近乎即時的方式顯示在 Amazon Braket 主控台中。

> **Note**
>
> 如果您想要使用以 GPU 為基礎的執行個體，請務必使用 Braket 上嵌入式模擬器提供的其中一種 GPU 模擬器 (例如)。`lightning.gpu`如果您選擇其中一個以 CPU 為基礎的嵌入式模擬器 (例如、或`braket:default-simulator`)`lightning.qubit`，將不會使用 GPU，而且可能會產生不必要的費用。

建立您的第一個混合 Job

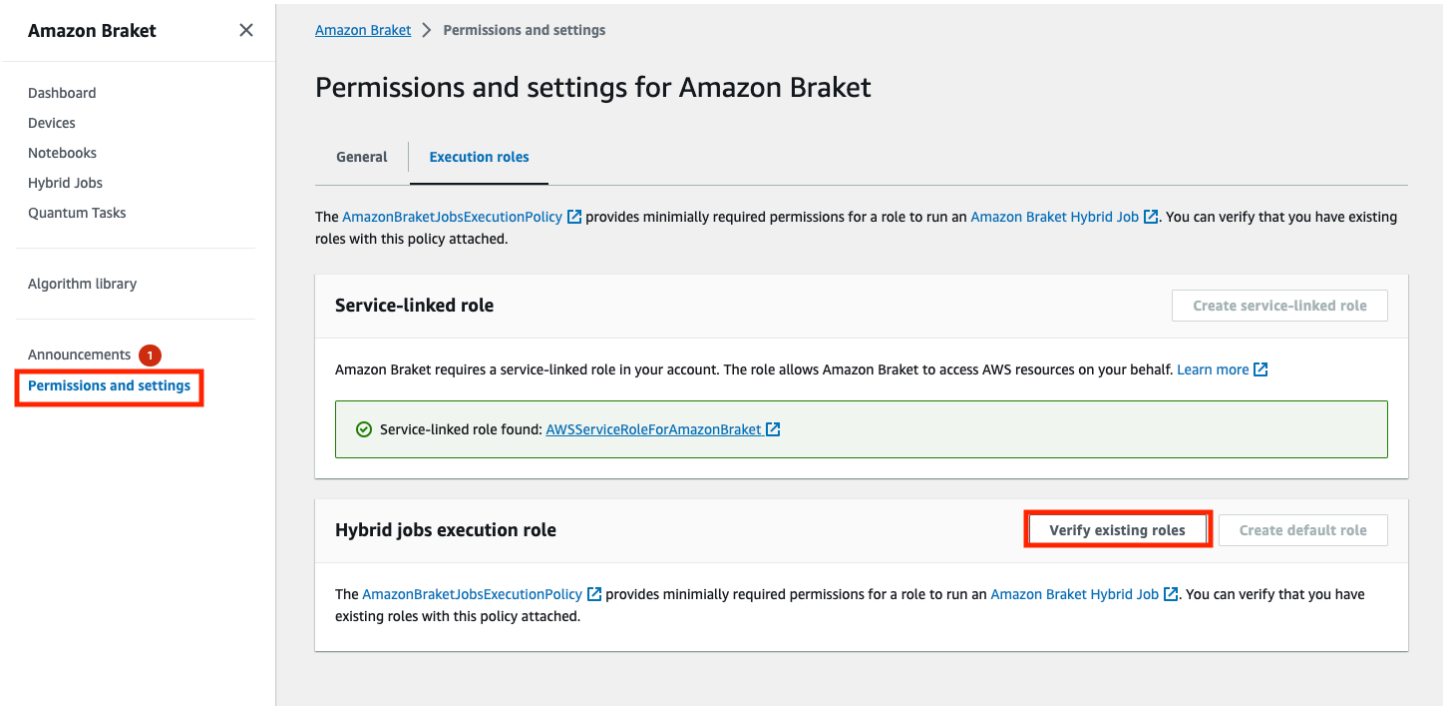
本節說明如何使用 Python 指令碼建立混合 Job。或者，若要使用本機 Python 程式碼建立混合作業，例如您偏好的整合式開發環境 (IDE) 或 Braket 筆記本，請參閱以混合工作的形式執行您的本機程式碼。

在本節中：

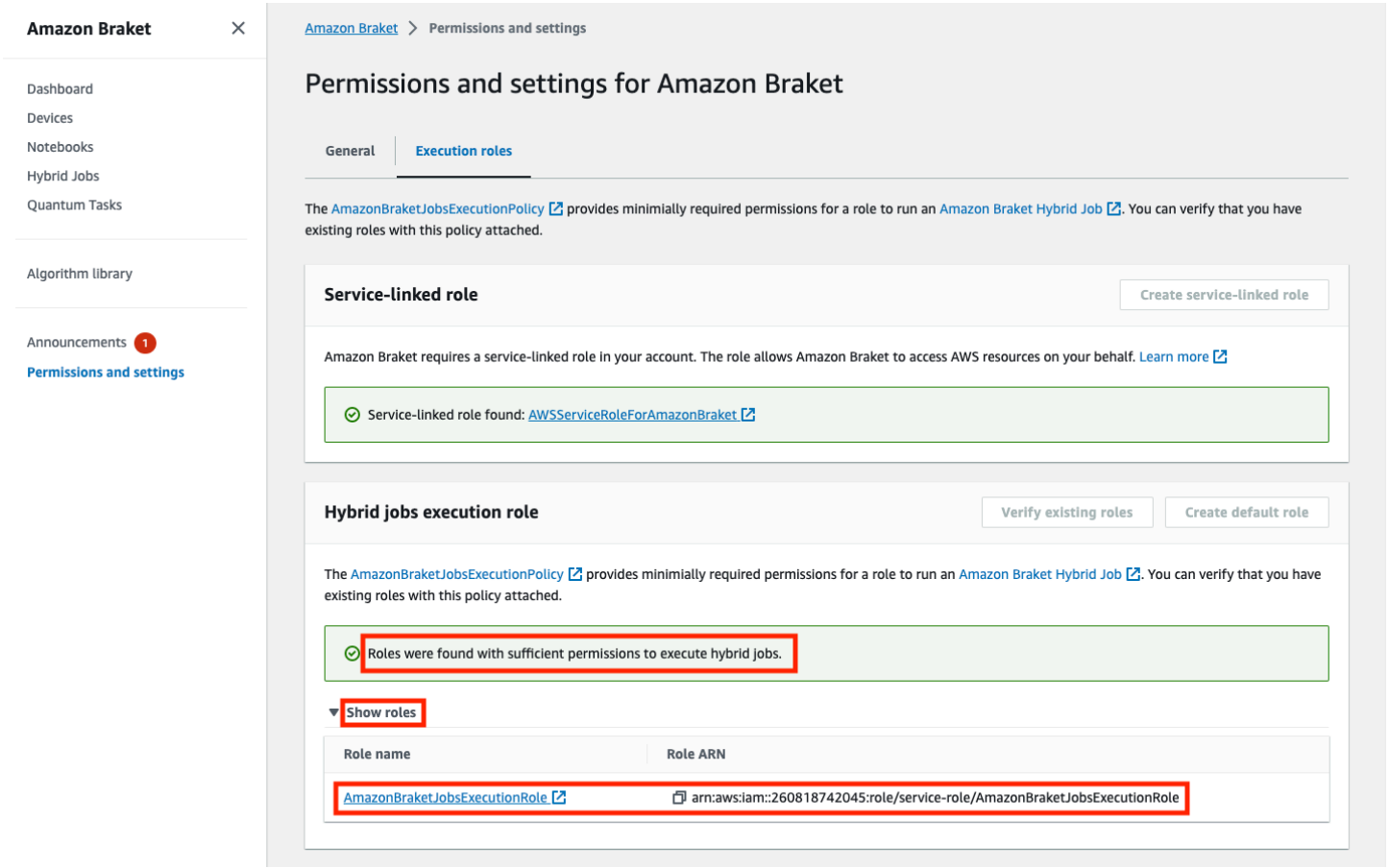
- 設定權限
- 建立並執行
- 監控結果

設定權限

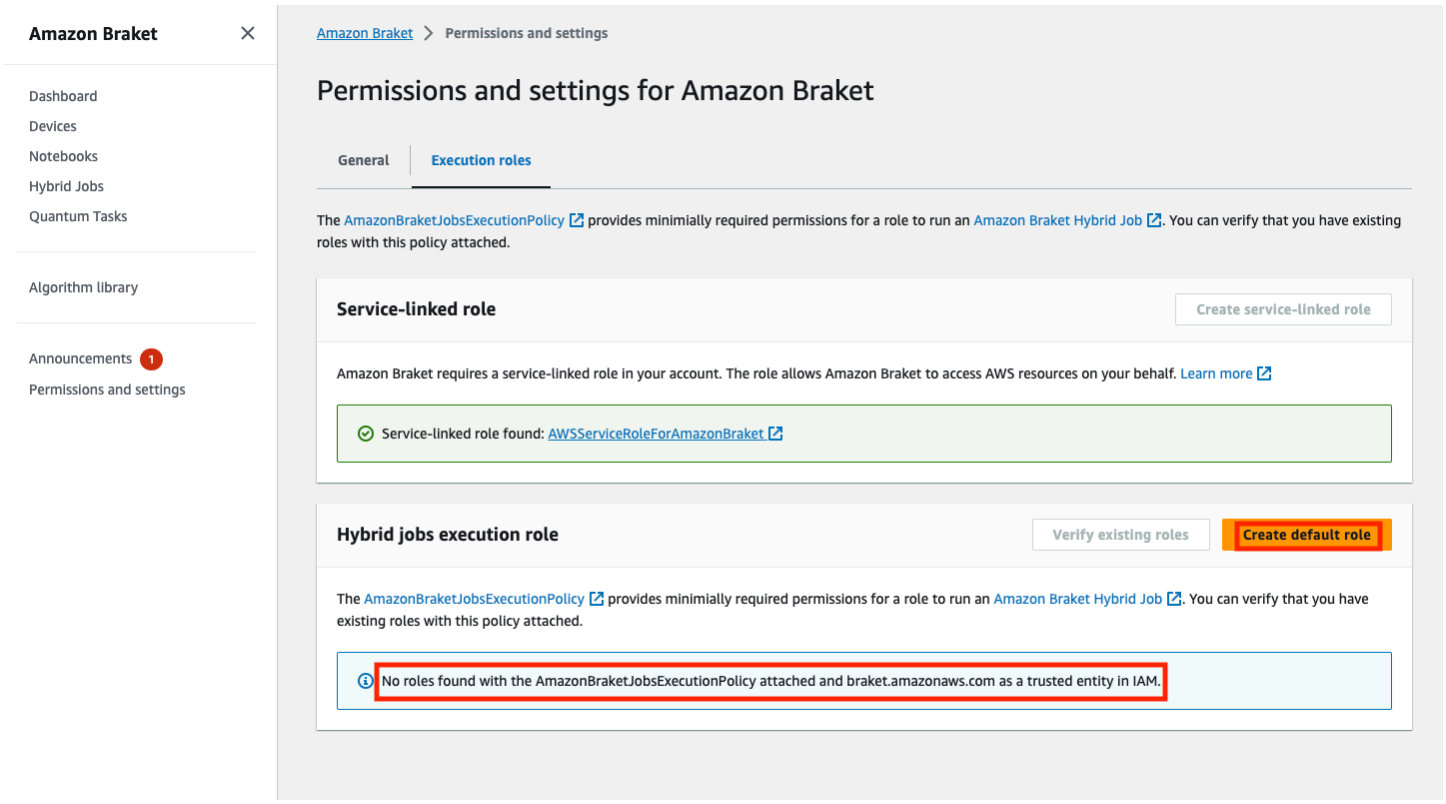
執行第一個混合式工作之前，您必須確定您有足夠的權限可以繼續執行此工作。要確定您擁有正確的權限，請從 Braket 控制台左側的菜單中選擇「權限」。Amazon Braket 的許可管理頁面可協助您確認其中一個現有角色是否具有足以執行混合任務的許可，或引導您建立可用於執行混合任務的預設角色 (如果您尚未擁有此類角色)。



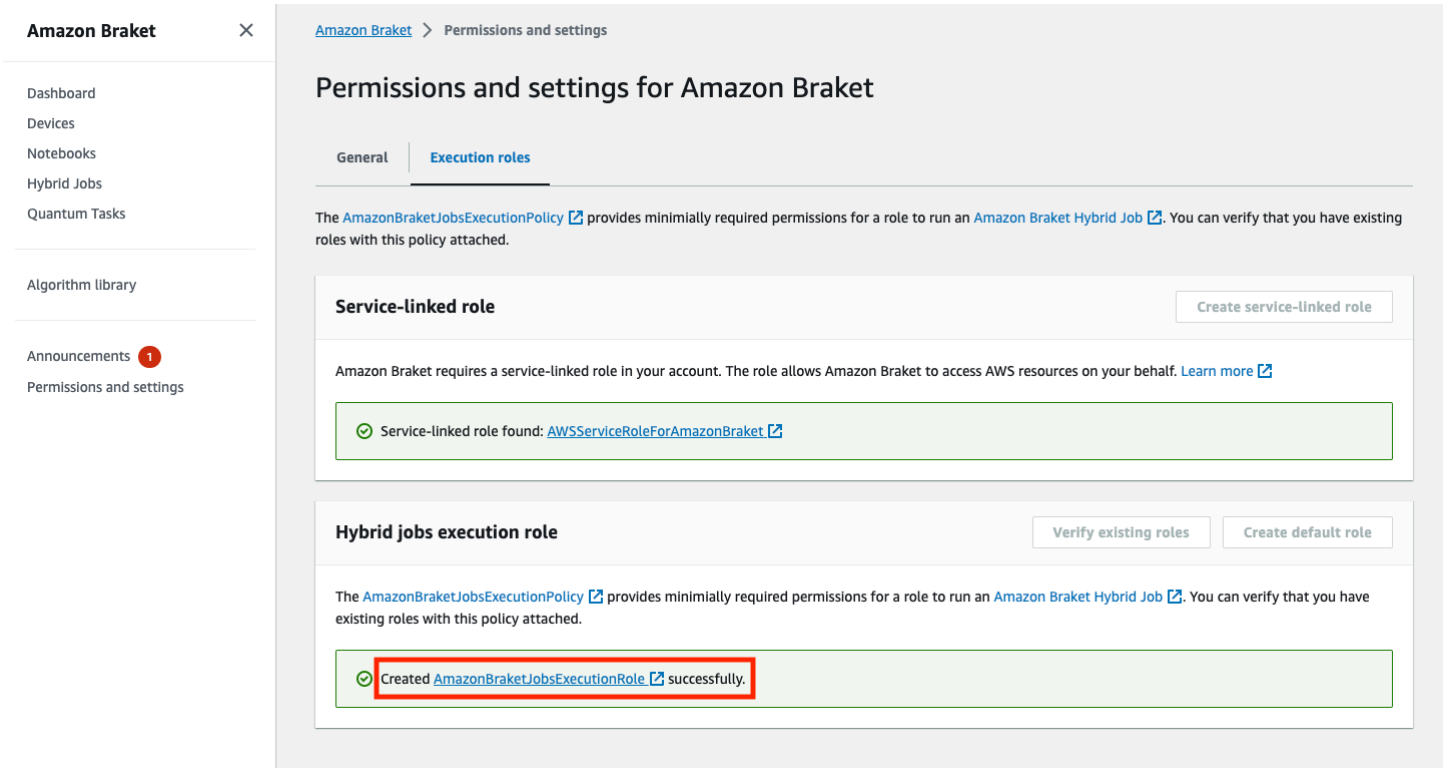
若要確認您的角色具有足夠權限來執行混合式工作，請選取驗證現有角色按鈕。如果這樣做，您會收到找到角色的訊息。要查看角色的名稱及其角色 ARN，請選擇「顯示角色」按鈕。



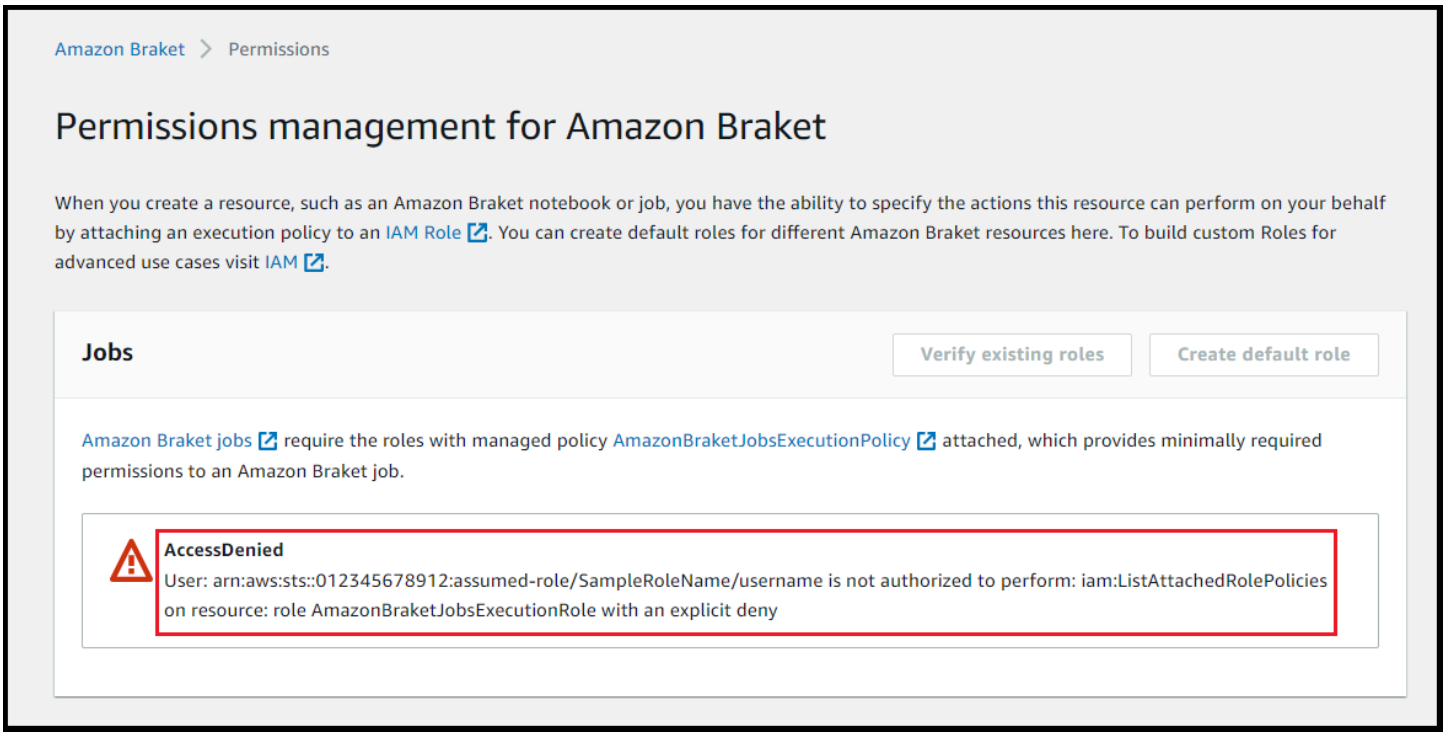
如果您沒有足夠權限執行混合工作的角色，您會收到一則訊息，指出找不到此類角色。選取 [建立預設角色] 按鈕以取得具有足夠權限的角色。



如果成功建立角色，您會收到一則訊息，確認這一點。

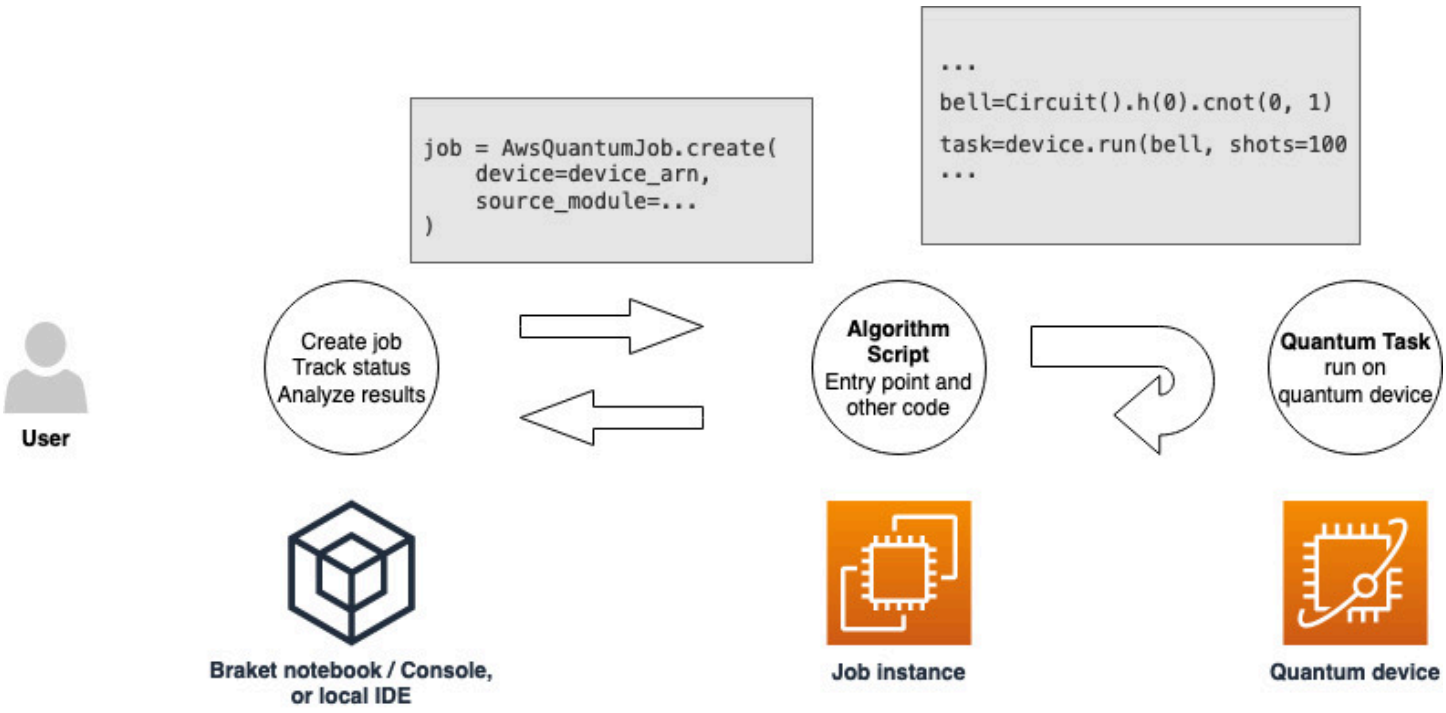


如果您沒有進行此查詢的權限，您將被拒絕訪問。在此情況下，請連絡您的內部 AWS 管理員。



建立並執行

一旦您擁有具有執行混合式工作權限的角色，就可以繼續進行了。您的第一個 Braket 混合作業的關鍵部分是算法腳本。它定義了您想要運行的算法，並包含經典邏輯和量子任務，這些任務是算法的一部分。除了演算法指令碼之外，您還可以提供其他相依性檔案。算法腳本及其依賴關係被稱為源模塊。進入點定義了混合式工作啟動時要在來源模組中執行的第一個檔案或函數。



首先，請考慮下列演算法指令碼的基本範例，該範例會建立五個鈴鐺狀態並列印對應的量測結果。

```
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!")
```

將此檔案名稱為 algorithm_script.py 儲存在 Braket 筆記本或本機環境的目前工作目錄中。該 algorithm_script.py 文件具有`start_here()`作為計劃的入口點。

接下來，在與 algorithm_script.py 文件相同的目錄中創建一個 Python 文件或 Python 筆記本。此指令碼會啟動混合作業，並處理任何非同步處理，例如列印我們感興趣的狀態或關鍵結果。此指令碼至少需要指定混合式工作指令碼和主要裝置。

> **Note**
>
> 如需有關如何在筆記本的相同目錄中建立 Braket 筆記本或上傳檔案 (例如 algorithm_script.py 檔案) 的詳細資訊，請參閱使用 Amazon Braket Python SDK 執行您的第一個電路

對於這個基本的第一種情況，你定位一個模擬器。無論您鎖定哪種類型的量子裝置、模擬器或實際的量子處理單元 (QPU)，您在下列指令碼`device`中指定的裝置都可用來排程混合工作，並且可供演算法指令碼做為環境變數`AMZN_BRAKET_DEVICE_ARN`使用。

> **Note**
>
> 您只能使用混合式工作中可用 AWS 區域 的裝置。Amazon Braket SDK 會 auto 選擇這個 AWS 區域。例如，us-east-1 中的混合式工作可以使用IonQ、、和TN1裝置 SV1DM1，但不能使用裝置。Rigetti

如果您選擇量子計算機而不是模擬器，Braket 會安排您的混合任務以優先訪問運行所有量子任務。

```
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
)
```

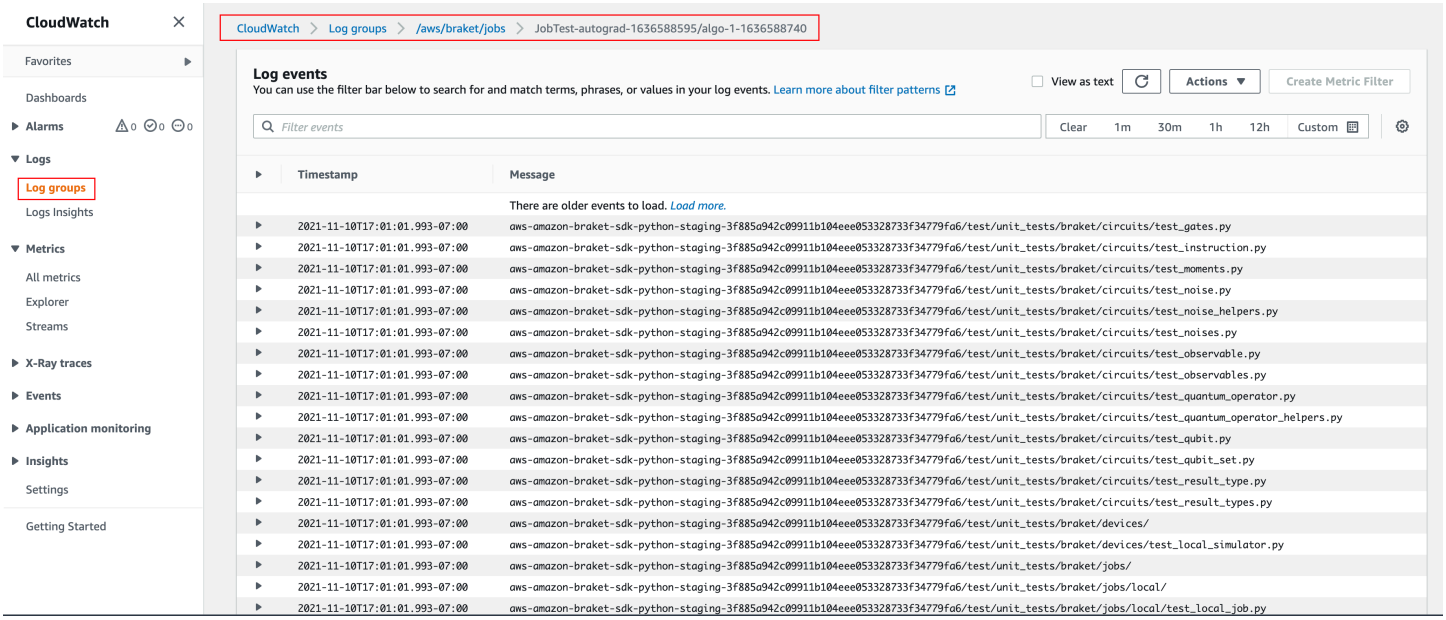
此參數會wait_until_complete=True設定詳細模式，以便您的工作在執行時列印實際工作的輸出。您應該會看到類似下列範例的輸出。

```
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
```

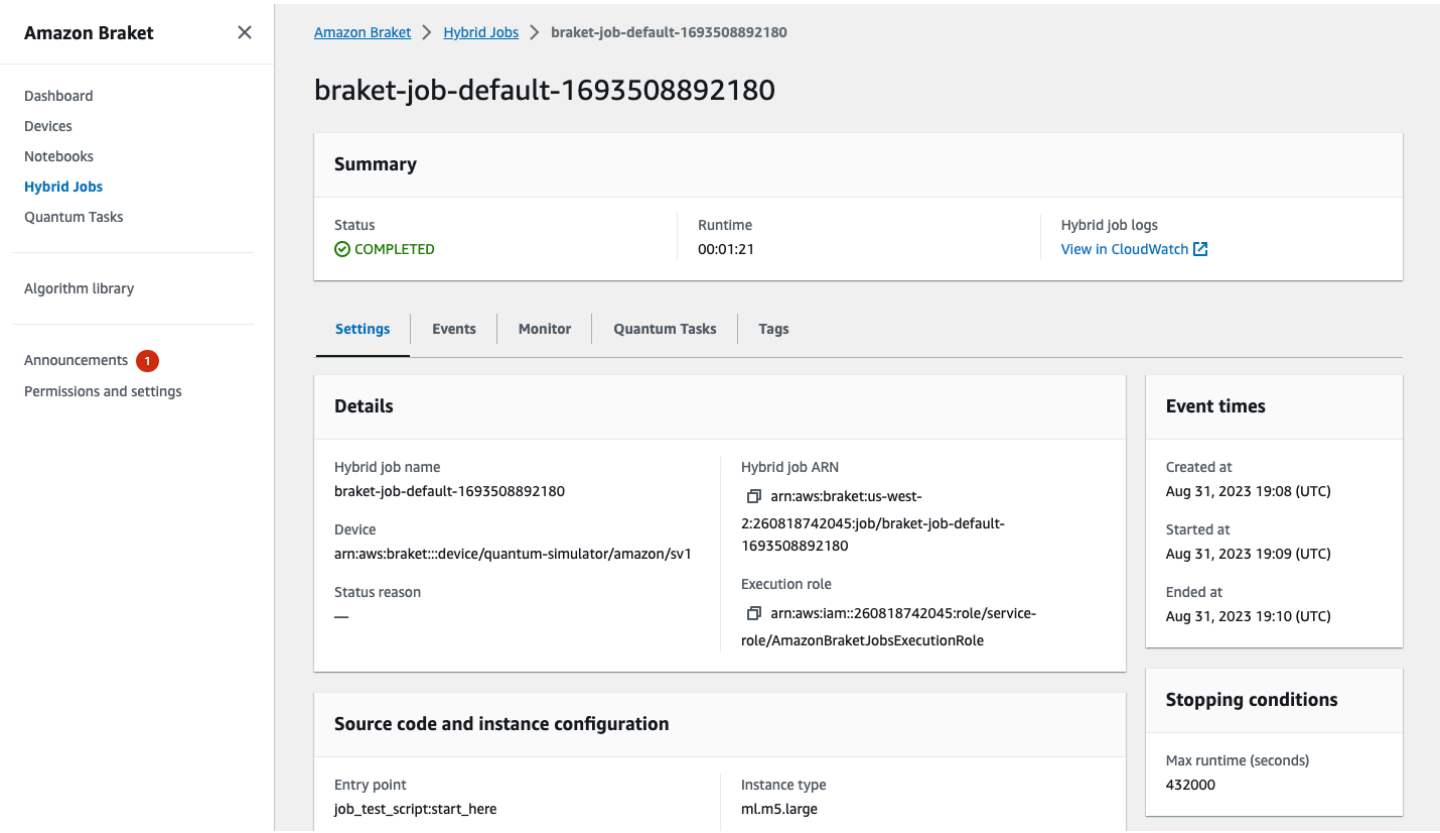
> **Note**
>
> 您也可以將自訂模組與 AwsQuantumJob.create 方法搭配使用，方法是傳遞其位置 (本機目錄或檔案的路徑，或 tar.gz 檔案的 S3 URI)。有關工作示例，請參閱 Amazon Braket 示例 Github 存儲庫中混合任務文件夾中的平行集訓練 _For_QML.IPynb 文件。

監控結果

或者，您可以從 Amazon 訪問日誌輸出 CloudWatch。若要這麼做，請移至工作詳細資料頁面左側功能表的 [記錄群組] 索引標籤，選取記錄群組aws/braket/jobs，然後選擇包含工作名稱的記錄資料流。在上述範例中，即為 braket-job-default-1631915042705/algo-1-1631915190。



您也可以選取 [混合式工作] 頁面，然後選擇 [設定]，在主控台中檢視混合工作的狀態。



您的混合任務在 Amazon S3 執行時會產生一些成品。預設的 S3 儲存貯體名稱為，amazon-braket-<region>-<accountid>且內容位於目jobs/<jobname>/<timestamp>錄中。您可以在使用 Braket Python SDK 建立混合任務code_location時指定不同的位置，來設定儲存這些成品的 S3 位置。

> **Note**
>
> 此 S3 儲存貯體必須與您的工作指令碼位於相同 AWS 區域 的位置。

目jobs/<jobname>/<timestamp>錄包含一個子資料夾，其中包含來自檔案中入口點腳本model.tar.gz檔的輸出。還有一個名為的目錄script，在文件中包含您的算法腳本工source.tar.gz件。實際量子任務的結果位於名為的目錄中jobs/<jobname>/tasks。

輸入、輸出、環境變數和輔助函數

除了構成完整演算法指令碼的檔案或檔案之外，混合式工作還可以有其他輸入和輸出。當您的混合任務開始時，AmazonBraket 會將混合任務建立過程中提供的輸入複製到執行演算法指令碼的容器中。混合任務完成後，演算法期間定義的所有輸出都會複製到指定的 Amazon S3 位置。

> **Note**
>
> 演算法指標會即時報告，且不遵循此輸出程序。

AmazonBraket 還提供了幾個環境變量和輔助函數，以簡化與容器輸入和輸出的交互。

本節說明 Amazon Braket Python SDK 所提供之`AwsQuantumJob.create`函式的關鍵概念，以及它們對應至容器檔案結構的關鍵概念。

在本節中：

- 輸入
- 輸出
- 環境變數
- 輔助函數

輸入

輸入數據：輸入數據可以通過指定輸入數據文件提供給混合算法，該文件設置為字典，帶有`input_data`參數。使用者在 SDK 中的`AwsQuantumJob.create`函`input_data`數內定義引數。這會將輸入資料複製到容器檔案系統，位於環境變數所指定的位置`"AMZN_BRAKET_INPUT_DIR"`。如需如何在混合演算法中使用輸入資料的幾個範例，請參閱 Amazon Braket 混合式任務的 QAOA 以 PennyLane及 Amazon Braket 混合式工作 J upyter 筆記型電腦中的量子機器學習。

> **Note**
>
> 當輸入資料較大 (>1GB) 時，在提交混合工作之前會有很長的等待時間。這是因為本機輸入資料會先上傳到 S3 儲存貯體，然後 S3 路徑會新增至混合任務請求，最後將混合任務請求提交至 Braket 服務。

超參數：如果您傳入`hyperparameters`，它們可以在環境變`"AMZN_BRAKET_HP_FILE"`數下使用。

> **Note**
>
> 如需如何建立超參數和輸入資料，然後將此資訊傳遞至混合式工作指令碼的詳細資訊，請參閱使用超參數一節和此 github 頁面。

檢查點：若要指定job-arn要在新混合工作中使用的檢查點，請使用指copy_checkpoints_from_job令。此命令會將檢查點資料複製到新checkpoint_configs3Uri的混合式工作，使其可在工作執行AMZN_BRAKET_CHECKPOINT_DIR時於環境變數指定的路徑中使用。預設值為None，表示新的混合式工作不會使用來自另一個混合工作的檢查點資料。

輸出

量子任務：量子任務結果存放在 S3 位置s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks。

Job 結果：演算法指令碼儲存至環境變數指定之目錄的所有內容"AMZN_BRAKET_JOB_RESULTS_DIR"都會複製到中指定的 S3 位置output_data_config。如果未指定此值，則預設值為s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data。我們提供 SDK 輔助函數 **save_job_result**，當從算法腳本調用時，您可以使用它以字典的形式方便地存儲結果。

檢查點：如果要使用檢查點，可以將它們儲存在環境變數所指定的目錄中。"AMZN_BRAKET_CHECKPOINT_DIR"您也可以使用 SDK 輔助函數來save_job_checkpoint代替。

演算法指標：您可以將演算法指標定義為演算法指令碼的一部分，這些指標會在混合任務執行時傳送至 Amazon， CloudWatch 並在 Amazon Braket 主控台中即時顯示。如需如何使用演算法指標的範例，請參閱使用 Amazon Braket 混合任務執行 QAOA 演算法。

環境變數

AmazonBraket 提供了幾個環境變量，以簡化與容器輸入和輸出的交互。下面的代碼列出了 Braket 使用的環境變量。

```
# 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"]
```

輔助函數

AmazonBraket 提供了幾個輔助功能，以簡化與容器輸入和輸出的交互。這些輔助函數將從用於運行混合 Job 的算法腳本中調用。下面的例子演示了如何使用它們。

```
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
```

儲存工作結果

您可以儲存演算法指令碼所產生的結果，以便可從混合任務指令碼中的混合任務物件以及 Amazon S3 的輸出資料夾取得結果 (位於名為 model.tar.gz 的 tar-zip 壓縮檔案中)。

輸出必須使用 JavaScript 物件標記法 (JSON) 格式儲存在檔案中。如果數據不能很容易地序列化為文本，就像在 numpy 數組的情況下，您可以傳入一個選項以使用醃製數據格式序列化。有關更多詳細信息，請參見編輯器 .jobs.data _持久性模塊。

若要儲存混合式工作的結果，您可以在演算法指令碼中加入以 #ADD 註解的下列幾行。

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

接著，您可以透過附加**print(job.result())**註解 #ADD 的行來顯示工作命令檔中的工作結果。

```
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
```

在這個例子中，我們已經刪除wait_until_complete=True了抑制詳細輸出。您可以將其重新加入以進行偵錯。當您執行此混合作業時，它會每隔 10 秒輸出一次識別碼和job-arn，然後輸出混合作業的狀態，直到混合作業為止COMPLETED，之後會顯示鈴聲電路的結果。請參閱以下範例。

```
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}]}
```

使用檢查點儲存並重新啟動混合式工作

您可以使用檢查點儲存混合式工作的中間重複項目。在上一節的演算法指令碼範例中，您可以新增以#ADD 註解的下列幾行，以建立檢查點檔案。

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

當您執行混合作業時，它會使用預設路徑在檢查點目錄中的混合作業成品中建立檔案-checkpoint 1.json <jobname>。/opt/jobs/checkpoints除非您要變更此預設路徑，否則混合式工作命令檔會保持不變。

如果您想要從先前混合作業產生的檢查點載入混合作業，演算法指令碼會使用from braket.jobs import load_job_checkpoint。在演算法指令碼中載入的邏輯如下。

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

載入此檢查點之後，您可以根據載入的內容繼續邏輯checkpoint-1。

> **Note**
>
> 檢查點 _file_suffix 必須在創建檢查點時先前指定的後綴匹配。

您的協調流程指令碼需要指定job-arn先前的混合工作，並加上 #ADD 註解行。

```
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
```

```
)
```

定義演算法指令碼的環境

AmazonBraket 支援由容器為您的演算法指令碼定義的三種環境：

- 基底容器 (預設值，如果沒`image_uri`有指定)
- 一個帶有張量流和 PennyLane
- 具有 PyTorch 和的容器 PennyLane

下表提供有關容器及其包含之程式庫的詳細資訊。

Amazon Braket 容器

Type	PennyLane 與 TensorFlow	PennyLane 與 PyTorch	薄荷油
Base	我們東amazon-braket-tensorflow-jobs部-阿馬遜網站/: 最新	我們西部-amazon-braket-pytorch-jobs 亞馬遜公司/: 最新	我們西部-amazon-braket-base-jobs 亞馬遜公司/: 最新
繼承程式庫	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy	
其他圖書館	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • 伊皮內核 • 喀拉斯 • 馬特洛特利卜	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • 伊皮內核 • 喀拉斯 • 馬特洛特利卜	• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • boto3 • 伊皮內核 • 馬特洛特利卜

Type	PennyLane 與 TensorFlow	PennyLane 與 PyTorch	薄荷油
	• 網絡 • 開放巴貝爾 • PennyLane • 原生物 • psi4 • rsa • PennyLane-閃電 GPU • 量子	• 網絡 • 開放巴貝爾 • PennyLane • 原生物 • psi4 • rsa • PennyLane-閃電 GPU • 量子	• 網絡 • numpy • 開放巴貝爾 • pandas • PennyLane • 原生物 • psi4 • rsa • scipy

您可以在 aws/ amazon-braket-containers 中檢視和存取開放原始碼容器定義。選擇最符合您使用案例的容器。容器必須位於您呼叫混合工作的 AWS 區域 來源中。您可以在建立混合式工作指令碼時，將下列三個引數其中一個新增至您的`create(…)` 呼叫，以指定容器映像檔。您可以將其他依賴項安裝到您在運行時選擇的容器中（以啟動或運行時為代價），因為 Amazon Braket 容器具有互聯網連接。下列範例適用於 us-west-2 區域。

- 基本影amazon-braket-base-jobs像圖片 = "292282985366.dr. 我們-西部-2. 亞馬遜. COM /: 1.0-CPU-匹配 39 英寸
- 張量流圖像圖片 = "292282985366.dkr.dr. 美國東部-1. 亞馬遜. COM/: 2.11.0-克 PU-聚氨酯 -39-cu112-比例 20.04」amazon-braket-tensorflow-jobs
- PyTorch 圖像amazon-braket-pytorch-jobs圖片 = "292282985366.dr. 我們-西部-2. 亞馬遜. COM/: 1.13.1-克普 -39-cu117-比特幣 20.04」

也`image-uris`可以使用開發套件中的`retrieve_image()`功能來擷取。Amazon下面的例子演示了如何從 us-we AWS 區域 st-2 檢索它們。

```
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")
```

使用超參數

建立混合式工作時，您可以定義演算法所需的超參數，例如學習速率或步數大小。超參數值通常用於控制演算法的各個層面，而且通常可以調整以最佳化演算法的效能。若要在 Braket 混合工作中使用超參數，您需要明確指定其名稱和值作為字典。請注意，這些值必須是字符串數據類型。搜尋最佳值集時，您可以指定要測試的超參數值。使用超參數的第一個步驟是將超參數設定並定義為字典，可在下列程式碼中看到：

```
#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
}
```

然後，您將傳遞上面給出的代碼片段中定義的超參數，以便在您選擇的算法中使用，並具有如下所示的內容：

```
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**
>
> 若要進一步瞭解輸入資料，請參閱「輸入」區段。

然後，超參數會使用下列程式碼載入混合式工作指令碼中：

```
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**
>
> 有關如何將輸入數據和設備 arn 等信息傳遞給混合作業腳本的更多信息，請參閱此 github 頁面。

QAOA 在 Amazon Braket 混合任務教程中提供了一些對於學習如何使用超參數非常有用的指南與 Amazon Braket 混合任務 PennyLane和量子機器學習。

設定混合式工作執行個體以執行演算法指令碼

根據您的算法，您可能有不同的要求。根據預設，AmazonBraket 會在執行個`ml.m5.large`體上執行您的演算法指令碼。不過，當您使用下列匯入和組態引數建立混合式工作時，您可以自訂此執行個體類型。

```
from braket.jobs.config import InstanceConfig

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

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

如果您正在執行內嵌模擬，並且已在裝置組態中指定了本機裝置，則可以 InstanceConfig 透過指定 instanceCount 並將其設定為大於一個，在中另外要求一個以上的執行個體。上限為 5。例如，您可以選擇 3 個實例，如下所示。

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

當您使用多個執行個體時，請考慮使用資料 parallel 功能來分配混合式工作。有關操作方法的更多詳細信息，請參閱下面的示例筆記本。

下表列出標準、運算最佳化和加速運算執行個體的可用執行個體類型和規格。

> **Note**
>
> 若要檢視混合式工作的預設傳統運算執行個體配額，請參閱此頁面。

標準執行個體	vCPU	記憶體
毫升 5. 大 (預設值)	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

標準執行個體	vCPU	記憶體
ml.m4.xlarge	4	16 GiB
ml.m4.2xlarge	8	32 GiB
ml.m4.4xlarge	16	64 GiB
ml.m4.10xlarge	40	256 GiB

運算優化執行個體	vCPU	記憶體
ml.c4.xlarge	4	7.5 GiB
ml.c4.2xlarge	8	15 GiB
ml.c4.4xlarge	16	30 GiB
ml.c4.8xlarge	36	192 GiB
ml.c5.xlarge	4	8 GiB
ml.c5.2xlarge	8	16 GiB
ml.c5.4xlarge	16	32 GiB
ml.c5.9xlarge	36	72 GiB
ml.c5.18xlarge	72	144 GiB
毫升 c5n. 大	4	10.5 GiB 博
毫升 c5n.2 倍大	8	21 GiB
毫升 c5n.4 倍大	16	42 GiB
毫升 c5n.9 倍大	36	96 GiB
毫升 c5n.18 倍大	72	192 GiB

加速壓縮執行個體	vCPU	記憶體
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
ml.g4dn.8xlarge	32	128 GiB
ml.g4dn.12xlarge	48	192 GiB
ml.g4dn.16xlarge	64	256 GiB

> **Note**
>
> 在 us-west-1 中不提供 p3 執行個體。如果您的混合式工作無法佈建要求的 ML 運算容量，請使用其他區域。

每個執行個體使用 30 GB 的預設資料儲存 (SSD) 組態。但是，您可以使用與配置`instanceType`. 下列範例顯示如何將總儲存空間增加到 50 GB。

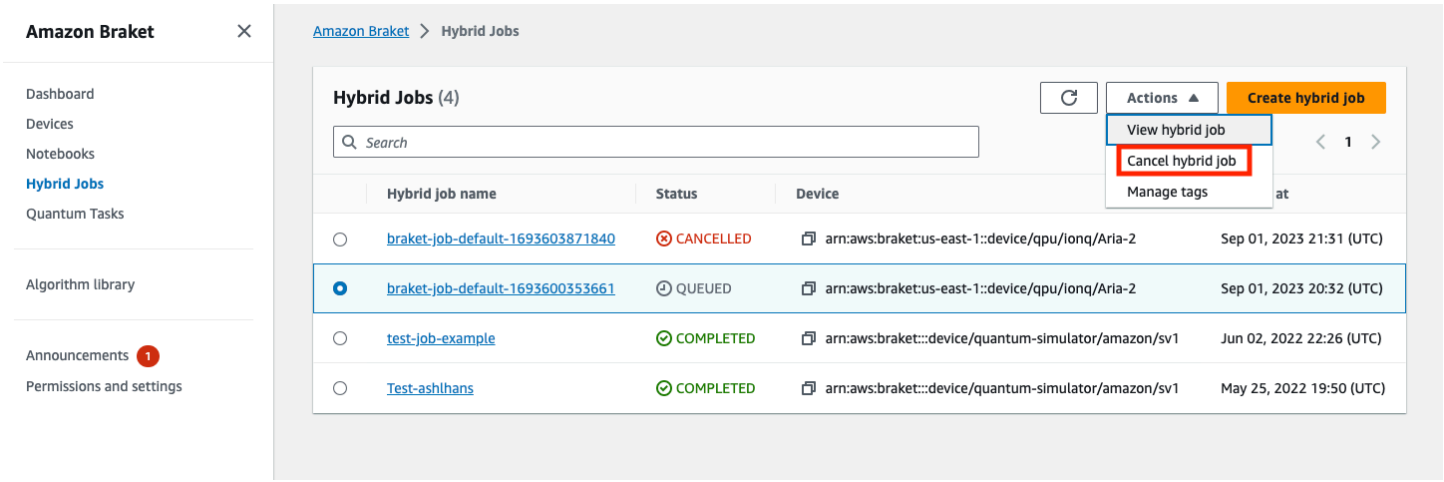
```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(
```

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

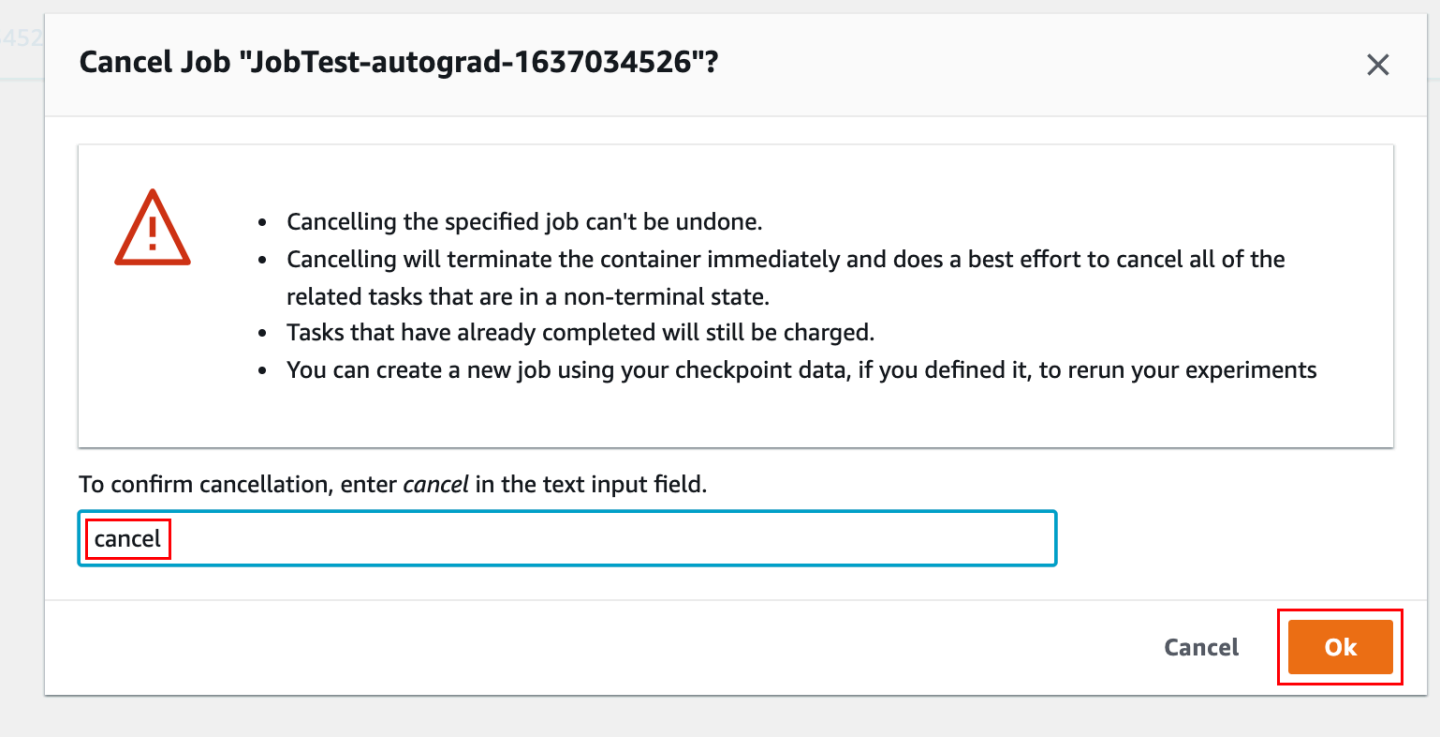
取消混合式 Job

您可能需要取消處於非終止狀態的混合式工作。這可以在控制台或代碼中完成。

若要在主控台中取消混合式工作，請從 [混合式工作] 頁面中選取要取消的混合式工作，然後從 [動作] 下拉式功能表中選取 [取消混合式工作]。



若要確認取消，請在出現提示時在輸入欄位中輸入 cancel，然後選取 [確定]。



要取消使用代碼從 Braket Python SDK 的混合作業，使用`job_arn`來識別混合作業，然後調用它的`cancel`命令，如下面的代碼。

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

該`cancel`命令立即終止傳統的混合作業容器，並盡最大努力取消所有仍處於非終端狀態的相關量子任務。

使用參數化編譯加速混合作業

Amazon Braket 支持某些 QPU 上的參數編譯。這使您可以通過僅編譯一次電路而不是混合算法中的每次迭代來減少與計算昂貴的編譯步驟相關的開銷。這樣可以大幅改善混合式工作的執行時間，因為您不需要在每個步驟中重新編譯電路。只需將參數化電路提交到我們支持的 QPU 之一，就可以作為 Braket 混合 Job。對於長時間執行的混合式工作，Braket 會在編譯電路時自動使用硬體供應商提供的更新校準資料，以確保最高品質的結果。

若要建立參數式電路，您首先需要在演算法指令碼中提供參數做為輸入。在這個例子中，我們使用一個小的參數電路，並忽略每次迭代之間的任何經典處理。對於典型的工作負載，您可以批次提交許多電路，並執行傳統處理，例如更新每次迭代中的參數。

```
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.")
```

您可以使用下列 Job 指令碼提交演算法指令碼，以「混合工作」的形式執行。在支援參數化編譯的 QPU 上執行混合 Job 時，僅在第一次執行時才會編譯電路。在接下來的執行中，會重複使用編譯的電路，以增加混合 Job 的執行階段效能，而不需要任何額外的程式碼行。

```
from braket.aws import AwsQuantumJob

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

> **Note**
>
> 除了脈衝級別程式之外，所有超導、閘極式 QPU Oxford Quantum Circuits 均支援參數編譯。Rigetti Computing

PennyLane 可搭配 Amazon Braket 使用

混合算法是包含傳統和量子指令的算法。經典指令在傳統硬件（EC2 實例或筆記本電腦）上運行，並且量子指令在模擬器或量子計算機上運行。建議您使用「混合式工作」功能執行混合演算法。如需詳細資訊，請參閱何時使用 Amazon Braket 工作。

AmazonBraket 可讓您在 Braket PennyLane 外掛程式的協助下，或使用 Amazon Amazon Braket Python SDK 和範例筆記型電腦儲存庫來設定和執行混合量子演算法。 Amazon基於 SDK 的 Braket 範例筆記本，可讓您在不使用 PennyLane 外掛程式的情況下設定和執行某些混合演算法。但是，我們建議使用， PennyLane 因為它提供了更豐富的體驗。

關於混合量子演算法

混合量子演算法對現今的產業很重要，因為當代量子運算裝置通常會產生雜訊，因此會產生錯誤。加入計算中的每個量子閘都會增加雜訊的機會；因此，長時間執行的演算法可能會因雜訊而不堪重負，這會導致計算錯誤。

純量子算法，如 Shor（量子相位估計示例）或格羅弗（格羅弗的例子）需要數千或數百萬次的操作。因此，它們對於現有的量子器件而言可能是不切實際的，這些量子器件通常稱為嘈雜的中間尺度量子（N ISQ）設備。

在混合量子演算法中，量子處理單元 (QPU) 可作為傳統 CPU 的協同處理器，特別是為了加速傳統演算法中的某些計算。在當今設備的功能範圍內，電路執行變得更短。

Amazon Braket PennyLane

AmazonBraket 提供了一個圍繞量子可差化編程概念構建的開源軟件框架的支持。PennyLane您可以使用此架構來訓練量子電路，就像訓練神經網路尋找量子化學、量子機器學習和最佳化中運算問題的解決方案一樣。

該 PennyLane 庫提供熟悉的機器學習工具的介面，包括 PyTorch 和 TensorFlow，使量子電路訓練快速且直觀。

- 圖 PennyLane 書館-預先安裝 PennyLane 在Amazon布拉克特筆記本中。若要從中存取 Amazon Braket 裝置 PennyLane，請開啟筆記本並使用下列指令匯入 PennyLane 資料庫。

```
import pennylane as qml
```

教學課程筆記本可協助您快速入門。或者，您可以從您選擇的 IDE PennyLane 上使用 Amazon Braket。

- Amazon布拉克特 PennyLane 插件-要使用自己的 IDE，您可以手動安裝 Amazon Braket PennyLane 插件。該插件 PennyLane 與 Amazon Braket Python 開發套件連接，因此您可以在 Amazon Braket 設備 PennyLane 上運行電路。要安裝 PennyLane 插件，請使用以下命令。

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

以下實例演示瞭如何在 PennyLane中設置對 Amazon Braket 設備的訪問權限：

```
# 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)
```

如需教學課程範例和詳細資訊 PennyLane，請參閱 Amazon Braket 範例儲存庫。

AmazonBraket PennyLane 插件使您可以通過一行代碼在 Amazon Braket QPU 和嵌入式模擬器設 PennyLane 備之間切換。它提供了兩個 Amazon Braket 量子設備可以使用 PennyLane：

- `braket.aws.qubit`用於與 Amazon Braket 服務的量子設備一起運行，包括 QPU 和模擬器
- `braket.local.qubit`用於使用 Amazon Braket SDK 的本地模擬器運行

Amazon布拉克特 PennyLane 插件是開源的。您可以從PennyLane 插件 GitHub 存儲庫安裝它。

如需詳細資訊 PennyLane，請參閱PennyLane 網站上的文件。

Amazon Braket 局範例筆記型電腦中的混合式

AmazonBraket 確實提供了各種不依賴 PennyLane 插件運行混合算法的示例筆記本電腦。您可以開始使用 Amazon Braket 混合式範例筆記型電腦，這些筆記型電腦說明變異方法，例如量子近似最佳化演算法 (QAOA) 或變分量子特徵求解析器 (VQE)。

標Amazon題範例筆記本電腦依賴於 Amazon Braket Python 開發套件。SDK 提供了一個框架，可通過 Amazon Braket 與量子計算硬件設備進行交互。它是一個開放原始碼程式庫，旨在協助您處理混合式工作流程的量子部分。

您可以使用我們的範例筆記本進一步探索 Amazon Braket。

含嵌入式 PennyLane 模擬器的混合演算法

AmazonBraket 混合作業現在配備了高性能 CPU 和 GPU 的嵌入式模擬器。PennyLane這個嵌入式模擬器系列可以直接嵌入到您的混合式工作容器中，包括快速狀態向量`lightning.qubit`模擬器、使用 NVIDIA CuQuantum 程式庫加速的`lightning.gpu`模擬器等。這些嵌入式模擬器非常適合用於變分算法，例如量子機器學習，可以從諸如伴隨差異化方法之類的先進方法中受益。您可以在一或多個 CPU 或 GPU 執行個體上執行這些內嵌模擬器。

有了混合式工作，您現在可以使用傳統協處理器和 QPU、Amazon Braket 隨選模擬器 (例如) 或直接使用嵌入式模擬器來執行變動演算法程式碼。SV1 PennyLane

嵌入式模擬器已經與混合作業容器一起使用，您只需要使用裝飾`@hybrid_job`器來裝飾您的主 Python 函數即可。若要使用 PennyLane `lightning.gpu`模擬器，您還需要在中指定 GPU 執行個體，`InstanceConfig`如下列程式碼片段所示：

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


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

請參閱範例筆記本，以開始使用具有混合式工作的 PennyLane 嵌入式模擬器。

PennyLane 與 Amazon Braket 模擬器相伴漸變

使用 Amazon Braket 的PennyLane外掛程式，您可以在本機狀態向量模擬器或 SV1 上執行時，使用隨附的差異化方法來計算漸層。

注意：要使用伴隨區分方法，您必須**`diff_method='device'`**在您的**`qnode`**，而不是指定。`diff_method='adjoint'`請參閱以下範例。

```
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將計算 QAOA 哈密頓人的分組索引，並使用它們將哈密頓分成多個期望值。如果您想要在執行 QAOA 時使用 SV1 的伴隨區分功能PennyLane，則需要移除分組索引來重建成本哈密頓軸，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 譯器混合任務並執 PennyLane行 QAOA 演算法

在本節中，您將使用您學到的知識來使用參數編譯來編寫實際 PennyLane 的混合程序。您可以使用演算法指令碼來解決量子近似最佳化演算法 (QAOA) 問題。該程序創建了一個與傳統的 Max Cut 優化問題相對應的成本函數，指定參數化的量子電路，並使用簡單的梯度下降方法來優化參數，從而使成本函數最小化。在此範例中，為了簡單起見，我們會在演算法指令碼中產生問題圖，但對於較典型的使用案例，最佳做法是透過輸入資料組態中的專屬通道提供問題規格。此旗標`parametrize_differentiable`預設為，`True`因此您可以透過支援的 QPU 上的參數編譯，自動獲得改善執行階段效能的好處。

```
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**
>
> 除了脈衝級別程式之外，所有超導、閘極式 QPU Oxford Quantum Circuits 均支援參數編譯。Rigetti Computing

利用內嵌式模擬器加速您的混合式工作負載 PennyLane

讓我們來看看如何使用 Amazon Braket 混合任務 PennyLane 上的內嵌模擬器來執行混合式工作負載。彭尼蘭基於 GPU 的嵌入式模擬器使用 Nvidia 庫量子庫來加速電路模擬。`lightning.gpu`嵌入式 GPU 模擬器已在所有 Braket 工作容器中預先配置，用戶可以立即使用。在此頁面中，我們將向您展示如何使用`lightning.gpu`來加速混合式工作負載。

用lightning.gpu於量子近似優化演算法工作負載

考慮本筆記本中的量子近似優化算法（QAOA）示例。要選擇一個嵌入的模擬器，您可以將device參數指定為以下格式的字符串："local:<provider>/<simulator_name>"。例如，您可以將設定"local:pennylane/lightning.gpu"為lightning.gpu。啟動時提供給「混合式 Job」的裝置字串會以環境變數的形式傳遞至工作"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)
```

在此頁面中，讓我們比較兩個嵌入式 PennyLane 狀態向量模擬器lightning.qubit（基於 CPU）和lightning.gpu（基於 GPU）。您需要為模擬器提供一些自定義門分解才能計算各種漸變。

現在您已準備好準備混合作業啟動指令碼。您將使用兩種執行個體類型來執行 QAOA 演算法：m5.2xlarge和。p3.2xlargem5.2xlarge執行個體類型與標準開發人員筆記型電腦相當。這p3.2xlarge是一個加速的計算實例，具有一個單一的 NVIDIA 沃爾特 GPU 與 16GB 的內存。

對hyperparameters於您所有的混合工作將是相同的。您需要做的就是嘗試不同的實例和模擬器，如下所示更改兩行。

```
# 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：

```
# 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**
>
> 如果您將使用 GPU 型執行個體指定instance_config為，但選擇做device為內嵌 CPU 型模擬器 (lightning.qubit)，則不會使用 GPU。如果您希望定位 GPU，請確保使用嵌入式 GPU 模擬器！

首先，您可以創建兩個混合作業，並在具有 18 個頂點的圖上使用 QAOA 解決最大切割。這可轉換成 18 量子位元電路，相對較小且可在筆記型電腦或執行個體上快速執行。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實例的平均迭代時間約為 25 秒，而對於p3.2xlarge實例來說，它大約是 12 秒。對於這個 18 量子位元工作流程，GPU 執行個體為我們提供了 2 倍的加速。如果您查看 Amazon Braket 混合任務定價頁面，您可以看到執行個體的每分鐘費用為 0.00768 USD，而m5.2xlarge執行個體的費用為 0.06375 美元。p3.2xlarge要運行 5 次總迭代，就像您在這裡一樣，使用 CPU 實例的費用為 0.016 美元，或使用 GPU 實例的 0.06375 美元-兩者都非常便宜！

現在讓我們更難解決問題，並嘗試在 24 頂點圖上解決最大切割問題，該圖將轉換為 24 個量子比特。在相同的兩個執行個體上再次執行混合式作業，並比較成本。

> **Note**
>
> 你會看到，在 CPU 實例上運行這個混合作業的時間可能是大約五個小時！

```
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實例的平均迭代時間大約是一個小時，而對於p3.2xlarge實例來說，它大約是兩分鐘。對於這個更大的問題，GPU 實例的速度要快一個數量級！為了從這種加速中受益，您所要做的就是更改兩行代碼，將實例類型和使用的本地模擬器交換出來。要運行 5 次總迭代，就像這裡所做的那樣，使用 CPU 實例的費用約為 2.27072 美元，或者使用 GPU 實例的費用約為 0.775625 美元。CPU 使用率不僅更昂貴，而且還需要更多的時間來運行。使用 NVIDIA 支援的嵌入式模擬器使用上 AWS PennyLane的 GPU 執行個體加速此工作流程 CuQuantum，可讓您以較少的總成本和更少的時間執行具有中間量子位元計數 (介於 20 到 30 之間) 的工作流程。這表示即使是在筆記型電腦或類似大小的執行個體上太大而無法快速執行的問題，您也可以嘗試量子運算。

量子機器學習與資料平行

如果您的工作負載類型是針對資料集進行訓練的量子機器學習 (QML)，則可以使用資料平行處理原則進一步加速工作負載。在 QML 中，該模型包含一個或多個量子電路。該模型可能包含也可能不包含經典的神經網絡。使用資料集訓練模型時，會更新模型中的參數以將遺失函數降到最低。損失函數通常是針對單一資料點定義的，以及整個資料集中平均損失的總損失。在 QML 中，損失通常是以序列方式計算，然後再將梯度計算為總損耗的平均值。此程序非常耗時，尤其是當有數百個資料點時。

由於一個數據點的損失不依賴於其他數據點，因此可以 parallel 評估損失！可以同時評估與不同資料點相關的損失和梯度。這就是所謂的資料平行處理原則。透 SageMaker過分散式資料 parallel 程式庫，Amazon Braket 混合任務可讓您更輕鬆地利用資料平行處理來加速訓練。

請考慮下列資料平行處理原則的 QML 工作負載，這些工作負載使用來自已知 UCI 儲存庫的 Sonar 資料集作為二進位分類的範例。聲納數據集有 208 個數據點，每個數據點都具有 60 個功能，這些功能是從聲納信號彈跳材料中收集的。每個資料點會標示為「M」(表示地雷) 或「R」表示岩石。我們的 QML 模型由輸入層、一個量子電路作為隱藏層和輸出層組成。輸入和輸出層是在中實現的經典神經網絡 PyTorch。量子電路與使用 PennyLane的 qml.qnn 模塊的 PyTorch 神經網絡集成在一起。如需工作負載的詳細資訊，請參閱範例筆記本。就像上面的 QAOA 範例一樣，您可以利用 GPU 的嵌入式模擬器`lightning.gpu`來提高內嵌式 CPU 模擬器的效能，藉此運用 GPU PennyLane 的強大功能。

若要建立混合工作，您可以透過其關鍵字引數呼叫`AwsQuantumJob.create`並指定演算法指令碼、裝置和其他組態。

```
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,
    ...
)
```

若要使用資料平行處理原則，您需要修改 SageMaker 分散式程式庫的演算法指令碼中的幾行程式碼，才能正確地平行化訓練。首先，您需要匯入套件，該`smdistributed`套件可以在多個 GPU 和多個執行個體之間分配工作負載，完成大部分工作負載的工作負載。該包裝在胸針 PyTorch 和 TensorFlow 容器中預先配置。該`dist`模塊告訴我們的算法腳本培訓（`world_size`）的 GPU 總數以及 GPU 核心`local_rank`的`rank`和。 `rank`是所有執行個體中 GPU 的絕對索引，而`local_rank`是執行個體內 GPU 的索引。例如，如果有四個執行個體，每個執行個體都配置了八個 GPU 供訓練使用，則`rank`範圍介於 0 到 31 之間，`local_rank`範圍介於 0 到 7 之間。

```
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)
```

接下來，您可以DistributedSampler根據 and 定義，world_sizerank然後將其傳遞至資料載入器。此取樣程式可避免 GPU 存取資料集的相同片段。

```
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,
)
```

接下來，您可以使用DistributedDataParallel類別來啟用資料平行處理原則。

```
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"])
```

以上是使用資料平行處理原則所需的變更。在 QML 中，您經常想要儲存結果和列印訓練進度。如果每個 GPU 都執行儲存和列印命令，記錄檔將會充斥重複的資訊，結果會相互覆寫。為了避免這種情況，您只能從具有 rank 0 的 GPU 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 混合式任務支援 SageMaker 分散式資料 parallel 程式庫的ml.p3.16xlarge執行個體類型。您可以透過「混合式工作」中的InstanceConfig引數設定執行個體類型。若要讓 SageMaker 分散式資料 parallel 程式庫知道資料平行處理原則已啟用，您需要新增兩個額外的超參數，"true"並"sagemaker_instance_type"將其"sagemaker_distributed_dataparallel_enabled"設定為您正在使用的執行個體類型。這兩個超參數由smdistributed封裝使用。您的算法腳本不需要明確使用它們。在 Amazon Braket SDK 中，它提供了一個方便的關鍵字參數distribution。透distribution="data_parallel"過建立混合任務，Amazon Braket SDK 會自動為您插入兩個超參數。如果您使用 Amazon Braket 輯器 API，則需要包括這兩個超參數。

設定執行個體和資料平行處理原則後，您現在可以提交混合式工作。一個執行個體中有 8 ml.p3.16xlarge 個 GPU。當您設定時instanceCount=1，工作負載會分散到執行個體中的 8 個 GPU 上。當您設定instanceCount大於一個時，工作負載會分散到所有執行個體中可用的 GPU。使用多個執行個體時，每個執行個體都會根據您使用的時間收取費用。例如，當您使用四個執行個體時，可計費時間是每個執行個體的四倍執行時間，因為有四個執行個體同時執行您的工作負載。

```
instance_config = InstanceConfig(instanceType='ml.p3.16xlarge',
                                 instanceCount=1,
)

hyperparameters={"nwires": "10",
                 "ndata": "32",
                 ...,
}

job = AwsQuantumJob.create(
    device="local:pennylane/lightning.gpu",
    source_module="qml_source",
    entry_point="qml_source.train_dp",
    hyperparameters=hyperparameters,
    instance_config=instance_config,
    distribution="data_parallel",
    ...
)
```

> **Note**
>
> 在上述混合作業創建中，train_dp.py是使用數據並行性修改的算法腳本。請記住，當您根據上述章節修改演算法指令碼時，資料平行處理原則才能正常運作。如果在沒有正確修改演算

法指令碼的情況下啟用資料平行處理原則選項，則混合工作可能會擲回錯誤，或者每個 GPU 可能會重複處理相同的資料片段，這是效率低下的。

讓我們在一個示例中比較運行時間和成本，其中使用 26 量子比特量子電路訓練模型以解決上述二進制分類問題。此範例中使用的`ml.p3.16xlarge`執行個體每分鐘費用為 0.4692 USD。如果沒有數據並行性，模擬器需要大約 45 分鐘的時間來訓練模型 1 個時代（即超過 208 個數據點），它的成本約為 20 美元。透過 1 個執行個體和 4 個執行個體的資料平行處理，只需 6 分鐘和 1.5 分鐘，兩者的轉換大約為 2.8 美元。透過跨 4 個執行個體使用資料並行處理原則，您不僅可以將執行時間縮短 30 倍，還可以將成本降低一個數量級！

使用本機模式建置和偵錯混合式工作

如果您正在構建新的混合算法，本地模式可幫助您調試和測試算法腳本。本機模式是一項功能，可讓您執行計劃在 Amazon Braket 混合作業中使用的程式碼，但不需要 Braket 管理執行混合式作業的基礎架構。相反地，您可以在 Braket Notebook 執行個體或偏好的用戶端 (例如筆記型電腦或桌上型電腦) 上執行混合式工作。在本機模式中，您仍然可以將量子工作傳送至實際裝置，但在本機模式下針對實際 QPU 執行時，無法獲得效能優勢。

若要使用本機模式，請修改`AwsQuantumJob`為發生此模式的位`LocalQuantumJob`置。例如，若要從 [建立您的第一個混合工作] 執行範例，請依照下列步驟編輯混合式工作指令碼。

```
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**
>
> 已預先安裝在 Amazon Braket 筆記型電腦中的 Docker，必須安裝在您的本機環境中才能使用此功能。有關安裝 Docker 的說明可以在這裡找到。此外，本機模式並非所有參數都受到支援。

攜帶您自己的容器 (BYOC)

Amazon Braket 混合任務提供三個預先建置的容器，可在不同的環境中執行程式碼。如果其中一個容器支援您的使用案例，則只需在建立混合作業時提供演算法指令碼。可以使用從算法腳本或`requirements.txt`文件中添加次要缺少的依賴關係`pip`。

如果這些容器都不支援您的使用案例，或者您希望擴充這些容器，Braket Hybrid Jobs 支援使用您自己的自訂Docker容器映像執行混合式作業，或自攜容器 (BYOC)。但是在深入研究之前，讓我們確保它實際上是適合您使用案例的正確功能。

什麼時候帶我自己的容器是正確的決定？

將您自己的容器 (BYOC) 帶到 Braket 混合作業中，可以靈活地使用您自己的軟體，方法是將其安裝在封裝的環境中。根據您的特定需求，可能有一些方法可以實現相同的靈活性，而無需完成完整的BYOC Docker構建-Amazon ECR 上傳-自訂映像 URI 週期。

> **Note**
>
> 如果您想要新增少量可公開使用的其他 Python 套件 (通常少於 10 個)，BYOC 可能不是正確的選擇。例如，如果您正在使用 PyPi.

在這種情況下，您可以使用其中一個預先構建的 Braket 圖像，然後在作業提交時將`requirements.txt`文件包含在源目錄中。檔案會自動讀取，並`pip`會如常一樣安裝具有指定版本的套件。如果您要安裝大量套件，工作的執行時間可能會大幅增加。檢查您要用來測試軟件是否可以工作的預構建容器的 Python 和 CUDA 版本（如果適用）。

當您想要為工作指令碼使用非 Python 語言 (例如 C++ 或 Rust)，或者您想要使用預先建置容器無法使用的 Python 版本時，BYOC 就是必要的。這也是一個不錯的選擇，如果：

- 您正在使用具有許可證密鑰的軟件，並且需要對許可服務器對該密鑰進行身份驗證才能運行該軟件。使用 BYOC，您可以將授權金鑰嵌入Docker映像中，並包含驗證程式碼。
- 您正在使用非公開提供的軟體。例如，該軟件託管在私有 GitLab 或 GitHub 存儲庫上，您需要特定的 SSH 密鑰才能訪問。
- 您需要安裝一套未打包在 Braket 提供的容器中的大型軟件。BYOC 可讓您消除因為軟體安裝而導致混合式作業容器的長時間啟動時間。

BYOC 還可讓您透過軟體建置Docker容器並將其提供給使用者，讓客戶可以使用自訂 SDK 或演算法。您可以透過在 Amazon ECR 中設定適當的許可來執行此操作。

> **Note**
>
> 您必須遵守所有適用的軟體授權。

攜帶自己的容器的食譜

在本節中，我們提供了 Braket Hybrid Jobs 所bring your own container (BYOC)需的 step-by-step 指南-腳本，文件和步驟來組合它們，以便啟動並運行您的自定義Docker映像。我們為兩種常見情況提供食譜：

1. 在Docker影像中安裝其他軟體，並在工作中僅使用 Python 演算法指令碼。
2. 使用以非 Python 語言撰寫的演算法指令碼搭配混合工作，或使用 x86 以外的 CPU 架構。

對於案例 2，定義容器條目腳本更為複雜。

Braket 執行混合 Job 務時，它會啟動請求的 Amazon EC2 執行個體數量和類型，然後執行Docker映像 URI 輸入指定的映像，以便在其上建立任務。使用 BYOC 功能時，您可以指定在您擁有讀取存取權限的私有 Amazon ECR 儲存庫中託管的映像 URI。Braket 混合作業會使用該自訂映像來執行工作。

建置可與混合工作搭配使用的Docker映像所需的特定元件。如果您不熟悉編寫和構建`Dockerfiles`，我們建議您在閱讀這些說明時根據需要參考 Dockerfile Amazon ECR CLI文檔和文檔。

以下是您需要的概述：

- 您的碼頭文件的基本圖像
- (選擇性) 修改後的容器進入點指令碼
- 安裝Dockerfile任何必要軟體並包含容器指令碼的

您的碼頭文件的基本圖像

如果您使用的是 Python，並且希望在 Braket 提供的容器中提供的軟件之上安裝軟件，則基本映像的選項是在我們的GitHub 存儲庫和 Amazon ECR 上託管的 Braket 容器映像之一。您將需要向 Amazon ECR 進行身份驗證，以提取映像並在其上建置。例如，BYOC Docker 檔案的第一行可能是：`FROM [IMAGE_URI_HERE]`

接下來，填寫其餘部分Dockerfile以安裝並設置要添加到容器的軟件。預先構建的 Braket 映像將已經包含適當的容器入口點腳本，因此您無需擔心包含該腳本。

如果您想要使用非 Python 語言，例如 C++、Rust 或 Julia，或者如果您想要為非 x86 CPU 架構 (例如 ARM) 建立映像檔，您可能需要在準系統公開映像檔之上建立映像檔。您可以在 Amazon 彈性容器註冊表公共圖庫中找到許多此類圖像。確保您選擇適合 CPU 架構的一種，並在必要時選擇要使用的 GPU。

(選擇性) 修改後的容器進入點指令碼

> **Note**
>
> 如果您只在預先構建的 Braket 映像中添加其他軟件，則可以跳過本節。

若要在混合作業中執行非 Python 程式碼，您需要修改定義容器入口點的 Python 指令碼。例如，Amazon Braket Github 上的 `braket_container.py` python 腳本。這是 Braket 預先構建的圖像的腳本，用於啟動算法腳本並設置適當的環境變量。容器入口點腳本本身必須在 Python 中，但可以啟動非 Python 腳本。在預構建的示例中，您可以看到 Python 算法腳本作為 Python 子進程或作為全新進程啟動。透過修改此邏輯，您可以啟用入口點指令碼以啟動非 Python 演算法指令碼。例如，您可以根據副檔名結尾來修改`thekick_off_customer_script()`函數來啟動 Rust 程序。

您也可以選擇編寫一個全新的`braket_container.py`. 它應該將輸入資料、來源存檔和其他必要檔案從 Amazon S3 複製到容器中，並定義適當的環境變數。

安裝**`Dockerfile`**任何必要軟體並包含容器指令碼的

> **Note**
>
> 如果您使用預先構建的 Braket 映像作為Docker基礎映像，則容器腳本已經存在。

如果您在上一個步驟中建立了修改過的容器指令碼，則需要將其複製到容器中，並將環境變數定義`SAGEMAKER_PROGRAM`為`braket_container.py`，或者您已命名為新容器入口點指令碼的名稱。

以下是可讓您在 GPU 加速工作執行個體上使用 Julia 的範例：`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
```

此範例會下載並執行由提供的指令碼， AWS 以確保符合所有相關開放原始碼授權。例如，透過適當地歸因任何已安裝的程式碼，由. MIT license

如果您需要包含非公開程式碼，例如在私人 GitHub 或 GitLab 儲存庫中託管的程式碼，請勿在Docker映像中內嵌安全殼層金鑰來存取它。相反，在構建Docker Compose時使用Docker以允許訪問其構建的主機上的 SSH。有關更多信息，請參閱 Docker 中的安全使用 SSH 密鑰訪問私有 Github 存儲庫指南。

建立和上傳您的Docker圖片

如果定義正確Dockerfile，您現在可以按照步驟建立私有 Amazon ECR 儲存庫 (如果尚未存在)。您也可以建置、標記容器映像檔，並將其上傳至儲存庫。

您已準備好建立、標記和推送映像檔。請參閱 Docker 構建文檔以獲取選項的完整說明docker build和一些示例。

對於上面定義的示例文件，您可以運行：

```
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
```

指派適當的 Amazon ECR 許可

Braket Hybrid Jobs Docker映像檔必須託管在私有的 Amazon ECR 儲存庫中。根據預設，私有 Amazon ECR 存放庫不提供讀取存取權限給想要使用您影像的任何其他使用者，例如協作者或學生。Braket Hybrid Jobs IAM role您必須設定儲存庫原則，才能授與適當的權限。一般而言，只授予您想要存取影像的特定使用者和IAM角色的權限，而不允許任何具有影像的人image URI提取影像。

在您自己的容器中運行 Braket 混合作業

要使用自己的容器創建混合作業，請AwsQuantumJob.create()使用image_uri指定的參數調用。您可以使用 QPU、隨選模擬器，或在 Braket 混合作業提供的傳統處理器上本機執行程式碼。我們建議在真正的 QPU 上執行之前，先在 SV1、DM1 或 TN1 等模擬器上測試您的程式碼。

若要在傳統的處理器上執行程式碼，請透過更新來指定instanceCount您使用的InstanceConfig。instanceType請注意，如果您指定 instance_count > 1，則需要確保您的代碼可以在多個主機上運行。您可以選擇的執行個體數目上限為 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(instanceType="ml.p3.8xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

> **Note**
>
> 使用裝置 ARN 來追蹤您用作混合工作中繼資料的模擬器。可接受的值必須遵循格式device = "local:<provider>/<simulator_name>"。請記住，<provider>並且<simulator_name>必須僅由字母，數字，_-，和組成.。字串的長度限制為 256 個字元。
>
> 如果您打算使用 BYOC，但並未使用 Braket SDK 來建立量子任務，則應將環境變數的值傳遞給AMZN_BRAKET_JOB_TOKEN要求中的jobToken參數。CreateQuantumTask如果不這樣做，量子任務將不會獲得優先級，並作為常規的獨立量子任務計費。

在中設定預設值區 AwsSession

提供您自己AwsSession的選項可為您提供更大的靈活性，例如，在默認值區的位置。預設情況下，預設值區位置為AwsSessionf"amazon-braket-{id}-{region}"。但

是您可以在創建AwsSession. 使用者可以選擇性地使用參數名稱將AwsSession物件傳入AwsQuantumJob.create，aws_session如下列程式碼範例所示。

```
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
)
```

直接與混合式工作互動 API

您可以使用直接存取 Amazon Braket 混合任務並與之API互動。但是，API直接使用時，預設值和便利方法不可用。

> **Note**
>
> 我們強烈建議您使用 Amazon Braket 開發套件與 Amazon Braket 譯混合任務進 Python 互動。它提供方便的預設值和保護功能，協助您的混合式工作成功執行。

本主題涵蓋使用API. 如果您選擇使用 API，請記住，這種方法可能會更複雜，並準備好進行多次迭代以運行混合任務。

若要使用 API，您的帳戶應具有AmazonBraketFullAccess受管政策的角色。

> **Note**
>
> 如需如何透過AmazonBraketFullAccess受管政策取得角色的詳細資訊，請參閱啟用 Amazon Braket 頁面。

此外，您還需要執行角色。此角色將傳遞至服務。您可以使用 Amazon Braket 主控台建立角色。使用 [權限和設定] 頁面上的 [執行角色] 索引標籤，為混合式工作建立預設角色。

CreateJobAPI需要您指定混合式工作的所有必要參數。若要使用 Python，請將演算法指令碼檔案壓縮成 tar 套件 (例如 input.tar.gz 檔案)，然後執行下列指令碼。更新角括號 (<>) 內的程式碼部分，以符合您的帳戶資訊和進入點，這些資訊指定了混合式工作開始的路徑、檔案和方法。

```
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
    },
)
```

建立混合式工作後，您可以透過GetJobAPI或主控台存取混合式工作詳細資料。若要從執行程式createJob碼的 Python 工作階段取得混合作業詳細資訊，如前面的範例，請使用下列 Python 命令。

```
getJob = client.get_job(jobArn=job["jobArn"])
```

若要取消混合式工作，請CancelJobAPI使用該工作Amazon Resource Name的 ('JobArn') 呼叫。

```
cancelJob = client.cancel_job(jobArn=job["jobArn"])
```

您可以使用checkpointConfig參數將檢查點指定為createJobAPI的一部分。

```
checkpointConfig = {
    "localPath" : "/opt/omega/checkpoints",
    "s3Uri": f"s3://{bucket}/{s3_prefix}/checkpoints"
},
```

> **Note**
>
> 的 LocalPath checkpointConfig 不能以下列任何保留路徑開頭：/opt/ml/opt/braket、/tmp、或/usr/local/nvidia。

錯誤緩解

量子誤差緩解是一組旨在減少量子計算機錯誤的影響的技術。

量子器件受到環境噪聲影響，這會降低執行計算的質量。儘管容錯量子計算可以解決此問題，但目前的量子器件受到量子位數和相對較高的誤差率的限制。為了在短期內解決這個問題，研究人員正在研究提高雜訊量子計算準確性的方法。這種稱為量子誤差緩解的方法涉及使用各種技術，從雜訊的量測資料中擷取最佳訊號。

錯誤緩解 IonQ Aria

緩解錯誤包括運行多個物理電路，並將其測量結合以獲得改進的結果。該IonQ Aria設備具有一種稱為去偏置的錯誤緩解方法。

解偏置將電路映射為多個變體，這些變體起作用於不同的量子位排列或具有不同的門分解。通過使用不同的電路實現，否則可能導致偏置測量結果，從而減少了諸如閘極過旋轉或單個故障量子位元等系統誤差的影響。這需要額外的開銷來校準多個量子位和門。

如需有關解除偏移的詳細資訊，請參閱透過對稱化增強量子電腦效能。

> **Note**
>
> 使用去偏移至少需要 2500 張照片。

您可以使用以下代碼在設備上運行具有去偏IoQ Aria置的量子任務：

```
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
```

量子任務完成後，您可以查看量子任務中的測量概率和任何結果類型。所有變體的測量概率和計數會彙總成單一分佈。電路中指定的任何結果類型 (例如期望值) 均使用彙總測量計數來計算。

銳利化

您也可以存取使用稱為銳利化的不同後處理策略計算的量測機率。銳利化會比較每個變體的結果，並捨棄不一致的鏡頭，有利於跨變體最有可能的測量結果。如需詳細資訊，請參閱透過對稱化增強量子電腦效能。

重要的是，銳利化假定輸出分佈的形式是稀疏的，具有很少的高概率狀態和許多零概率狀態。如果這個假設無效，它可能會扭曲概率分佈。

您可以在 Braket Python SDK 中的`additional_metadata`欄位中存取銳利分佈的`GateModelTaskResult`機率。請注意，銳利化不會傳回度量計數，而是傳回重新標準化的機率分佈。下面的代碼片段演示了如何在銳利化後訪問分佈。

```
print(result.additional_metadata.ionqMetadata.sharpenedProbabilities)
>>> {"00": 0.51, "11": 0.549} # sharpened probabilities
```

布拉基直接

借助 Braket Direct，您可以預留對所選不同量子設備的專用訪問權限，與量子計算專家聯繫以獲得有關工作負載的指導，並儘早獲得新一代功能，例如可用性有限的新量子設備。

在本節中：

保留

保留使您可以獨家訪問您選擇的量子設備。您可以在方便的時候安排保留，以便確切知道工作負載何時開始和結束執行。預訂以 1 小時為單位提供，並可提前 48 小時取消，無需額外付費。您可以選擇為即將到來的預約提前將量子工作和混合式工作排入佇列，或在保留期間提交工作負載。

無論您在 Quantum 處理單元 (QPU) 上執行了多少個量子任務和混合任務，專用裝置存取的費用取決於您保留的持續時間。

以下量子計算機可供預訂：

- 伊奧諾 Q 的詠嘆調
- IQM 石榴石
- QuEra的拉奎拉
- 里盖蒂的阿斯彭 M-3

何時使用預訂

利用專用裝置存取與保留功能，讓您能夠確切知道量子工作負載何時開始和結束執行時間，方便且可預測。與隨需提交任務和混合任務相比，您不需要在其他客戶任務的隊列中等待。由於您可以在保留期間對裝置的獨佔存取權，因此只有您的工作負載會在整個保留區內在裝置上執行。

我們建議您在研究的設計和原型設計階段使用隨需存取，以便快速且符合成本效益的演算法迭代。準備好產生最終實驗結果後，請考慮在方便時排程裝置預約，以確保您能夠符合專案或發佈截止日期。當您希望在特定時間執行工作時，例如當您在量子電腦上執行現場示範或研討會時，我們也建議您使用保留。

在本節中：

建立保留區

若要建立保留區，請依照下列步驟聯絡 Braket 團隊：

1. 打開 Amazon Braket 控制台。
2. 在左窗格中選擇 Braket Direct，然後在「保留」區段中選擇「保留裝置」。
3. 選擇您要預約的裝置。
4. 提供您的聯繫信息，包括姓名和電子郵件。請務必提供您定期檢查的有效電子郵件地址。
5. 在 [告訴我們您的工作負載] 下，提供有關使用保留區執行之工作負載的任何詳細資料。例如，所需的保留長度、相關限制或所需的排程。
6. 如果您有興趣在確認預訂後與 Braket 專家聯繫以進行預訂準備會話，可選擇選擇「我對準備會話感興趣」。

您也可以按照以下步驟與我們聯繫以創建預訂：

1. 打開 Amazon Braket 控制台。
2. 在左窗格中選擇設備，然後選擇要保留的設備。
3. 在「摘要」區段中，選擇「保留裝置」。
4. 請遵循上一個程序中的步驟 4-6。

提交表格後，您會收到來自 Braket 團隊的電子郵件，其中包含創建預訂的後續步驟。預訂確認後，您將通過電子郵件收到預訂 ARN。

> **Note**
>
> 您的預訂只有在您收到預訂 ARN 後才能確認。

保留區以最少 1 小時為單位提供，而某些裝置可能會有額外的保留長度限制 (包括最短和最長保留期間)。在確認預訂之前，Braket 團隊將與您分享任何相關信息。

如果您對預訂準備會議表示有興趣，Braket 團隊會通過電子郵件與您聯繫，以安排與 Braket 專家進行 30 分鐘的會議。

使用保留區執行您的工作負載

在保留期間，只有您的工作負載會在裝置上執行。若要指定要在裝置保留期間執行的量子工作和混合式工作，您必須使用有效的保留區 ARN。

> **Note**
>
> 保留是 AWS 帳戶和設備特定的。只有建立保留區的 AWS 帳戶可以使用您的保留區 ARN。此外，保留 ARN 僅在選擇的開始和結束時間在保留的裝置上有效。

為了充分利用保留時間，您可以選擇在保留之前將任務和工作排入佇列。這些工作負載會保持狀QUEUED態，直到保留區開始為止。保留區開始時，任何佇列的工作負載都會按照提交的順序執行。Job 任務優先於獨立的量子任務。

> **Note**
>
> 由於只有您的工作負載在保留期間執行，因此使用保留區 ARN 提交的工作和工作不會顯示佇列。

為保留項目建立量子工作的程式碼範例：

1. 定義一個電路，以準備在開放式質量的 GHZ 狀態。

```
// 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. 使用電路和保留區 ARN 建立量子任務。

```
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>"
)
```

為 Braket Direct 保留區建立混合式工作的程式碼範例：

1. 定義您的演算法指令碼。

```
//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. 使用演算法指令碼和保留區 ARN 建立混合式工作。

```
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. 使用遠端裝飾器建立混合作業。

```
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
```

預訂結束時會發生什麼

預訂結束後，您將無法再擁有該裝置的專用存取權。系統會自動取消排入此保留區佇列的任何剩餘工作負載。

> **Note**
>
> 保留結束時處於RUNNING狀態的任何工作都將被取消。我們建議您在方便時使用檢查點來儲存和重新啟動工作。

正在進行的預訂，例如預訂開始後和預訂結束之前，無法延長，因為每個保留都代表獨立的專用裝置存取權限。例如，會將兩個 back-to-back 保留區視為單獨的，而第一個保留項目中的任何擱置工作都會自動取消。他們不會在第二次保留中恢復。

> **Note**
>
> 保留項目代表您 AWS 帳戶的專用裝置存取權。即使設備處於閒置狀態，也沒有其他客戶可以使用它。因此，無論使用時間如何，都會按照保留時間的長度收費。

取消或重新排程現有的預留

您可以在預定預訂開始時間前 48 小時內取消預訂。要取消，請回復您收到與取消請求一起收到的預訂確認電子郵件。

若要重新排程，您必須取消現有的預留，然後建立新的保留。

專家建議

直接在 Braket 管理主控台與量子運算專家 Connect 絡，以取得有關您工作負載的額外指引。

若要透過 Braket Direct 探索專家建議選項，請開啟 Braket 主控台，在左窗格中選擇「Braket Direct」，然後前往「專家建議」部分。以下是可用的專家建議選項：

- Braket 辦公時間：Braket 辦公時間為 1：1 會，先到先得，每個月進行。每個可用的辦公時間為 30 分鐘，免費。與 Braket 專家交談可透過探索 use-case-to-device 配合度、找出最佳利用 Braket 的選項，以及獲得有關如何使用某些 Braket 功能 (例如 Amazon Braket 混合任務、Braket Pulse 或類比哈密頓模擬) 的建議，協助您更快地從構想到執行。
  - 要註冊 Braket 辦公時間，請選擇註冊並填寫聯繫信息，工作負載詳細信息以及您想要的討論主題。
  - 您將通過電子郵件收到下一個可用位置的日曆邀請。

  > **Note**
  >
  > 對於緊急問題或快速疑難排解問題，我們建議您聯繫 Sup AWS port。對於非緊急問題，您也可以使用 AWS Re: Post 論壇或量子計算堆疊交換，您可以在其中瀏覽先前回答的問題並提出新問題。

- 量子硬體供應商產品：iONQ Oxford Quantum Circuits QuEra、、和 Rigettti 均透過. AWS Marketplace
  - 若要探索他們的產品，請選取 [Connect] 並瀏覽其清單。
  - 若要深入瞭解上的專業服務供應項目 AWS Marketplace，請參閱專業服務產品。
- Amazon量子解決方案實驗室（QSL）：QSL 是一個由量子計算專家組成的協作研究和專業服務團隊，他們可以幫助您有效地探索量子計算並評估該技術的當前性能。
  - 要聯繫 QSL，請選擇「Connect」，然後填寫聯繫信息和用例詳細信息。
  - QSL 團隊將通過電子郵件與您聯繫，並進行後續步驟。

實驗能力

為了提升您的研究工作負載，快速存取新的創新功能非常重要。借助 Braket Direct，您可以直接在 Braket 控制台中請求訪問可用的實驗功能，例如可用性有限的新量子設備。

某些實驗功能在標準裝置規格之外運作，因此需要針對您的使用案例量身打造的實作指導。為確保您的工作負載設定成功，可透過 Braket Direct 提出要求存取。

僅預約訪問 IonQ 复地

隨著布拉克特直接, 你得到預訂只訪問 IonQ 复地 QPU. 由於該設備的可用性有限，此設備僅可通過 Braket Direct 提供。

要了解更多信息並請求訪問 IonQ Forte，請按照以下步驟操作：

1. 打開 Amazon Braket 控制台。
2. 選擇直接在左側菜單中，然後在實驗功能中導航到 i ONQ 复地。選擇 [檢視裝置]。
3. 在 Forte 裝置詳細資料頁面的摘要中，選擇 [保留裝置]。
4. 提供您的聯繫信息，包括姓名和電子郵件。提供您定期檢查的有效電子郵件地址。
5. 在 [告訴我們您的工作負載] 下方，提供有關使用保留項目執行之工作負載的詳細資訊，例如所需的保留時間長度、相關限制或所需排程。
6. （可選）如果您有興趣在預訂確認後與 Braket 專家聯繫以進行預訂準備會話，請選擇我對準備會話感興趣。

提交表格後，Braket 團隊將與您聯繫並進行後續步驟。

> **Note**
>
> 由於裝置可用性有限，使用 Forte 的權限受到限制。聯繫我們以了解更多信息。

訪問阿奎拉上的本地解調 QuEra

使用 Braket Direct，您可以在 QPU 上進行編程時請求訪問以控制本地解調。QuEra Aquila使用此功能，您可以調整驅動場對每個特定量子位的影響程度。

若要深入瞭解並要求存取此功能，請依照下列步驟執行：

1. 打開 Amazon Braket 控制台。
2. 選擇布拉克特直接在左側菜單中，然後在實驗功能導航到QuEra 拉奎拉-本地解調。選擇「取得存取權」
3. 提供您的聯繫信息，包括姓名和電子郵件。提供您定期檢查的有效電子郵件地址。

4. 在 [告訴我們您的工作負載] 底下，提供有關工作負載以及您打算使用此功能的位置的詳細資訊。

訪問阿奎拉上 QuEra 的高大幾何形狀

使用 Braket Direct，您可以在 QPU 上進行編程時請求訪問擴展的幾何圖形。QuEra Aquila透過此功能，您可以嘗試超越標準裝置功能，並指定晶格高度增加的幾何圖形。

若要深入瞭解並要求存取此功能，請依照下列步驟執行：

1. 打開 Amazon Braket 控制台。
2. 選擇布拉克特直接在左側菜單中，然後在實驗功能導航到 QuEra Aquila-高大的幾何形狀。選擇取得存取權。
3. 提供您的聯繫信息，包括姓名和電子郵件。提供您定期檢查的有效電子郵件地址。
4. 在 [告訴我們您的工作負載] 底下，提供有關工作負載以及您打算使用此功能的位置的詳細資訊。

在 QuEra 拉奎拉上獲得緊密的幾何形狀

使用 Braket Direct，您可以在 QPU 上進行編程時請求訪問擴展的幾何圖形。QuEra Aquila透過此功能，您可以嘗試超越標準裝置功能，並以更緊密的垂直間距排列格子列。

若要深入瞭解並要求存取此功能，請依照下列步驟執行：

1. 打開 Amazon Braket 控制台。
2. 選擇布拉克特直接在左側菜單中，然後在實驗功能導航到 QuEra Aquila-高大的幾何形狀。選擇取得存取權。
3. 提供您的聯繫信息，包括姓名和電子郵件。提供您定期檢查的有效電子郵件地址。
4. 在 [告訴我們您的工作負載] 底下，提供有關工作負載以及您打算使用此功能的位置的詳細資訊。

記錄和監控

提交量子任務後，您可以通過 Amazon Braket SDK 和控制台跟踪其狀態。量子任務完成後，Braket 會將結果儲存在您指定的 Amazon S3 位置。完成可能需要一些時間，特別是對於 QPU 設備，具體取決於隊列的長度。狀態類型包括：

- CREATED-Amazon Braket 收到了您的量子任務。
- QUEUED— Amazon Braket 處理了您的量子任務，現在正在等待在設備上運行。
- RUNNING— 您的量子任務正在 QPU 或按需模擬器上運行。
- COMPLETED— 您的量子任務完成了在 QPU 或按需模擬器上運行。
- FAILED— 您的量子任務試圖運行並失敗. 根據您的量子任務失敗的原因，請嘗試再次提交量子任務。
- CANCELLED-你取消了量子任務 量子任務沒有運行。

在本節中：

從 Amazon Braket 開發套件追蹤量子任務

該命令`device.run(…)` 定義具有唯一量子任務 ID 的量子任務。您可以查詢和跟踪狀態，如下面的例子。`task.state()`

注意：`task = device.run()`是一個異步操作，這意味著您可以在系統在後台處理量子任務時繼續工作。

擷取結果

當您呼叫時task.result()，SDK 會開始輪詢 Amazon Braket 以查看量子任務是否已完成。SDK 會使用您在中定義的輪詢參數.run()。量子任務完成後，SDK 會從 S3 儲存貯體擷取結果，並將其作為QuantumTaskResult物件傳回。

```
# 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: RUNNING
Status: RUNNING
Status: COMPLETED
```

取消量子任務

若要取消量子工作，請呼叫cancel()方法，如下列範例所示。

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

```
Status of task: CANCELLING
```

檢查元數據

您可以檢查已完成量子工作的中繼資料，如下列範例所示。

```
# 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
```

擷取量子任務或結果

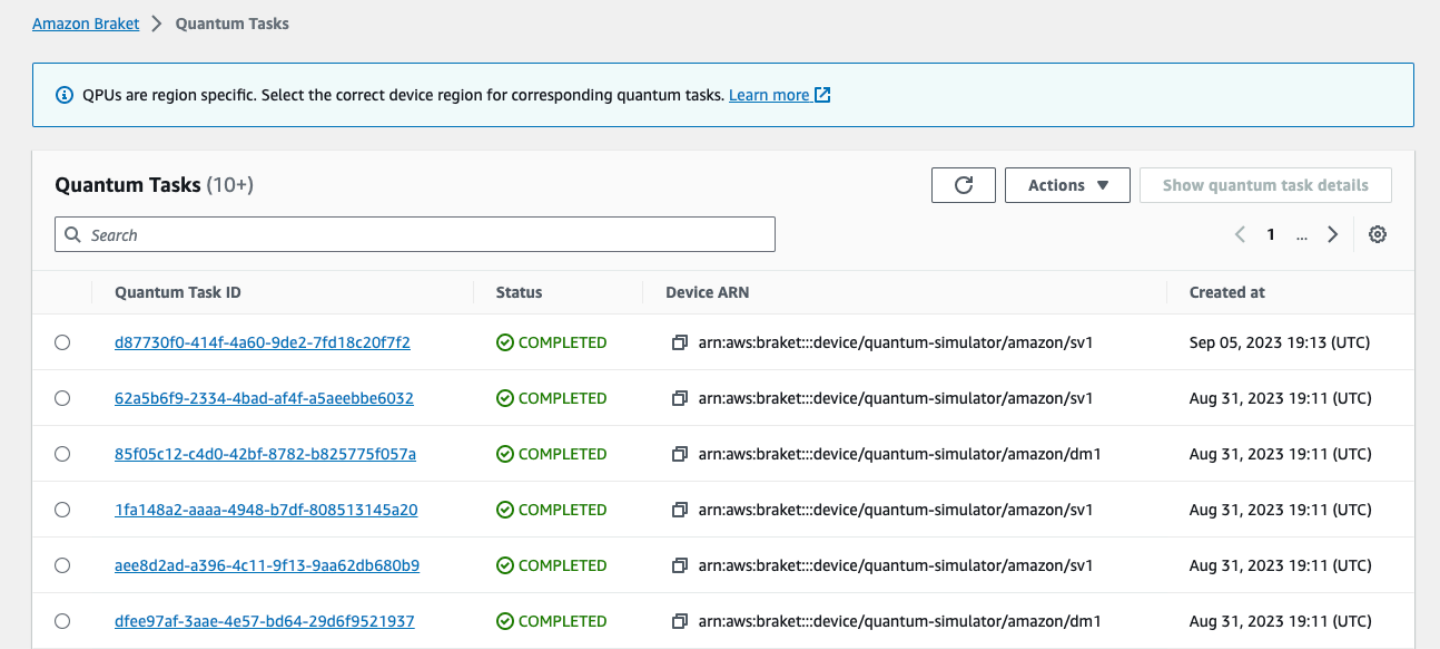
如果您的核心在您提交量子任務後死亡，或者關閉筆記本電腦或計算機，則可以使用其唯一的 ARN（量子任務 ID）重建該`task`對象。然後，您可`task.result()`以調用從存儲它的 S3 存儲桶中獲取結果。

```
from braket.aws import AwsSession, AwsQuantumTask

# restore task with unique arn
task_load = AwsQuantumTask(arn=task_id)
# retrieve the result of the task
```

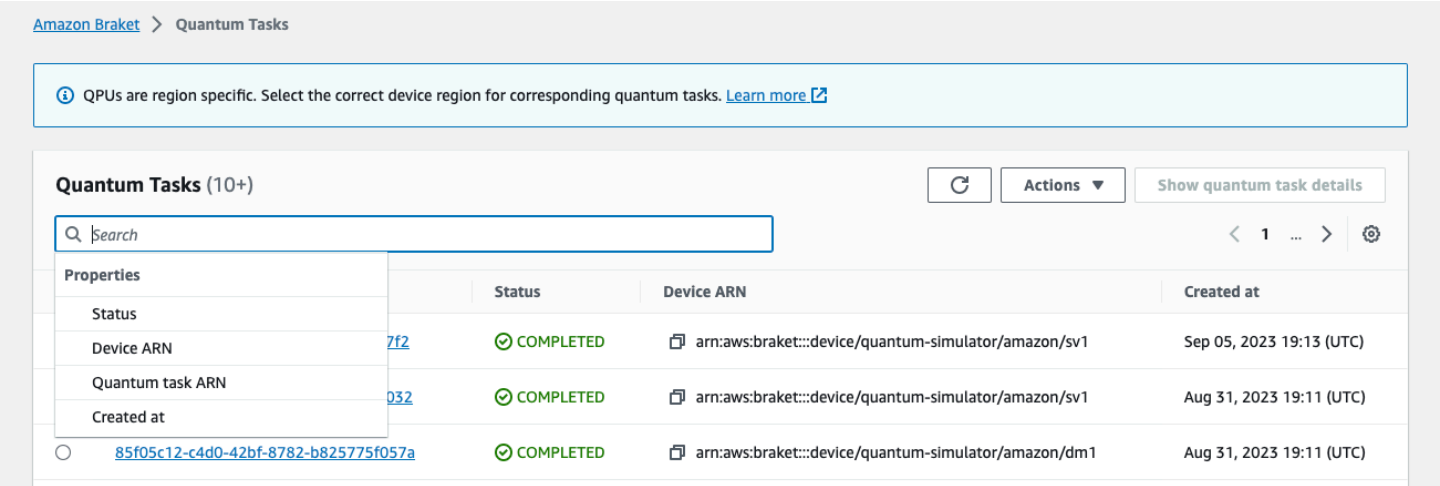
```
result = task_load.result()
```

透過 Amazon Braket 主控台監控量子任務

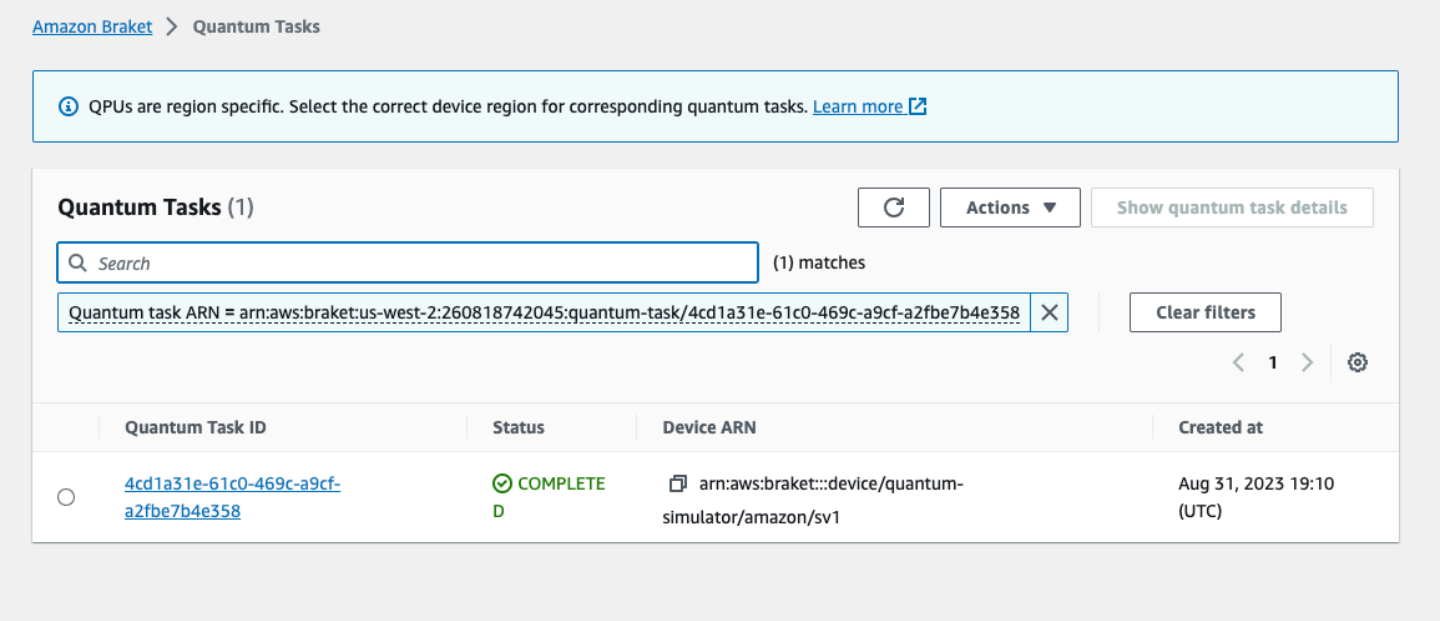
AmazonBraket 提供了一種透過 Amazon Braket 主控台監控量子任務的便利方式。所有提交的量子任務都列在「量子任務」字段中，如下圖所示。此服務是特定於區域的，這意味著您只能查看在特定項目中創建的量子任務。AWS 區域



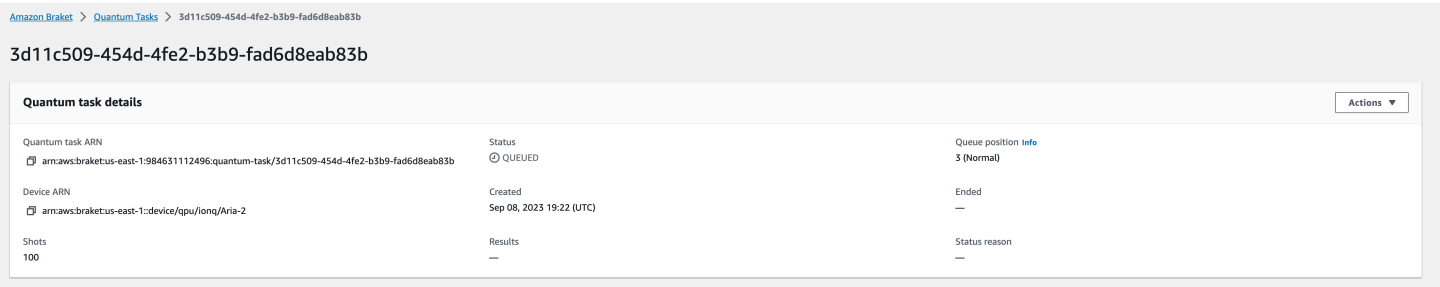
您可以通過導航欄搜索特定的量子任務。可以根據量子任務 ARN（ID），狀態，設備和創建時間進行搜索。當您選取導覽列時，選項會自動顯示，如下列範例所示。



下圖顯示了根據量子任務唯一的量子任務 ID 來搜索量子任務的示例，該任務可通過調用獲取`task.id`。



另外，如下圖所示，量子任務處於狀態時可以監視量子任務的狀QUEUED態。按一下量子任務 ID 會顯示詳細資訊頁面。此頁面顯示量子任務的動態佇列位置，相對於它將處理的裝置。



作為混合任務一部分提交的量子任務在隊列中將具有優先級。在混合任務外提交的量子任務將具有正常的佇列優先級。

希望查詢 Braket SDK 的客戶可以透過程式設計方式取得量子任務和混合工作佇列位置。如需詳細資訊，請參閱我的工作何時執行頁面。

標記亞馬遜標記資源

標籤是一種自訂屬性標籤，可讓您指派或由 AWS 指派給 AWS 資源。標籤是告訴更多有關您資源的元數據。每個標籤皆包含鍵與值。這些合稱為鍵值組。對於您指派的標籤，您可以定義鍵與值。

在 Amazon Braket 主控台中，您可以瀏覽至量子任務或筆記本，並檢視與其關聯的標籤清單。您可以新增標籤、移除標籤或修改標籤。您可以在建立量子任務或筆記本時標記，然後透過主控台AWS CLI、或管理關聯的標籤API。

使用標籤

標籤可以將您的資源組織成對您有用的類別。例如，您可以指定「部門」標籤來指定擁有此資源的部門。

每個標籤有兩個部分：

- 標籤鍵 (例如 CostCenter，環境或專案)。標籤鍵會區分大小寫。
- 稱為標籤值的選擇性欄位 (例如，111122223 333 或正式作業)。忽略標籤值基本上等同於使用空字串。與標籤鍵相同，標籤值會區分大小寫。

標籤可幫助您執行以下操作：

- 識別和組織您的AWS資源。許多 AWS 服務支援標記，因此您可以對來自不同服務的資源指派相同的標籤，指出資源是相關的。
- 追蹤您的AWS成本。您可以在 AWS Billing and Cost Management 儀表板上啟用這些標籤。AWS會使用標籤分類您的成本，並交付每月成本配置報告給您。如需詳細資訊，請參閱 AWS Billing and Cost Management 使用者指南中的使用成本配置標籤。
- 控制對AWS資源的存取。如需詳細資訊，請參閱使用標籤控制存取。

更多關於AWS和標籤

- 有關標籤的一般資訊，包括命名和使用慣例，請參閱AWS一般參考中的標記AWS資源。
- 若要取得有關標籤限制的資訊，請參閱AWS一般參考中的標籤命名限制和需求。
- 有關最佳實踐和標籤策略，請參閱標記最佳實踐和AWS標籤策略。
- 關於支援標記的服務清單，請參閱Resource Groups 標記 API 參考。

以下各節提供有關 Amazon Braket 標籤的更具體資訊。

亞馬遜文字架中支援的資源

AmazonBraket 中的以下資源類型支援標記：

- `quantum-task` 資源
- 資源名稱：`AWS::Service::Braket`

- ARN 正則表達式：`arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

注意：儘管筆記本實際上是 Amazon SageMaker 資源，但您Amazon可以使用控制台導航到筆記本資源，在 Amazon Braket 控制台中應用和管理 Braket 筆記本的標籤。如需詳細資訊，請參閱 SageMaker 文件中的記事本執行個體中繼資料

標籤限制

下列基本限制適用於 Amazon Braket 資源上的標籤：

- 可指派給資源的標籤數目上限：50
- 索引鍵長度上限：128 個 Unicode 字元
- 數值長度上限：256 個 Unicode 字元
- 索引鍵和 value: `a-z, A-Z, 0-9, space` 和下列字元的有效字元：`_ . : / = + -`和 `@`
- 金鑰和值會區分大小寫。
- 不要用`aws`作密鑰的前綴; 它保留供AWS使用。

在 Amazon Braket 標籤管理標籤

您可以將標籤設定為資源上的屬性。您可以透過 Amazon Braket 主控台、Braket 或檢視、新增、修改、列出Amazon和刪除標籤。API AWS CLI如需詳細資訊，請參閱亞馬遜標題 API 參考資料。

新增標籤

您可以在下列時間將標籤新增至可加上標籤的資源：

- 建立資源時：使用主控台，或將`Tags`參數與`Create`作業一起包含在 AWSAPI 中。
- 建立資源之後：使用主控台導覽至量子工作或筆記本資源，或呼叫 AWSAPI 中的`TagResource`作業。

若要在建立資源時將標籤新增至資源，您還需要建立指定類型資源的權限。

檢視標籤

您可以使用主控台導覽至任務或記事本資源，或呼叫作業，來檢視 Amazon Braket 中任何可加標籤資源上的標籤。AWS `ListTagsForResource` API

您可以使用以下AWSAPI命令來查看資源上的標籤：

- AWS API: `ListTagsForResource`

編輯標籤

您可以使用控制台瀏覽至量子任務或筆記本資源來編輯標籤，或者您可以使用以下命令修改附加到可標記資源的標籤值。當您指定已存在的標籤鍵時，會覆寫該鍵的值：

- AWS API: `TagResource`

移除標籤

您可以透過指定要移除的金鑰、使用主控台瀏覽至量子工作或筆記本資源，或在呼叫`UntagResource`作業時，從資源中移除標籤。

- AWS API: `UntagResource`

Amazon Braket 網頁中的 CLI 標記示例

如果您正在使用 AWS CLI，這裡是一個示例命令，顯示如何創建一個標籤，該標籤適用於您使用 Rigetti QPU SV1 的參數設置創建的量子任務。請注意，標籤是在範例指令的結尾指定的。在這種情況下，Key 被賦予的值**`state`**和值給出的值`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標記

- 如果您正在使用 Amazon Braket API 在資源上設置標籤，請調用 TagResourceAPI.

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

- 若要從資源中移除標籤，請呼叫 UntagResourceAPI.

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- 若要列出附加至特定資源的所有標籤，請呼叫 ListTagsForResourceAPI.

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city
\",\"state\"]"
```

Amazon 與 Amazon Braket 的事件和自動化操作 EventBridge

Amazon EventBridge 監控 Amazon Braket 量子任務中的狀態變更事件。來自 Amazon Braket 的事件幾乎是實時交付的。 EventBridge您可編寫簡單的規則，來指示您在意的事件，包括當事件符合規則時所要自動執行的動作。可觸發的自動動作包括：

- 呼叫 AWS Lambda 函數
- 啟動 AWS Step Functions 狀態機器
- 通知 Amazon SNS 主題

EventBridge 監視這些文Amazon件狀態變更事件：

- 卡恩特姆任務狀態變化

AmazonBraket 保證量子任務狀態變化事件的交付。這些事件至少會傳送一次，但可能會出現故障。

如需詳細資訊，請參閱中的事件和事件模式 EventBridge。

在本節中：

使用監控量子任務狀態 EventBridge

使用 EventBridge，您可以建立規則來定義 Amazon Braket 傳送有關 Braket 量子任務狀態變更的通知時要採取的動作。例如，您可以建立規則，在每次量子任務狀態變更時傳送電子郵件訊息給您。

1. 使用具有AWS使用權限 EventBridge 和 Amazon Braket 的帳戶登錄。
2. 在以下位置打開 Amazon EventBridge 控制台 https://console.aws.amazon.com/events/。
3. 使用下列值建立 EventBridge 規則：
   - 針對 Rule type (規則類型) 選擇 Rule with an event pattern (具有事件模式的規則)。
   - 在 Event source (事件來源) 中，選擇 Other (其他)。
   - 在 [事件模式] 區段中，選擇 [自訂模式 (JSON 編輯器)]，然後將下列事件模式貼到文字區域中：

     ```
     {
       "source": [
         "aws.braket"
       ],
       "detail-type": [
         "Braket Task State Change"
       ]
     }
     ```

     若要擷取 Amazon Braket 中的所有事件，請排除下列程式碼所示的`detail-type`區段：

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

   - 對於目標類型 AWS 服務，選擇和選取目標，然後選擇一個目標，例如 Amazon SNS 主題或AWS Lambda函數。當從 Amazon Braket 收到量子任務狀態更改事件時觸發目標。

例如，使用 Amazon Simple Notification Service (SNS) 主題在事件發生時傳送電子郵件或文字訊息。若要這麼做，請先使用 Amazon SNS 主控台建立 Amazon SNS 主題。若要進一步了解，請參閱使用 Amazon SNS 傳送使用者通知。

如需有關建立規則的詳細資訊，請參閱建立可回應事件的 Amazon EventBridge 規則。

示例 Amazon Braket EventBridge 事件

如需 Amazon Braket Quantum 任務狀態變更事件欄位的相關資訊，請參閱中的事件和事件模式 EventBridge。

下列屬性會顯示在 JSON「詳細資料」欄位中。

- **quantumTaskArn**(str)：產生此事件的量子任務。
- **status**（可選 [str]）：量子任務轉移到的狀態。
- **deviceArn**(str)：由建立此量子任務的使用者所指定的裝置。
- shots（int）：用戶shots請求的數量。
- **outputS3Bucket**(str)：使用者指定的輸出值區。
- **outputS3Directory**（str）：用戶指定的輸出 key prefix。
- **createdAt**（str）：將量子任務創建時間作為 ISO-8601 字符串。
- **endedAt**（可選 [str]）：量子任務達到終端狀態的時間。只有當量子任務已轉換為終端狀態時，此欄位才會出現。

以下 JSON 代碼顯示了 Amazon Braket 量子任務狀態更改事件的示例。

```
{
    "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 Amazon 監控 Amazon Braket CloudWatch

您可以使用 Amazon Braket 監控 Amazon Braket CloudWatch，該 Amazon 會收集原始資料並將其處理為可讀且接近即時的指標。您可以在 Amazon CloudWatch 主控台中檢視 15 個月前產生的歷史資訊，或搜尋過去 2 週更新過的指標，以更好地瞭解 Amazon Braket 的效能。若要深入瞭解，請參閱使用 CloudWatch 量度。

亞馬遜標誌指標和維度

度量標準是中的基本概念 CloudWatch。量度代表發佈至 CloudWatch的一組時間順序的資料點。每個量度都有一組維度。若要進一步瞭解中的量度維度 CloudWatch，請參閱CloudWatch 維度。

Amazon Braket 會將 Amazon 標誌特定的以下指標資料傳送到亞馬遜 CloudWatch 指標中：

量子任務指標

如果存在量子任務，則可以使用指標。它們會顯示在主控台的 AWS/制動器/ 依裝置下方。CloudWatch

指標	描述
計數	量子任務的數量。
Latency (延遲)	當量子任務已經完成此度量被發射。它代表從量子任務初始化到完成的總時間。

量子任務指標的維度

量子任務度量與基於參數的維度一起發布，該deviceArn參數的格式為 arn: aw: braket:: 設備/xxx。

支援的裝置

如需支援裝置和裝置 ARN 的清單，請參閱 Braket 裝置。

> **Note**
>
> 您可以瀏覽至 Amazon SageMaker 主控台上的筆記本詳細資料頁面，檢視 Amazon Braket 筆記型電腦的 CloudWatch 日誌串流。您可透過SageMaker 主控台取得其他 Amazon Braket 筆記型電腦設定。

Amazon Braket 日誌記錄與 CloudTrail

AmazonBraket 集成了一種服務AWS CloudTrail，該服務提供了用戶，角色或 Amazon Braket AWS 服務 中的操作的記錄。 CloudTrail 捕獲所有對Amazon布拉克特的API呼叫作為事件。捕獲的呼叫包括來自 Amazon Braket 控制台的呼叫以及對 Braket Amazon Braket 操作的代碼調用。如果您建立追蹤，您可以啟用持續交付 CloudTrail 事件到 Amazon S3 儲存貯體，包括 Amazon Braket 的事件。如果您未設定追蹤，您仍然可以在 [事件歷程記錄] 中檢視 CloudTrail 主控台中最近的事件。使用收集的信息 CloudTrail，您可以確定向 Amazon Braket 提出的請求，提出請求的 IP 地址，提出請求的人員，提出請求的時間以及其他詳細信息。

若要進一步了解 CloudTrail，請參閱使AWS CloudTrail用者指南。

Amazon Braket 信息 CloudTrail

CloudTrail 在您創建帳戶AWS 帳戶時啟用。當活動在 Amazon Braket 中發生時，該活動會與事件歷史記錄中的其他 CloudTrail 事件一起記錄在AWS 服務事件中。您可以檢視、搜尋和下載 AWS 帳戶 的最新事件。如需詳細資訊，請參閱檢視具有事 CloudTrail 件記錄的事件。

對於您的事件的持續記錄AWS 帳戶，包括 Amazon Braket 的事件，請創建一個跟踪。追蹤可 CloudTrail 將日誌檔交付到 Amazon S3 儲存貯體。依預設，當您在主控台中建立追蹤時，該追蹤會套用至所有的 AWS 區域。該追蹤會記錄來自 AWS 分割區中所有區域的事件，並將日誌檔案交付到您指定的 Amazon S3 儲存貯體。此外，您可以設定其他，AWS 服務以進一步分析 CloudTrail 記錄中收集的事件資料並採取行動。如需詳細資訊，請參閱下列內容：

- 建立追蹤的概觀
- CloudTrail 支援的服務與整合
- 設定的 Amazon SNS 通知 CloudTrail
- 從多個區域接收 CloudTrail 記錄檔並從多個帳戶接收 CloudTrail 記錄檔

所有Amazon布拉克特動作都由 CloudTrail記錄。例如，呼叫GetQuantumTask或動GetDevice作會在 CloudTrail 記錄檔中產生項目。

每一筆事件或日誌項目都會包含產生請求者的資訊。身分資訊可協助您判斷下列事項：

- 提出該請求時，是否使用了特定角色或聯合身分使用者的暫時安全憑證。
- 該請求是否由另一項 AWS 服務 服務提出。

如需詳細資訊，請參閱 CloudTrail 使用者身分元素。

了解 Amazon Braket 日誌文件條目

追蹤是一種組態，可讓事件以日誌檔的形式傳遞到您指定的 Amazon S3 儲存貯體。 CloudTrail 記錄檔包含一或多個記錄項目。事件代表來自任何來源的單一請求，包括有關請求的操作，動作的日期和時間，請求參數等信息。 CloudTrail 日誌文件不是公共API調用的有序堆棧跟踪，因此它們不會以任何特定順序顯示。

下列範例是GetQuantumTask動作的記錄項目，它會取得量子工作的詳細資訊。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "foobar",
    "arn": "foobar",
    "accountId": "foobar",
    "accessKeyId": "foobar",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "foobar",
        "arn": "foobar",
        "accountId": "foobar",
        "userName": "foobar"
```

```
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-08-07T00:56:57Z"
      }
    }
  },
  "eventTime": "2020-08-07T01:00:08Z",
  "eventSource": "braket.amazonaws.com",
  "eventName": "GetQuantumTask",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "foobar",
  "userAgent": "aws-cli/1.18.110 Python/3.6.10
 Linux/4.9.184-0.1.ac.235.83.329.metal1.x86_64 botocore/1.17.33",
  "requestParameters": {
    "quantumTaskArn": "foobar"
  },
  "responseElements": null,
  "requestID": "20e8000c-29b8-4137-9cbc-af77d1dd12f7",
  "eventID": "4a2fdb22-a73d-414a-b30f-c0797c088f7c",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "recipientAccountId": "foobar"
}
```

以下顯示GetDevice動作的記錄項目，該項目會傳回裝置事件的詳細資訊。

```
{
  "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"
}
```

使用以下方法建立 Amazon Braket 筆記本執行個 AWS CloudFormation

您可以用AWS CloudFormation來管理您的 Amazon Braket 筆記型電腦執行個體。Braket 筆記本實例建立在 Amazon SageMaker 上。使用 CloudFormation，您可以使用描述預定組態的範本檔案來佈建筆記本執行個體。範本檔案會以 JSON 或 YAML 格式撰寫。您可以以有序且可重複的方式建立、更新和刪除執行個體。當您管理多個 Braket 筆記本實例時，您可能會發現這很有用AWS 帳戶。

建立 Braket 筆記本的 CloudFormation 範本後，您可AWS CloudFormation以用來部署資源。如需詳細資訊，請參閱AWS CloudFormation使用指南中的在AWS CloudFormation主控台上建立堆疊。

若要使用建立 Braket 筆記本實例 CloudFormation，請執行下列三個步驟：

1. 建立 Amazon SageMaker 生命週期組態指令碼。

2. 建立要承擔的 AWS Identity and Access Management (IAM) 角色 SageMaker。

3. 使用前置詞建立 SageMaker 筆記本執行個體 **amazon-braket-**

您可以為您建立的所有 Braket 筆記本重複使用生命週期組態。您也可以針對指派相同執行權限的 Braket 筆記本重複使用 IAM 角色。

步驟 1：建立 Amazon SageMaker 生命週期組態指令碼

使用下列範本建立SageMaker 生命週期組態指令碼。此指令碼會自訂 Braket 的 SageMaker 筆記本執行個體。如需生命週期 CloudFormation 資源的組態選項，請參閱AWS CloudFormation使用指南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
```

步驟 2：創建 Amazon 假定的 IAM 角色 SageMaker

當您使用 Braket 筆記本執行個體時， SageMaker 會代表您執行作業。例如，假設您使用支援裝置上的電路執行 Braket 筆記型電腦。在筆記型電腦執 SageMaker 行個體中，為您執行 Braket 上的作業。筆記本執行角色定義了允許代表您執行的確切作業。 SageMaker 如需詳細資訊，請參閱 Amazon SageMaker 開發人員指南中的SageMaker 角色。

使用下列範例建立具有所需權限的 Braket 筆記本執行角色。您可以根據需要修改策略。

> **Note**
>
> 請確定角色具有前置詞為的 Amazon S3 儲存貯體`s3:ListBucket`和`s3:GetObject`操作的`braketnotebookcdk-"`權限。生命週期組態指令碼需要這些權限才能複製 Braket 筆記本安裝指令碼。

```
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: "*"
```

步驟 3：建立具有前置詞的 Amazon SageMaker 筆記本執行個體 **amazon-braket-**

使用在步驟 1 和步驟 2 中建立的 SageMaker 生命週期指令碼和 IAM 角色建立 SageMaker 筆記本執行個體。筆記型電腦執行個體是針對 Braket 客製化的，可透過 Amazon Braket 主控台存取。若要取得有關此 CloudFormation 資源之組態選項的更多資訊，請參閱AWS CloudFormation使用指南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
```

進階記錄

您可以使用記錄器記錄整個任務處理過程。這些進階記錄技術可讓您查看背景輪詢，並建立記錄以供稍後偵錯。

要使用記錄器，我們建議您更改poll_timeout_seconds和poll_interval_seconds參數，以便量子任務可以長時間運行並連續記錄量子任務狀態，並將結果保存到文件中。您可以將此代碼傳輸到 Python 腳本而不是 Jupyter 筆記本，以便腳本可以在後台作為進程運行。

配置記錄器

首先，設定記錄器，讓所有記錄檔都會自動寫入文字檔案中，如下列範例行所示。

```
# 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
```

建立並執行電路

現在，您可以創建一個電路，將其提交到要運行的設備，並查看此示例中所示的情況。

```
# 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
)
```

檢查記錄檔

您可以通過輸入以下命令來檢查寫入文件中的內容。

```
# 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})
```

從日誌文件中獲取 ARN

您可以從傳回的記錄檔輸出 (如上述範例所示) 取得 ARN 資訊。使用 ARN ID，您可以檢索完成的量子任務的結果。

```
# 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()
```

Amazon Braket 中的安全性

本章可協助您瞭解如何在使用 Amazon Braket 時應用共同的責任模型。本文說明如何設定 Amazon Braket 以符合您的安全性和合規目標。您也會學到如何使用其AWS 服務他協助您監控和保護 Amazon Braket 資源的其他資源。

雲端安全是 AWS 最重視的一環。身為 AWS 客戶的您，將能從資料中心和網路架構的建置中獲益，以滿足組織最為敏感的安全要求。您必須對其他因素負責，包括資料的敏感性、公司的要求以及適用的法律和法規。

對安全的共同責任

安全是 AWS 與您共同的責任。共同的責任模型 將此描述為雲端 本身 的安全和雲端內部的安全：

- 雲端本身的安全 – AWS 負責保護在 AWS 雲端 中執行 AWS 服務 的基礎設施。AWS 也提供您可安全使用的服務。在 AWS 合規計劃中，第三方稽核員會定期測試並驗證我們的安全功效。若要了解適用於 Amazon Braket 的合規計劃，請參閱合規計劃適用範圍的AWS服務。
- 雲端中的安全性 — 您有責任維持對此AWS基礎架構上託管內容的控制權。此內容包括您所使用 AWS 服務 的安全組態和管理任務。

資料保護

AWS共同責任模型適用於 Amazon Braket 中的資料保護。如此模型所述，AWS 負責保護執行所有 AWS 雲端 的全球基礎設施。您負責維護在此基礎設施上託管內容的控制權。您也必須負責您所使用 AWS 服務 的安全組態和管理任務。如需有關資料隱私權的更多相關資訊，請參閱資料隱私權常見問答集。如需有關歐洲資料保護的相關資訊，請參閱 AWS 安全性部落格上的 AWS 共同的責任模型和 GDPR 部落格文章。

基於資料保護目的，建議您使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 保護 AWS 帳戶 憑證，並設定個人使用者。如此一來，每個使用者都只會獲得授與完成其任務所必須的許可。我們也建議您採用下列方式保護資料：

- 每個帳戶均要使用多重要素驗證 (MFA)。
- 使用 SSL/TLS 與 AWS 資源通訊。我們需要 TLS 1.2 並建議使用 TLS 1.3。
- 使用 AWS CloudTrail 設定 API 和使用者活動日誌記錄。
- 使用 AWS 加密解決方案，以及 AWS 服務 內的所有預設安全控制項。

- 使用進階的受管安全服務（例如 Amazon Macie），協助探索和保護儲存在 Amazon S3 的敏感資料。
- 如果您在透過命令列介面或 API 存取 AWS 時，需要 FIPS 140-2 驗證的加密模組，請使用 FIPS 端點。如需有關 FIPS 和 FIPS 端點的更多相關資訊，請參閱聯邦資訊處理標準 (FIPS) 140-2 概觀。

我們強烈建議您絕對不要將客戶的電子郵件地址等機密或敏感資訊，放在標籤或自由格式的文字欄位中，例如 Name (名稱) 欄位。這包括當您使用主控台、API 或開發套件AWS 服務使用 Amazon Braket 或其他軟體AWS開發套件時。AWS CLI您在標籤或自由格式文字欄位中輸入的任何資料都可能用於計費或診斷日誌。如果您提供外部伺服器的 URL，我們強烈建議請勿在驗證您對該伺服器請求的 URL 中包含憑證資訊。

資料保留

90 天後，Amazon Braket 會自動移除與您的量子任務相關聯的所有量子任務 ID 和其他中繼資料。根據此資料保留政策，儘管這些任務和結果仍然存放在 S3 儲存貯體中，但無法再透過 Amazon Braket 主控台搜尋來擷取這些任務和結果。

如果您需要存取存放在 S3 儲存貯體中超過 90 天的歷史量子任務和結果，則必須單獨保留任務 ID 和與該資料相關聯的其他中繼資料的記錄。請務必在 90 天之前儲存資訊。您可以使用該保存的信息來檢索歷史數據。

管理對 Amazon Braket 的訪問

本章說明執行 Amazon Braket 或限制特定使用者和角色存取所需的權限。您可以授予 (或拒絕) 必要的權限給您帳戶中的任何使用者或角色。若要這麼做，請將適當的 Amazon Braket 政策附加到帳戶中的該使用者或角色，如以下各節所述。

作為先決條件，您必須啟用 Amazon Braket。若要啟用 Braket，請務必以具有 (1) 管理員許可或 (2) 的使用者或角色登入，或 (2) 已指派AmazonBraketFullAccess政策，並具有建立 Amazon Simple Storage Service (Amazon S3) 貯體的權限。

在本節中：

- Amazon Braket AWS 管理政策的更新
- 限制使用者對特定筆記本執行個體
- 限制使用者存取特定 S3 儲存貯體

Amazon Braket 資源

Braket 創建一種類型的資源：量子任務資源。此資源類型的 Amazon 資源名稱 (ARN) 如下所示：

- 資源名稱::AWS: 服務:: 文字
- ARN 正則表達式：ARN：$ {分區}：口袋：$ {區域}：$ {帳戶}：量子任務/$ {} RandomId

筆記本和角色

您可以在布拉基特中使用記事博資源類型。筆記本是 Braket 能夠共享的 Amazon SageMaker 資源。若要搭配 Braket 使用筆記本，您必須指定名稱開頭為的 IAM 角色。AmazonBraketServiceSageMakerNotebook

若要建立筆記本，您必須使用具有管理員權限或附加下列內嵌原則的角色。

```
{
   "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*"
                    ]
                }
            }
        }
    ]
}
```

若要建立角色，請遵循 [建立記事本] 頁面中提供的步驟，或讓管理員為您建立角色。確定已附加AmazonBraketFullAccess原則。

建立角色之後，您可以在 future 啟動的所有記事本中重複使用該角色。

關於本 AmazonBraketFullAccess政策

該AmazonBraketFullAccess政策授予 Amazon Braket 操作的許可，包括這些任務的許可：

- 從 Amazon 彈性容器登錄下載容器 — 讀取和下載用於 Amazon Braket 混合任務功能的容器映像。容器必須符合格式「arn: awn:: ecr:: 存儲庫/亞馬遜字符」。
- 保留 AWS CloudTrail 記錄 — 除了啟動和停止查詢、測試指標篩選器和篩選記錄事件之外的所有描述、取得和列出動作。 AWS CloudTrail 日誌檔包含您帳戶中發生的所有 Amazon Braket API 活動的記錄。
- 利用角色來控制資源 — 在您的帳戶中建立服務連結角色。服務連結角色可以代表您存取 AWS 資源。它只能由 Amazon Braket 服務使用。此外，將 IAM 角色傳遞給 Amazon Braket，`CreateJob`API並建立角色並將範圍內的政策附加 AmazonBraketFullAccess 到該角色。
- 建立日誌群組、日誌事件和查詢日誌群組以維護帳戶的使用日誌檔 — 建立、存放和檢視有關帳戶中 Amazon Braket 使用情況的記錄資訊。在混合式工作記錄群組上查詢量度。包括正確的 Braket 路徑，並允許把日誌數據。將指標資料放入 CloudWatch。
- 在 Amazon S3 儲存貯體中建立和存放資料，並列出所有儲存貯體 — 若要建立 S3 儲存貯體，請在帳戶中列出 S3 儲存貯體，然後將物件放入帳戶中名稱以 amazon-bra ket-開頭的任何儲存貯體並從

其中取得物件。Braket 需要這些權限才能將包含已處理量子任務結果的文件放入存儲桶並從存儲桶中檢索它們。

- 傳遞 IAM 角色 — 將 IAM 角色傳遞給 `CreateJob`API.
- Amazon SageMaker 筆記本 — 創建和管理範圍為「arn: aws: SAGEMAKER:: SageMaker 筆記本-實例/亞馬遜-braket-」的範圍內的筆記本實例。
- 驗證服務配額 — 若要建立 SageMaker筆記本和 Amazon Braket 混合任務，您的資源數量不得超過帳戶的配額。

政策內容

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

關於本 AmazonBraketJobsExecutionPolicy政策

該AmazonBraketJobsExecutionPolicy政策授予 Amazon Braket 混合任務中使用的執行角色的許可，如下所示：

- 從 Amazon 彈性容器登錄下載容器-讀取和下載用於 Amazon Braket 混合任務功能之容器映像的許可。容器必須符合格式「arn: awn: ecr: *: *: 儲存庫/亞馬遜胸罩 *」。
- 建立日誌群組、日誌事件以及查詢日誌群組，以維護帳戶的使用日誌檔 — 建立、存放和檢視帳戶中 Amazon Braket 使用情況的記錄資訊。在混合式工作記錄群組上查詢量度。包括正確的 Braket 路徑，並允許把日誌數據。將指標資料放入 CloudWatch。
- 將資料存放在 Amazon S3 儲存貯體 — 列出帳戶中的 S3 儲存貯體、將物件放入帳戶中任何以 amazon-braket 開頭的儲存貯體，並從其名稱中取得物件。Braket 需要這些權限才能將包含已處理量子任務結果的文件放入存儲桶中，並從存儲桶中檢索它們。
- 傳遞 IAM 角色 — 將 IAM 角色傳遞給 CreateJob API. 角色必須符合格式的 arn: aw:iam:: *。:role/service-role/AmazonBraketJobsExecutionRole

```
"Version": "2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "s3:GetObject",
   "s3:PutObject",
   "s3:ListBucket",
   "s3:CreateBucket",
   "s3:PutBucketPublicAccessBlock",
```

```
    "s3:PutBucketPolicy"
   ],
   "Resource": "arn:aws:s3:::amazon-braket-*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "ecr:GetDownloadUrlForLayer",
    "ecr:BatchGetImage",
    "ecr:BatchCheckLayerAvailability"
   ],
   "Resource": "arn:aws:ecr:*:*:repository/amazon-braket*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "ecr:GetAuthorizationToken"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "braket:CancelJob",
    "braket:CancelQuantumTask",
    "braket:CreateJob",
    "braket:CreateQuantumTask",
    "braket:GetDevice",
    "braket:GetJob",
    "braket:GetQuantumTask",
    "braket:SearchDevices",
    "braket:SearchJobs",
    "braket:SearchQuantumTasks",
    "braket:ListTagsForResource",
    "braket:TagResource",
    "braket:UntagResource"
   ],
   "Resource": "*"
  },
  {
   "Effect": "Allow",
   "Action": [
    "iam:PassRole"
   ],
```

```
    "Resource": "arn:aws:iam::*:role/service-role/AmazonBraketJobsExecutionRole*",
    "Condition": {
     "StringLike": {
      "iam:PassedToService": [
       "braket.amazonaws.com"
      ]
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iam:ListRoles"
    ],
    "Resource": "arn:aws:iam::*:role/*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "logs:GetQueryResults"
    ],
    "Resource": [
     "arn:aws:logs:*:*:log-group:*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "logs:PutLogEvents",
     "logs:CreateLogStream",
     "logs:CreateLogGroup",
     "logs:GetLogEvents",
     "logs:DescribeLogStreams",
     "logs:StartQuery",
     "logs:StopQuery"
    ],
    "Resource": "arn:aws:logs:*:*:log-group:/aws/braket*"
   },
   {
    "Effect": "Allow",
    "Action": "cloudwatch:PutMetricData",
    "Resource": "*",
    "Condition": {
     "StringEquals": {
```

```
      "cloudwatch:namespace": "/aws/braket"
    }
   }
  }
 ]
}
```

限制使用者存取特定裝置

若要限制某些使用者存取特定 Braket 裝置，您可以將拒絕權限原則新增至特定IAM角色。

使用此類權限可以限制下列動作：

- `CreateQuantumTask`-拒絕在指定裝置上建立量子任務。
- `CreateJob`-拒絕在指定裝置上建立混合工作。
- `GetDevice`-拒絕取得指定裝置的詳細資料。

下列範例會限制對的所有 QPU 的存取。 AWS 帳戶 123456789012

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
    "Effect": "Deny",
    "Action": [
      "braket:CreateQuantumTask",
      "braket:CreateJob",
      "braket:GetDevice"
    ],
    "Resource": [
    "arn:aws:braket:*:*:device/qpu/*"
    ]
   }
  ]
}
```

若要調整此程式碼，請將受限裝置的Amazon資源編號 (ARN) 替換為上一個範例中顯示的字串。此字串提供資源值。在 Braket 中，一個設備代表一個 QPU 或模擬器，你可以調用它來運行量子任務。可用的裝置會列在「裝置」頁面上。有兩種結構描述可用來指定對這些裝置的存取權：

- `arn:aws:braket:<region>:<account id>:device/qpu/<provider>/<device_id>`

- arn:aws:braket:<region>:<account id>:device/quantum-simulator/<provider>/<device_id>

以下是各種類型設備訪問的示例

- 若要選取所有區域中的所有 QPU：arn:aws:braket:*:*:device/qpu/*
- 若要僅選取美國-西部 -2 區域中的所有 QPU，請執行下列動作：arn:aws:braket:us-west-2:123456789012:device/qpu/*
- 同樣地，若要僅選取 us-west-2 區域中的所有 QPU (因為裝置是服務資源，而非客戶資源)：arn:aws:braket:us-west-2:* :device/qpu/*
- 若要限制對所有隨選模擬器裝置的存取：arn:aws:braket:* :123456789012:device/quantum-simulator/*
- 若要限制 us-east-1 區域中IonQ Harmony裝置的存取，請執行下列動作：arn:aws:braket:us-east-1:123456789012:device/ionq/Harmony
- 若要限制存取特定供應商的裝置 (例如，RigettiQPU裝置)：arn:aws:braket:* :123456789012:device/qpu/rigetti/*
- 若要限制對TN1裝置的存取：arn:aws:braket:* :123456789012:device/quantum-simulator/amazon/tn1

Amazon Braket AWS 管理政策的更新

下表提供有關 Braket AWS 受管理原則的更新詳細資訊，因為此服務開始追蹤這些變更。

變更	Description	日期
AmazonBraketFullAccess-布拉克特的完整訪問政策	已新增服務GetServiceQuota與雲觀察：要包含在政策中的GetMetricData動作。AmazonBraketFullAccess .	2023 年 3 月 24 日
AmazonBraketFullAccess-布拉克特的完整訪問政策	布拉克特調整 iam：PassRole 許 AmazonBraketFullAccess 可包括service-role/ 路徑。	2021 年 11 月 29 日

變更	Description	日期
AmazonBraketJobsExecutionPolicy-Amazon Braket 混合式工作的混合作業執行政策	Braket 更新了混合作業執行角色 ARN 以包含路徑。`service-role/`	2021 年 11 月 29 日
布拉克特開始跟踪更改	Braket 開始追蹤其 AWS 受管理政策的變更。	2021 年 11 月 29 日

限制使用者對特定筆記本執行個體

若要限制特定使用者存取特定 Braket 筆記本執行個體，您可以將拒絕權限原則新增至特定角色、使用者或群組。

下列範例使用原則變數來有效地限制啟動、停止和存取中特定筆記本執行個體的權限 AWS 帳戶 `123456789012`，這些執行個體是根據應具有存取權的使用者命名 (例如，使用者`Alice`可以存取名為的筆記本執行個體`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}*"
      ]
    }
  ]
}
```

限制使用者存取特定 S3 儲存貯體

若要限制特定使用者存取特定 Amazon S3 儲存貯體，您可以向特定角色、使用者或群組新增拒絕政策。

下列範例會限制擷取物件並將物件放入特定S3值區 (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) 的權限，並限制這些物件的清單。

```
{
  "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/*"
      ]
    }
  ]
}
```

若要限制特定筆記本執行個體對值區的存取，您可以將上述原則新增至筆記本執行角色。

亞馬遜字幕服務鏈接角色

當您啟用 Amazon Braket 時，會在您的帳戶中建立服務連結角色。

服務連結角色是一種獨特的 IAM 角色類型，在此情況下，它會直接連結至 Amazon Braket。Amazon Braket 服務連結角色已預先定義為包含 Braket 代表您呼叫其他AWS 服務人時所需的所有許可。

服務連結角色可讓您輕鬆設定 Amazon Braket，因為您不必手動新增必要的許可。Amazon Braket 定義了其服務連結角色的許可。除非您變更這些定義，否則只有 Amazon Braket 可以擔任其角色。定義的權限包括信任原則和權限原則。許可原則無法附加到其他任何 IAM 實體。

Amazon Braket 設定的服務連結角色是 AWS Identity and Access Management (IAM) 服務連結角色功能的一部分。如需其他AWS 服務支援服AWS務連結角色的相關資訊，請參閱搭配 IAM 使用的服務，並在服務連結角色欄中尋找具有是的服務。選擇具有連結的 Yes (是)，以檢視該服務的服務連結角色文件。

亞馬遜網站的服務連結角色許可

亞馬遜網站使用信任胸罩的`AWSServiceRoleForAmazonBraket`服務連結角色來擔任該角色。

您必須設定許可，以允許 IAM 實體 (例如群組或角色) 建立、編輯或刪除服務連結角色。如需詳細資訊，請參閱服務連結角色權限。

Amazon Braket 中的服務連結角色預設會被授與下列許可：

- Amazon S3 — 列出帳戶中存儲桶的許可，並將對象放入帳戶中的任何存儲桶並從其中獲取對象，名稱以 amazon-bra ket-開頭。
- Amazon CloudWatch 日誌 — 列出和建立日誌群組、建立關聯的日誌串流，以及將事件放入為 Amazon Braket 建立的日誌群組的許可。

下列原則附加至**`AWSServiceRoleForAmazonBraket`**服務連結角色：

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

亞馬遜布拉基特的韌性

AWS全球基礎架構是圍繞AWS 區域和可用區域建立的。

每個區域提供多個實體分離和隔離的可用區域。這些可用區域 (AZ) 透過低延遲、高輸送量和高度備援的網路連線。因此，與傳統的單一或多資料中心基礎架構相比，可用性區域具有更高的可用性、容錯能力和可擴充性。

您可以設計和操作在 AZ 之間自動容錯移轉的應用程式和資料庫，而不會中斷。

如需有關AWS 區域和可用區域的詳細資訊，請參閱AWS全域基礎結構。

亞馬遜文字架的合規驗證

第三方稽核員會定期評估 Amazon Braket 的安全性和合規性，以及我們與第三方硬體供應商的整合。有關 Braket 的合規性信息up-to-date列表，請參閱AWS 服務合規計劃的範圍。如需一般資訊，請參閱AWS符合性。

您可使用 AWS Artifact 下載第三方稽核報告。如需詳細資訊，請參閱在中下載報告AWS Artifact。

> **Note**
>
> AWS合規性報告不涵蓋第三方硬體供應商所提供的 QPU，他們可以選擇進行自己的獨立稽核。

使用 Amazon Braket 時的合規責任取決於資料的敏感度、公司的合規目標以及適用的法律和法規。AWS提供下列資源以協助遵循法規：

- 安全與合規快速入門指南 – 這些部署指南討論在 AWS 上部署以安全及合規為重心基準環境的架構考量和步驟。
- AWS 合規資源 – 這組手冊和指南可能適用於您的產業和位置。

亞馬遜網站中的基礎設施安全

作為受管服務，Amazon Braket 受到安AWS全流程概觀白皮書中所述的全球網路安全程序的保護。AWS

若要透過網路存取 Amazon Braket，您可以呼叫已發佈的 AWS API。用戶端必須支援 Transport Layer Security (TLS) 1.2 或更新版本。客戶還必須支持具有完美正向保密性（PFS）的密碼套件，例如短暫的迪菲-赫爾曼（DHE）或橢圓曲線短暫迪菲-赫爾曼（ECDHE）。現代系統 (如 Java 7 和更新版本) 大多會支援這些模式。

此外，請求必須使用存取金鑰 ID 和與 IAM 主體相關聯的私密存取金鑰來簽署。或者，您可以使用 AWS Security Token Service (AWS STS) 來產生暫時安全憑證來簽署請求。

Amazon Braket 硬件供應商的安全

亞馬遜硬體上的 QPU 是由第三方硬體供應商代管。當您在 QPU 上執行量子任務時，Amazon Braket 會在將電路傳送至指定的 QPU 進行處理時，使用 DeviceEearn 做為識別碼。

如果您使用 Amazon Braket 存取由第三方硬體供應商操作的量子運算硬體，則您的電路及其相關資料將由硬體供應商處理，該硬體供應商會在由其操作的設施之外進行處理AWS。有關實體位置及AWS每個 QPU 可用的區域位於裝置資料Amazon Braket 控制台的部分。

您的內容是匿名處理的。僅將處理電路所需的內容發送給第三方。AWS 帳戶信息不會傳輸給第三方。

所有資料一律於靜態及傳輸中加密。解密資料僅供處理之用。除了處理您的電路以外，Amazon Braket 第三方供應商不得存放或使用您的內容作為其他用途。電路完成後，結果將返回到 Amazon Braket 並存放在您的 S3 儲存貯體中。

Amazon Braket 第三方量子硬體供應商的安全性會定期稽核，以確保符合網路安全、存取控制、資料保護和實體安全標準。

Amazon VPC 端點適用於 Amazon Braket

您可以透過建立介面 VPC 端點，在 VPC 和 Amazon Braket 之間建立私人連線。介面端點採用這項技術 AWS PrivateLink，可在沒有網際網路閘道、NAT 裝置、VPN 連線或AWS Direct Connect連線的情況下存取 Braket API。VPC 中的執行個體不需要公有 IP 位址即可與 Braket API 進行通訊。

每個介面端點都是由您子網路中的一或多個彈性網路介面表示。

VPC 和 Braket 之間的流量不會離開Amazon網路，這會增加您與雲端應用程式共用資料的安全性，因為這樣可以減少資料暴露在公用網際網路上的風險。 PrivateLink如需詳細資訊，請參閱 Amazon VPC 使用者指南中的介面虛擬私人雲端端點 (AWS PrivateLink)。

Amazon Braket 路磁碟機 VPC 端點的注意事項

在為 Braket 設定介面 VPC 端點之前，請務必先查看 Amazon VPC 使用者指南中的界面端點屬性和限制。

Braket 支援從您的 VPC 呼叫其所有 API 動作。

預設情況下，允許透過 VPC 端點完全存取 Braket。如果您指定 VPC 端點原則，則可以控制存取權。如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的使用 VPC 端點控制對服務的存取。

設置胸針和 PrivateLink

若要AWS PrivateLink搭配 Amazon Braket 使用，您必須建立一個 Amazon Virtual Private Cloud 端 (Amazon VPC) 端點作為界面，然後透過 Amazon Bra API ket 服務連接到端點。

以下是此過程的一般步驟，將在後面的章節中詳細說明。

- 設定並啟動 Amazon VPC 來託管您的AWS資源。如果您已有 VPC，則可以略過此步驟。
- 為布拉基特創建一個 Amazon VPC 端點
- 通過您的端點 Connect 和運行 Braket 量子任務

步驟 1：如果需要，啟動 Amazon VPC

請記住，如果您的帳戶已經運行了 VPC，則可以跳過此步驟。

VPC 可控制您的網路設定，例如 IP 位址範圍、子網路、路由表和網路閘道。基本上，您是在自訂虛擬網路中啟動AWS資源。如需 VPC 的詳細資訊，請參閱 Amazon VPC 使用者指南。

開啟 Amazon VPC 主控台並建立具有子網路、安全群組和網路閘道的新 VPC。

步驟 2：為 Braket 創建一個接口 VPC 端點

您可以使用 Amazon VPC 主控台或 () 為 Braket 服務建立 VPC 端點。AWS Command Line Interface AWS CLI如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的建立介面端點。

若要在主控台中建立 VPC 端點，請開啟 Amazon VPC 主控台，開啟「端點」頁面，然後繼續建立新端點。記下端點 ID 以供以後參考。當您對 Braket API 進行某些呼叫時，它是`–endpoint-url`標誌的一部分所必需的。

使用以下服務名稱為 Braket 建立 VPC 端點：

- `com.amazonaws.substitute_your_region.braket`

注意：如果您為端點啟用私有 DNS，則可以使用該區域的預設 DNS 名稱向 Braket 提出API要求，`braket.us-east-1.amazonaws.com`例如。

如需詳細資訊，請參閱《Amazon VPC 使用者指南》中的透過介面端點存取服務。

步驟 3：透過您的端點 Connect 並執行 Braket 量子任務

建立 VPC 端點之後，您可以執行包含`endpoint-url`參數的 CLI 命令，以指定API或執行階段的介面端點，例如下列範例：

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

如果您為 VPC 端點啟用私人 DNS 主機名稱，則不需要在 CLI 命令中將端點指定為 URL。相反地，CLI 和 Amazon Braket SDK 預設使用的 Braket API DNS 主機名稱會解析為您的 VPC 端點。它具有以下示例所示的形式：

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

名為透過使用AWS PrivateLink端點從 Amazon VPC 直接存取 Amazon SageMaker 筆記本的部落格文章提供了一個範例，說明如何設定端點以安全連線到 SageMaker 筆記型電腦，這些筆記型電腦類似於 Amazon Braket 筆記型電腦。

如果您按照博客文章中的步驟進行操作，請記住替換 Amazon SageMaker 的名稱 AmazonBraket。對於服務名稱，如果您的地區不是 us-ea st-1，請在該字符串中輸入`com.amazonaws.us-east-1.braket`或替換正確的AWS 區域名稱。

有關建立端點的更多資訊

- 如需如何使用私有子網路建立 VPC 的相關資訊，請參閱使用私有子網路建立 VPC
- 如需使用 Amazon VPC 主控台或 AWS CLI，請參閱 Amazon VPC 使用者指南中的建立界面端點。
- 如需使用建立和設定端點的相關資訊AWS CloudFormation，請參閱使用指南中的AWS:: EC2:: vpcendPoint 資源。AWS CloudFormation

使用 Amazon VPC 端點政策控制存取

若要控制對 Amazon Braket 的連線存取，您可以將 AWS Identity and Access Management (IAM) 端點政策附加到您的 Amazon VPC 端點。此政策會指定下列資訊：

- 可以執行動作的主參與者 (使用者或角色)。
- 可執行的動作。
- 可供執行動作的資源。

如需詳細資訊，請參閱 Amazon VPC 使用者指南中的使用 VPC 端點控制對服務的存取。

範例：用於編輯動作的 VPC 端點原則

下列範例顯示 Braket 的端點原則。連接到端點時，此策略會授予對所有資源上所有主參與者列出的 Braket 動作的存取權。

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

您可以透過附加多個端點政策來建立複雜的 IAM 規則。如需詳細資訊和範例，請參閱：

- 適用於 Step Functions 的 Amazon Virtual Private Cloud 端端點政策
- 為非管理員使用者建立精細的 IAM 許可
- 使用 VPC 端點控制對服務的存取

Amazon Braket 故障排除

使用本節中的疑難排解資訊和解決方案來協助解決 Amazon Braket 的問題。

在本節中：

AccessDeniedException

如果您在啟用或使用 Braket AccessDeniedException時收到，您可能會嘗試在您受限制角色無法存取的區域中啟用或使用 Braket。

在這種情況下，您應該聯絡您的內部 AWS 管理員，以瞭解下列哪些條件適用：

- 如果有角色限制阻止訪問區域。
- 如果您嘗試使用的角色被允許使用 Braket。

如果您的角色在使用 Braket 時無法存取特定區域，則您將無法使用該特定地區的裝置。

調用 CreateQuantumTask 操作時ValidationException發生錯誤（ ）

如果您收到類似下列類似的錯誤：`An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller doesn't have access to amazon-braket-…` 請檢查您所參考的是現有的 s3_folder。硬碟不會 auto 為您建立新的 Amazon S3 儲存貯體和前置字元。

如果您API直接存取並收到類似下列類似的錯誤：`Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET`請檢查您是否未包含`s3://`在 Amazon S3 儲存貯體路徑中。

SDK 功能不起作用

您的 Python 版本必須是 3.9 或更高版本。對於 Amazon Braket 混合任務，我們建議使用 Python 3.10。

確認您的 SDK 和結構描述是否正確 up-to-date。若要從筆記本或 Python 編輯器更新 SDK，請執行下列命令：

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

若要更新結構描述，請執行下列命令：

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

如果您是從自己的用戶端存取 Amazon Braket，請確認您的AWS 區域已設定為 Amazon Braket 支援的區域。

混合式工作失敗，原因是 ServiceQuotaExceededException

如果您超過目標模擬器裝置的並行量子任務限制，則在 Amazon Braket 模擬器上執行量子任務的混合任務可能會無法建立。如需服務限制的詳細資訊，請參閱配額主題。

如果您在帳戶中的多個混合作業中針對模擬器裝置執行並行工作，則可能會遇到此錯誤。

若要查看針對特定模擬器裝置的並行量子工作數目，請使用 `search-quantum-tasks`API,，如下列程式碼範例所示。

```
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
```

您也可以使用 Amazon CloudWatch 指標，在裝置上檢視建立的量子任務：「Braket」>「按裝置」。

為了避免遇到這些錯誤：

1. 要求提高模擬器裝置的並行量子任務數量的服務配額。這僅適用於設SV1備。
2. 處理代碼中的ServiceQuotaExceeded異常並重試。

組件在筆記本實例中停止工作

如果筆記型電腦的某些元件停止運作，請嘗試下列動作：

1. 將您建立或修改的任何筆記本下載至本機磁碟機。
2. 停止您的筆記本實例。
3. 刪除筆記本執行個體。
4. 使用不同的名稱建立新的記事本執行個體。
5. 將記事本上傳到新的執行個體。

Amazon Braket 配額

下表列出亞馬遜網站的服務配額。服務配額 (也稱為限制) 是您的服務資源或作業數目上限 AWS 帳戶。

可以增加一些配額。如需詳細資訊，請參閱 AWS 服務 配額。

- 突發率配額無法增加。
- 可調配額的最大速率增加 (突發速率除外，無法調整) 是指定預設速率限制的 2 倍。例如，預設配額 60 可調整為最大 120。
- 並發 SV1 (DM1) 量子任務的可調配額允許每個最多 60 個 AWS 區域。
- 混合式工作允許的最大運算執行個體數量為 5，且配額可調整。

資源	描述	限制	可調整
API要求率	在目前區域中，您可以在此帳戶中每秒傳送的要求數目上限。	140	是

資源	描述	限制	可調整
API請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外請求數目上限 (RPS)。	600	否
CreateQuantumTask 要求率	每個區域每秒可傳送的CreateQuantumTask 要求數目上限。	20	是
CreateQuantumTask 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外CreateQuantumTask 請求數目上限 (RPS)。	40	否
SearchQuantumTasks 要求率	每個區域每秒可傳送的SearchQuantumTasks 要求數目上限。	5	是
SearchQuantumTasks 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外SearchQuantumTasks 請求數目上限 (RPS)。	50	否
GetQuantumTask 要求率	每個區域每秒可傳送的GetQuantumTask 要求數目上限。	100	是

資源	描述	限制	可調整
GetQuantumTask 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外GetQuantumTask 請求數目上限 (RPS)。	500	否
CancelQuantumTask 要求率	每個區域每秒可傳送的CancelQuantumTask 要求數目上限。	2	是
CancelQuantumTask 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外CancelQuantumTask 請求數目上限 (RPS)。	20	否
GetDevice 要求率	每個區域每秒可傳送的GetDevice 要求數目上限。	5	是
GetDevice 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外GetDevice 請求數目上限 (RPS)。	50	否
SearchDevices 要求率	每個區域每秒可傳送的SearchDevices 要求數目上限。	5	是

資源	描述	限制	可調整
SearchDevices 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外SearchDevices 請求數目上限 (RPS)。	50	否
CreateJob 要求率	每個區域每秒可傳送的CreateJob 要求數目上限。	1	是
CreateJob 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外CreateJob 請求數目上限 (RPS)。	5	否
SearchJob 要求率	每個區域每秒可傳送的SearchJob 要求數目上限。	5	是
SearchJob 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外SearchJob 請求數目上限 (RPS)。	50	否
GetJob要求率	每個區域每秒可傳送的GetJob要求數目上限。	5	是
GetJob請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外GetJob請求數目上限 (RPS)。	25	否

資源	描述	限制	可調整
CancelJob 要求率	每個區域每秒可傳送的CancelJob 要求數目上限。	2	是
CancelJob 請求突發率	在目前區域的此帳戶中，您每秒可以傳送的額外CancelJob 請求數目上限 (RPS)。	5	否
並發SV1量子任務的數量	目前 Region 中在狀態向量模擬器 (SV1) 上執行的並行量子工作的最大數量。	100 us-east-1, 美國西部 50 號， 100 us-west-2, 歐盟西部 50 號	否
並發DM1量子任務的數量	在當前 Region 中密度矩陣模擬器（DM1）上運行的並發量子任務的最大數量。	100 us-east-1, 美國西部 50 號， 100 us-west-2, 歐盟西部 50 號	否
並發TN1量子任務的數量	在當前區域中，張量網絡模擬器（TN1）上運行的並發量子任務的最大數量。	10 us-east-1, 美國西部 10 號， 5 eu-west-2，	是
並行混合式作業數目	目前區域中同時混合式工作的最大數目。	5	是
混合式工作執行時間	混合式工作可以執行的時間上限 (以天為單位)。	5	否

以下是混合式工作的預設傳統運算執行個體配額。若要提高這些配額，請聯絡 AWS Support。此外，還會針對每個執行個體指定可用區域。

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c4.xlarge 的執行個體數目上限。	5	是	是	是	是	是	否
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c4.2xlarge 的執行個體數目上限。	5	是	是	是	是	是	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (ml.c4.4xlarge)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c4.4xlarge 的執行個體數目上限。	5	是	是	是	是	是	否
混合式作業的最大執行個體數目上限 (ml.c4.8x 大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c4.8xlarge 的執行個體數目上限。	5	是	是	是	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式作業的最大執行個體數目上限 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5.xlarge 的執行個體數目上限。	5	是	是	是	是	是	是
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5.2xlarge 的執行個體數目上限。	5	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5.4xlarge 的執行個體數目上限。	1	是	是	是	是	是	是
混合式作業的最大執行個體數目上限 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5.9xlarge 的執行個體數目上限。	1	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式作業的最大執行個體數目上限 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5.18xlarge 的執行個體數目上限。	0	是	是	是	是	是	是
混合式作業的最大執行個體數目上限 (ml.c5n.xlarge)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5n.xlarge 的執行個體數目上限。	0	是	是	是	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式作業的最大執行個體數目上限 (ml.c5n.2XL)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5n.2xlarge 的執行個體數目上限。	0	是	是	是	是	否	否
混合式作業的最大執行個體數目上限為 ml.c5n.4xlarge	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5n.4xlarge 的執行個體數目上限。	0	是	是	是	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式作業的最大執行個體數目上限為 ml.c5n.9x 大	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5n.9xlarge 的執行個體數目上限。	0	是	是	是	是	否	否
混合式作業的最大執行個體數量 (ml.c5n.18x 大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.c5n.18xlarge 的最大執行個體數量。	0	是	是	是	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (ml.g4dn.xlarge)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.xlarge 的執行個體數目上限。	0	是	是	是	是	是	是
混合式工作的最大執行個體數量 (ml.g4dn.2 x 大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.2xlarge 的執行個體數目上限。	0	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數量 (ml.g4dn.4xlarge)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.4xlarge 的執行個體數目上限。	0	是	是	是	是	是	是
混合式工作的最大執行個體數量 (ml.g4dn.8 x 大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.8xlarge 的執行個體數目上限。	0	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數量 (ml.g4dn.12 個大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.12xlarge 的執行個體數目上限。	0	是	是	是	是	是	是
混合式工作的最大執行個體數量 (ml.g4dn.16 x 大)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.g4dn.16xlarge 的執行個體數目上限。	0	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m4.xlarge 的執行個體數目上限。	5	是	是	是	是	是	否
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m4.2xlarge 的執行個體數目上限。	5	是	是	是	是	是	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目毫升 .m4.4xlarge	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m4.4xlarge 的執行個體數目上限。	2	是	是	是	是	是	否
混合式工作的最大執行個體數目毫升 .m4.10x 大	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m4.10xlarge 的執行個體數目上限。	0	是	是	是	是	是	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目毫升 .m4.16	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m4.16› large 的執行個體數目上限。	0	是	是	是	是	是	否
混合式作業的最大執行個體數目 (ml.m5.large)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.large 的執行個體數目上限。	5	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目上限 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.xlarge 的執行個體數目上限。	5	是	是	是	是	是	是
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.2xlarge 的執行個體數目上限。	5	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.4xlarge 的執行個體數目上限。	5	是	是	是	是	是	是
混合式作業的最大執行個體數量 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.12xlarge 的執行個體數目上限。	0	是	是	是	是	是	是

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數量 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.m5.24› large 的執行個體數目上限。	0	是	是	是	是	是	是
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p2.xlarge 的執行個體數目上限。	0	是	是	否	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式作業的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p2.8xlarge 的執行個體數目上限。	0	是	是	否	是	否	否
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p2.16xlarge 的執行個體數目上限。	0	是	是	否	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p3.2xlarge 的執行個體數目上限。	0	是	是	否	是	否	否
混合式作業的執行個體數目上限為 ml.p4d.24 小時	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p4d.24xlarge 的最大執行個體數量。	0	是	是	否	是	否	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的執行個體數目上限為 ml.p3dn.24 小時	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p3dn.24xlarge 的最大執行個體數量。	0	是	是	否	是	否	否
混合式作業的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p3.8xlarge 的執行個體數目上限。	0	是	是	否	是	是	否

資源	描述	限制	可調整	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
混合式工作的最大執行個體數目 (毫升)	此帳戶和區域中所有 Amazon Braket 混合式任務允許類型為 ml.p3.16xlarge 的執行個體數目上限。	0	是	是	否	是	是	否

要求限制更新

如果您收到執行個體類型的 ServiceQuotaExceeded 例外狀況，但沒有足夠的執行個體可用，則可以從 AWS 主控台的「Ser vice Quotas」頁面要求提高限制，然後在「服 AWS 務」下搜尋 Amazon Braket。

> **Note**
>
> 如果您的混合式工作無法佈建要求的 ML 運算容量，請使用其他區域。此外，如果您在表格中沒有看到執行個體，則無法用於「混合式工作」。

額外配額和限制

- Amazon Braket 量子任務動作的大小限制為 3MB。
- SV1、和Rigetti裝置每項工作允許的最DM1大拍攝數量為 100,000 張。
- 每個任務允許的最大拍攝數量TN1為 1000 張。

- 對於阿里亞 -1 和阿麗亞 -2 設備，每個任務IonQ的最大值為 5,000 張。對於IonQ的和諧和長處設備和OQC設備，最大值為 10,000。
- 對於QuEra，每個任務允許的最大射門數為 1000。
- 對於TN1和QPU裝置，每個工作的快照必須大於 0。

疑難排解開啟問題

本節提供在使用 OpenQASM 3.0 時遇到錯誤時可能會很有用的疑難排解指標。

在本節中：

包含陳述式錯誤

布拉科特目前沒有要包含在 OpenQASM 程序中的標準門庫文件。例如，下列範例會引發剖析器錯誤。

```
OPENQASM 3;
include "standardlib.inc";
```

此代碼生成錯誤消息：`No terminal matches '"' in the current parser context, at line 2 col 17.`

非連續錯誤 qubits

`true`在設備功能中設置為的設備qubits上使用非連續`requiresContiguousQubitIndices`會導致錯誤。

在模擬器上運行量子任務時IonQ，下面的程序觸發錯誤。

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

此代碼生成錯誤消息：`Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

混合物理qubits與虛擬qubits錯誤

不允許在同一程序qubits中將物理qubits與虛擬混合使用，並導致錯誤。下列程式碼會產生錯誤。

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

此代碼生成錯誤消息：`[line 4] mixes physical qubits and qubits registers.`

請求結果類型並qubits在同一程序錯誤中進行測量

請求結果類型和在相同程序中明確測量的結果qubits會導致錯誤。下列程式碼會產生錯誤。

```
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])
```

此代碼生成錯誤消息：`Qubits should not be explicitly measured when result types are requested.`

經典和qubit寄存器限制超過錯誤

只允許一個傳統的寄存器和一個qubit寄存器。下列程式碼會產生錯誤。

```
OPENQASM 3;

qubit[2] q0;
qubit[2] q1;
```

此代碼生成錯誤消息：`[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

框前面沒有逐字編譯錯誤

所有框必須在前面加上逐字編譯。下列程式碼會產生錯誤。

```
box{
rx(0.5) $0;
}
```

此代碼生成錯誤消息：`In verbatim boxes, native gates are required. x is not a device native gate.`

逐字框缺少本地門錯誤

逐字框應該具有本地門和物理門。qubits下面的代碼生成本機門錯誤。

```
#pragma braket verbatim
box{
x $0;
}
```

此代碼生成錯誤消息：`In verbatim boxes, native gates are required. x is not a device native gate.`

逐字框缺少物理錯誤 qubits

逐字框必須有物理的。qubits下列程式碼會產生遺失的實體qubits錯誤。

```
qubit[2] q;

#pragma braket verbatim
box{
rx(0.1) q[0];
}
```

此代碼生成錯誤消息：`Physical qubits are required in verbatim box.`

逐字編譯缺少「胸章」錯誤

您必須在逐字編譯中包含「胸章」。下列程式碼會產生錯誤。

```
#pragma braket verbatim          // Correct
#pragma verbatim                 // wrong
```

此代碼生成錯誤消息：`You must include “braket” in the verbatim pragma`

單一qubits無法編入索引錯誤

單一qubits無法編製索引。下列程式碼會產生錯誤。

```
OPENQASM 3;

qubit q;
h q[0];
```

此代碼生成錯誤：`[line 4] single qubit cannot be indexed.`

但是，單個qubit數組可以按如下方式進行索引：

```
OPENQASM 3;

qubit[1] q;
h q[0];   // This is valid
```

兩qubit門qubits中的物理未連接錯誤

若要使用實體qubits，請先檢查以確認裝置是否使用實體，`device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits`然後qubits透過檢查`device.properties.paradigm.connectivity.connectivityGraph`或來驗證連線圖表`device.properties.paradigm.connectivity.fullyConnected`。

```
OPENQASM 3;

cnot $0, $14;
```

此代碼生成錯誤消息：`[line 3] has disconnected qubits 0 and 14`

GetDevice 不會傳回開啟品質管理結果錯誤

如果您在使用開發套件時沒有看到 OpenQASM 的 GetDevice 回應結果，您可能需要設定 AWS_EXECTION_ENV 環境變數來設定使用者代理程式。有關如何針對 Go 和 Java SDK 執行此操作，請參閱下面提供的代碼示例。

若要在使用下列項目時設定 AWS_EXECION_ENV 環境變數以設定使用者代理程式： 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]
```

若要設定 AW_EXECION_ENV 環境變數，以便在使用 Boto3 時設定使用者代理程式：

```
import boto3
import botocore

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

若要設定 AW_EXECION_ENV 環境變數，以便在 JavaScript使用/(SDK v2) 時設定使用者代理程式：TypeScript

```
import Braket from 'aws-sdk/clients/braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0'  });
```

若要設定 AW_EXECION_ENV 環境變數，以便在 JavaScript使用/(SDK v3) 時設定使用者代理程式：TypeScript

```
import { Braket } from '@aws-sdk/client-braket';
const client = new Braket({ region: 'us-west-2', credentials: AWS_CREDENTIALS,
 customUserAgent: 'BraketSchemas/1.8.0' });
```

若要設定 AWS_EXECION_ENV 環境變數，以便在使用 Go SDK 時設定使用者代理程式：

```
os.Setenv("AWS_EXECUTION_ENV", "BraketGo BraketSchemas/1.8.0")
mySession := session.Must(session.NewSession())
svc := braket.New(mySession)
```

若要設定 AW_ 執行環境變數，以便在使用 Java SDK 時設定使用者代理程式：

```
ClientConfiguration config = new ClientConfiguration();
config.setUserAgentSuffix("BraketSchemas/1.8.0");
BraketClient braketClient =
 BraketClientBuilder.standard().withClientConfiguration(config).build();
```

本地模擬器支持警告

這些功能LocalSimulator支援 OpenQASM 中可能無法在 QPU 或隨選模擬器上使用的進階功能。如果您的程式只包含特定於的語言功能LocalSimulator，如下列範例所示，您將收到警告。

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

此程式碼會產生警告：`此程式使用僅支援的 OpenQASM 語言功能。 LocalSimulatorQPU 或隨選模擬器可能不支援其中某些功能。

如需有關支援 OpenQASM 功能的詳細資訊，請按一下這裡。

亞馬遜標題的 API 和 SDK 參考指南

Amazon Braket 提供 API、開發套件和命令列界面，可讓您用來建立和管理筆記本執行個體，以及訓練和部署模型。

- 亞馬遜蟒蛇 SDK（推薦）
- 亞馬遜標題 API 參考
- 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

您也可以從 Amazon Braket 教學課程GitHub儲存庫取得程式碼範例。

- 布拉克教程 GitHub

文件歷史記錄

下表說明此版本 Amazon Braket 的說明文件。

- API版本：二 ○○ 二年四月二十八日
- API參考資料最新更新：2023 年 9 月 25 日
- 最新文件更新：2024 年 5 月 22 日

變更	Description	日期
新裝置IQM Garnet與地區 Europe North 1	新增對 IQM 石榴石裝置的支援。具有方形晶格拓撲的 20 量子位元裝置。將布拉基特支持的地區擴展到歐洲北部 1（斯德哥爾摩）。	2024年5月22 日
發行局部解調	實驗功能現在包括 Aquila QPU 的 QuEra局部解調功能。	2024年4月11日
筆記本不活動管理器發布	建立筆記本執行個體時，請啟用閒置管理員並設定閒置持續時間以自動重設 Braket 筆記本執行個體。	2024年3月27日
目錄返工	重新整理 Amazon Braket 目錄，以遵守 AWS 樣式指南要求，並改善客戶體驗的內容流程。	2023 年 12 月 12 日
發行 布拉基直接	增加了對 Braket 直接功能的支持，包括： • 保留 • 專家建議 • 實驗能力	2023 年 11 月 27 日

已更新 創建一個亞馬遜電腦筆記本實例	已更新文件，以新增資訊，以便為新的和現有的 Amazon Braket 客戶建立筆記本執行個體。	2023 年 11 月 27 日
已更新 攜帶您自己的容器 (BYOC)	已更新文件，以新增有關何時到 BYOC、BYOC 的配方，以及在容器上執行 Braket 混合作業的相關資訊。	2023 年 10 月 18 日
混合作業裝飾器發布	添加以混合工作的形式執行您的本機程式碼頁面。包含範例： • 從本地 Python 代碼創建一個混合任務 • 安裝額外的 Python 軟件包和源代碼 • 將資料儲存並載入混合式作業執行個體 • 混合工作裝飾器的最佳實踐	2023 年 10 月 16 日
新增佇列可見性	更新了開發人員指南文檔以包含queue depth和queue position。 更新 API 文件化，以反映佇列可見度的新 API 變更。	2023 年 9 月 25 日
標準化文件中的命名	更新了文檔以將「工作」的任何實例更改為「混合任務」，「任務」更改為「量子任務」	2023 年 9 月 11 日
新裝置 IonQ Aria 2	增加了對IonQ Aria 2設備的支持	2023 年 9 月 8 日

更新了本地門	已更新文件，以新增有關從 Rigetti 以程式設計方式存取原生閘門的相關資訊。	2023 年 8 月 16 日
Xanadu出發	更新了文檔以刪除所有Xanadu設備	2023 年 6 月 2 日
新裝置 IonQ Aria	增加了對IonQ Aria設備的支持	2023 年 5 月 16 日
淘汰的Rigetti裝置	停止支援 Rigetti Aspen-M-2	2023 年 5 月 2 日
更新的AmazonBraketFullAccess政策資訊	更新了定義AmazonBraketFullAccess政策內容的指令碼，以包含服務GetServiceQuota和 cloudwatch:GetMetricData 動作以及有關配額限制的資訊。	2023 年 4 月 19 日
引導式旅程啟動	更改了文檔以反映 Braket 入職的最新和簡化方法。	2023 年 4 月 5 日
新裝置 Rigetti Aspen-M-3	增加了對Rigetti Aspen-M-3設備的支持	2023 年 1 月 17 日
新的伴隨漸變功能	已新增關於由提供的伴隨漸層功能的資訊 SV1	2022 年 12 月 7 日
新的演算法庫功能	已新增 Braket 演算法程式庫的相關資訊，該函式庫提供預先建置的量子演算法目錄	2022 年 11 月 28 日
D-Wave出發	更新了文檔，以適應去除所有D-Wave設備	2022 年 11 月 17 日
新裝置 QuEra Aquila	增加了對QuEra Aquila設備的支持	2022 年 10 月 31 日

Support 煞車脈衝	增加了對 Braket 脈衝的支持，允許在設備上Rigetti使用脈衝控制 OQC	2022 年 10 月 20 日
Support IonQ 原生閘門	增加了對 ionQ 設備提供的本地門組的支持	2022 年 9 月 13 日
新執行個體配額	更新與混合式工作相關的預設傳統運算執行個體配額	2022 年 8 月 22 日
新服務儀表板	已更新主控台螢幕擷取畫面以包含服務	2022 年 8 月 17 日
新裝置 Rigetti Aspen-M-2	增加了對Rigetti Aspen-M-2設備的支持	2022 年 8 月 12 日
全新開啟品質管理功能	增加了 OpenQASM 功能支持本地模擬器（布拉克 _sv 和制動器 _dm）	2022 年 8 月 4 日
新的成本追蹤程序	新增如何取得模擬器和硬體工作負載的近乎即時的最大成本估算	2022 年 7 月 18 日
新Xanadu Borealis裝置	增加了對Xanadu Borealis設備的支持	2022 年 6 月 2 日
新的入職簡化程序	已新增簡化新上線程序如何運作的資訊	2022 年 5 月 16 日
新裝置 D-Wave Advantage _system6.1	增加了對D-WaveAdvantage_system6.1設備的支持	2022 年 5 月 12 日
Support 嵌入式模擬器	新增如何使用混合式工作執行嵌入式模擬，以及如何使用 PennyLane 閃電模擬器	2022 年 5 月 4 日

AmazonBraketFullAccess - Amazon Braket 的完整訪問政策	新增 s3：ListAllMyBucket s 允許使用者檢視和檢查為 Amazon Braket 建立和使用的儲存貯體的許可	2022 年 3 月 31 日
Support 開放式品質管理	新增支援閘門式量子裝置和模擬器的 OpenQASM 3.0	2022 年 3 月 7 日
新的量子硬件供應商Oxford Quantum Circuits和新地區，eu-west-2	增加了對 OQC 和 eu-west-2 的支持	2022 年 2 月 28 日
新Rigetti裝置	新增對 Rigetti Aspen M-1 的支援	2022 年 2 月 15 日
新資源限制	同時DM1和SV1任務的最大數量從 55 增加到 100	2022 年 1 月 5 日
新Rigetti裝置	新增對 Rigetti Aspen-11 的支援	2021 年 12 月 20 日
淘汰的Rigetti裝置	為Rigetti Aspen-10設備停止支持	2021 年 12 月 20 日
新結果類型	局部密度矩陣模擬器和DM1設備支持的降低密度矩陣結果類型	2021 年 12 月 20 日
更新的策略說明	Amazon Braket 更新角色 ARN 包括路徑。`servicerole/` 如需政策更新的相關資訊，請參閱 Amazon Braket 對 AWS 受管政策的更新表。	2021 年 11 月 29 日
Amazon Braket 工作	Amazon Braket 混合任務的用戶指南並API添加	2021 年 11 月 29 日
新Rigetti裝置	新增對 Rigetti Aspen-10 的支援	2021年11月20日

淘汰的D-Wave裝置	已停止支援 D-Wave QPU，Advantage_system1	2021 年 11 月 4 日
新D-Wave裝置	增加了對額外 D-Wave QPU 的支持，Advantage_system4	2021 年 10 月 5 日
全新噪音模擬器	增加了對密度矩陣模擬器（DM1）的支持，該模擬器可以模擬多達 17 的電路qubits和局部噪聲模擬器 bra ket_dm	2021 年 5 月 25 日
PennyLane 支持	增加了對 Amazon Bra PennyLane ket 的支持	2020 年 12 月 8 日
新的模擬器	增加了對張量網絡模擬器（TN1）的支持，該模擬器允許更大的電路	2020 年 12 月 8 日
工作批次處理	支援客戶任務批次處理	2020 年 11 月 24 日
手動qubit配置	支持設備上的Rigetti手動qubit分配	2020 年 11 月 24 日
可調配額	Braket 為您的任務資源提供自助式調整配額	2020 年 10 月 30 日
Support PrivateLink	您可以為您的 Braket 任務設定私有 VPC 端點	2020 年 10 月 30 日
標籤的支援	Braket 支持量子任務資源的API基礎標籤	2020 年 10 月 30 日
新D-Wave裝置	增加了對額外 D-Wave QPU 的支持，Advantage_system1	2020 年 9 月 29 日
初始版本	Amazon Braket 文件的初始版本	2020 年 8 月 12 日

AWS 詞彙表

如需最新的AWS術語，請參閱AWS詞彙參考中的AWS詞彙表。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。