



Guide du développeur

Amazon Braket



Amazon Braket: Guide du développeur

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Les marques et la présentation commerciale d'Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n'est pas d'Amazon, d'une manière susceptible de créer une confusion parmi les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Les marques et la présentation commerciale d'Amazon ne peuvent être utilisées en relation avec un produit ou un service qui n'est pas d'Amazon, d'une manière susceptible de créer une confusion parmi les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce qu'Amazon Braket ?	1
Comment ça marche	4
Flux de tâches quantique Amazon Braket	5
Traitement des données par des tiers	6
Termes et concepts d'Amazon Braket	6
AWS terminologie et astuces pour Amazon Braket	11
Suivi des coûts et économies	12
Définition des limites de dépenses pour Amazon Braket QPUs	12
Suivi des coûts en temps quasi réel	16
Les meilleures pratiques pour réduire les coûts	18
Références d'API et dépôts	20
Référentiels principaux	21
Plug-ins	21
Régions et appareils pris en charge	22
Régions et points de terminaison	26
Prise en main	28
Activer Amazon Braket	28
Conditions préalables	29
Étapes pour activer Amazon Braket	29
Création d'une instance de bloc-notes Amazon Braket	30
(Avancé) Créez un bloc-notes Braket à l'aide de CloudFormation	32
Étape 1 : Création d'un script de configuration du cycle de vie de l' SageMaker IA	33
Étape 2 : créer le rôle IAM assumé par Amazon SageMaker AI	34
Étape 3 : créer une instance de bloc-notes SageMaker AI avec le préfixe amazon-braket-	35
Génération	36
Construisez votre premier circuit	36
Création de vos premiers algorithmes quantiques	41
Construire des circuits dans le SDK	42
Inspection du circuit	58
Liste des types de résultats	60
Obtenir des conseils d'experts	66
(Avancé) Exécutez vos circuits avec OpenQASM 3.0	67
Qu'est-ce qu'OpenQASM 3.0 ?	68

Quand utiliser OpenQASM 3.0	69
Comment fonctionne OpenQASM 3.0	69
Conditions préalables	69
Quelles sont les fonctionnalités d'OpenQASM prises en charge par Braket ?	69
Créez et soumettez un exemple de tâche quantique OpenQASM 3.0	76
Support d'OpenQASM sur différents appareils Braket	78
Simuler le bruit	89
Qubitrecâblage	90
Compilation textuelle	91
La console Braket	92
Ressources supplémentaires	92
Gradients de calcul	92
Mesurer des qubits spécifiques	93
(Avancé) Explorez les capacités expérimentales	94
Accès au dérèglage local sur Aquila QuEra	95
Accès à de hautes géométries sur Aquila QuEra	97
Accès à des géométries serrées sur Aquila QuEra	98
Circuits dynamiques sur les appareils IQM	100
Contrôle du pouls (avancé) sur Amazon Braket	102
Frames (Images)	103
Ports	103
Formes d'onde	104
Travailler avec Hello Pulse	105
Accès aux portes natives à l'aide d'impulsions	109
Simulation hamiltonienne analogique (avancée)	111
Bonjour AHS : Exécutez votre première simulation hamiltonienne analogique	112
Soumettre un programme analogique à l'aide d' QuEra Aquila	126
(Avancé) Travailler avec AWS Boto3	145
Activez le client Amazon Braket Boto3	146
Configuration AWS CLI des profils pour Boto3 et le SDK Braket	149
test	152
Soumission de tâches quantiques à des simulateurs	152
Simulateur vectoriel d'état local (braket_sv)	154
Simulateur de matrice de densité locale (braket_dm)	154
Simulateur AHS local (braket_ahs)	155
Simulateur de vecteurs d'état (SV1)	155

Simulateur de matrice de densité (DM1)	156
Simulateur de réseau Tensor () TN1	157
À propos des simulateurs embarqués	159
Comparez les simulateurs	159
Exemples de tâches quantiques sur Amazon Braket	163
Tester une tâche quantique avec le simulateur local	169
Émulateur de dispositif quantique local	171
Avantages de l'émulation locale	171
Création d'un émulateur local	172
Exécuter	173
Soumission de tâches quantiques à QPUs	174
AQT	175
IonQ	176
IQM	177
Rigetti	177
QuEra	178
Exemple : Soumission d'une tâche quantique à un QPU	178
Inspection des circuits compilés	182
Exécution de plusieurs programmes	182
À propos de l'ensemble du programme et des coûts	184
À propos du traitement par lots de tâches quantiques et des coûts	185
Traitement par lots de tâches quantiques et PennyLane	185
Traitement par lots de tâches et circuits paramétrés	186
Quand s'exécutera ma tâche quantique ?	187
Fenêtres de disponibilité et état du QPU	187
Visibilité de la file	188
Configurer les notifications par e-mail ou par SMS	190
(Avancé) Utilisation des réservations	190
Comment créer une réservation	191
Exécution de tâches quantiques lors d'une réservation	192
Exécution de tâches hybrides lors d'une réservation	196
Que se passe-t-il à la fin de votre réservation	198
Annuler ou replanifier une réservation existante	198
Techniques (avancées) d'atténuation des erreurs	198
Techniques d'atténuation des erreurs sur IonQ les appareils	199
Offres d'emploi chez Amazon Braket Hybrid	202

Quand utiliser Amazon Braket Hybrid Jobs	203
Exécution d'une tâche hybride avec Amazon Braket Hybrid Jobs	204
Concepts clés	206
Inputs	206
Sorties	207
Variables d'environnement	208
Fonctions d'assistance	209
Conditions préalables	209
Créez un Job hybride	213
Créez et exécutez	214
Surveillez vos résultats	217
Enregistrez vos résultats	219
Utilisation des points de contrôle	221
Exécutez votre code local en tant que tâche hybride	222
Utilisation de l'API avec des tâches hybrides	231
Création et débogage d'une tâche hybride en mode local	235
Annuler un Job hybride	235
Personnalisation de votre Hybrid Job	237
Définissez l'environnement de votre script d'algorithme	237
Utilisation d'hyperparamètres	249
Configurez votre instance de tâche hybride	250
Utilisation de la compilation paramétrique pour accélérer les tâches hybrides	254
(Avancé) PennyLane avec Amazon Braket	255
Amazon Braket avec PennyLane	256
Exemples d'algorithmes hybrides dans les carnets de notes Amazon Braket	258
Algorithmes hybrides avec PennyLane simulateurs intégrés	258
Dégradé adjoint activé PennyLane avec les simulateurs Amazon Braket	259
Utilisation de tâches hybrides et exécution PennyLane d'un algorithme QAOA	260
Exécutez des charges de travail hybrides avec des simulateurs PennyLane intégrés	263
(Avancé) CUDA-Q avec Amazon Braket	269
CUDA-Q en NBIs	270
CUDA-Q dans les emplois hybrides	270
Résolution des problèmes	274
AccessDeniedException	274
Une erreur s'est produite (ValidationException) lors de l'appel de l' CreateQuantumTask opération	274

Une fonctionnalité du SDK ne fonctionne pas	275
Une tâche hybride échoue en raison de ServiceQuotaExceededException	275
Les composants ont cessé de fonctionner dans une instance de bloc-notes	276
Résolution des problèmes liés à la mise à niveau de Python 3.12	276
Présentation de	277
Messages d'erreur courants	277
Ordinateurs portables gérés par Braket	278
Job Decorator Hybrid	278
Bring-Your-Own-Container (BYOC)	279
Mise à niveau de l'instance Braket Notebook	280
Résolution des problèmes liés à OpenQASM	281
Inclure une erreur de déclaration	281
Erreur non contiguë qubits	281
Mélange d'erreur physique qubits et d'qubitserreur virtuelle	282
Erreur lors de la demande de types de résultats et de la mesure qubits dans le même programme	282
Erreur de dépassement des limites classiques et des limites de qubit registre	283
Boîte non précédée d'une erreur de pragma textuelle	283
Erreur concernant les portes natives manquantes dans les boîtes Verbatim	283
Erreur physique manquante dans les boîtes verbatim qubits	284
Il manque une erreur « braket » dans le pragma textuel	284
Une seule qubits erreur ne peut pas être indexée	284
L'erreur physique qubits dans une qubit porte à deux portes n'est pas connectée	285
Avertissement de support du simulateur local	285
Sécurité	287
Responsabilité partagée en matière de sécurité	288
Protection des données	288
Conservation des données	289
Gérer l'accès à Amazon Braket	290
Ressources Amazon Braket	290
Carnets de notes et rôles	290
AWS politiques gérées	292
Restreindre l'accès des utilisateurs à certains appareils	296
Restreindre l'accès des utilisateurs à certaines instances de blocs-notes	298
Restreindre l'accès des utilisateurs à certains compartiments S3	299
Rôle lié à un service	300

Validation de conformité	301
Sécurité de l'infrastructure	302
Sécurité par des tiers	302
Points de terminaison de VPC (PrivateLink)	303
Considérations relatives aux points de terminaison Amazon Braket VPC	303
Configurez Braket et PrivateLink	304
Informations supplémentaires sur la création d'un point de terminaison	306
Contrôlez l'accès avec les politiques relatives aux points de terminaison Amazon VPC	306
Journalisation et surveillance	308
Suivi des tâches quantiques à partir du SDK Amazon Braket	309
Surveillance des tâches quantiques via la console Amazon Braket	311
Balisage de ressources	313
Utilisation d'étiquettes	314
Ressources prises en charge pour le balisage dans Amazon Braket	315
Balisage avec l'Amazon Braket API	315
Restrictions de balisage	316
Gestion des tags dans Amazon Braket	316
Exemple de AWS CLI balisage dans Amazon Braket	317
Surveillez vos tâches quantiques avec EventBridge	318
Surveillez l'état des tâches quantiques avec EventBridge	319
Exemple d'événement Amazon Braket EventBridge	320
Surveillez vos indicateurs avec CloudWatch	321
Statistiques et dimensions d'Amazon Braket	322
Enregistrez vos tâches quantiques avec CloudTrail	322
Informations sur Amazon Braket dans CloudTrail	323
Comprendre les entrées du fichier journal Amazon Braket	324
Journalisation (avancée)	326
Quotas	329
Quotas et limites supplémentaires	371
Historique de la documentation	372
.....	ccclxxxvi

Qu'est-ce qu'Amazon Braket ?



Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Amazon Braket est une solution entièrement gérée Service AWS qui aide les chercheurs, les scientifiques et les développeurs à se lancer dans l'informatique quantique. L'informatique quantique a le potentiel de résoudre des problèmes informatiques qui sont hors de portée des ordinateurs classiques, car elle exploite les lois de la mécanique quantique pour traiter l'information de nouvelles manières.

L'accès au matériel informatique quantique peut être coûteux et peu pratique. L'accès limité complique l'exécution d'algorithmes, l'optimisation des conceptions, l'évaluation de l'état actuel de la technologie et la planification du moment où investir vos ressources pour en tirer le meilleur parti. Braket vous aide à relever ces défis.

Braket offre un point d'accès unique à une variété de technologies informatiques quantiques. Avec Braket, vous pouvez :

- Explorez et concevez des algorithmes quantiques et hybrides.
- Testez des algorithmes sur différents simulateurs de circuits quantiques.
- Exécutez des algorithmes sur différents types d'ordinateurs quantiques.
- Créez des applications de preuve de concept.

La définition des problèmes quantiques et la programmation d'ordinateurs quantiques pour les résoudre nécessitent de nouvelles compétences. Pour vous aider à acquérir ces compétences, Braket propose différents environnements pour simuler et exécuter vos algorithmes quantiques. Vous pouvez trouver l'approche la mieux adaptée à vos besoins et démarrer rapidement avec un ensemble d'exemples d'environnements appelés ordinateurs portables.

Le développement de Braket comporte trois étapes :

- [Build](#) - Braket fournit des environnements de blocs-notes Jupyter entièrement gérés qui facilitent la prise en main. Les blocs-notes Braket sont préinstallés avec des exemples d'algorithmes, de ressources et d'outils de développement, notamment le SDK Amazon Braket. Avec le SDK Amazon Braket, vous pouvez créer des algorithmes quantiques, puis les tester et les exécuter sur différents ordinateurs quantiques et simulateurs en modifiant une seule ligne de code.
- [Test](#) - Braket donne accès à des simulateurs de circuits quantiques à hautes performances entièrement gérés. Vous pouvez tester et valider vos circuits. Braket gère tous les composants logiciels sous-jacents et les clusters Amazon Elastic Compute Cloud (Amazon EC2) afin de simplifier la simulation de circuits quantiques sur une infrastructure de calcul haute performance (HPC) classique.
- [Run](#) - Braket fournit un accès sécurisé à la demande à différents types d'ordinateurs quantiques. Vous avez accès à des ordinateurs quantiques basés sur des portes depuis AQT, IonQ, et IQM Rigetti, ainsi qu'à un simulateur hamiltonien analogique de QuEra. Vous n'avez également aucun engagement initial et vous n'avez pas besoin de vous procurer un accès auprès de fournisseurs individuels.

À propos de l'informatique quantique et de Braket

L'informatique quantique n'en est qu'à ses débuts. Il est important de comprendre qu'il n'existe actuellement aucun ordinateur quantique universel tolérant aux pannes. Par conséquent, certains types de matériel quantique sont mieux adaptés à chaque cas d'utilisation et il est essentiel d'avoir accès à une variété de matériel informatique. Braket propose une variété de matériels par le biais de fournisseurs tiers.

Le matériel quantique existant est limité en raison du bruit, qui introduit des erreurs. L'industrie est entrée dans l'ère du Noisy Intermediate Scale Quantum (NISQ). À l'ère du NISQ, les appareils informatiques quantiques sont trop bruyants pour supporter des algorithmes quantiques purs, tels que l'algorithme de Shor ou l'algorithme de Grover. Jusqu'à ce qu'une meilleure correction des erreurs quantiques soit disponible, l'informatique quantique la plus pratique nécessite la combinaison de ressources informatiques classiques (traditionnelles) avec des ordinateurs quantiques pour créer des algorithmes hybrides. Braket vous aide à travailler avec des algorithmes quantiques hybrides.

Dans les algorithmes quantiques hybrides, les unités de traitement quantique (QPUs) sont utilisées comme coprocesseurs CPUs, accélérant ainsi les calculs spécifiques dans un algorithme classique. Ces algorithmes utilisent un traitement itératif, dans lequel le calcul passe d'un ordinateur classique à un ordinateur quantique. Par exemple, les applications actuelles de l'informatique quantique en chimie, en optimisation et en apprentissage automatique sont basées sur des algorithmes quantiques

variationnels, qui sont un type d'algorithme quantique hybride. Dans les algorithmes quantiques variationnels, les routines d'optimisation classiques ajustent les paramètres d'un circuit quantique paramétré de manière itérative, de la même manière que les poids d'un réseau neuronal sont ajustés de manière itérative en fonction de l'erreur d'un ensemble d'apprentissage automatique. Braket donne accès à la bibliothèque de logiciels PennyLane open source, qui vous aide à utiliser des algorithmes quantiques variationnels.

L'informatique quantique gagne du terrain pour les calculs dans quatre domaines principaux :

- Théorie des nombres, y compris la factorisation et la cryptographie (par exemple, l'algorithme de Shor est la principale méthode quantique pour les calculs de théorie des nombres)
- Optimisation, y compris la satisfaction des contraintes, la résolution de systèmes linéaires et l'apprentissage automatique
- Informatique oraculaire, y compris la recherche, les sous-groupes cachés et la recherche d'ordres (par exemple, l'algorithme de Grover est la principale méthode quantique pour les calculs oraculaires)
- Simulation : y compris la simulation directe, les invariants de nœuds et les applications d'algorithmes d'optimisation approximative quantique (QAOA)

Ces catégories de calculs peuvent être utilisées dans les secteurs des services financiers, de la biotechnologie, de la fabrication et des produits pharmaceutiques, pour n'en nommer que quelques-uns. Braket propose des fonctionnalités et des exemples de blocs-notes qui peuvent déjà être appliqués à de nombreux problèmes de validation de concept, en plus de certains problèmes pratiques.

Dans cette section :

- [Comment fonctionne Amazon Braket](#)
- [Termes et concepts d'Amazon Braket](#)
- [Suivi des coûts et économies](#)
- [Références d'API et dépôts pour Amazon Braket](#)
- [Régions et appareils pris en charge par Amazon Braket](#)

Comment fonctionne Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Amazon Braket fournit un accès à la demande à des dispositifs informatiques quantiques, notamment à des simulateurs de circuits à la demande et à différents types d'unités de traitement quantique (). QPUs Dans Amazon Braket, la demande atomique envoyée à un appareil est une tâche quantique. Pour les appareils basés sur un portail, cette demande inclut le circuit quantique (y compris les instructions de mesure et le nombre de prises de vue) et les autres métadonnées de la demande. Pour les simulateurs hamiltoniens analogiques, la tâche quantique contient la disposition physique du registre quantique et la dépendance temporelle et spatiale des champs de manipulation.

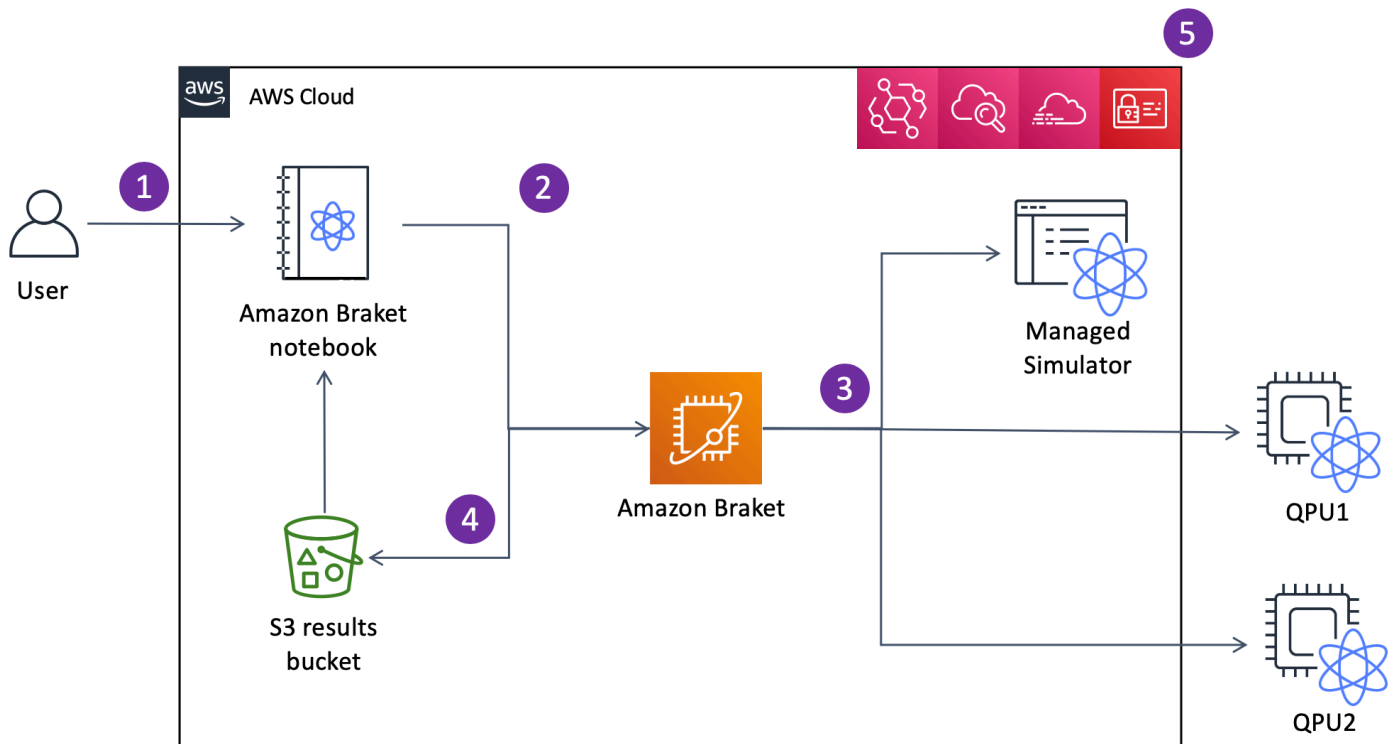
Braket Direct est un programme qui élargit la façon dont vous pouvez explorer l'informatique quantique AWS, accélérant ainsi la recherche et l'innovation. Vous pouvez réserver de la capacité dédiée sur divers appareils quantiques, contacter directement des spécialistes de l'informatique quantique et avoir un accès anticipé aux capacités de nouvelle génération, y compris le dernier dispositif à ions piégés de Forte. IonQ

Dans cette section, nous allons découvrir le flux de haut niveau lié à l'exécution de tâches quantiques sur Amazon Braket.

Dans cette section :

- [Flux de tâches quantique Amazon Braket](#)
- [Traitement des données par des tiers](#)

Flux de tâches quantique Amazon Braket



Avec Jupyter les blocs-notes, vous pouvez définir, soumettre et surveiller vos tâches quantiques depuis la console [Amazon Braket](#) ou à l'aide du SDK Amazon [Braket](#). Vous pouvez créer vos circuits quantiques directement dans le SDK. Toutefois, pour les simulateurs hamiltoniens analogiques, vous définissez la disposition des registres et les champs de contrôle (1). Une fois votre tâche quantique définie, vous pouvez choisir un appareil sur lequel l'exécuter et l'envoyer à l'API Amazon Braket (2). Selon l'appareil que vous avez choisi, la tâche quantique est mise en file d'attente jusqu'à ce que le périphérique soit disponible et la tâche est envoyée au QPU ou au simulateur pour mise en œuvre (3). Amazon Braket vous donne accès à une variété d'[appareils quantiques compatibles](#) QPUs, notamment des simulateurs à la demande, des simulateurs locaux et un simulateur intégré.

Après avoir traité votre tâche quantique, Amazon Braket renvoie les résultats dans un compartiment Amazon S3, où les données sont stockées dans votre Compte AWS (4). Dans le même temps, le SDK interroge les résultats en arrière-plan et les charge dans le bloc-notes Jupyter à la fin de la tâche quantique. Vous pouvez également consulter et gérer vos tâches quantiques sur la page Quantum Tasks de la console Amazon Braket ou en utilisant le `GetQuantumTask` fonctionnement d'Amazon Braket. API

Amazon Braket est intégré à Gestion des identités et des accès AWS (IAM), Amazon CloudWatch et AWS CloudTrail Amazon EventBridge pour la gestion, la surveillance et la journalisation des accès des utilisateurs, ainsi que pour le traitement basé sur les événements (5).

Traitement des données par des tiers

Les tâches quantiques soumises à un dispositif QPU sont traitées sur des ordinateurs quantiques situés dans des installations exploitées par des fournisseurs tiers. Pour en savoir plus sur la sécurité et le traitement par des tiers dans Amazon Braket, consultez la section [Sécurité des fournisseurs de matériel Amazon Braket](#).

Termes et concepts d'Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Les termes et concepts suivants sont utilisés dans Braket :

Simulation hamiltonienne analogique

La simulation hamiltonienne analogique (AHS) est un paradigme informatique quantique distinct pour la simulation directe de la dynamique quantique dépendante du temps de systèmes à plusieurs corps. Dans AHS, les utilisateurs spécifient directement un hamiltonien dépendant du temps et l'ordinateur quantique est réglé de telle sorte qu'il imite directement l'évolution continue du temps sous cet hamiltonien. Les dispositifs AHS sont généralement des appareils à usage spécial et non des ordinateurs quantiques universels tels que les appareils basés sur un portail. Ils sont limités à une classe de Hamiltoniens qu'ils peuvent simuler. Cependant, étant donné que ces hamiltoniens sont naturellement implémentés sur le dispositif, AHS ne souffre pas de la surcharge requise pour formuler des algorithmes sous forme de circuits et implémenter des opérations de porte.

Support

Nous avons baptisé le service Braket d'après la notation [bra-ket, une notation](#) standard en mécanique quantique. Elle a été introduite par Paul Dirac en 1939 pour décrire l'état des systèmes quantiques, et elle est également connue sous le nom de notation de Dirac.

Braket Direct

Avec Braket Direct, vous pouvez réserver un accès dédié aux différents appareils quantiques de votre choix, entrer en contact avec des spécialistes de l'informatique quantique pour obtenir des conseils sur votre charge de travail et accéder rapidement aux capacités de nouvelle génération, telles que les nouveaux dispositifs quantiques dont la disponibilité est limitée.

Tâche hybride chez Braket

Amazon Braket dispose d'une fonctionnalité appelée Amazon Braket Hybrid Jobs qui fournit des exécutions entièrement gérées d'algorithmes hybrides. Un job hybride Braket comprend trois éléments :

1. La définition de votre algorithme, qui peut être fournie sous forme de script, de module Python ou de conteneur Docker.
2. L'instance de tâche hybride, basée sur Amazon EC2, sur laquelle exécuter votre algorithme. La valeur par défaut est une instance ml.m5.xlarge.
3. L'appareil quantique sur lequel exécuter les tâches quantiques qui font partie de votre algorithme. Une seule tâche hybride contient généralement un ensemble de nombreuses tâches quantiques.

Appareil

Dans Amazon Braket, un appareil est un backend capable d'exécuter des tâches quantiques. Un appareil peut être un QPU ou un simulateur de circuit quantique. Pour en savoir plus, consultez les [appareils compatibles avec Amazon Braket](#).

Réduction des erreurs

L'atténuation des erreurs implique l'exécution de plusieurs circuits physiques et la combinaison de leurs mesures pour obtenir un meilleur résultat. Pour plus d'informations, consultez la section [Techniques d'atténuation des erreurs](#).

Informatique quantique basée sur un portail

Dans l'informatique quantique basée sur des portes (QC), également appelée QC basée sur des circuits, les calculs sont décomposés en opérations élémentaires (portes). Certains ensembles

de portes sont universels, ce qui signifie que chaque calcul peut être exprimé sous la forme d'une séquence finie de ces portes. Les portes sont les éléments constitutifs des circuits quantiques et sont analogues aux portes logiques des circuits numériques classiques.

Limite de Gateshot

Une limite de tirs fait référence au nombre total de portes par tir (somme de tous les types de portes) et au nombre de tirs par tâche. Mathématiquement, la limite de prise de vue peut être exprimée comme suit :

$$\text{Gateshot limit} = (\text{Gate count per shot}) * (\text{Shot count per task})$$

hamiltonien

La dynamique quantique d'un système physique est déterminée par son hamiltonien, qui code toutes les informations relatives aux interactions entre les constituants du système et aux effets des forces motrices exogènes. L'hamiltonien d'un système à N qubits est généralement représenté sous la forme d'une matrice 2^N sur 2^N de nombres complexes sur les machines classiques. En exécutant une simulation hamiltonienne analogique sur un appareil quantique, vous pouvez éviter ces besoins exponentiels en ressources.

Pouls

Une impulsion est un signal physique transitoire transmis aux qubits. Il est décrit par une forme d'onde jouée dans une trame qui sert de support au signal porteur et est liée au canal ou au port matériel. Les clients peuvent concevoir leurs propres impulsions en fournissant l'enveloppe analogique qui module le signal porteur sinusoïdal à haute fréquence. Le cadre est décrit de manière unique par une fréquence et une phase qui sont souvent choisies pour être en résonance avec la séparation d'énergie entre les niveaux d'énergie pour $|0\rangle$ et $|1\rangle$ du qubit. Les portes sont ainsi créées sous forme d'impulsions ayant une forme prédéterminée et des paramètres calibrés tels que leur amplitude, leur fréquence et leur durée. Les cas d'utilisation qui ne sont pas couverts par les modèles de formes d'onde seront activés par le biais de formes d'onde personnalisées qui seront spécifiées à la résolution d'un échantillon unique en fournissant une liste de valeurs séparées par un temps de cycle physique fixe.

Circuit quantique

Un circuit quantique est le jeu d'instructions qui définit un calcul sur un ordinateur quantique basé sur un portail. Un circuit quantique est une séquence de portes quantiques, qui sont des transformations réversibles sur un qubit registre, associées à des instructions de mesure.

Simulateur de circuits quantiques

Un simulateur de circuit quantique est un programme informatique qui fonctionne sur des ordinateurs classiques et calcule les résultats de mesure d'un circuit quantique. Pour les circuits généraux, les besoins en ressources d'une simulation quantique augmentent de façon exponentielle avec le nombre de circuits qubits à simuler. Braket donne accès à des simulateurs de circuits quantiques gérés (accessibles via le BraketAPI) et locaux (inclus dans le SDK Amazon Braket).

Ordinateur quantique

Un ordinateur quantique est un appareil physique qui utilise des phénomènes de mécanique quantique, tels que la superposition et l'intrication, pour effectuer des calculs. Il existe différents paradigmes en informatique quantique (QC), tels que le QC basé sur des portes.

Unité de traitement quantique (QPU)

Un QPU est un dispositif informatique quantique physique qui peut fonctionner sur une tâche quantique. QPU peut être basé sur différents paradigmes de contrôle qualité, tels que le contrôle qualité basé sur des portes. Pour en savoir plus, consultez les [appareils compatibles avec Amazon Braket](#).

Portails natifs QPU

Les portes natives QPU peuvent être directement mappées pour contrôler les impulsions par le système de contrôle QPU. Les portes natives peuvent être exécutées sur le périphérique QPU sans autre compilation. Sous-ensemble de portes prises en charge par le QPU. Vous trouverez les portes natives d'un appareil sur la page Appareils de la console Amazon Braket et via le SDK Braket.

Portails supportés par QPU

Les portes compatibles QPU sont les portes acceptées par l'appareil QPU. Ces portes peuvent ne pas fonctionner directement sur le QPU, ce qui signifie qu'il peut être nécessaire de les décomposer en portes natives. Vous trouverez les portes compatibles d'un appareil sur la page Appareils de la console Amazon Braket et via le SDK Amazon Braket.

Tâche quantique

Dans Braket, une tâche quantique est la demande atomique adressée à un appareil. Pour les dispositifs de contrôle qualité basés sur une porte, cela inclut le circuit quantique (y compris les instructions de mesure et le nombre deshots) et les autres métadonnées de demande. Vous pouvez créer des tâches quantiques via le SDK Amazon Braket ou en utilisant directement

l'opération `CreateQuantumTaskAPI`. Une fois que vous avez créé une tâche quantique, elle est mise en file d'attente jusqu'à ce que le périphérique demandé soit disponible. Vous pouvez consulter vos tâches quantiques sur la page Quantum Tasks de la console Amazon Braket ou en utilisant les opérations `GetQuantumTask` or `SearchQuantumTasksAPI`.

Qubit

L'unité d'information de base d'un ordinateur quantique s'appelle un qubit (bit quantique), un peu comme un bit dans l'informatique classique. A qubit est un système quantique à deux niveaux qui peut être réalisé par différentes implémentations physiques, telles que des circuits supraconducteurs ou des ions et des atomes individuels. D'autres qubit types sont basés sur des photons, des spins électroniques ou nucléaires, ou des systèmes quantiques plus exotiques.

Queue depth

Queue depth fait référence au nombre de tâches quantiques et de tâches hybrides mises en file d'attente pour un appareil donné. Les tâches quantiques et le nombre de files d'attente de tâches hybrides d'un appareil sont accessibles via le Braket Software Development Kit (SDK) ou la console Amazon Braket Management Console.

1. La profondeur de la file d'attente des tâches fait référence au nombre total de tâches quantiques en attente d'exécution en priorité normale.
2. La profondeur de la file d'attente des tâches prioritaires fait référence au nombre total de tâches quantiques soumises en attente d'exécution Amazon Braket Hybrid Jobs. Ces tâches ont la priorité sur les tâches autonomes une fois qu'une tâche hybride démarre.
3. La profondeur de la file d'attente des tâches hybrides fait référence au nombre total de tâches hybrides actuellement en file d'attente sur un appareil. Les tâches quantiques soumises dans le cadre d'une tâche hybride sont prioritaires et sont agrégées dans la Priority Task Queue.

Queue position

Queue position fait référence à la position actuelle de votre tâche quantique ou de votre tâche hybride dans une file d'attente d'appareils correspondante. Il peut être obtenu pour des tâches quantiques ou des tâches hybrides via le Braket Software Development Kit (SDK) ou la console Amazon Braket Management Console.

Shots

L'informatique quantique étant intrinsèquement probabiliste, tout circuit doit être évalué plusieurs fois pour obtenir un résultat précis. L'exécution et la mesure d'un circuit unique s'appellent un tir. Le nombre de tirs (exécutions répétées) pour un circuit est choisi en fonction de la précision souhaitée pour le résultat.

AWS terminologie et astuces pour Amazon Braket

Politiques IAM

Une politique IAM est un document qui autorise ou refuse les autorisations Services AWS et les ressources. Les politiques IAM vous permettent de personnaliser les niveaux d'accès des utilisateurs aux ressources. Par exemple, vous pouvez autoriser les utilisateurs à accéder à tous les compartiments Amazon S3 de votre Compte AWS compartiment ou uniquement à un compartiment spécifique.

- Bonne pratique : Respectez le principe de sécurité du moindre privilège lorsque vous accordez des autorisations. En suivant ce principe, vous contribuez à empêcher les utilisateurs ou les rôles de disposer de plus d'autorisations que nécessaire pour effectuer leurs tâches quantiques. Par exemple, si un employé n'a besoin d'accéder qu'à un compartiment spécifique, spécifiez le compartiment dans la politique IAM au lieu de lui accorder l'accès à tous les compartiments de votre. Compte AWS

Rôles IAM

Un rôle IAM est une identité que vous pouvez assumer pour obtenir un accès temporaire aux autorisations. Avant qu'un utilisateur, une application ou un service puisse assumer un rôle IAM, il doit être autorisé à passer à ce rôle. Lorsqu'une personne assume un rôle IAM, elle abandonne toutes les autorisations qu'elle détenait dans le cadre d'un rôle précédent et assume les autorisations du nouveau rôle.

- Bonne pratique : les rôles IAM sont idéaux pour les situations dans lesquelles l'accès aux services ou aux ressources doit être accordé de manière temporaire plutôt qu'à long terme.

Compartiment Amazon S3

Amazon Simple Storage Service (Amazon S3) vous permet de stocker des données sous forme Service AWS d'objets dans des compartiments. Les compartiments Amazon S3 offrent un espace de stockage illimité. La taille maximale d'un objet dans un compartiment Amazon S3 est de 5 To. Vous pouvez télécharger tout type de données de fichier dans un compartiment Amazon S3, telles que des images, des vidéos, des fichiers texte, des fichiers de sauvegarde, des fichiers multimédia pour un site Web, des documents archivés et les résultats de vos tâches quantiques Braket.

- Bonne pratique : vous pouvez définir des autorisations pour contrôler l'accès à votre compartiment S3. Pour plus d'informations, consultez les [politiques relatives](#) aux compartiments dans la documentation Amazon S3.

Suivi des coûts et économies

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Avec Amazon Braket, vous avez accès à des ressources informatiques quantiques à la demande sans engagement initial. Vous ne payez que ce que vous utilisez. Pour en savoir plus sur les tarifs, consultez notre [page de tarification](#).

Dans cette section :

- [Définition des limites de dépenses pour Amazon Braket QPUs](#)
- [Suivi des coûts en temps quasi réel](#)
- [Les meilleures pratiques pour réduire les coûts](#)

Définition des limites de dépenses pour Amazon Braket QPUs

Les limites de dépenses Amazon Braket fournissent un contrôle facultatif des coûts par appareil pour les unités de traitement quantique (). QPUs

Comment fonctionnent les limites de dépenses : Amazon Braket suit vos dépenses cumulées et valide chaque demande de création de tâche par rapport à la limite que vous avez configurée. Si le coût estimé d'une tâche dépasse votre limite de dépenses restante, Amazon Braket rejette immédiatement la tâche avec une erreur de validation. Vous pouvez éventuellement configurer une période pour votre limite de dépenses. En configurant une période, vous pouvez vous assurer que les tâches ne peuvent être soumises que pendant cette période spécifiée. Les tâches soumises en dehors de cette période seront rejetées.

Conception optionnelle : les flux de travail existants ne seront pas affectés à moins que vous n'activiez explicitement les contrôles. Vous pouvez supprimer toutes les restrictions en supprimant la limite de dépenses.

 Note

Les limites de dépenses s'appliquent uniquement aux [tâches QPU](#) à la demande et hybrides. Ils excluent les [simulateurs](#), les [ordinateurs portables gérés](#), les coûts des instances [Hybrid Job](#) EC2 et les réservations [Braket Direct](#). Pour une gestion complète des coûts de tous les services AWS, continuez à utiliser [AWS Budgets](#).

Liste des mesures de limitation des dépenses

Rechercher

À l'aide de la commande AWS CLI suivante, vous pouvez rechercher et répertorier les limites de dépenses dans une région AWS spécifique et pour un appareil Braket spécifique.

```
aws --region {device_region} braket search-spending-limits --filters
name=deviceArn,operator=EQUAL,values={device_arn}
```

Créez

À l'aide de la commande AWS CLI suivante, vous pouvez créer une nouvelle limite de dépenses pour un dispositif quantique spécifique dans une région spécifique. La demande est rejetée s'il existe déjà une limite de dépenses pour l'appareil.

```
aws --region {device_region} braket create-spending-limit --device-arn {device_arn}
--spending-limit {max_spend}
```

Mettre à jour

À l'aide de la commande AWS CLI suivante, vous pouvez mettre à jour une limite de dépenses existante vers une nouvelle valeur de dépense maximale. La demande est rejetée si la somme des dépenses actuelles et des dépenses en attente est déjà supérieure à la nouvelle dépense maximale demandée.

```
aws --region {device_region} braket update-spending-limit --spending-limit-arn
{spending_limit_arn} --spending-limit {new_max_spend}
```

Vous pouvez indiquer une période à la place ou en plus de la nouvelle dépense maximale, comme dans l'exemple ci-dessus.

Supprimer

À l'aide de la commande AWS CLI suivante, vous pouvez supprimer une limite de dépenses existante.

```
aws --region {device_region} braket delete-spending-limit --spending-limit-arn  
{spending_limit_arn}
```

Vous pouvez indiquer une période à la place ou en plus de la nouvelle dépense maximale, comme dans l'exemple ci-dessus.

Bien que facultatif, il est recommandé de toujours spécifier le paramètre de région. Les commandes exécutées dans une région différente de celle de l'appareil échoueront ou, dans le cas contraire `SearchSpendingLimits`, renverront des résultats incorrects.

Pour plus d'exemples sur l'utilisation des limites de dépenses, consultez l'[exemple de bloc-notes](#).

Comment fonctionne la validation des tâches

Lorsque le compte AWS envoie une `CreateQuantumTask` demande par ailleurs valide, il est soumis au comportement de blocage suivant. Remarque : Le budget restant correspond à la différence entre la limite de dépenses et la somme des dépenses en attente et des dépenses en cours. (Voir la section suivante)

- Cas 1 : Il n'y a aucune limite de dépenses pour le dispositif de tâches : la tâche est créée.
- Cas 2 : Il existe une limite de dépenses pour l'appareil cible, et l'heure actuelle se situe dans la période de limite de dépenses :
 - Si le coût estimé de la tâche est inférieur ou égal au budget restant : si le résultat `CreateQuantumTask` est atteint, la tâche est créée.
 - Si le coût estimé est supérieur au budget restant : `CreateQuantumTask` échec et aucune tâche n'est créée.
- Cas 3 : Il existe une limite de dépenses pour l'appareil cible, et l'heure actuelle est en dehors de la période de limite de dépenses : `CreateQuantumTask` échec et aucune tâche n'est créée.

Comment le budget restant est calculé

Le budget restant correspond à la différence entre la limite de dépenses et la somme des dépenses en cours et des dépenses en attente.

Lorsqu'une tâche est créée pour un appareil soumis à une limite de dépenses, les dépenses en file d'attente sont augmentées du coût estimé de la tâche. Cet événement est répertorié dans la première ligne du tableau suivant. Le tableau suivant indique ce qu'il advient des dépenses en file d'attente et des dépenses actuelles, en fonction de la progression de la tâche.

Ancien état d'une tâche quantique	Nouvel état de tâche quantique	Passage aux dépenses en file d'attente	Modification des dépenses actuelles
-	CRÉÉ	Augmenté par le coût estimé	Pas de modification
CRÉÉ	QUEUED	Pas de modification	Pas de modification
N'importe lequel	RUNNING	Pas de modification	Pas de modification
N'importe lequel	ANNULATION	Pas de modification	Pas de modification
ANNULATION	CANCELLED	Réduit par le coût estimé	Pas de changement
N'importe lequel	ÉCHEC	Réduit par le coût estimé	Pas de modification
RUNNING	TERMINÉ	Réduit par le coût estimé	Augmenté du coût estimatif (ajusté en conséquence pour les tâches partiellement achevées)

Étuis Edge

Q : Lors de la création d'une limite de dépenses, les tâches déjà présentes dans la file d'attente sont-elles prises en compte dans les dépenses en file d'attente ?

R : Non. Les tâches déjà créées, mises en file d'attente ou en cours ne sont pas prises en compte dans les dépenses en file d'attente d'une limite de dépenses nouvellement créée.

Q : L'abaissement de la limite de dépenses, en la mettant à jour, entraîne-t-il l'arrêt anticipé d'une tâche quantique créée, en file d'attente ou en cours de réalisation ?

R : Non.

Q : Le fait d'atteindre l'heure de fin de la limite de dépenses entraîne-t-il l'arrêt anticipé d'une tâche quantique créée, en file d'attente ou en cours de réalisation ?

R : Non. Les tâches créées, mises en file d'attente ou en cours peuvent être effectuées indépendamment de l'état de la limite de dépenses.

Q : En quoi l'absence de limite de dépenses diffère-t-elle d'une limite de dépenses de zéro dollar ?

R : Aucune limite de dépenses ne permet de créer des tâches quantiques sans restrictions. Une limite de dépenses de zéro dollar bloque toutes les tâches quantiques.

Q : Une limite de dépenses nulle dans le passé ou dans le futur bloque-t-elle toute création de tâches quantiques ?

A : Oui.

Q : Lors de la création d'une limite de dépenses, le coût estimé des tâches déjà en attente sera-t-il pris en compte dans les dépenses actuelles une fois ces tâches terminées ?

R : Non. Seules les tâches soumises alors qu'une limite de dépenses est active sont prises en compte dans les dépenses cumulées.

Suivi des coûts en temps quasi réel

Le SDK Braket vous offre la possibilité d'ajouter un suivi des coûts en temps quasi réel à vos charges de travail quantiques. Chacun de nos carnets d'exemples inclut un code de suivi des coûts pour vous fournir une estimation maximale des coûts des unités de traitement quantique (QPUs) et des simulateurs à la demande de Braket. Les estimations des coûts maximaux seront affichées en dollars américains et n'incluent aucun crédit ni aucune réduction.

Note

Les frais indiqués sont des estimations basées sur l'utilisation des tâches de votre simulateur Amazon Braket et de votre unité de traitement quantique (QPU). Les frais estimés indiqués peuvent différer de vos frais réels. Les frais estimés ne tiennent pas compte des remises

ou des crédits et des frais supplémentaires peuvent vous être facturés en fonction de votre utilisation d'autres services tels qu'Amazon Elastic Compute Cloud (Amazon EC2).

Suivi des coûts pour SV1

Afin de démontrer comment la fonction de suivi des coûts peut être utilisée, nous allons construire un circuit Bell State et l'exécuter sur notre SV1 simulateur. Commencez par importer les modules du SDK Braket, définissez un Bell State et ajoutez la `Tracker()` fonction à notre circuit :

```
#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)
```

```
Counter({'00': 500, '11': 500})
```

Lorsque vous utilisez votre ordinateur portable, vous pouvez vous attendre au résultat suivant pour votre simulation de Bell State. La fonction de suivi vous indiquera le nombre de tirs envoyés, les tâches quantiques terminées, la durée d'exécution, la durée d'exécution facturée et votre coût maximum en dollars américains. Le temps d'exécution peut varier pour chaque simulation.

```
import datetime

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()
```

```
Decimal('0.0037500000')
```

Utiliser le suivi des coûts pour fixer les coûts maximaux

Vous pouvez utiliser le suivi des coûts pour définir les coûts maximaux d'un programme. Vous pouvez avoir un seuil maximum pour le montant que vous souhaitez dépenser pour un programme donné. De cette façon, vous pouvez utiliser le suivi des coûts pour élaborer une logique de contrôle des coûts dans votre code d'exécution. L'exemple suivant prend le même circuit sur un Rigetti QPU et limite le coût à 1 USD. Le coût d'exécution d'une itération du circuit dans notre code est de 0,30 USD. Nous avons défini la logique pour répéter les itérations jusqu'à ce que le coût total dépasse 1 USD ; par conséquent, l'extrait de code sera exécuté trois fois jusqu'à ce que la prochaine itération dépasse 1 USD. En général, un programme continue à itérer jusqu'à ce qu'il atteigne le coût maximum souhaité, dans ce cas, trois itérations.

```
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-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/Ankaa-3': {'shots': 600, 'tasks':
{'COMPLETED': 3}}}
```

1.4400000000 USD

Note

Le suivi des coûts ne suivra pas la durée des tâches TN1 quantiques ayant échoué. Au cours d'une TN1 simulation, si votre répétition se termine, mais que l'étape de contraction échoue, vos frais de répétition ne seront pas affichés dans le suivi des coûts.

Les meilleures pratiques pour réduire les coûts

Tenez compte des bonnes pratiques suivantes pour utiliser Amazon Braket. Gagnez du temps, minimisez les coûts et évitez les erreurs courantes.

Vérifiez à l'aide de simulateurs

- Vérifiez vos circuits à l'aide d'un simulateur avant de l'exécuter sur un QPU, afin de pouvoir affiner votre circuit sans encourir de frais pour l'utilisation du QPU.
- Bien que les résultats de l'exécution du circuit sur un simulateur ne soient pas identiques à ceux de l'exécution du circuit sur un QPU, vous pouvez identifier les erreurs de codage ou les problèmes de configuration à l'aide d'un simulateur.

Restreindre l'accès des utilisateurs à certains appareils

- Vous pouvez définir des restrictions qui empêchent les utilisateurs non autorisés de soumettre des tâches quantiques sur certains appareils. La méthode recommandée pour restreindre l'accès est d'utiliser AWS IAM. Pour plus d'informations sur la procédure à suivre, consultez [Restreindre l'accès](#).
- Nous vous recommandons de ne pas utiliser votre compte administrateur pour accorder ou restreindre l'accès des utilisateurs aux appareils Amazon Braket.

Définissez des alarmes de facturation

- Vous pouvez définir une alarme de facturation pour vous avertir lorsque votre facture atteint une limite prédéfinie. La méthode recommandée pour configurer une alarme est la méthode suivante AWS Budgets. Vous pouvez définir des budgets personnalisés et recevoir des alertes lorsque vos coûts ou votre utilisation peuvent dépasser le montant budgétisé. Les informations sont disponibles à l'adresse [AWS Budgets](#).

Testez des tâches TN1 quantiques avec un faible nombre de tirs

- Les simulateurs coûtent moins cher QPUs, mais certains peuvent être coûteux si les tâches quantiques sont exécutées avec un nombre de coups élevé. Nous vous recommandons de tester vos TN1 tâches avec un faible shot nombre de tâches. Shotle nombre n'a aucune incidence sur le coût SV1 et les tâches du simulateur local.

Vérifiez toutes les régions pour les tâches quantiques

- La console affiche les tâches quantiques uniquement pour votre activité actuelle Région AWS. Lorsque vous recherchez des tâches quantiques facturables qui ont été soumises, assurez-vous de cocher toutes les régions.
- Vous pouvez consulter la liste des appareils et de leurs régions associées sur la page de documentation [des appareils pris en charge](#).

Références d'API et dépôts pour Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Amazon Braket fournit APIs SDKs, ainsi qu'une interface de ligne de commande que vous pouvez utiliser pour créer et gérer des instances de blocs-notes, ainsi que pour former et déployer des modèles.

- [SDK Amazon Braket Python \(recommandé\)](#)
- [Référence de l'API Amazon Braket](#)
- [AWS Command Line Interface](#)
- [AWS SDK pour .NET](#)
- [AWS SDK pour C++](#)
- [AWS SDK pour GoAPI Reference](#)
- [AWS SDK pour Java](#)
- [AWS SDK pour JavaScript](#)
- [AWS SDK pour PHP](#)
- [AWS SDK pour Python \(Boto\)](#)
- [AWS SDK pour Ruby](#)

Vous pouvez également obtenir des exemples de code dans le référentiel Amazon Braket Tutorials GitHub .

- [Tutoriels Braket GitHub](#)

Référentiels principaux

Voici une liste des référentiels principaux contenant les principaux packages utilisés pour Braket :

- [SDK Python Braket - Utilisez le SDK](#) Python Braket pour configurer votre code Jupyter sur des ordinateurs portables dans le langage de programmation Python. Une fois vos Jupyter ordinateurs portables configurés, vous pouvez exécuter votre code sur les appareils et simulateurs Braket
- [Braket Schemas](#) - Le contrat entre le SDK Braket et le service Braket.
- [Simulateur par défaut de Braket](#) - Tous nos simulateurs quantiques locaux pour Braket (vecteur d'état et matrice de densité).

Plug-ins

Ensuite, il y a les différents plugins utilisés avec divers appareils et outils de programmation. Il s'agit notamment des plugins pris en charge par Braket ainsi que des plugins pris en charge par des tiers, comme indiqué ci-dessous.

Amazon Braket a pris en charge :

- [Bibliothèque d'algorithmes Amazon Braket](#) : catalogue d'algorithmes quantiques prédéfinis écrits en Python. Exécutez-les tels quels ou utilisez-les comme point de départ pour créer des algorithmes plus complexes.
- [PennyLane Plugin Braket](#) - À utiliser PennyLane comme framework QML sur Braket.

Tierce partie (l'équipe Braket surveille et contribue) :

- [Fournisseur Qiskit-Braket](#) : utilisez le Qiskit SDK pour accéder aux ressources de Braket.
- [SDK Braket-Julia](#) - (EXPÉRIMENTAL) Une version native Julia du SDK Braket

Régions et appareils pris en charge par Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Dans Amazon Braket, un appareil représente une unité de traitement quantique (QPU) ou un simulateur que vous pouvez appeler pour exécuter des tâches quantiques. Amazon Braket permet d'accéder aux appareils QPU depuis AQT, IonQIQM, QuEra et Rigetti. En outre, AWS offre un accès à des simulateurs locaux, intégrés et à la demande. Pour plus d'informations sur les simulateurs intégrés, voir [À propos des simulateurs intégrés](#).

Pour plus d'informations sur les fournisseurs de matériel quantique pris en charge, voir [Soumission de tâches quantiques à QPUs](#). Pour plus d'informations sur les simulateurs disponibles, voir [Soumission de tâches quantiques aux simulateurs](#). Le tableau suivant affiche la liste des appareils et simulateurs disponibles.

Fournisseur	Nom du périphérique	Paradigm	Type	ARN de l'appareil	Région
AQT	IBEX-Q1	basé sur un portail	QPU	arn:aws:braket:eu-north-1::-Q1 device/qpu/aqt/lbex	eu-north-1
IonQ	Aria-1	basé sur un portail	QPU	arn:aws:braket:us-east-1::-1 device/qpu/ionq/Aria	us-east-1
IonQ	Forte-1	basé sur un portail	QPU	arn:aws:braket:us-east-1::-1 device/qpu/ionq/Forte	us-east-1
IonQ	Forte-Enterprise-1	basé sur un portail	QPU	arn:aws:braket:us-east-1::-Enterprise-1 device/qpu/ionq/Forte	us-east-1

Fournisseur	Nom du périphérique	Paradigm	Type	ARN de l'appareil	Région
IQM	Garnet	basé sur un portail	QPU	arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet	eu-north-1
IQM	Emerald	basé sur un portail	QPU	arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald	eu-north-1
QuEra	Aquila	Simulation hamiltonienne analogique	QPU	arn:aws:braket:us-east-1::device/qpu/quera/Aquila	us-east-1
Rigetti	Ankaa-3	basé sur un portail	QPU	arn:aws:braket:us-west-1:-3device/qpu/rigetti/Ankaa	us-west-1
AWS	braket_sv	basé sur un portail	Simulateur local	N/A (simulateur local dans le SDK Braket)	N/A
AWS	Braket_DM	basé sur un portail	Simulateur local	N/A (simulateur local dans le SDK Braket)	N/A
AWS	braket_ahs	Simulation hamiltonienne analogique	Simulateur local	N/A (simulateur local dans le SDK Braket)	N/A

Fournisseur	Nom du périphérique	Paradigm	Type	ARN de l'appareil	Région
AWS	SV1	basé sur un portail	Simulateur à la demande	arn:aws:braket::1:device/quantum-simulator/amazon/sv	us-east-1, us-west-1, us-west-2, eu-west-2
AWS	DM1	basé sur un portail	Simulateur à la demande	arn:aws:braket::1:device/quantum-simulator/amazon/dm	us-east-1, us-west-1, us-west-2, eu-west-2
AWS	TN1	basé sur un portail	Simulateur à la demande	arn:aws:braket::1:device/quantum-simulator/amazon/tn	us-east-1, us-west-2 et eu-west-2

Note

ARNs Les appareils distinguent les majuscules et minuscules. Par exemple, lorsque vous utilisez l'AQT IBEX-Q1 appareil, vérifiez que son ARN contient 'Ibex-Q1'.

Pour obtenir des informations supplémentaires sur les ordinateurs QPUs que vous pouvez utiliser avec Amazon Braket, consultez Amazon Braket [Quantum Computers](#).

Propriétés de l'appareil

Pour tous les appareils, vous pouvez trouver d'autres propriétés de l'appareil, telles que la topologie de l'appareil, les données d'étalonnage et les ensembles de portes natifs, dans l'onglet Appareils de la console Amazon Braket ou via l'GetDeviceAPI. Lorsque vous construisez un circuit avec les

simulateurs, Amazon Braket exige que vous utilisiez des qubits ou des indices contigus. Lorsque vous utilisez le SDK, l'exemple de code suivant montre comment accéder aux propriétés de chaque périphérique et simulateur disponibles.

```
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:eu-north-1::device/qpu/aqt/Ibex-Q1')
# AQT IBEX-Q1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1')
# IonQ Aria-1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1')
# IonQ Forte-1
# device = AwsDevice('arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1')
# IonQ Forte-Enterprise-1
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet')
# IQM Garnet
# device = AwsDevice('arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald')
# IQM Emerald
# 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/Ankaa-3')
# Rigetti Ankaa-3

# Get device properties
```

```
device.properties
```

Régions et points de terminaison pour Amazon Braket

Pour une liste complète des régions et des points de terminaison, consultez la [référence AWS générale](#).

Les tâches quantiques exécutées sur un appareil QPU peuvent être consultées dans la console Amazon Braket de la région de cet appareil. Lorsque vous utilisez le SDK Amazon Braket, vous pouvez soumettre des tâches quantiques à n'importe quel appareil QPU, quelle que soit la région dans laquelle vous travaillez. Le SDK crée automatiquement une session dans la région pour le QPU spécifié.

Amazon Braket est disponible dans les versions suivantes : Régions AWS

Nom de la région	Région	Points de terminaison du support
USA Est (Virginie du Nord)	us-east-1	braket.us-east-1.amazonaws.com (IPv4 uniquement) braket.us-east-1.api.aws (double pile)
USA Ouest (Californie du Nord)	us-west-1	braket.us-west-1.amazonaws.com (IPv4 uniquement) braket.us-west-1.api.aws (double pile)
USA West 2 (Oregon)	us-west-2	braket.us-west-2.amazonaws.com (IPv4 uniquement) braket.us-west-2.api.aws (double pile)
UE North 1 (Stockholm)	eu-north-1	braket.eu-north-1.amazonaws.com (IPv4 uniquement)

Nom de la région	Région	Points de terminaison du support
		braket.eu-north-1.api.aws (double pile)
EU West 2 (Londres)	eu-west-2	braket.eu-west-2.amazonaws.com (IPv4 uniquement) braket.eu-west-2.api.aws (double pile)

 Note

Le SDK Amazon Braket ne prend pas en charge IPv6 les réseaux uniquement.

Commencer à utiliser Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Après avoir suivi les instructions de la section [Activer Amazon Braket](#), vous pouvez commencer à utiliser Amazon Braket.

Les étapes pour commencer sont les suivantes :

- [Activer Amazon Braket](#)
- [Création d'une instance de bloc-notes Amazon Braket](#)
- [Créez une instance de bloc-notes Braket à l'aide de CloudFormation](#)

Activer Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

[Vous pouvez activer Amazon Braket dans votre compte via la AWS console.](#)

Dans cette section :

- [Conditions préalables](#)
- [Étapes pour activer Amazon Braket](#)

Conditions préalables

Pour activer et exécuter Amazon Braket, vous devez disposer d'un utilisateur ou d'un rôle autorisé à lancer des actions Amazon Braket. Ces autorisations sont incluses dans la politique IAM (AmazonBraketFullAccess arn:aws:iam : :aws:policy/). AmazonBraketFullAccess

Note

Si vous êtes administrateur :

Pour permettre à d'autres utilisateurs d'accéder à Amazon Braket, accordez-leur des autorisations en joignant la AmazonBraketFullAccesspolitique ou en joignant une politique personnalisée que vous créez. Pour en savoir plus sur les autorisations nécessaires pour utiliser Amazon Braket, consultez [Gérer l'accès à Amazon Braket](#).

Étapes pour activer Amazon Braket

1. Connectez-vous à la [console Amazon Braket avec votre](#) Compte AWS
2. Ouvrez la console Amazon Braket.
3. Sur la page d'accueil de Braket, cliquez sur Commencer pour accéder à la page du tableau de bord du service. L'alerte en haut de votre tableau de bord de service vous guidera à travers les trois étapes suivantes :
 - a. Création de [rôles liés à un service \(SLR\)](#)
 - b. Permettre l'accès à des ordinateurs quantiques tiers
 - c. Création d'une nouvelle instance de bloc-notes Jupyter

Pour utiliser des appareils quantiques tiers, vous devez accepter certaines conditions concernant le transfert de données entre vous et ces appareils. AWS Les termes et conditions de cet accord sont fournis dans l'onglet Général de la page Autorisations et paramètres de la console Amazon Braket.

Note

Les appareils Quantum qui n'impliquent aucun tiers, tels que les simulateurs locaux Braket ou les simulateurs à la demande, peuvent être utilisés sans accepter le contrat Enable Third Devices.

L'acceptation de ces conditions pour permettre l'utilisation d'appareils tiers ne doit être effectuée qu'une seule fois par compte si vous accédez à du matériel tiers.

Création d'une instance de bloc-notes Amazon Braket

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Amazon Braket fournit des blocs-notes Jupyter entièrement gérés pour vous aider à démarrer. Les instances de bloc-notes Amazon Braket sont basées sur les instances de [bloc-notes Amazon SageMaker AI](#). Les étapes suivantes expliquent comment créer une nouvelle instance de bloc-notes pour les clients nouveaux et existants.

Nouveaux clients d'Amazon Braket :

1. Ouvrez la [console Amazon Braket](#) et accédez à la page du tableau de bord dans le volet de gauche.
2. Cliquez sur Commencer dans le modal Bienvenue sur Amazon Braket au centre de la page de votre tableau de bord. Entrez un nom de bloc-notes pour créer un bloc-notes Jupyter par défaut.
 - a. La création de votre bloc-notes peut prendre plusieurs minutes.
 - b. Votre bloc-notes sera répertorié sur la page Carnets avec le statut En attente.
 - c. Lorsque votre instance de bloc-notes est prête à être utilisée, son statut passe à InService.
 - d. Actualisez la page pour afficher l'état actualisé du bloc-notes.

Clients actuels d'Amazon Braket :

1. Ouvrez la [console Amazon Braket](#) et sélectionnez Notebooks dans le volet de gauche.
2. Sélectionnez Créer une instance de bloc-notes.
 - a. Si vous n'avez aucun bloc-notes, sélectionnez la configuration standard pour créer un bloc-notes Jupyter par défaut.

3. Entrez le nom d'une instance de Notebook, en utilisant uniquement des caractères alphanumériques et des tirets, et sélectionnez le mode visuel de votre choix.
4. Activez ou désactivez le gestionnaire d'inactivité du bloc-notes pour votre bloc-notes.
 - a. Si cette option est activée, sélectionnez la durée d'inactivité souhaitée avant la réinitialisation de l'ordinateur portable. Lorsqu'un ordinateur portable est réinitialisé, les frais de calcul cessent d'être facturés, mais les frais de stockage se poursuivent.
 - b. Pour vérifier le temps d'inactivité restant pour votre instance de bloc-notes, accédez à la barre de commandes, sélectionnez l'onglet Braket, puis l'onglet Inactivity Manager.

 Note

Pour enregistrer votre travail, intégrez votre [instance de bloc-notes SageMaker AI à un référentiel Git](#). Vous pouvez également déplacer votre travail en dehors des /Braket Examples dossiers /Braket Algorithms et afin qu'il ne soit pas remplacé par le redémarrage de l'instance du bloc-notes.

5. (Facultatif) Avec la configuration avancée, vous pouvez créer un bloc-notes avec des autorisations d'accès, des configurations supplémentaires et des paramètres d'accès au réseau :
 - a. Dans Configuration du bloc-notes, choisissez votre type d'instance.
 - i. Le type d'instance standard et économique, ml.t3.medium, est choisi par défaut. Pour en savoir plus sur la tarification des instances, consultez la [tarification d'Amazon SageMaker AI](#).
 - b. Pour associer un dépôt Github public à votre instance de bloc-notes, cliquez sur le menu déroulant du dépôt Git et sélectionnez Cloner un dépôt public git depuis une URL dans le menu déroulant du référentiel. Entrez l'URL du dépôt dans la barre de texte de l'URL du dépôt Git.
 - c. Dans Autorisations d'accès, configurez les rôles IAM facultatifs, l'accès root et les clés de chiffrement.
 - d. Dans Accès réseau, configurez des paramètres réseau et d'accès personnalisés pour votre Jupyter Notebook instance.
6. Vérifiez vos paramètres et définissez des balises pour identifier votre instance de bloc-notes. Cliquez sur Lancer.

Note

Consultez et gérez vos instances de bloc-notes Amazon Braket dans les consoles Amazon Braket et SageMaker Amazon AI. [Les paramètres supplémentaires du bloc-notes Amazon Braket sont disponibles via la SageMaker console.](#)

Si vous travaillez dans la console Amazon Braket dans le SDK Amazon Braket et AWS que les plugins sont préchargés dans les blocs-notes que vous avez créés. Pour l'exécuter sur votre propre machine, installez le SDK et les plug-ins lorsque vous exécutez la commande `pip install amazon-braket-sdk` ou lorsque vous exécutez la commande `pip install amazon-braket-pennylane-plugin` pour les PennyLane plug-ins.

Créez une instance de bloc-notes Braket à l'aide de CloudFormation

Tip

Découvrez les bases de l'informatique quantique avec AWS ! Inscrivez-vous au plan d'[apprentissage numérique Amazon Braket](#) et obtenez votre propre badge numérique après avoir suivi une série de cours de formation et une évaluation numérique.

Vous pouvez l'utiliser CloudFormation pour gérer vos instances de bloc-notes Amazon Braket. Les instances de bloc-notes Braket sont basées sur Amazon SageMaker AI. Avec CloudFormation, vous pouvez approvisionner une instance de bloc-notes avec un fichier modèle qui décrit la configuration prévue. Le fichier modèle est écrit au format JSON ou YAML. Vous pouvez créer, mettre à jour et supprimer des instances de manière ordonnée et reproductible. Cela peut vous être utile lorsque vous gérez plusieurs instances de bloc-notes Braket dans votre Compte AWS.

Après avoir créé un CloudFormation modèle pour un bloc-notes Braket, vous pouvez l'utiliser CloudFormation pour déployer la ressource. Pour plus d'informations, consultez la section [Création d'une pile sur la CloudFormation console](#) dans le guide de CloudFormation l'utilisateur.

Pour créer une instance de bloc-notes Braket à l'aide de CloudFormation, vous devez effectuer les trois étapes suivantes :

1. Créez un script de configuration du cycle de vie de l' SageMaker IA.

2. Créez un rôle Gestion des identités et des accès AWS (IAM) à assumer par l' SageMaker IA.
3. Créez une instance de bloc-notes SageMaker AI avec le préfixe **amazon-braket-**

Vous pouvez réutiliser la configuration du cycle de vie pour tous les blocs-notes Braket que vous créez. Vous pouvez également réutiliser le rôle IAM pour les blocs-notes Braket auxquels vous attribuez les mêmes autorisations d'exécution.

Dans cette section :

- [Étape 1 : Création d'un script de configuration du cycle de vie de l' SageMaker IA](#)
- [Étape 2 : créer le rôle IAM assumé par Amazon SageMaker AI](#)
- [Étape 3 : créer une instance de bloc-notes SageMaker AI avec le préfixe amazon-braket-](#)

Étape 1 : Création d'un script de configuration du cycle de vie de l' SageMaker IA

Utilisez le modèle suivant pour créer un [script de configuration du cycle de vie de l'SageMaker IA](#). Le script personnalise une instance de bloc-notes SageMaker AI pour Braket. Pour les options de configuration de la CloudFormation ressource de cycle de vie, reportez-vous [AWS::SageMaker::NotebookInstanceLifecycleConfig](#) au guide de CloudFormation l'utilisateur.

```
BraketNotebookInstanceLifecycleConfig:
  Type: "AWS::SageMaker::NotebookInstanceLifecycleConfig"
  Properties:
    NotebookInstanceLifecycleConfigName: BraketLifecycleConfig-${AWS::StackName}
    OnStart:
      - Content:
          Fn::Base64: |
            #!/usr/bin/env bash
            sudo -u ec2-user -i #EOS
            curl -o braket-notebook-lcc.zip https://d3ded4lzb1lnme.cloudfront.net/
notebook/braket-notebook-lcc.zip
            unzip braket-notebook-lcc.zip
            ./install.sh
            EOS

            exit 0
```

Étape 2 : créer le rôle IAM assumé par Amazon SageMaker AI

Lorsque vous utilisez une instance de bloc-notes Braket, l' SageMaker IA effectue des opérations en votre nom. Supposons, par exemple, que vous utilisiez un bloc-notes Braket à l'aide d'un circuit sur un appareil compatible. Dans l'instance de bloc-notes, l' SageMaker IA exécute l'opération sur Braket pour vous. Le rôle d'exécution du bloc-notes définit les opérations exactes que l' SageMaker IA est autorisée à exécuter en votre nom. Pour plus d'informations, consultez SageMaker la section [Rôles](#) de l' SageMaker IA dans le guide du développeur Amazon AI.

Utilisez l'exemple suivant pour créer un rôle d'exécution de bloc-notes Braket avec les autorisations requises. Vous pouvez modifier les politiques en fonction de vos besoins.

Note

Assurez-vous que le rôle dispose d'une autorisation pour les `s3:GetObject` opérations `s3:ListBucket` et sur les compartiments Amazon S3 préfixées par `braketnotebookcdk-` Le script de configuration du cycle de vie nécessite ces autorisations pour copier le script d'installation du bloc-notes Braket.

```
ExecutionRole:
  Type: "AWS::IAM::Role"
  Properties:
    RoleName: !Sub AmazonBraketNotebookRole-${AWS::StackName}
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        -
          Effect: "Allow"
          Principal:
            Service:
              - "sagemaker.amazonaws.com"
          Action:
            - "sts:AssumeRole"
      Path: "/service-role/"
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/AmazonBraketFullAccess
    Policies:
      -
        PolicyName: "AmazonBraketNotebookPolicy"
        PolicyDocument:
```

```

Version: "2012-10-17"
Statement:
  - Effect: Allow
    Action:
      - s3:GetObject
      - s3:PutObject
      - s3:ListBucket
    Resource:
      - arn:aws:s3:::amazon-braket-*
      - arn:aws:s3:::braketnotebookcdk-*
  - Effect: "Allow"
    Action:
      - "logs:CreateLogStream"
      - "logs:PutLogEvents"
      - "logs:CreateLogGroup"
      - "logs:DescribeLogStreams"
    Resource:
      - !Sub "arn:aws:logs:*:${AWS::AccountId}:log-group:/aws/sagemaker/*"
  - Effect: "Allow"
    Action:
      - braket:*
    Resource: "*"

```

Étape 3 : créer une instance de bloc-notes SageMaker AI avec le préfixe **amazon-braket-**

Utilisez le script de cycle de vie SageMaker AI et le rôle IAM créés aux étapes 1 et 2 pour créer une instance de bloc-notes SageMaker AI. L'instance de bloc-notes est personnalisée pour Braket et est accessible via la console Amazon Braket. Pour plus d'informations sur les options de configuration de cette CloudFormation ressource, consultez [AWS::SageMaker::NotebookInstance](#) le guide de CloudFormation l'utilisateur.

```

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

```

Créez vos tâches quantiques avec Amazon Braket

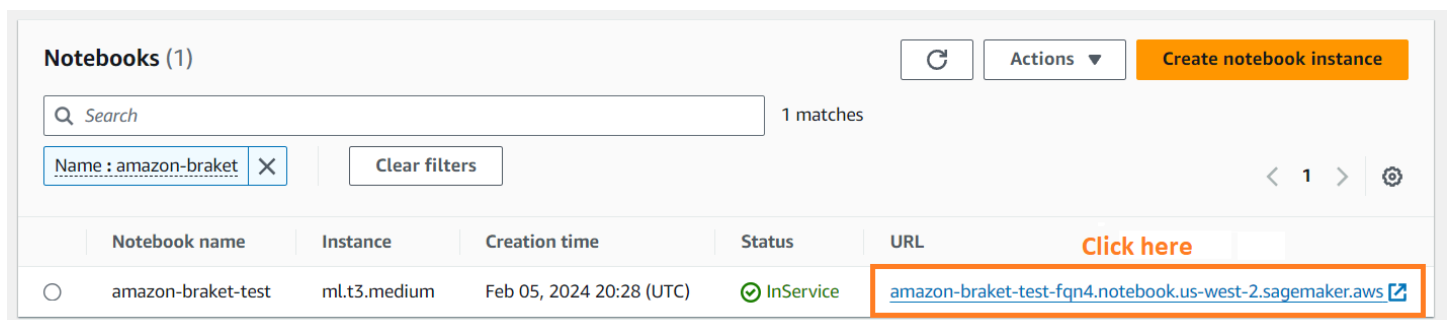
Braket fournit des environnements de blocs-notes Jupyter entièrement gérés qui facilitent la prise en main. Les blocs-notes Braket sont préinstallés avec des exemples d'algorithmes, de ressources et d'outils de développement, notamment le SDK Amazon Braket. Avec le SDK Amazon Braket, vous pouvez créer des algorithmes quantiques, puis les tester et les exécuter sur différents ordinateurs quantiques et simulateurs en modifiant une seule ligne de code.

Dans cette section :

- [Construisez votre premier circuit](#)
- [Obtenir des conseils d'experts](#)
- [Exécutez vos circuits avec OpenQASM 3.0](#)
- [Explorez les capacités expérimentales](#)
- [Contrôle du pouls sur Amazon Braket](#)
- [Simulation hamiltonienne analogique](#)
- [Travailler avec AWS Boto3](#)

Construisez votre premier circuit

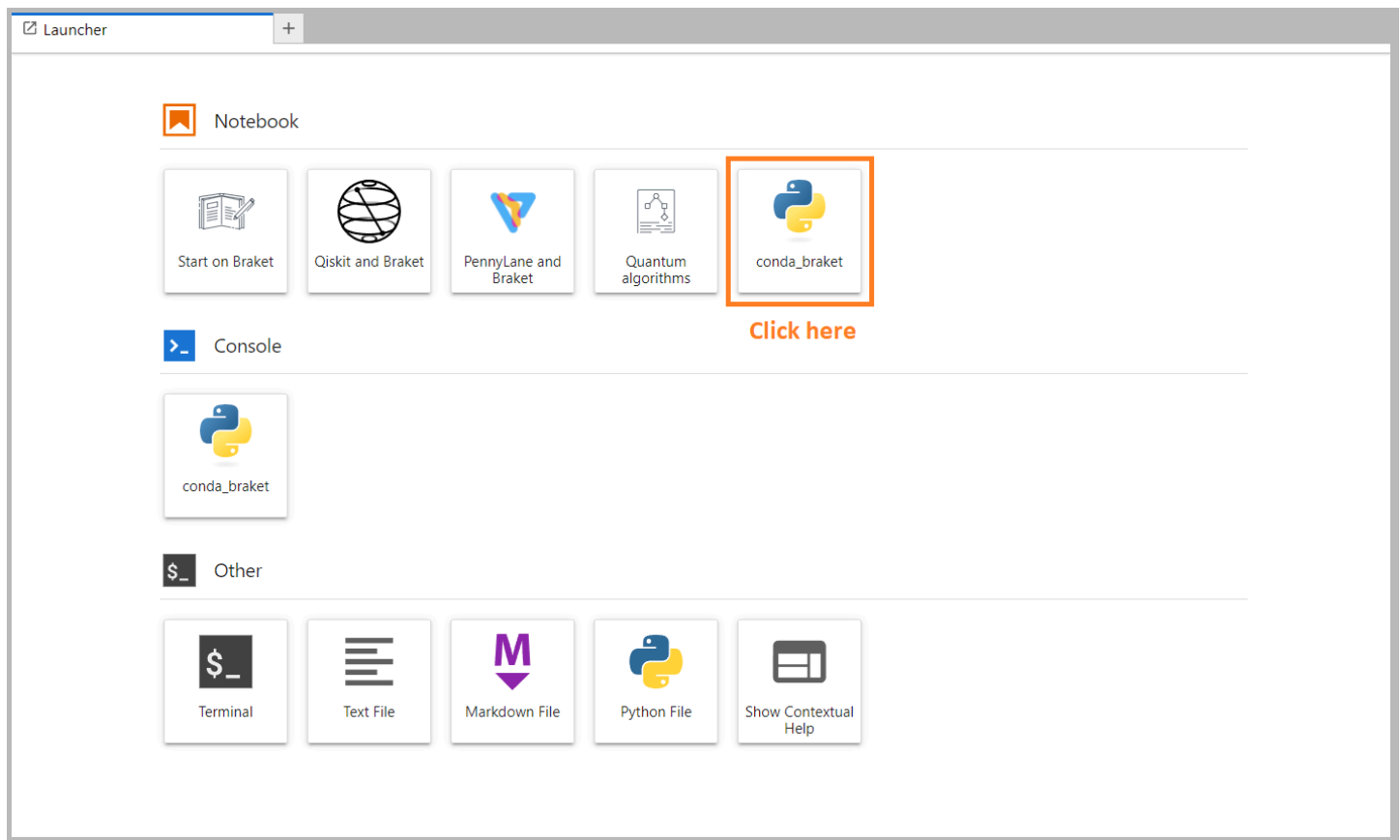
Une fois votre instance de bloc-notes lancée, ouvrez-la avec une interface Jupyter standard en choisissant le bloc-notes que vous venez de créer.



The screenshot shows the Amazon Braket console interface. At the top, there's a 'Notebooks (1)' section with a search bar and a 'Create notebook instance' button. Below the search bar, a filter is applied: 'Name : amazon-braket'. A table lists the notebook instances. The first instance, 'amazon-braket-test', is highlighted. Its URL, 'amazon-braket-test-fqn4.notebook.us-west-2.sagemaker.aws', is highlighted with an orange box. A 'Click here' link is also visible next to the URL.

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

Les instances du bloc-notes Amazon Braket sont préinstallées avec le SDK Amazon Braket et toutes ses dépendances. Commencez par créer un nouveau bloc-notes avec `conda_braket` le noyau.



Vous pouvez commencer par un simple « Bonjour tout le monde ! » exemple. Construisez d'abord un circuit qui prépare un état de Bell, puis exécutez ce circuit sur différents appareils pour obtenir les résultats.

Commencez par importer les modules du SDK Amazon Braket, en définissant un module SDK simple `BRACKETlong` et en définissant un circuit Bell State de base.

```
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)
```

Vous pouvez visualiser le circuit avec cette commande :

```
print(bell)
```

```

T : # 0 # 1 #
    #####
q0 : ## H #####
    ##### #
        #####
q1 : ##### X ##
    #####
T : # 0 # 1 #

```

Exécutez votre circuit sur le simulateur local

Ensuite, choisissez le dispositif quantique sur lequel exécuter le circuit. Le SDK Amazon Braket est fourni avec un simulateur local pour le prototypage et les tests rapides. Nous vous recommandons d'utiliser le simulateur local pour les circuits plus petits, qui peuvent atteindre 25 qubits (selon votre matériel local).

Pour instancier le simulateur local :

```

# Instantiate the local simulator
local_sim = LocalSimulator()

```

et lancez le circuit :

```

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

```

Vous devriez voir un résultat semblable à celui-ci :

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

L'état de Bell spécifique que vous avez préparé est une superposition égale de $|00\rangle$ et $|11\rangle$, et une distribution à peu près égale (jusqu'au shot bruit) de 00 et 11 en tant que résultats de mesure, comme prévu.

Exécutez votre circuit sur un simulateur à la demande

Amazon Braket donne également accès à un simulateur haute performance à la demande SV1, pour exécuter de plus grands circuits. SV1 est un simulateur vectoriel d'état à la demande qui permet de simuler des circuits quantiques allant jusqu'à 34 qubits. Vous trouverez plus d'informations SV1

dans la section [Appareils pris en charge](#) et dans la AWS console. Lorsque vous exécutez des tâches quantiques sur SV1 (et sur TN1 ou sur n'importe quel QPU), les résultats de votre tâche quantique sont stockés dans un compartiment S3 de votre compte. Si vous ne spécifiez aucun compartiment, le SDK Braket crée un compartiment par défaut `amazon-braket-{region}-{accountID}` pour vous. Pour en savoir plus, consultez [Gérer l'accès à Amazon Braket](#).

Note

Entrez le nom de votre bucket actuel, là où l'exemple suivant l'indique `amazon-braket-s3-demo-bucket` comme nom de compartiment. Les noms des compartiments pour Amazon Braket commencent `amazon-braket-` toujours par les autres caractères d'identification que vous ajoutez. Si vous avez besoin d'informations sur la configuration d'un compartiment S3, consultez [Getting started with Amazon S3](#).

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

# The name of the bucket
my_bucket = "amazon-braket-s3-demo-bucket"

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

Pour exécuter un circuitSV1, vous devez fournir l'emplacement du compartiment S3 que vous avez précédemment sélectionné comme argument positionnel dans l'. `run()` appel.

```
# 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)
```

La console Amazon Braket fournit des informations supplémentaires sur votre tâche quantique. Accédez à l'onglet Quantum Tasks de la console et votre tâche quantique devrait figurer en haut de

la liste. Vous pouvez également rechercher votre tâche quantique à l'aide de l'identifiant unique de la tâche quantique ou d'autres critères.

Note

Au bout de 90 jours, Amazon Braket supprime automatiquement toutes les tâches quantiques IDs et autres métadonnées associées à vos tâches quantiques. Pour plus d'informations, consultez la section [Conservation des données](#).

Exécution sur un QPU

Avec Amazon Braket, vous pouvez exécuter l'exemple de circuit quantique précédent sur un ordinateur quantique physique en modifiant simplement une seule ligne de code. Amazon Braket donne accès à une variété d'unités de traitement quantique (QPU). Vous trouverez des informations sur les différents appareils et les fenêtres de disponibilité dans la section [Appareils pris en charge](#) et dans la AWS console sous l'onglet Appareils. L'exemple suivant montre comment instancier un IQM appareil.

```
# Choose the IQM hardware to run your circuit
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")
```

Ou choisissez un IonQ appareil avec ce code :

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

Après avoir sélectionné un appareil et avant d'exécuter votre charge de travail, vous pouvez demander la profondeur de la file d'attente des appareils à l'aide du code suivant pour déterminer le nombre de tâches quantiques ou de tâches hybrides. En outre, les clients peuvent consulter la profondeur des files d'attente spécifiques aux appareils sur la page Appareils du 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)
# Returns the number of hybrid jobs queued on the device
# '2'
```

Lorsque vous exécutez votre tâche, le SDK Amazon Braket interroge pour obtenir un résultat (avec un délai d'expiration par défaut de 5 jours). Vous pouvez modifier cette valeur par défaut en modifiant le `poll_timeout_seconds` paramètre dans la `.run()` commande, comme indiqué dans l'exemple suivant. N'oubliez pas que si votre délai d'interrogation est trop court, les résultats risquent de ne pas être renvoyés dans le délai imparti, par exemple lorsqu'un QPU n'est pas disponible et qu'une erreur de temporisation locale est renvoyée. Vous pouvez relancer le sondage en appelant la `task.result()` fonction.

```
# 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)
```

De plus, après avoir soumis votre tâche quantique ou votre tâche hybride, vous pouvez appeler la `queue_position()` fonction pour vérifier votre position dans la file d'attente.

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

Création de vos premiers algorithmes quantiques

La bibliothèque d'algorithmes Amazon Braket est un catalogue d'algorithmes quantiques prédéfinis écrits en Python. Exécutez ces algorithmes tels quels ou utilisez-les comme points de départ pour créer des algorithmes plus complexes. Vous pouvez accéder à la bibliothèque d'algorithmes depuis la console Braket. Pour plus d'informations, consultez la bibliothèque d'[algorithmes Braket Github](#).

Amazon Braket X

Dashboard
Devices
Notebooks
Hybrid Jobs
Quantum Tasks

Algorithm library

Announcements 1
Permissions and settings

Amazon Braket > Algorithm library

Algorithm library

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

Algorithms (11) Open notebook ▼

Q Filter algorithms

Bernstein Vazirani algorithm [GitHub](#)

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

Tags: [Textbook](#)

Deutsch-Jozsa algorithm [GitHub](#)

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

Tags: [Textbook](#)

Grover's algorithm [GitHub](#)

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

Quantum Approximate Optimization Algorithm [GitHub](#)

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

La console Braket fournit une description de chaque algorithme disponible dans la bibliothèque d'algorithmes. Cliquez GitHub sur un lien pour voir les détails de chaque algorithme, ou choisissez Ouvrir un bloc-notes pour ouvrir ou créer un bloc-notes contenant tous les algorithmes disponibles. Si vous choisissez l'option bloc-notes, vous pouvez alors trouver la bibliothèque d'algorithmes Braket dans le dossier racine de votre bloc-notes.

Construire des circuits dans le SDK

Cette section fournit des exemples de définition d'un circuit, de visualisation des portes disponibles, d'extension d'un circuit et de visualisation des portes prises en charge par chaque appareil. Il contient également des instructions sur la façon d'allouer manuellement qubits, de demander au compilateur d'exécuter vos circuits exactement tels que définis et de créer des circuits bruyants à l'aide d'un simulateur de bruit.

Vous pouvez également travailler au niveau du pouls dans Braket pour différents portails avec certains QPUs. Pour plus d'informations, consultez [Pulse Control sur Amazon Braket](#).

Dans cette section :

- [Portails et circuits](#)
- [Ensembles de programmes](#)
- [Mesure partielle](#)
- [qubitAllocation manuelle](#)

- [Compilation textuelle](#)
- [Simulation du bruit](#)

Portails et circuits

Les portes et circuits quantiques sont définis dans la [braket.circuits](#) classe du SDK Amazon Braket Python. À partir du SDK, vous pouvez instancier un nouvel objet de circuit en appelant `Circuit()`

Exemple : définition d'un circuit

L'exemple commence par définir un exemple de circuit de quatre qubits (étiquetés `q0`, `q1`, `q2`, et `q3`) composé de portes Hadamard standard à un qubit et de portes CNOT à deux qubits. Vous pouvez visualiser ce circuit en appelant la `print` fonction, comme le montre l'exemple suivant.

```
# 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 #####
    ##### #
    ##### #
q1 : ## H #####
    ##### # #
    ##### ##### #
q2 : ## H ### X #####
    ##### ##### #
    ##### #####
q3 : ## H ##### X ##
    ##### #####
T : # 0 # 1 #
```

Exemple : définition d'un circuit paramétré

Dans cet exemple, nous définissons un circuit dont les portes dépendent de paramètres libres. Nous pouvons spécifier les valeurs de ces paramètres pour créer un nouveau circuit ou, lors de la soumission du circuit, pour qu'il soit exécuté en tant que tâche quantique sur certains appareils.

```
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)
```

Vous pouvez créer un nouveau circuit non paramétré à partir d'un circuit paramétré en fournissant soit un seul float (qui est la valeur que prendront tous les paramètres libres), soit des arguments par mot-clé spécifiant la valeur de chaque paramètre au circuit comme suit.

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

Notez qu'il `my_circuit` n'est pas modifié, vous pouvez donc l'utiliser pour instancier de nombreux nouveaux circuits avec des valeurs de paramètres fixes.

Exemple : modifier les portes d'un circuit

L'exemple suivant définit un circuit avec des portes qui utilisent des modificateurs de commande et de puissance. Vous pouvez utiliser ces modifications pour créer de nouvelles portes, telles que la Ry porte contrôlée.

```
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)
```

Les modificateurs de porte ne sont pris en charge que sur le simulateur local.

Exemple : Voir toutes les portes disponibles

L'exemple suivant montre comment examiner toutes les portes disponibles dans Amazon Braket.

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

La sortie de ce code répertorie toutes les portes.

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

N'importe laquelle de ces portes peut être ajoutée à un circuit en appelant la méthode correspondant à ce type de circuit. Par exemple, appelez `circ.h(0)`, pour ajouter une porte Hadamard à la première. qubit

Note

Les portes sont ajoutées en place, et l'exemple suivant ajoute toutes les portes répertoriées dans l'exemple précédent au même circuit.

```
circ = Circuit()
# toffoli gate with q0, q1 the control qubits and q2 the target.
circ.ccnnot(0, 1, 2)
# cnot gate
circ.cnot(0, 1)
# controlled-phase gate that phases the |11> state, cphaseshift(phi) =
  diag((1,1,1,exp(1j*phi))), where phi=0.15 in the examples below
circ.cphaseshift(0, 1, 0.15)
# controlled-phase gate that phases the |00> state, cphaseshift00(phi) =
  diag([exp(1j*phi),1,1,1])
circ.cphaseshift00(0, 1, 0.15)
```

```

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

```

```

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

```

Outre le jeu de portes prédéfini, vous pouvez également appliquer des portes unitaires auto-définies au circuit. Il peut s'agir de portes à un seul qubit (comme indiqué dans le code source suivant) ou de portes à plusieurs qubits appliquées à la valeur qubits définie par le paramètre. `targets`

```

import numpy as np

# Apply a general unitary
my_unitary = np.array([[0, 1],[1, 0]])
circ.unitary(matrix=my_unitary, targets=[0])

```

Exemple : étendre les circuits existants

Vous pouvez étendre les circuits existants en ajoutant des instructions. Une `Instruction` est une directive quantique qui décrit la tâche quantique à effectuer sur un dispositif quantique. Les opérateurs incluent `Gate` uniquement les objets de type.

```

# 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, 6])
new_instr = instr.copy(target_mapping={0: 5, 1: 6})

```

Exemple : Afficher les barrières prises en charge par chaque appareil

Les simulateurs prennent en charge toutes les portes du SDK Braket, mais les appareils QPU en prennent en charge un sous-ensemble plus restreint. Vous pouvez trouver les portes compatibles d'un appareil dans les propriétés de l'appareil. Voici un exemple avec un appareil IonQ :

```

# Import the device module
from braket.aws import AwsDevice

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

# 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 Aria 1:


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

Les portes prises en charge devront peut-être être compilées dans des portes natives avant de pouvoir fonctionner sur du matériel quantique. Lorsque vous soumettez un circuit, Amazon Braket effectue cette compilation automatiquement.

Exemple : récupérer par programmation la fidélité des portes natives prises en charge par un appareil

Vous pouvez consulter les informations de fidélité sur la page Appareils de la console Braket. Il est parfois utile d'accéder aux mêmes informations par le biais d'un programme. Le code suivant montre comment extraire la fidélité à deux qubit portes entre deux portes d'un QPU.

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

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

# Specify the qubits
a=10
b=11
edge_properties_entry =
    device.properties.standardized.twoQubitProperties['10-11'].twoQubitGateFidelity
gate_name = edge_properties_entry[0].gateName
fidelity = edge_properties_entry[0].fidelity
print(f"Fidelity of the {gate_name} gate between qubits {a} and {b}: {fidelity}")
```

Ensembles de programmes

Les ensembles de programmes exécutent efficacement plusieurs circuits quantiques en une seule tâche quantique. Dans le cadre de cette tâche, vous pouvez soumettre jusqu'à 100 circuits quantiques ou un seul circuit paramétrique avec jusqu'à 100 ensembles de paramètres différents. Cette opération minimise le temps entre les exécutions de circuits suivantes et réduit la surcharge de traitement des tâches quantiques. Actuellement, les ensembles de programmes sont pris en charge sur Amazon Braket Local Simulator et sur AQT les appareils IQM. Rigetti

Définition d'un ProgramSet

Le premier exemple de code suivant montre comment créer un ProgramSet en utilisant à la fois des circuits paramétrés et des circuits sans paramètres.

```

from braket.aws import AwsDevice
from braket.circuits import Circuit, FreeParameter
from braket.program_sets.circuit_binding import CircuitBinding
from braket.program_sets import ProgramSet

# Initialize the quantum device
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")

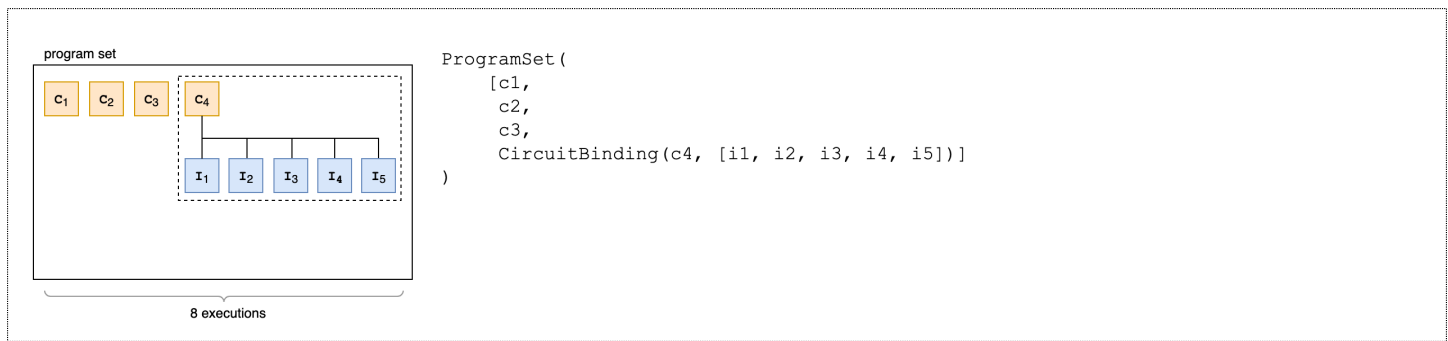
# Define circuits
circ1 = Circuit().h(0).cnot(0, 1)
circ2 = Circuit().rx(0, 0.785).ry(1, 0.393).cnot(1, 0)
circ3 = Circuit().t(0).t(1).cz(0, 1).s(0).cz(1, 2).s(1).s(2)
parameterize_circuit = Circuit().rx(0, FreeParameter("alpha")).cnot(0, 1).ry(1,
    FreeParameter("beta"))

# Create circuit bindings with different parameters
circuit_binding = CircuitBinding(
    circuit=parameterize_circuit,
    input_sets={
        'alpha': (0.10, 0.11, 0.22, 0.34, 0.45),
        'beta': (1.01, 1.01, 1.03, 1.04, 1.04),
    })

# Creating the program set
program_set_1 = ProgramSet([
    circ1,
    circ2,
    circ3,
    circuit_binding,
])

```

Cet ensemble de programmes contient quatre programmes uniques : `circ1`, `circ2`, `circ3`, `circuit_binding`. Le `circuit_binding` programme s'exécute avec cinq liaisons de paramètres différentes, créant ainsi cinq exécutables. Les trois autres programmes sans paramètres créent chacun un exécutable. Cela donne huit exécutables au total, comme le montre l'image suivante.



Le deuxième exemple de code suivant montre comment utiliser la `product()` méthode pour associer le même ensemble d'observables à chaque exécutable de l'ensemble de programmes.

```

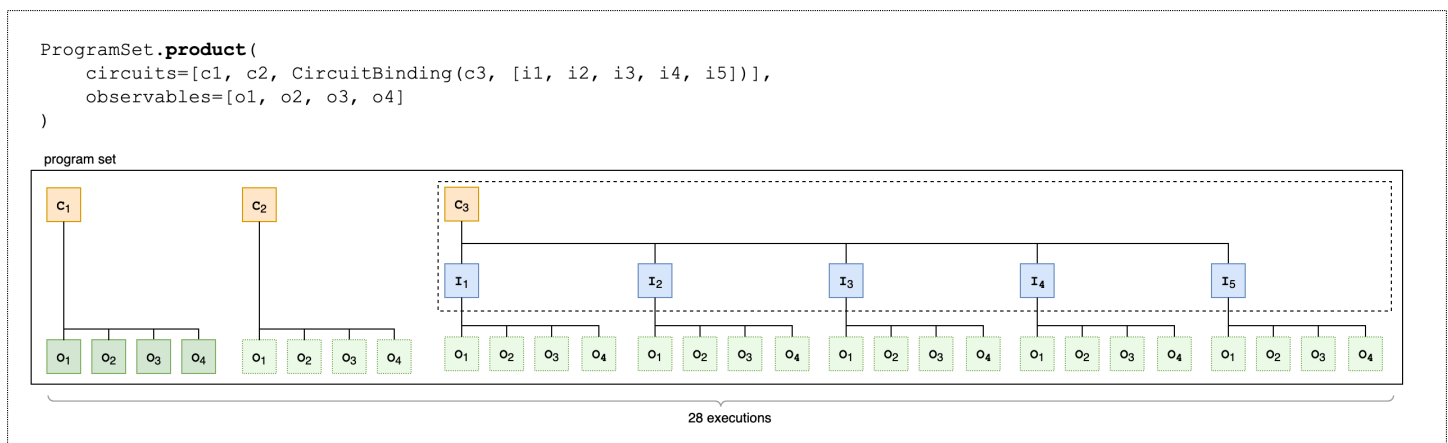
from braket.circuits.observables import I, X, Y, Z

observables = [Z(0) @ Z(1), X(0) @ X(1), Z(0) @ X(1), X(0) @ Z(1)]

program_set_2 = ProgramSet.product(
    circuits=[circ1, circ2, circuit_binding],
    observables=observables
)

```

Pour les programmes sans paramètres, chaque observable est mesuré pour chaque circuit. Pour les programmes paramétriques, chaque observable est mesuré pour chaque ensemble d'entrées, comme indiqué dans l'image suivante.



Le troisième exemple de code suivant montre comment utiliser la `zip()` méthode pour associer des observables individuels à des ensembles de paramètres spécifiques dans le `ProgramSet`.

```

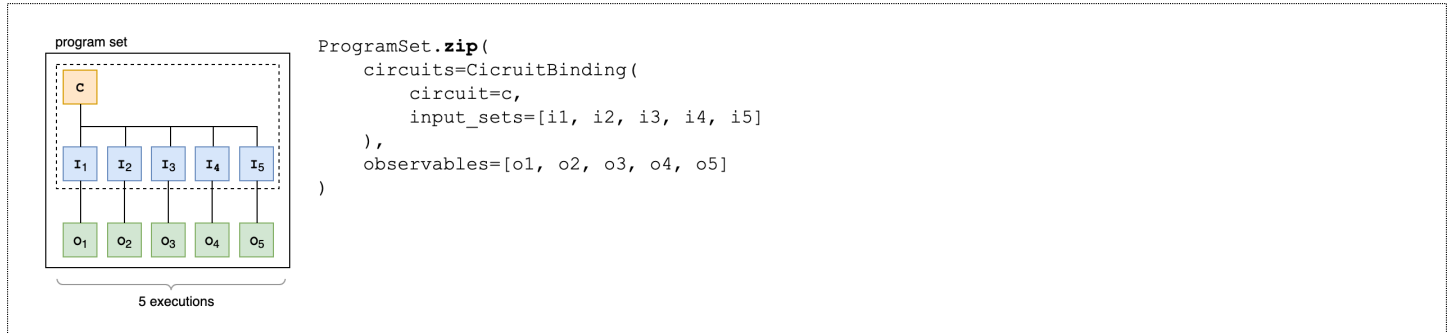
program_set_3 = ProgramSet.zip(
    circuits=circuit_binding,

```

```

observables=observables + [Y(0) @ Y(1)]
)

```

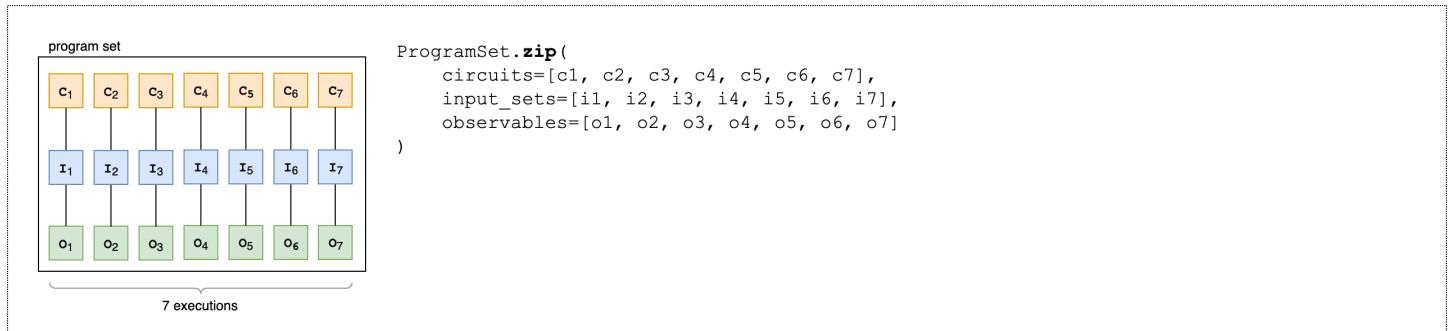


Au lieu de `circuitBinding()`, vous pouvez directement compresser une liste d'observables avec une liste de circuits et de jeux d'entrées.

```

program_set_4 = ProgramSet.zip(
    circuits=[c1, c2, c3],
    input_sets=[[], {}, {}],
    observables=observables[:3]
)

```



Pour plus d'informations et des exemples sur les ensembles de programmes, consultez le [dossier Program set](#) sur [amazon-braket-examples Github](#).

Inspecter et exécuter un programme défini sur un appareil

Le nombre d'exécutables d'un ensemble de programmes est égal à son nombre de circuits uniques liés à des paramètres. Calculez le nombre total d'exécutables et de plans de circuit à l'aide de l'exemple de code suivant.

```
# Number of shots per executable
```

```
shots = 10
num_executables = program_set_1.total_executables

# Calculate total number of shots across all executables
total_num_shots = shots*num_executables
```

Note

Avec les ensembles de programmes, vous payez des frais uniques par tâche et des frais par tir basés sur le nombre total de plans sur tous les circuits d'un ensemble de programmes.

Pour exécuter l'ensemble de programmes, utilisez l'exemple de code suivant.

```
# Run the program set
task = device.run(
    program_set_1, shots=total_num_shots,
)
```

Lorsque vous utilisez Rigetti des appareils, votre ensemble de programmes peut rester dans RUNNING cet état alors que les tâches sont partiellement terminées et partiellement mises en file d'attente. Pour obtenir des résultats plus rapidement, pensez à soumettre votre programme défini en tant que [Job hybride](#).

Analyse des résultats

Exécutez le code suivant pour analyser et mesurer les résultats des exécutables dans un `ProgramSet`

```
# Get the results from a program set
result = task.result()

# Get the first executable
first_program = result[0]
first_executable = first_program[0]

# Inspect the results of the first executable
measurements_from_first_executable = first_executable.measurements
print(measurements_from_first_executable)
```

Mesure partielle

Au lieu de mesurer tous les qubits d'un circuit quantique, utilisez une mesure partielle pour mesurer des qubits individuels ou un sous-ensemble de qubits.

Note

Des fonctionnalités supplémentaires telles que la mesure en milieu de circuit et les opérations d'anticipation sont disponibles en tant que capacités expérimentales, voir [Accès aux circuits dynamiques sur](#) les appareils IQM.

Exemple : mesurer un sous-ensemble de qubits

L'exemple de code suivant illustre une mesure partielle en mesurant uniquement le qubit 0 dans un circuit à états de Bell.

```
from braket.devices import LocalSimulator
from braket.circuits import Circuit

# 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)
```

qubitAllocation manuelle

Lorsque vous exécutez un circuit quantique sur des ordinateurs quantiques à partir de Rigetti, vous pouvez éventuellement utiliser qubit l'allocation manuelle pour contrôler ceux qui qubits sont utilisés

pour votre algorithme. La [console Amazon Braket](#) et le SDK [Amazon Braket](#) vous aident à inspecter les données d'étalonnage les plus récentes de l'unité de traitement quantique (QPU) que vous avez sélectionnée, afin que vous puissiez sélectionner celle qui convient le mieux à votre expérience.

qubits

qubitL'allocation manuelle vous permet d'exécuter des circuits avec une plus grande précision et d'étudier les qubit propriétés individuelles. Les chercheurs et les utilisateurs expérimentés optimisent la conception de leurs circuits en fonction des dernières données d'étalonnage des appareils et peuvent obtenir des résultats plus précis.

L'exemple suivant montre comment allouer de qubits manière explicite.

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

device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
circ = Circuit().h(0).cnot(0, 7) # Indices of actual qubits in the QPU

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

my_task = device.run(circ, s3_location, shots=100, disable_qubit_rewiring=True)
```

Pour plus d'informations, consultez [les exemples d'Amazon Braket sur GitHub](#), ou plus précisément, ce bloc-notes : [Allocation de qubits sur les appareils QPU](#).

Compilation textuelle

Lorsque vous exécutez un circuit quantique sur des ordinateurs quantiques basés sur des portes, vous pouvez demander au compilateur d'exécuter vos circuits exactement tels que définis, sans aucune modification. À l'aide de la compilation textuelle, vous pouvez spécifier soit qu'un circuit entier soit préservé exactement comme spécifié, soit que seules des parties spécifiques de celui-ci soient préservées (prises en charge Rigetti uniquement par). Lorsque vous développez des algorithmes pour l'analyse comparative du matériel ou des protocoles d'atténuation des erreurs, vous devez avoir la possibilité de spécifier exactement les portes et les configurations de circuits qui s'exécutent sur le matériel. La compilation Verbatim vous permet de contrôler directement le processus de compilation en désactivant certaines étapes d'optimisation, garantissant ainsi que vos circuits fonctionnent exactement comme prévu.

La compilation Verbatim est prise en charge sur les Rigetti appareils AQT IonQIQM,, et nécessite l'utilisation de portes natives. Lors de l'utilisation de la compilation textuelle, il est conseillé de vérifier la topologie de l'appareil pour s'assurer que les portes sont connectées qubits et que le circuit utilise les portes natives prises en charge par le matériel. L'exemple suivant montre comment accéder par programmation à la liste des portes natives prises en charge par un appareil.

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

En effet Rigetti, qubit le recâblage doit être désactivé en le configurant `disableQubitRewiring=True` pour une utilisation avec la compilation textuelle. S'il `disableQubitRewiring=False` est défini lors de l'utilisation de boîtes verbatim dans une compilation, le circuit quantique échoue à la validation et ne s'exécute pas.

Si la compilation textuelle est activée pour un circuit et exécutée sur un QPU qui ne le prend pas en charge, une erreur est générée indiquant qu'une opération non prise en charge a entraîné l'échec de la tâche. Au fur et à mesure que de plus en plus de matériels quantiques prennent en charge nativement les fonctions de compilation, cette fonctionnalité sera étendue pour inclure ces appareils. Les appareils qui prennent en charge la compilation textuelle l'incluent en tant qu'opération prise en charge lorsqu'ils sont interrogés avec le code suivant.

```
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/Ankaa-3")
device.properties.action[DeviceActionType.OPENQASM].supportedPragmas
```

Il n'y a aucun coût supplémentaire associé à l'utilisation de la compilation textuelle. Les tâches quantiques exécutées sur les appareils Braket QPU, les instances d'ordinateurs portables et les simulateurs à la demande continuent de vous être facturées sur la base des tarifs actuels, tels que spécifiés sur la page de tarification d'[Amazon](#) Braket. Pour plus d'informations, consultez le bloc-notes d'exemple de [compilation Verbatim](#).

Note

Si vous utilisez OpenQASM pour écrire vos circuits pour les IonQ périphériques AQT et, et que vous souhaitez mapper votre circuit directement aux qubits physiques, vous devez utiliser le `#pragma braket verbatim` car le `disableQubitRewiring` drapeau est ignoré par OpenQASM.

Simulation du bruit

Pour instancier le simulateur de bruit local, vous pouvez modifier le backend comme suit.

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

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

Vous pouvez créer des circuits bruyants de deux manières :

1. Construisez le circuit bruyant de bas en haut.
2. Prenez un circuit sans bruit existant et injectez du bruit partout.

L'exemple suivant montre les approches utilisant un circuit de base avec un bruit dépolarisant et un canal Kraus personnalisé.

```
import scipy.stats
import numpy as np

# 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)
```

```
from braket.circuits import Noise

# 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)
```

L'exécution d'un circuit offre la même expérience utilisateur qu'auparavant, comme le montrent les deux exemples suivants.

Exemple 1

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

Or

Exemple 2

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

Pour plus d'exemples, consultez [l'exemple d'introduction du simulateur de bruit Braket](#)

Inspection du circuit

Les circuits quantiques d'Amazon Braket ont un concept de pseudo-temps appelé Moments. Chacun qubit peut bénéficier d'une seule porte par Moment. L'objectif Moments est de faciliter l'adressage des circuits et de leurs portes et de fournir une structure temporelle.

Note

Les moments ne correspondent généralement pas au temps réel auquel les portes sont exécutées sur un QPU.

La profondeur d'un circuit est donnée par le nombre total de moments qu'il contient. Vous pouvez visualiser la profondeur du circuit appelant la méthode `circuit.depth`, comme indiqué dans l'exemple suivant.

```
from braket.circuits import Circuit

# 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) ##### X ##
    ##### #      #####
    ##### #      #####
q1 : ## Ry(0.20) ##### ZZ(0.15) #####
    ##### #      #####
    ##### #
q2 : ##### X #####
    ##### #
    #####
q3 : ##### ZZ(0.15) #####
    #####
T : #      0      #      1      # 2 #
Total circuit depth: 3

```

La profondeur totale du circuit ci-dessus est de 3 (exprimée en moments 01, et2). Vous pouvez vérifier le fonctionnement du portail à chaque instant.

Moments fonctionne comme un dictionnaire de paires clé-valeur.

- La clé est qu'un `MomentsKey()` il contient du pseudo-temps et des qubit informations.
- La valeur est attribuée dans le type de `Instructions()`.

```

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

```

```

MomentsKey(time=0, qubits=QubitSet([Qubit(0)]), moment_type=<MomentType.GATE: 'gate'>,
noise_index=0, subindex=0)
Instruction('operator': Rx('angle': 0.15, 'qubit_count': 1), 'target':
QubitSet([Qubit(0)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=0, qubits=QubitSet([Qubit(1)]), moment_type=<MomentType.GATE: 'gate'>,
noise_index=0, subindex=0)
Instruction('operator': Ry('angle': 0.2, 'qubit_count': 1), 'target':
QubitSet([Qubit(1)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=1, qubits=QubitSet([Qubit(0), Qubit(2)]), moment_type=<MomentType.GATE:
'gate'>, noise_index=0, subindex=0)

```

```

Instruction('operator': CNot('qubit_count': 2), 'target': QubitSet([Qubit(0),
    Qubit(2)]), 'control': QubitSet([]), 'control_state': (), 'power': 1)

MomentsKey(time=1, qubits=QubitSet([Qubit(1), Qubit(3)]), moment_type=<MomentType.GATE:
    'gate'>, noise_index=0, subindex=0)
Instruction('operator': ZZ('angle': 0.15, 'qubit_count': 2), 'target':
    QubitSet([Qubit(1), Qubit(3)]), 'control': QubitSet([]), 'control_state': (), 'power':
    1)

MomentsKey(time=2, qubits=QubitSet([Qubit(0)]), moment_type=<MomentType.GATE: 'gate'>,
    noise_index=0, subindex=0)
Instruction('operator': X('qubit_count': 1), 'target': QubitSet([Qubit(0)]), 'control':
    QubitSet([]), 'control_state': (), 'power': 1)

```

Vous pouvez également ajouter des portes à un circuitMoments.

```

from braket.circuits import Instruction, Gate

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 : ##### H ##
        #####
T : # 0 # 1 # 2 #

```

Liste des types de résultats

Amazon Braket peut renvoyer différents types de résultats lorsqu'un circuit est mesuré à l'aide de. `ResultType` Un circuit peut renvoyer les types de résultats suivants.

- `AdjointGradient` renvoie le gradient (dérivé vectoriel) de la valeur attendue d'un observable fourni. Cet observable agit sur une cible donnée par rapport à des paramètres spécifiés en utilisant la méthode de différenciation adjointe. Vous ne pouvez utiliser cette méthode que lorsque `shots=0`.
- `Amplitude` renvoie l'amplitude des états quantiques spécifiés dans la fonction d'onde de sortie. Il n'est disponible que sur les simulateurs locaux SV1 et sur les simulateurs.
- `Expectation` renvoie la valeur attendue d'un observable donné, qui peut être spécifiée avec la `Observable` classe présentée plus loin dans ce chapitre. La cible qubits utilisée pour mesurer l'observable doit être spécifiée, et le nombre de cibles spécifiées doit être égal au nombre de cibles qubits sur lesquelles l'observable agit. Si aucune cible n'est spécifiée, l'observable ne doit fonctionner que sur 1 qubit et il est appliqué à tous qubits en parallèle.
- `Probability` renvoie les probabilités de mesure des états de base de calcul. Si aucune cible n'est spécifiée, `Probability` renvoie la probabilité de mesurer tous les états de base. Si des cibles sont spécifiées, seules les probabilités marginales des vecteurs de base sur les cibles spécifiées qubits sont renvoyées. Les simulateurs gérés QPUs sont limités à 15 qubits maximum, et les simulateurs locaux sont limités à la taille de la mémoire du système.
- `Reduced density matrix` renvoie une matrice de densité pour un sous-système d'une cible spécifiée qubits à partir d'un système de qubits. Pour limiter la taille de ce type de résultat, Braket limite le nombre de cibles qubits à un maximum de 8.
- `StateVector` renvoie le vecteur d'état complet. Il est disponible sur le simulateur local.
- `Sampler` renvoie le nombre de mesures d'un qubit ensemble cible spécifié et observable. Si aucune cible n'est spécifiée, l'observable ne doit fonctionner que sur 1 qubit et il est appliqué à tous qubits en parallèle. Si des cibles sont spécifiées, le nombre de cibles spécifiées doit être égal au nombre de cibles qubits sur lesquelles l'observable agit.
- `Variance` renvoie la variance ($\text{mean}([x - \text{mean}(x)]^2)$) de l'ensemble de cibles spécifié et observable en tant que type de résultat demandé. Si aucune cible n'est spécifiée, l'observable ne doit fonctionner que sur 1 qubit et il est appliqué à tous qubits en parallèle. Sinon, le nombre de cibles spécifiées doit être égal qubits au nombre auquel l'observable peut être appliqué.

Les types de résultats pris en charge par les différents fournisseurs :

SIM locale	SV1	DM1	TN1	AQT	IonQ	IQM	Rigetti
---------------	-----	-----	-----	-----	------	-----	---------

Gradient adjoint	N	Y	N	N	N	N	N	N
Amplitude	Y	Y	N	N	N	N	N	N
Espérance	Y	Y	Y	Y	Y	Y	Y	Y
Probabilité	Y	Y	Y	N	Y	Y	Y	Y
Matrice à densité réduite	Y	N	Y	N	N	N	N	N
Vecteur d'état	Y	N	N	N	N	N	N	N
Exemple	Y	Y	Y	Y	Y	Y	Y	Y
Variance	Y	Y	Y	Y	Y	Y	Y	Y

Vous pouvez vérifier les types de résultats pris en charge en examinant les propriétés de l'appareil, comme indiqué dans l'exemple suivant.

```
from braket.aws import AwsDevice

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

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

```
name='Sample' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Expectation' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Variance' observables=['x', 'y', 'z', 'h', 'i'] minShots=10 maxShots=50000
name='Probability' observables=None minShots=10 maxShots=50000
```

Pour appeler `aResultType`, ajoutez-le à un circuit, comme indiqué dans l'exemple suivant.

```
from braket.circuits import Circuit, 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

Différents appareils quantiques fournissent des résultats sous différents formats. Par exemple, les Rigetti appareils renvoient des mesures, tandis que IonQ les appareils fournissent des probabilités. Le SDK Amazon Braket propose une propriété de mesures pour tous les résultats. Toutefois, pour les appareils qui renvoient des probabilités, ces mesures sont post-calculées et basées sur les probabilités, car les mesures par tir ne sont pas disponibles. Pour déterminer si un résultat a été post-calculé, cochez la case `measurements_copied_from_device` sur l'objet du résultat. Cette opération est détaillée dans le fichier [gate_model_quantum_task_result.py](#) du référentiel Amazon Braket SDK GitHub .

Observables

La `Observable` classe d'Amazon Braket vous permet de mesurer un observable spécifique.

Vous ne pouvez appliquer qu'une seule observable de non-identité unique à chacun. qubit Une erreur se produit si vous spécifiez deux ou plusieurs observables non identitaires différents pour la même chose. qubit À cette fin, chaque facteur d'un produit tensoriel compte comme un observable individuel. Cela signifie que vous pouvez avoir plusieurs produits tensoriels sur le même produitqubit, à condition que les facteurs agissant sur ceux-ci qubit restent les mêmes.

Un observable peut être redimensionné et ajouter d'autres observables (mis à l'échelle ou non). Cela crée un `Sum` qui peut être utilisé dans le type de `AdjointGradient` résultat.

La Observable classe inclut les observables suivants.

```
import numpy as np

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

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

# Get the tensor product of the 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())

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

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

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

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



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

Parameters

Les circuits peuvent intégrer des paramètres libres. Ces paramètres libres ne doivent être créés qu'une seule fois pour être exécutés plusieurs fois et peuvent être utilisés pour calculer des dégradés.

Chaque paramètre gratuit utilise un nom codé sous forme de chaîne qui permet de :

- Définir les valeurs des paramètres
- Identifier les paramètres à utiliser

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

theta = FreeParameter("theta")
phi = FreeParameter("phi")
circ = Circuit().h(0).rx(0, phi).ry(0, phi).cnot(0, 1).xx(0, 1, theta)
```

Gradient adjoint

L'SV1 appareil calcule le gradient adjoint d'une valeur attendue observable, y compris un hamiltonien à plusieurs termes. Pour différencier les paramètres, spécifiez leur nom (sous forme de chaîne) ou par référence directe.

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

device = AwsDevice(Devices.Amazon.SV1)

circ.adjoint_gradient(observable=3 * Observable.Z(0) @ Observable.Z(1) - 0.5 *
observables.X(0), parameters = ["phi", theta])
```

Le fait de transmettre des valeurs de paramètres fixes en tant qu'arguments à un circuit paramétré supprimera les paramètres libres. L'exécution de ce circuit `AdjointGradient` produit une erreur,

car les paramètres libres n'existent plus. L'exemple de code suivant illustre l'utilisation correcte et incorrecte :

```
# Will error, as no free parameters will be present
#device.run(circ(0.2), shots=0)

# Will succeed
device.run(circ, shots=0, inputs={'phi': 0.2, 'theta': 0.2})
```

Obtenir des conseils d'experts

Connectez-vous à des experts en informatique quantique directement dans la console de gestion Braket pour obtenir des conseils supplémentaires sur vos charges de travail.

Pour explorer les options de conseil d'expert via Braket Direct, ouvrez la console Braket, choisissez Braket Direct dans le volet de gauche et accédez à la section Conseils d'experts. Les options de conseil d'experts suivantes sont disponibles :

- **Heures de bureau de Braket** : Les heures de bureau de Braket sont des sessions individuelles, premier arrivé, premier servi, qui ont lieu tous les mois. Chaque créneau horaire disponible est de 30 minutes et est gratuit. Discuter avec des experts de Braket peut vous aider à passer plus rapidement de l'idée à l'exécution en explorant l' use-case-to-deviceajustement, en identifiant les options permettant d'utiliser au mieux Braket pour votre algorithme et en obtenant des recommandations sur la manière d'utiliser certaines fonctionnalités de Braket, telles qu'Amazon Braket Hybrid Jobs, Braket Pulse ou Analog Hamiltonian Simulation.
- Pour vous inscrire aux heures d'ouverture de Braket, sélectionnez S'inscrire et renseignez les informations de contact, les détails de la charge de travail et les sujets de discussion souhaités.
- Vous recevrez une invitation au calendrier pour le prochain créneau disponible par e-mail.

Note

Pour les problèmes urgents ou les questions de dépannage rapide, nous vous recommandons de contacter le [AWS Support](#). Pour les questions non urgentes, vous pouvez également utiliser le [forum AWS Re:post](#) ou le [Quantum Computing Stack Exchange](#), où vous pouvez consulter les réponses aux questions précédentes et en poser de nouvelles.

- Offres des fournisseurs de matériel quantique : IonQ, QuEra, et Rigetti chacun propose des offres de services professionnels via AWS Marketplace.
 - Pour découvrir leurs offres, sélectionnez Connect et parcourez leurs listes.
 - Pour en savoir plus sur les offres de services professionnels sur le AWS Marketplace, consultez la section [Produits de services professionnels](#).
- AmazonQuantum Solutions Lab (QSL) : Le QSL est une équipe de recherche collaborative et de services professionnels composée d'experts en informatique quantique qui peuvent vous aider à explorer efficacement l'informatique quantique et à évaluer les performances actuelles de cette technologie.
 - Pour contacter le QSL, sélectionnez Connect, puis renseignez les informations de contact et les détails du cas d'utilisation.
 - L'équipe QSL vous contactera par e-mail pour vous indiquer les prochaines étapes.

Exécutez vos circuits avec OpenQASM 3.0

AmazonBraket prend désormais en charge [OpenQASM 3.0](#) pour les dispositifs quantiques et les simulateurs basés sur des portes. Ce guide de l'utilisateur fournit des informations sur le sous-ensemble d'OpenQASM 3.0 pris en charge par Braket. [Les clients de Braket ont désormais le choix de soumettre des circuits Braket avec le SDK ou de fournir directement des chaînes OpenQASM 3.0 à tous les appareils basés sur un portail avec l'API Amazon Braket et le SDK Amazon Braket Python.](#)

Les rubriques de ce guide vous présentent divers exemples illustrant comment effectuer les tâches quantiques suivantes.

- [Créez et soumettez des tâches quantiques OpenQASM sur différents appareils Braket](#)
- [Accédez aux opérations prises en charge et aux types de résultats](#)
- [Simulez le bruit avec OpenQASM](#)
- [Utiliser la compilation textuelle avec OpenQASM](#)
- [Résoudre les problèmes liés à OpenQASM](#)

Ce guide fournit également une introduction à certaines fonctionnalités spécifiques au matériel qui peuvent être implémentées avec OpenQASM 3.0 sur Braket et des liens vers d'autres ressources.

Dans cette section :

- [Qu'est-ce qu'OpenQASM 3.0 ?](#)

- [Quand utiliser OpenQASM 3.0](#)
- [Comment fonctionne OpenQASM 3.0](#)
- [Conditions préalables](#)
- [Quelles sont les fonctionnalités d'OpenQASM prises en charge par Braket ?](#)
- [Créez et soumettez un exemple de tâche quantique OpenQASM 3.0](#)
- [Support d'OpenQASM sur différents appareils Braket](#)
- [Simulez le bruit avec OpenQASM 3.0](#)
- [Qubitrecâblage avec OpenQASM 3.0](#)
- [Compilation Verbatim avec OpenQASM 3.0](#)
- [La console Braket](#)
- [Ressources supplémentaires](#)
- [Calculer des dégradés avec OpenQASM 3.0](#)
- [Mesurer des qubits spécifiques avec OpenQASM 3.0](#)

Qu'est-ce qu'OpenQASM 3.0 ?

L'Open Quantum Assembly Language (OpenQASM) est une [représentation intermédiaire](#) pour les instructions quantiques. OpenQASM est un framework open source largement utilisé pour la spécification de programmes quantiques pour les appareils basés sur des portes. Avec OpenQASM, les utilisateurs peuvent programmer les portes quantiques et les opérations de mesure qui constituent les éléments de base du calcul quantique. La version précédente d'OpenQASM (2.0) était utilisée par un certain nombre de bibliothèques de programmation quantique pour décrire des programmes de base.

La nouvelle version d'OpenQASM (3.0) étend la version précédente pour inclure davantage de fonctionnalités, telles que le contrôle du niveau des impulsions, le chronométrage des portes et le flux de contrôle classique pour combler le fossé entre l'interface utilisateur final et le langage de description du matériel. Les détails et les spécifications de la version 3.0 actuelle sont disponibles sur la spécification GitHub [OpenQASM 3.x Live](#). Le futur développement d'OpenQASM est régi par le [comité de pilotage technique](#) d'OpenQASM 3.0, dont AWS il est membre aux côtés d'IBM, Microsoft et de l'université d'Innsbruck.

Quand utiliser OpenQASM 3.0

OpenQASM fournit un cadre expressif permettant de spécifier des programmes quantiques par le biais de contrôles de bas niveau qui ne sont pas spécifiques à une architecture, ce qui le rend parfaitement adapté en tant que représentation sur plusieurs appareils basés sur des portes. La prise en charge d'OpenQASM par Braket favorise son adoption en tant qu'approche cohérente du développement d'algorithmes quantiques basés sur des portes, réduisant ainsi le besoin pour les utilisateurs d'apprendre et de gérer des bibliothèques dans plusieurs frameworks.

Si vous avez des bibliothèques de programmes existantes dans OpenQASM 3.0, vous pouvez les adapter pour les utiliser avec Braket plutôt que de réécrire complètement ces circuits. Les chercheurs et les développeurs devraient également bénéficier d'un nombre croissant de bibliothèques tierces disponibles prenant en charge le développement d'algorithmes dans OpenQASM.

Comment fonctionne OpenQASM 3.0

Support d'OpenQASM 3.0 par Braket assure la parité des fonctionnalités avec la représentation intermédiaire actuelle. Cela signifie que tout ce que vous pouvez faire aujourd'hui sur des appareils matériels et des simulateurs à la demande avec Braket, vous pouvez le faire avec OpenQASM en utilisant Braket. API Vous pouvez exécuter des programmes OpenQASM 3.0 en fournissant directement des chaînes OpenQASM à tous les périphériques basés sur un portail, de la même manière que les circuits sont actuellement fournis aux périphériques sur Braket. Les utilisateurs de Braket peuvent également intégrer des bibliothèques tierces compatibles avec OpenQASM 3.0. Le reste de ce guide explique comment développer des représentations OpenQASM à utiliser avec Braket.

Conditions préalables

[Pour utiliser OpenQASM 3.0 sur Amazon Braket, vous devez disposer de la version v1.8.0 des schémas Python Amazon Braket et de la version v1.17.0 ou supérieure du SDK Amazon Braket Python.](#)

Si vous utilisez Amazon Braket pour la première fois, vous devez activer Amazon Braket. Pour obtenir des instructions, consultez [Activer Amazon Braket](#).

Quelles sont les fonctionnalités d'OpenQASM prises en charge par Braket ?

La section suivante répertorie les types de données, les instructions et les instructions pragma d'OpenQASM 3.0 pris en charge par Braket.

Dans cette section :

- [Types de données OpenQASM pris en charge](#)
- [Déclarations OpenQASM prises en charge](#)
- [Pragmas OpenQASM de Braket](#)
- [Support des fonctionnalités avancées pour OpenQASM sur le simulateur local](#)
- [Opérations et grammaire prises en charge avec OpenPulse](#)

Types de données OpenQASM pris en charge

Les types de données OpenQASM suivants sont pris en charge par Amazon Braket.

- Des entiers non négatifs sont utilisés pour les indices de qubits (virtuels et physiques) :
 - `cnot q[0], q[1];`
 - `h $0;`
- Des nombres à virgule flottante ou des constantes peuvent être utilisés pour les angles de rotation des portes :
 - `rx(-0.314) $0;`
 - `rx(pi/4) $0;`

Note

`pi` est une constante intégrée à OpenQASM et ne peut pas être utilisée comme nom de paramètre.

- Les tableaux de nombres complexes (avec la `im` notation OpenQASM pour les parties imaginaires) sont autorisés dans les pragmas de type résultat pour définir les observables hermitiens généraux et dans les pragmas unitaires :
 - `#pragma braket unitary [[0, -1im], [1im, 0]] q[0]`
 - `#pragma braket result expectation hermitian([[0, -1im], [1im, 0]]) q[0]`

Déclarations OpenQASM prises en charge

Les instructions OpenQASM suivantes sont prises en charge par Amazon Braket.

- Header: OPENQASM 3;
- Déclarations binaires classiques :
 - bit b1;(de manière équivalente,creg b1;)
 - bit[10] b2;(de manière équivalente,creg b2[10];)
- Déclarations Qubit :
 - qubit b1;(de manière équivalente,qreg b1;)
 - qubit[10] b2;(de manière équivalente,qreg b2[10];)
- Indexation au sein de tableaux : q[0]
- Entrée :input float alpha;
- spécification physique qubits : \$0
- Portails et opérations pris en charge sur un appareil :
 - h \$0;
 - iswap q[0], q[1];

Note

Les portes prises en charge par un périphérique se trouvent dans les propriétés de l'appareil pour les actions OpenQASM ; aucune définition de porte n'est nécessaire pour utiliser ces portes.

- Relevés verbatim box. Actuellement, nous ne prenons pas en charge la notation de la durée des boîtes. Les portes natives et physiques qubits sont obligatoires dans les boîtes de saisie des verbatims.

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

- Mesure et attribution de mesures sur qubits ou sur un qubit registre complet.
 - measure \$0;
 - measure q;

- `measure q[0];`
- `b = measure q;`
- `measure q # b;`
- Les instructions `Barrier` fournissent un contrôle explicite sur la compilation et l'exécution des circuits en empêchant la réorganisation des portes et les optimisations au-delà des limites des barrières. Ils appliquent également un ordre temporel strict lors de l'exécution, garantissant ainsi toutes les opérations avant la fin d'une barrière avant le début des opérations suivantes.
 - `barrier;`
 - `barrier q[0], q[1];`
 - `barrier $3, $6;`

Pragmas OpenQASM de Braket

Les instructions pragma OpenQASM suivantes sont prises en charge par Amazon Braket.

- Pragmas relatifs au bruit
 - `#pragma braket noise bit_flip(0.2) q[0]`
 - `#pragma braket noise phase_flip(0.1) q[0]`
 - `#pragma braket noise pauli_channel`
- Pragmas textuels
 - `#pragma braket verbatim`
- Type de résultat : pragmas
 - Types de résultats invariants de base :
 - Vecteur d'état : `#pragma braket result state_vector`
 - Matrice de densité : `#pragma braket result density_matrix`
 - Pragmas de calcul du gradient :
 - Dégagé adjoint : `#pragma braket result adjoint_gradient expectation(2.2 * x[0] @ x[1]) all`
 - Types de résultats de base Z :
 - Amplitude : `#pragma braket result amplitude "01"`
 - Probabilité : `#pragma braket result probability q[0], q[1]`

- Espérance : `#pragma braket result expectation x(q[0]) @ y([q1])`
- Écart : `#pragma braket result variance hermitian([[0, -1im], [1im, 0]])`
\$0
- Échantillon : `#pragma braket result sample h($1)`

Note

OpenQASM 3.0 est rétrocompatible avec OpenQASM 2.0, de sorte que les programmes écrits à l'aide de la version 2.0 peuvent s'exécuter sur Braket. Cependant, les fonctionnalités d'OpenQASM 3.0 prises en charge par Braket présentent quelques différences syntaxiques mineures, telles que `qreg vs` et `vs. creg qubit bit`. Il existe également des différences dans la syntaxe des mesures, et celles-ci doivent être prises en charge par leur syntaxe correcte.

Support des fonctionnalités avancées pour OpenQASM sur le simulateur local

Il `LocalSimulator` prend en charge les fonctionnalités avancées d'OpenQASM qui ne sont pas proposées dans le cadre des QPU de Braket ou des simulateurs à la demande. La liste de fonctionnalités suivante n'est prise en charge que dans `LocalSimulator` :

- Modificateurs de portail
- Portails intégrés OpenQASM
- Variables classiques
- Opérations classiques
- Portails personnalisés
- Contrôle classique
- fichiers QASM
- Sous-programmes

Pour des exemples de chaque fonctionnalité avancée, consultez cet [exemple de bloc-notes](#). [Pour la spécification complète d'OpenQASM, consultez le site Web d'OpenQASM.](#)

Opérations et grammaire prises en charge avec OpenPulse

Types de OpenPulse données pris en charge

Blocs d'appels :

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

Blocs de décalcomanie :

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

Cadres :

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

Formes d'onde :

```
// 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};
```

Exemple d'étalonnage de portail personnalisé :

```
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;
```

Exemple d'impulsion arbitraire :

```
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);
```

```
}
```

Créez et soumettez un exemple de tâche quantique OpenQASM 3.0

Vous pouvez utiliser le SDK Amazon Braket Python, Boto3 ou le pour envoyer des tâches quantiques OpenQASM 3.0 AWS CLI à un appareil Amazon Braket.

Dans cette section :

- [Exemple de programme OpenQASM 3.0](#)
- [Utilisez le SDK Python pour créer des tâches quantiques OpenQASM 3.0](#)
- [Utilisez Boto3 pour créer des tâches quantiques OpenQASM 3.0](#)
- [Utilisez le AWS CLI pour créer des tâches OpenQASM 3.0](#)

Exemple de programme OpenQASM 3.0

[Pour créer une tâche OpenQASM 3.0, vous pouvez commencer par un programme OpenQASM 3.0 de base \(ghz.qasm\) qui prépare un état GHZ comme indiqué dans l'exemple suivant.](#)

```
// 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;
```

Utilisez le SDK Python pour créer des tâches quantiques OpenQASM 3.0

Vous pouvez utiliser le [SDK Amazon Braket Python](#) pour envoyer ce programme à un appareil Amazon Braket à l'aide du code suivant. Assurez-vous de remplacer l'exemple d'emplacement du compartiment Amazon S3 « amzn-s3-demo-bucket » par le nom de votre propre compartiment Amazon S3.

```
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/Ankaa-3")
from braket.ir.openqasm import Program

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

# Specify an optional s3 bucket location and number of shots
s3_location = ("amzn-s3-demo-bucket", "openqasm-tasks")
my_task = device.run(
    program,
    s3_location,
    shots=100,
)

```

Utilisez Boto3 pour créer des tâches quantiques OpenQASM 3.0

Vous pouvez également utiliser le [SDK AWS Python pour Braket \(Boto3\)](#) pour créer les tâches quantiques à l'aide de chaînes OpenQASM 3.0, comme indiqué dans l'exemple suivant. [L'extrait de code suivant fait référence à ghz.qasm qui prépare un état GHZ comme indiqué ci-dessus.](#)

```

import boto3
import json

my_bucket = "amzn-s3-demo-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/Ankaa-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,
)
```

Utilisez le AWS CLI pour créer des tâches OpenQASM 3.0

La [AWS Command Line Interface \(CLI\)](#) peut également être utilisée pour soumettre des programmes OpenQASM 3.0, comme indiqué dans l'exemple suivant.

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

Support d'OpenQASM sur différents appareils Braket

Pour les appareils compatibles avec OpenQASM 3.0, le `action` champ prend en charge une nouvelle action par le biais de la `GetDevice` réponse, comme indiqué dans l'exemple suivant pour les Rigetti périphériques et. IonQ

```
//OpenQASM as available with the Rigetti device capabilities
```

```

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

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

```

Pour les appareils qui prennent en charge le contrôle du pouls, pulse le champ est affiché dans la GetDevice réponse. L'exemple suivant montre ce pulse champ pour le Rigetti périphérique.

```

// 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
}
}
}

```

Les champs précédents détaillent les éléments suivants :

Ports :

Décrit les ports de périphériques externes (extern) prédéfinis déclarés sur le QPU en plus des propriétés associées au port donné. Tous les ports répertoriés dans cette structure sont prédéclarés en tant qu'identifiants valides dans le OpenQASM 3.0 programme soumis par l'utilisateur. Les propriétés supplémentaires d'un port sont les suivantes :

- Identifiant du port (PortID)
 - Le nom de port déclaré comme identifiant dans OpenQASM 3.0.
- Direction (direction)
 - La direction du port. Les ports d'entraînement transmettent des impulsions (direction « tx »), tandis que les ports de mesure reçoivent des impulsions (direction « rx »).
- Type de port (PortType)
 - Type d'action dont ce port est responsable (par exemple, drive, capture ou ff - fast-flux).
- Dt (dt)
 - Durée en secondes qui représente un seul pas de temps d'échantillonnage sur le port donné.
- Mappages de qubits (QubitMappings)
 - Les qubits associés au port donné.
- Fréquences centrales (CenterFrequencies)
 - Liste des fréquences centrales associées pour toutes les trames prédéclarées ou définies par l'utilisateur sur le port. Pour plus d'informations, reportez-vous à la section Cadres.
- Propriétés spécifiques à QHP () qhpSpecificProperties
 - Une carte facultative détaillant les propriétés existantes concernant le port spécifique au QHP.

Cadres :

Décrit les cadres externes préfabriqués déclarés sur le QPU ainsi que les propriétés associées aux cadres. Toutes les trames répertoriées dans cette structure sont prédéclarées comme des identifiants valides dans le OpenQASM 3.0 programme soumis par l'utilisateur. Les propriétés supplémentaires d'un cadre sont les suivantes :

- ID du cadre (FrameID)
 - Le nom du cadre déclaré comme identifiant dans OpenQASM 3.0.
- Identifiant du port (ID du port)
 - Le port matériel associé à la trame.
- Fréquence (fréquence)
 - Fréquence initiale par défaut de la trame.
- Fréquence centrale (CenterFrequency)
 - Le centre de la bande passante de fréquence de la trame. Généralement, les trames ne peuvent être ajustées qu'à une certaine bande passante autour de la fréquence centrale. Par conséquent,

les ajustements de fréquence doivent rester dans un delta donné par rapport à la fréquence centrale. Vous pouvez trouver la valeur de bande passante dans les paramètres de validation.

- Phase (phase)
 - Phase initiale par défaut de la trame.
- Porte associée (porte associée)
 - Les portes associées à la trame donnée.
- Mappages de qubits (QubitMappings)
 - Les qubits associés à la trame donnée.
- Propriétés spécifiques à QHP () qhpSpecificProperties
 - Une carte optionnelle détaillant les propriétés existantes concernant le cadre spécifique au QHP.

SupportsDynamicFrames:

Décrit si un cadre peut être déclaré `cal` ou bloqué par défaut `cal` le biais de la `OpenPulse newframe` fonction. Si cette valeur est fausse, seules les images répertoriées dans la structure des cadres peuvent être utilisées dans le programme.

SupportedFunctions:

Décrit les `OpenPulse` fonctions prises en charge par le périphérique en plus des arguments, des types d'arguments et des types de retour associés aux fonctions données. Pour voir des exemples d'utilisation des `OpenPulse` fonctions, reportez-vous à la [OpenPulsespécification](#). À l'heure actuelle, Braket prend en charge :

- `shift_phase`
 - Déplace la phase d'une image d'une valeur spécifiée
- `set_phase`
 - Définit la phase du cadre à la valeur spécifiée
- `phases d'échange`
 - Échange les phases entre deux images.
- `shift_frequency`
 - Déplace la fréquence d'une image d'une valeur spécifiée
- `set_frequency`
 - Définit la fréquence de l'image à la valeur spécifiée

- jouer
 - Planifie une forme d'onde
- capture_v0
 - Renvoie la valeur d'une image de capture dans un registre de bits

SupportedQhpTemplateWaveforms:

Décrit les fonctions de forme d'onde prédéfinies disponibles sur le périphérique ainsi que les arguments et les types associés. Par défaut, Braket Pulse propose des routines de forme d'onde prédéfinies sur tous les appareils, à savoir :

Constante

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

τ est la longueur de la forme d'onde et iq est un nombre complexe.

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

Gaussien

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

τ est la longueur de la forme d'onde, σ la largeur de la gaussienne et l'amplitude A . Si elle est réglée ZaE sur `True`, la valeur gaussienne est décalée et redimensionnée de telle sorte qu'elle soit égale à zéro au début et à la fin de la forme d'onde, et qu'elle atteigne son maximum. A

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

DRAG Gaussien

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

τ est la longueur de la forme d'onde, σ est la largeur de la gaussienne, β est un paramètre libre et A est l'amplitude. Si cette valeur est ZaE définie sur `True`, la valeur gaussienne de suppression des

dérivées par porte adiabatique (DRAG) est décalée et redimensionnée de telle sorte qu'elle soit égale à zéro au début et à la fin de la forme d'onde, et que la partie réelle atteigne son maximum. A Pour plus d'informations sur la forme d'onde DRAG, consultez l'article [Simple Pulses for Elimination of Leak in Weakly Nonlinear](#) Qubits.

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

Place Erf

$$\text{Erf_Square}(t, L, W, \sigma, A = 1, Z_{aE} = 0) =$$

$$A \times \frac{\text{erf}((t - t_1)/\sigma) + \text{erf}(-(t - t_2)/\sigma)}{2 \times \text{erf}(W/2\sigma)}$$

Où L est la longueur, W est la largeur de la forme d'onde, σ définit la vitesse à laquelle les arêtes montent et descendent $t_2 = (L+W)/2$, $t_1 = (L-W)/2$ et quelle A est l'amplitude. Si elle est réglée Z_{aE} sur `True`, la valeur gaussienne est décalée et redimensionnée de telle sorte qu'elle soit égale à zéro au début et à la fin de la forme d'onde, et qu'elle atteigne son maximum. A L'équation suivante est la version redimensionnée de la forme d'onde.

$$\text{Erf_Square}(\dots, Z_{aE} = 1) = (a \times \text{Erf_Square}(\dots, Z_{aE} = 0) - bA)/(a - b)$$

Où $a = \text{erf}(W/2\sigma)$ et $b = \text{erf}(-t_1/\sigma)/2 + \text{erf}(t_2/\sigma)/2$.

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

`SupportsLocalPulseElements`:

Décrit si les éléments d'impulsion, tels que les ports, les trames et les formes d'onde, peuvent être définis localement par `defcal` blocs. Si la valeur est `false`, les éléments doivent être définis par `cal` blocs.

`SupportsNonNativeGatesWithPulses`:

Décrit si nous pouvons ou ne pouvons pas utiliser des portes non natives en combinaison avec des programmes d'impulsions. Par exemple, vous ne pouvez pas utiliser une porte non native

comme une H porte dans un programme sans d'abord définir la porte `defcal` pour le qubit utilisé. Vous pouvez trouver la liste des `nativeGateSet` clés de porte natives sous les fonctionnalités de l'appareil.

`ValidationParameters`:

Décrit les limites de validation des éléments d'impulsion, notamment :

- Échelle maximale/Valeurs d'amplitude maximales pour les formes d'onde (arbitraires et prédéfinies)
- Largeur de bande de fréquence maximale à partir de la fréquence centrale fournie en Hz
- Pouls minimal length/duration en secondes
- Pouls maximal length/duration en secondes

Opérations, résultats et types de résultats pris en charge avec OpenQASM

Pour savoir quelles fonctionnalités d'OpenQASM 3.0 sont prises en charge par chaque appareil, vous pouvez vous référer à la `braket.ir.openqasm.program` clé dans le `action` champ de sortie des capacités de l'appareil. Par exemple, les opérations prises en charge et les types de résultats disponibles pour le simulateur SV1 Braket State Vector sont les suivants.

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

```

Simulez le bruit avec OpenQASM 3.0

Pour simuler le bruit avec OpenQASM3, vous utilisez les instructions `pragma` pour ajouter des opérateurs de bruit. Par exemple, pour simuler la version bruyante du [programme GHZ](#) fournie précédemment, vous pouvez soumettre le programme OpenQASM suivant.

```

// 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;

```

Les spécifications de tous les opérateurs de bruit `pragma` pris en charge sont fournies dans la liste suivante.

```

#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
```

Opérateur Kraus

Pour générer un opérateur Kraus, vous pouvez parcourir une liste de matrices en imprimant chaque élément de la matrice sous forme d'expression complexe.

Lorsque vous utilisez les opérateurs Kraus, n'oubliez pas les points suivants :

- Le nombre de ne qubits doit pas dépasser 2. La [définition actuelle dans les schémas](#) définit cette limite.
- La longueur de la liste d'arguments doit être un multiple de 8. Cela signifie qu'il ne doit être composé que de matrices 2x2.
- La longueur totale ne dépasse pas $2^{\text{num_qubits}}$ matrices. Cela signifie 4 matrices pour 1 qubit et 16 pour 2 qubits.
- Toutes les matrices fournies sont [totalement conservatrices de traces positives \(CPTP\)](#).
- Le produit des opérateurs de Kraus et de leurs conjugués de transposition doit constituer une matrice d'identité.

Qubitrecâblage avec OpenQASM 3.0

[Amazon Braket prend en charge la qubit notation physique dans OpenQASM sur les Rigetti appareils \(pour en savoir plus, consultez cette page\)](#). Lorsque vous utilisez le physique qubits avec une [stratégie de recâblage naïve](#), assurez-vous qu'ils sont connectés sur le périphérique sélectionné. Sinon, si qubit des registres sont utilisés à la place, la stratégie de recâblage PARTIEL est activée par défaut sur Rigetti les appareils.

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

h $0;
```

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

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

Compilation Verbatim avec OpenQASM 3.0

Lorsque vous exécutez un circuit quantique sur des ordinateurs quantiques fournis par des fournisseurs tels que Rigetti, et IonQ, vous pouvez demander au compilateur d'exécuter vos circuits exactement comme définis, sans aucune modification. Cette fonctionnalité est connue sous le nom de compilation textuelle. Avec les appareils Rigetti, vous pouvez spécifier précisément ce qui doit être préservé, soit un circuit entier, soit uniquement des parties spécifiques de celui-ci. Pour ne conserver que des parties spécifiques d'un circuit, vous devrez utiliser des portes natives dans les régions préservées. Actuellement, il IonQ ne prend en charge que la compilation textuelle pour l'ensemble du circuit, de sorte que chaque instruction du circuit doit être incluse dans une boîte textuelle.

Avec OpenQASM, vous pouvez spécifier explicitement un pragma textuel autour d'une boîte de code qui est ensuite laissée intacte et non optimisée par la routine de compilation de bas niveau du matériel. L'exemple de code suivant montre comment utiliser la `#pragma braket verbatim` directive pour y parvenir.

```
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;
```

Pour des informations plus détaillées sur le processus de compilation des verbatim, y compris des exemples et des meilleures pratiques, consultez le bloc-notes d'exemples de [compilation Verbatim](#) disponible dans le référentiel github. `amazon-braket-examples`

La console Braket

Les tâches OpenQASM 3.0 sont disponibles et peuvent être gérées dans la console Amazon Braket. Sur la console, vous avez la même expérience en soumettant des tâches quantiques dans OpenQASM 3.0 qu'en soumettant des tâches quantiques existantes.

Ressources supplémentaires

OpenQASM est disponible dans toutes les régions Amazon Braket.

[Pour un exemple de bloc-notes permettant de démarrer avec OpenQASM sur Amazon Braket, consultez les didacticiels Braket. \[GitHub\]\(#\)](#)

Calculer des dégradés avec OpenQASM 3.0

Amazon Braket prend en charge le calcul des dégradés sur des simulateurs locaux et à la demande lors de l'exécution en mode (exact). `shots=0` Ceci est réalisé grâce à l'utilisation de la méthode de différenciation adjointe. Pour spécifier le gradient que vous souhaitez calculer, vous pouvez fournir le pragma approprié, comme illustré dans le code de l'exemple suivant.

```
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
```

Au lieu de répertorier explicitement tous les paramètres individuels, vous pouvez également spécifier le `all` mot-clé dans le pragma. Cela permettra de calculer le gradient par rapport à tous les input paramètres listés, ce qui peut être une option pratique lorsque le nombre de paramètres est très élevé. Dans ce cas, le pragma ressemblera au code de l'exemple suivant.

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

Tous les types d'observables sont pris en charge dans l'implémentation OpenQASM 3.0 d'Amazon Braket, y compris les opérateurs individuels, les produits tensoriels, les observables hermitiens et les observables. L'opérateur spécifique que vous souhaitez utiliser lors du calcul des dégradés doit être intégré à la `expectation()` fonction, et les qubits sur lesquels agit chaque terme de l'observable doivent être explicitement spécifiés.

Mesurer des qubits spécifiques avec OpenQASM 3.0

Le simulateur vectoriel d'état local et le simulateur de matrice de densité local fournis par Amazon Braket permettent de soumettre des OpenQASM programmes dans lesquels un sous-ensemble des qubits du circuit peut être mesuré de manière sélective. Cette capacité, souvent appelée mesure partielle, permet des calculs quantiques plus ciblés et plus efficaces. Par exemple, dans l'extrait de code suivant, vous pouvez créer un circuit à deux qubits et choisir de mesurer uniquement le premier qubit, tout en laissant le second qubit non mesuré.

```
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];
"""
```

Dans cet exemple, nous avons un circuit quantique à deux qubits `q[1]`, `q[0]` mais nous ne voulons mesurer que l'état du premier qubit. Ceci est réalisé par la ligne `b[0] = measure q[0]`, qui mesure l'état du qubit `[0]` et stocke le résultat dans le bit classique `b[0]`. Pour exécuter ce scénario de mesure partielle, nous pouvons exécuter le code suivant sur le simulateur vectoriel d'état local fourni par Amazon Braket.

```
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)
```

Vous pouvez vérifier si un appareil prend en charge les mesures partielles en inspectant le `requiresAllQubitsMeasurement` champ dans ses propriétés d'action ; si c'est le cas `False`, les mesures partielles sont prises en charge.

```
from braket.devices import Devices

AwsDevice(Devices.Rigetti.Ankaa3).properties.action['braket.ir.openqasm.program'].requiresAllQubitsMeasurementVoilàFalse,
```

ce qui indique que tous les qubits ne doivent pas être mesurés.

Explorez les capacités expérimentales

Les capacités expérimentales permettent d'accéder à du matériel dont la disponibilité est limitée et aux nouvelles fonctionnalités logicielles émergentes. Ces fonctionnalités peuvent avoir un impact sur les performances de l'appareil au-delà des spécifications standard. Vous pouvez activer automatiquement les fonctionnalités logicielles expérimentales par tâche via le SDK Amazon Braket.

Pour utiliser les capacités expérimentales, spécifiez le `experimental_capabilities` paramètre lorsque vous créez des tâches quantiques. Définissez ce paramètre sur "ALL" pour activer toutes les fonctionnalités expérimentales disponibles pour cette tâche. L'exemple suivant montre comment activer les fonctionnalités expérimentales lorsque vous exécutez un circuit sur un périphérique :

```
from braket.aws import AwsDevice

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

task = device.run(
    circuit,
    shots=1000,
    experimental_capabilities="ALL"
)
```

Note

Ces fonctionnalités sont expérimentales et peuvent être modifiées sans préavis. Les performances de l'appareil peuvent différer des spécifications publiées, et les résultats

peuvent différer des opérations standard. Vous devez activer explicitement les fonctionnalités expérimentales pour chaque tâche. Les tâches dépourvues de ce paramètre utiliseront uniquement les fonctionnalités standard de l'appareil.

Dans cette section :

- [Accès au dérèglage local sur Aquila QuEra](#)
- [Accès à de hautes géométries sur Aquila QuEra](#)
- [Accès à des géométries serrées sur Aquila QuEra](#)
- [Circuits dynamiques sur les appareils IQM](#)

Accès au dérèglage local sur Aquila QuEra

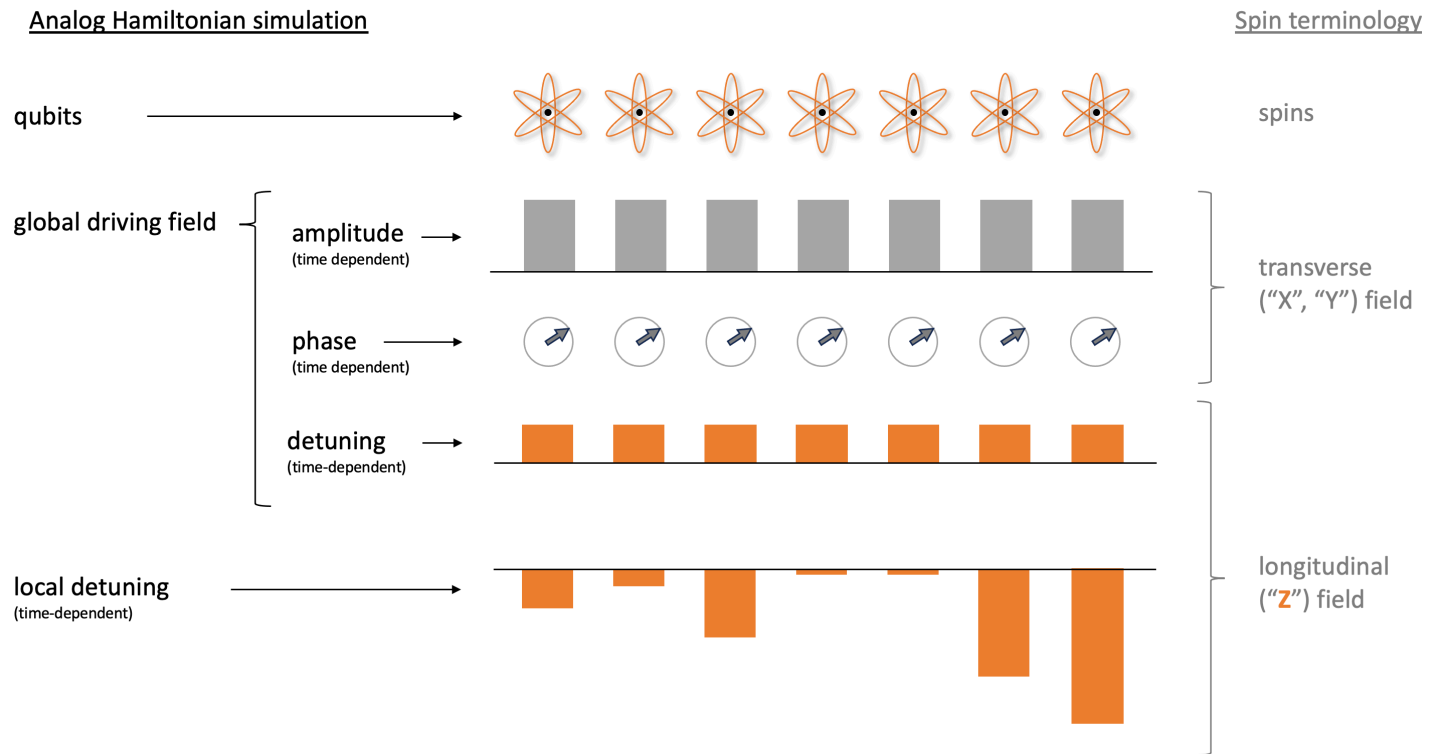
Le dérèglage local (LD) est un nouveau champ de contrôle dépendant du temps doté d'un modèle spatial personnalisable. Le champ LD affecte les qubits selon un schéma spatial personnalisable, réalisant différents hamiltoniens pour différents qubits, au-delà de ce que le champ de conduite uniforme et l'interaction Rydberg-Rydberg peuvent créer.

Contraintes :

Le modèle spatial du champ de désaccordage local est personnalisable pour chaque programme AHS, mais il est constant au cours d'un programme. La série chronologique du champ de désaccordage local doit commencer et se terminer à zéro, toutes les valeurs étant inférieures ou égales à zéro. De plus, les paramètres du champ de désaccordage local sont limités par des contraintes numériques, qui peuvent être consultées via le SDK Braket dans la section des propriétés spécifiques de l'appareil - `aquila_device.properties.paradigm.rydberg.rydbergLocal`

Limites :

Lors de l'exécution de programmes quantiques utilisant le champ de désaccordage local (même si son amplitude est fixée à zéro constant dans l'hamiltonien), le dispositif présente une décohérence plus rapide que le temps T2 indiqué dans la section des propriétés d'Aquila consacrée aux performances. Lorsque cela n'est pas nécessaire, il est recommandé d'omettre le champ de désaccordage local de l'hamiltonien du programme AHS.



Exemples :

1. Simulation de l'effet d'un champ magnétique longitudinal non uniforme dans les systèmes de spin

Alors que l'amplitude et la phase du champ moteur ont le même effet sur les qubits que le champ magnétique transverse sur les spins, la somme du désaccord du champ moteur et du désaccord local produit le même effet sur les qubits que le champ longitudinal sur les spins. Grâce au contrôle spatial du champ de désaccordage local, des systèmes de spin plus complexes peuvent être simulés.

2. Préparation des états initiaux hors équilibre

Le bloc-notes d'exemple [Simulation de la théorie des jauges en réseau avec des atomes de Rydberg](#) montre comment empêcher l'atome central d'un arrangement linéaire à 9 atomes d'être excité lors du recuit du système vers la phase ordonnée Z2. Après l'étape de préparation, le champ de désaccordage local est réduit et le programme AHS continue de simuler l'évolution temporelle du système à partir de cet état de non-équilibre particulier.

3. Résolution de problèmes d'optimisation pondérée

L'exemple du kit MWIS ([Maximum Weight Independent Set](#)) pour ordinateur portable montre comment résoudre un problème MWIS sur Aquila. Le champ de désaccordage local est utilisé

pour définir les poids sur les nœuds du graphe à disque unitaire, dont les arêtes sont réalisées par l'effet de blocage Rybderg. En partant de l'état fondamental uniforme et en augmentant progressivement le champ de désaccordage local, le système passe à l'état fondamental du MWIS Hamiltonian afin de trouver des solutions au problème.

Accès à de hautes géométries sur Aquila QuEra

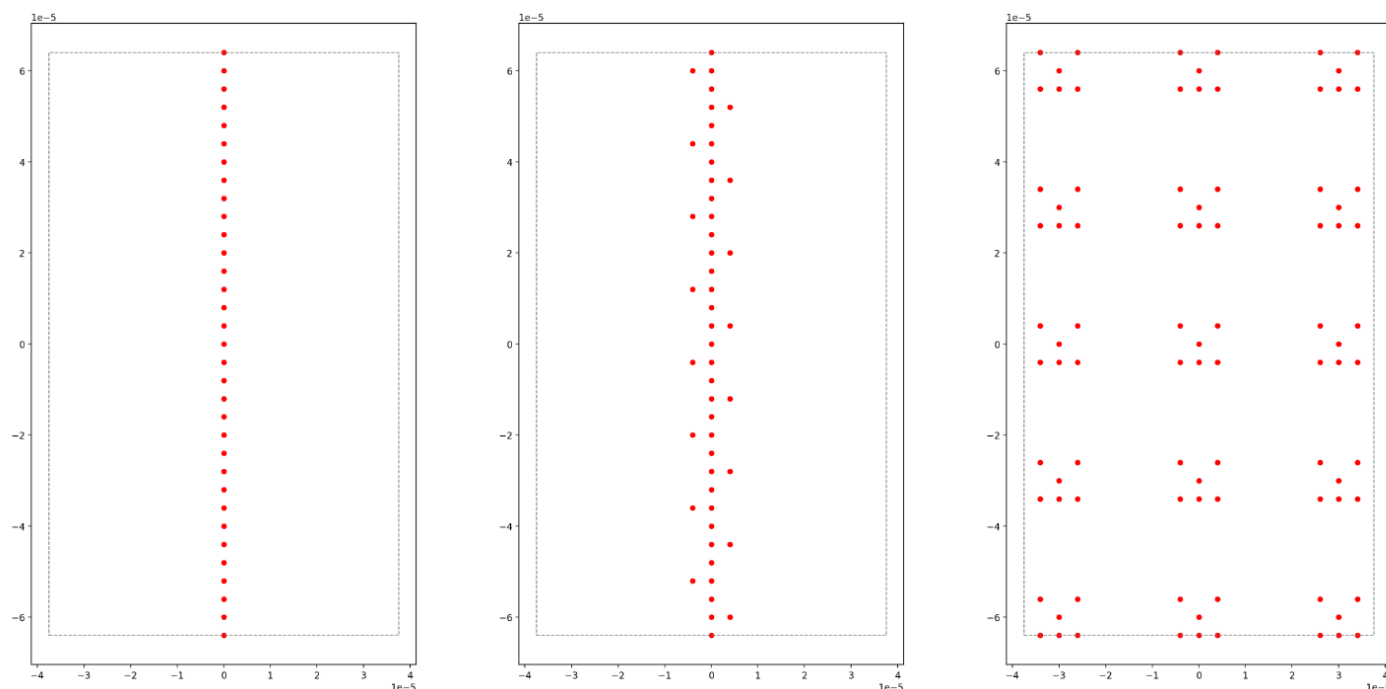
La fonction de géométries hautes vous permet de spécifier des géométries avec une hauteur accrue. Grâce à cette fonctionnalité, les arrangements atomiques de vos programmes AHS peuvent s'étendre sur une longueur supplémentaire dans la direction y au-delà des capacités habituelles d'Aquila.

Contraintes :

La hauteur maximale pour les géométries hautes est de 0,000128 m (128 um).

Limites :

Lorsque cette fonctionnalité expérimentale est activée pour votre compte, les fonctionnalités affichées sur la page des propriétés de l'appareil et lors de l'`GetDevice` appel continueront de refléter la limite inférieure normale de hauteur. Lorsqu'un programme AHS utilise des arrangements atomiques qui vont au-delà des capacités habituelles, l'erreur de remplissage devrait augmenter. Vous trouverez un nombre élevé de 0 inattendus dans la `pre_sequence` partie du résultat de la tâche, ce qui réduira les chances d'obtenir un arrangement parfaitement initialisé. Cet effet est le plus fort dans les rangées comportant de nombreux atomes.



Exemples :

1. Arrangements 1D et quasi-1D plus grands

Les chaînes d'atomes et les arrangements en forme d'échelle peuvent être étendus à des nombres d'atomes plus élevés. En orientant la direction longue parallèlement à y, il est possible de programmer des instances plus longues de ces modèles.

2. Plus d'espace pour le multiplexage de l'exécution de tâches avec de petites géométries

L'exemple de bloc-notes [Tâches quantiques parallèles sur Aquila](#) montre comment tirer le meilleur parti de la zone disponible : en plaçant des copies multiplexées de la géométrie en question dans un arrangement d'atomes. Avec la plus grande surface disponible, davantage de copies peuvent être placées.

Accès à des géométries serrées sur Aquila QuEra

La fonction de géométries serrées vous permet de spécifier des géométries avec un espacement plus court entre les lignes voisines. Dans un programme AHS, les atomes sont disposés en rangées, séparées par un espacement vertical minimal. La coordonnée y de deux sites atomiques doit être égale à zéro (même ligne) ou être différente de l'espacement minimal entre les lignes (ligne différente). Grâce à la fonctionnalité de géométries serrées, l'espacement minimal entre les rangées est réduit, ce qui permet de créer des arrangements atomiques plus serrés. Bien que cette extension

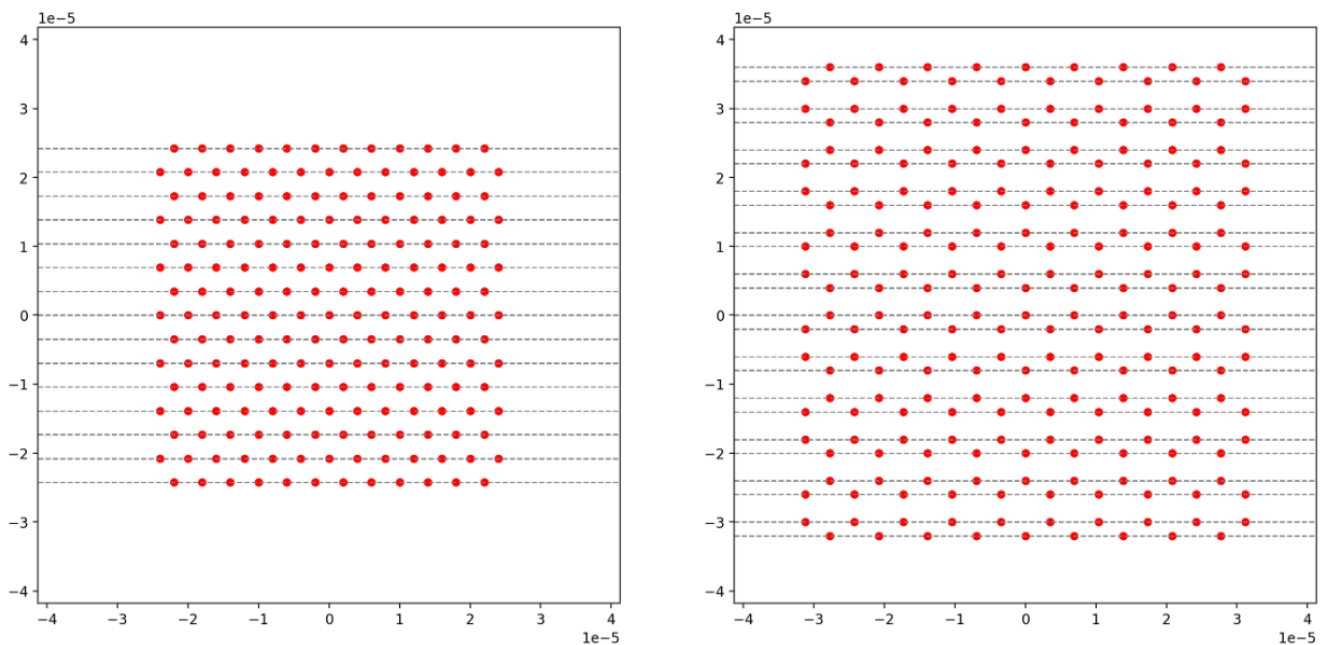
ne modifie pas l'exigence de distance euclidienne minimale entre les atomes, elle permet de créer des réseaux où des atomes distants occupent des rangées voisines plus proches les unes des autres, comme le réseau triangulaire en est un exemple notable.

Contraintes :

L'espacement minimal entre les rangées pour les géométries serrées est de 0,000002 m (2 μm).

Limites :

Lorsque cette fonctionnalité expérimentale est activée pour votre compte, les fonctionnalités affichées sur la page des propriétés de l'appareil et lors de l'`GetDevice` appel continueront de refléter la limite inférieure normale de hauteur. Lorsqu'un programme AHS utilise des arrangements atomiques qui vont au-delà des capacités habituelles, l'erreur de remplissage devrait augmenter. Les clients trouveront un nombre élevé de 0 inattendus dans la `pre_sequence` partie du résultat de la tâche, ce qui réduira les chances d'obtenir un arrangement parfaitement initialisé. Cet effet est le plus fort dans les rangées comportant de nombreux atomes.



Exemples :

1. Réseaux non rectangulaires avec de petites constantes de réseau

Un espacement des rangées plus serré permet de créer des réseaux où les voisins les plus proches de certains atomes se trouvent dans le sens de la diagonale. Les réseaux triangulaires, hexagonaux, les réseaux de Kagome et certains quasi-cristaux en sont des exemples notables.

2. Famille de réseaux réglables

Dans les programmes AHS, les interactions sont ajustées en ajustant la distance entre les paires d'atomes. Un espacement des rangées plus serré permet d'ajuster les interactions entre les différentes paires d'atomes les unes par rapport aux autres avec plus de liberté, car les angles et les distances qui définissent la structure des atomes sont moins limités par la contrainte minimale d'espacement des lignes. Un exemple notable est la famille des réseaux de Shastry-Sutherland avec différentes longueurs de liaison.

Circuits dynamiques sur les appareils IQM

Les circuits dynamiques des IQM appareils permettent des mesures en milieu de circuit (MCM) et des opérations anticipées. Ces fonctionnalités permettent aux chercheurs et aux développeurs quantiques de mettre en œuvre des algorithmes quantiques avancés dotés de fonctionnalités de logique conditionnelle et de réutilisation des qubits. Cette fonctionnalité expérimentale permet d'explorer des algorithmes quantiques avec une meilleure efficacité des ressources et d'étudier des schémas d'atténuation et de correction des erreurs quantiques.

Principales instructions :

- `measure_ff`: implémente des mesures pour le contrôle anticipé, en mesurant un qubit et en stockant le résultat à l'aide d'une clé de rétroaction.
- `cc_prx`: implémente une rotation contrôlée de manière classique qui s'applique uniquement lorsque le résultat associé à la clé de feedback donnée mesure un état $|1\rangle$.

Amazon Braket prend en charge les circuits dynamiques à travers OpenQASM, le Amazon Braket SDK, et le Amazon Braket Qiskit Provider

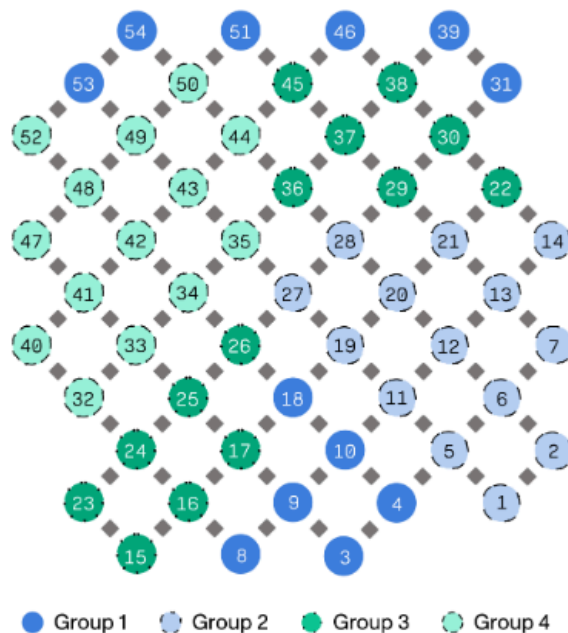
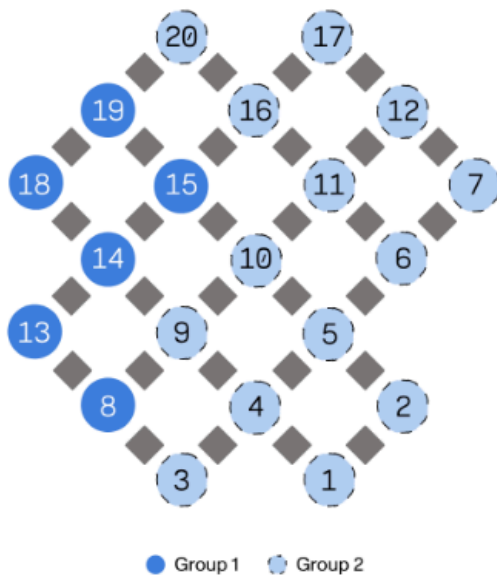
Contraintes :

1. Les clés de commentaires figurant dans `measure_ff` les instructions doivent être uniques.
2. Cela `cc_prx` doit se produire ensuite `measure_ff` avec la même clé de feedback.

3. Dans un circuit unique, le feed-forward sur un qubit ne peut être contrôlé que par un qubit, soit seul, soit par un autre qubit. Dans différents circuits, vous pouvez avoir différentes paires de commande.
 - a. Par exemple, si le qubit 1 est contrôlé par le qubit 2, il ne peut pas être contrôlé par le qubit 3 dans le même circuit. Il n'y a aucune contrainte quant au nombre de fois que le contrôle est appliqué entre le qubit 1 et le qubit 2. Le qubit 2 peut être contrôlé par le qubit 3 (ou le qubit 1), sauf si une réinitialisation active a été effectuée sur le qubit 2.
4. Le contrôle ne peut être appliqué qu'aux qubits d'un même groupe. Les groupes de qubits pour les Emerald appareils IQM Garnet et sont présentés dans les images suivantes.
5. Les programmes dotés de ces capacités doivent être soumis sous forme de programmes textuels. Pour en savoir plus sur les programmes Verbatim, voir [Compilation Verbatim avec OpenQASM 3.0](#).

Limites :

Le MCM ne peut être utilisé que pour le contrôle anticipé dans un programme. Les résultats MCM (0 ou 1) ne sont pas renvoyés dans le cadre du résultat d'une tâche.



Ces images montrent les groupements de qubits pour les deux IQM appareils. L'appareil Garnet à 20 qubits contient 2 groupes de qubits, tandis que l'appareil à Emerald 54 qubits contient 4 groupes de qubits.

Exemples :

1. Réutilisation des qubits par réinitialisation active

Le MCM avec des opérations de réinitialisation conditionnelle permet la réutilisation des qubits dans le cadre d'une exécution de circuit unique. Cela réduit les exigences de profondeur des circuits et améliore l'utilisation des ressources des dispositifs quantiques.

2. Protection active contre le retournement des embouts

Les circuits dynamiques détectent les erreurs de retournement de bits et appliquent des opérations correctives en fonction des résultats de mesure. Cette implémentation sert d'expérience de détection d'erreurs quantiques.

3. Expériences de téléportation

La téléportation d'État transfère les états des qubits à l'aide d'opérations quantiques locales et d'informations classiques provenant de. MCMs La téléportation par portes met en œuvre des portes entre les qubits sans opérations quantiques directes. Ces expériences mettent en évidence des sous-programmes fondamentaux dans trois domaines clés : la correction des erreurs quantiques, l'informatique quantique basée sur les mesures et la communication quantique.

4. Simulation de systèmes quantiques ouverts

Les circuits dynamiques modélisent le bruit dans les systèmes quantiques grâce à l'enchevêtrement des qubits de données et de l'environnement, ainsi qu'à des mesures environnementales. Cette approche utilise des qubits spécifiques pour représenter les données et les éléments de l'environnement. Un canal de bruit peut être conçu par les portes et les mesures appliquées à l'environnement.

Pour plus d'informations sur l'utilisation des circuits dynamiques, consultez des exemples supplémentaires dans le référentiel de [blocs-notes Amazon Braket](#).

Contrôle du pouls sur Amazon Braket

Les impulsions sont les signaux analogiques qui contrôlent les qubits d'un ordinateur quantique. Sur certains appareils d'Amazon Braket, vous pouvez accéder à la fonction de contrôle des impulsions pour soumettre des circuits à l'aide d'impulsions. Vous pouvez accéder au contrôle du pouls via le SDK Braket, à l'aide d'OpenQASM 3.0, ou directement via le Braket. APIs Tout d'abord, introduisez quelques concepts clés pour le contrôle du pouls dans Braket.

Dans cette section :

- [Frames \(Images\)](#)
- [Ports](#)
- [Formes d'onde](#)
- [Travailler avec Hello Pulse](#)
- [Accès aux portes natives à l'aide d'impulsions](#)

Frames (Images)

Un cadre est une abstraction logicielle qui agit à la fois comme une horloge dans le programme quantique et comme une phase. Le temps d'horloge est incrémenté à chaque utilisation et un signal porteur dynamique est défini par une fréquence. Lors de la transmission de signaux vers le qubit, une trame détermine la fréquence porteuse du qubit, le décalage de phase et l'heure à laquelle l'enveloppe de forme d'onde est émise. Dans Braket Pulse, la construction des cadres dépend de l'appareil, de la fréquence et de la phase. Selon le périphérique, vous pouvez choisir un cadre prédéfini ou instancier de nouveaux cadres en fournissant un port.

```
from braket.aws import AwsDevice
from braket.pulse import Frame, Port

# Predefined frame from a device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
drive_frame = device.frames["Transmon_5_charge_tx"]

# Create a custom frame
readout_frame = Frame(frame_id="r0_measure", port=Port("channel_0", dt=1e-9),
                      frequency=5e9, phase=0)
```

Ports

Un port est une abstraction logicielle représentant n'importe quel composant input/output matériel contrôlant les qubits. Il aide les fournisseurs de matériel à fournir une interface avec laquelle les utilisateurs peuvent interagir pour manipuler et observer les qubits. Les ports sont caractérisés par une chaîne unique qui représente le nom du connecteur. Cette chaîne expose également un incrément de temps minimum qui indique avec quelle précision nous pouvons définir les formes d'onde.

```
from braket.pulse import Port

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

Formes d'onde

Une forme d'onde est une enveloppe dépendant du temps que nous pouvons utiliser pour émettre des signaux sur un port de sortie ou pour capturer des signaux via un port d'entrée. Vous pouvez spécifier vos formes d'onde directement via une liste de nombres complexes ou en utilisant un modèle de forme d'onde pour générer une liste auprès du fournisseur du matériel.

```
from braket.pulse import ArbitraryWaveform, ConstantWaveform
import numpy as np

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

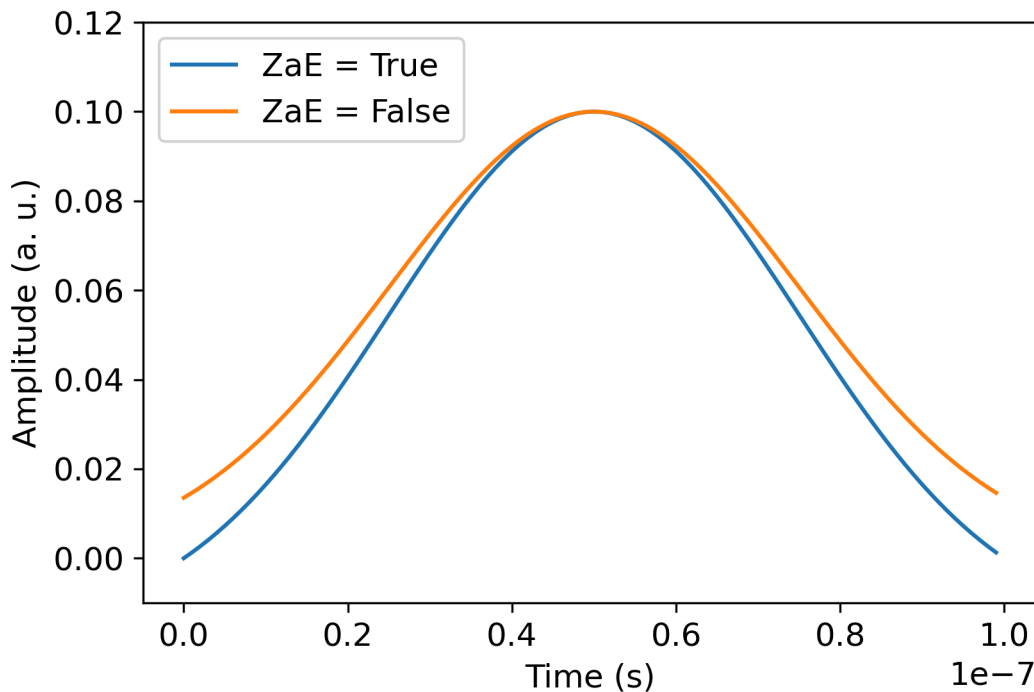
Braket Pulse fournit une bibliothèque standard de formes d'onde, notamment une forme d'onde constante, une forme d'onde gaussienne et une forme d'onde DRAG (Derivative Removal by Adiabatic Gate). Vous pouvez récupérer les données de forme d'onde à l'aide de la `sample` fonction pour dessiner la forme de la forme d'onde, comme indiqué dans l'exemple suivant.

```
from braket.pulse import GaussianWaveform
import numpy as np
import matplotlib.pyplot as plt

zero_at_edge1 = GaussianWaveform(1e-7, 25e-9, 0.1, True)
# or zero_at_edge1 = GaussianWaveform(1e-7, 25e-9, 0.1)
zero_at_edge2 = GaussianWaveform(1e-7, 25e-9, 0.1, False)

times_1 = np.arange(0, zero_at_edge1.length, drive_frame.port.dt)
times_2 = np.arange(0, zero_at_edge2.length, drive_frame.port.dt)

plt.plot(times_1, zero_at_edge1.sample(drive_frame.port.dt))
plt.plot(times_2, zero_at_edge2.sample(drive_frame.port.dt))
```

L'image précédente représente les formes d'onde gaussiennes créées à partir de.

`GaussianWaveform` Nous avons choisi une durée d'impulsion de 100 ns, une largeur de 25 ns et une amplitude de 0,1 (unités arbitraires). Les formes d'onde sont centrées dans la fenêtre d'impulsion. `GaussianWaveform` accepte un argument booléen `zero_at_edges` (ZaE dans la légende). Lorsqu'il est défini sur `True`, cet argument décale la forme d'onde gaussienne de telle sorte que les points à $t=0$ et $t=length$ soient à zéro et redimensionne son amplitude de telle sorte que la valeur maximale corresponde à l'argument. `amplitude`

Travailler avec Hello Pulse

Dans cette section, vous apprendrez à caractériser et à construire une porte à qubit unique directement en utilisant l'impulsion d'un Rigetti appareil. L'application d'un champ électromagnétique à un qubit entraîne une oscillation de Rabi, faisant passer le qubit entre son état 0 et son état 1. Avec une longueur et une phase d'impulsion calibrées, l'oscillation de Rabi peut calculer une seule porte qubit. Ici, nous allons déterminer la longueur d'impulsion optimale pour mesurer une impulsion $\pi/2$, un bloc élémentaire utilisé pour construire des séquences d'impulsions plus complexes.

Tout d'abord, pour créer une séquence d'impulsions, importez la `PulseSequence` classe.

```
from braket.aws import AwsDevice
```

```
from braket.circuits import FreeParameter
from braket.devices import Devices
from braket.pulse import PulseSequence, GaussianWaveform

import numpy as np
```

Ensuite, instanciez un nouveau périphérique Braket à l'aide de l'Amazon Resource NameARN du QPU. Le bloc de code suivant utilise Rigetti Ankaa-3.

```
device = AwsDevice(Devices.Rigetti.Ankaa3)
```

La séquence d'impulsions suivante comprend deux composantes : jouer une forme d'onde et mesurer un qubit. La séquence d'impulsions peut généralement être appliquée aux trames. À quelques exceptions près, telles que la barrière et le délai, qui peuvent être appliqués aux qubits. Avant de construire la séquence d'impulsions, vous devez récupérer les trames disponibles. Le cadre d'entraînement est utilisé pour appliquer l'impulsion nécessaire à l'oscillation de Rabi, et le cadre de lecture sert à mesurer l'état du qubit. Cet exemple utilise les images du qubit 25.

```
drive_frame = device.frames["Transmon_25_charge_tx"]
readout_frame = device.frames["Transmon_25_readout_rx"]
```

Créez maintenant la forme d'onde qui sera jouée dans le cadre du lecteur. L'objectif est de caractériser le comportement des qubits pour différentes durées d'impulsion. Vous allez jouer une forme d'onde avec des longueurs différentes à chaque fois. Au lieu d'instancier une nouvelle forme d'onde à chaque fois, utilisez la séquence d'impulsions supportée par le frein. `FreeParameter` Vous pouvez créer la forme d'onde et la séquence d'impulsions une seule fois avec des paramètres libres, puis exécuter la même séquence d'impulsions avec des valeurs d'entrée différentes.

```
waveform = GaussianWaveform(FreeParameter("length"), FreeParameter("length") * 0.25,
                             0.2, False)
```

Enfin, assemblez-les sous forme de séquence d'impulsions. Dans la séquence d'impulsions, `play` lit la forme d'onde spécifiée sur le cadre du lecteur et `capture_v0` mesure l'état à partir du cadre de lecture.

```
pulse_sequence = (
    PulseSequence()
    .play(drive_frame, waveform)
```

```
.capture_v0(readout_frame)
)
```

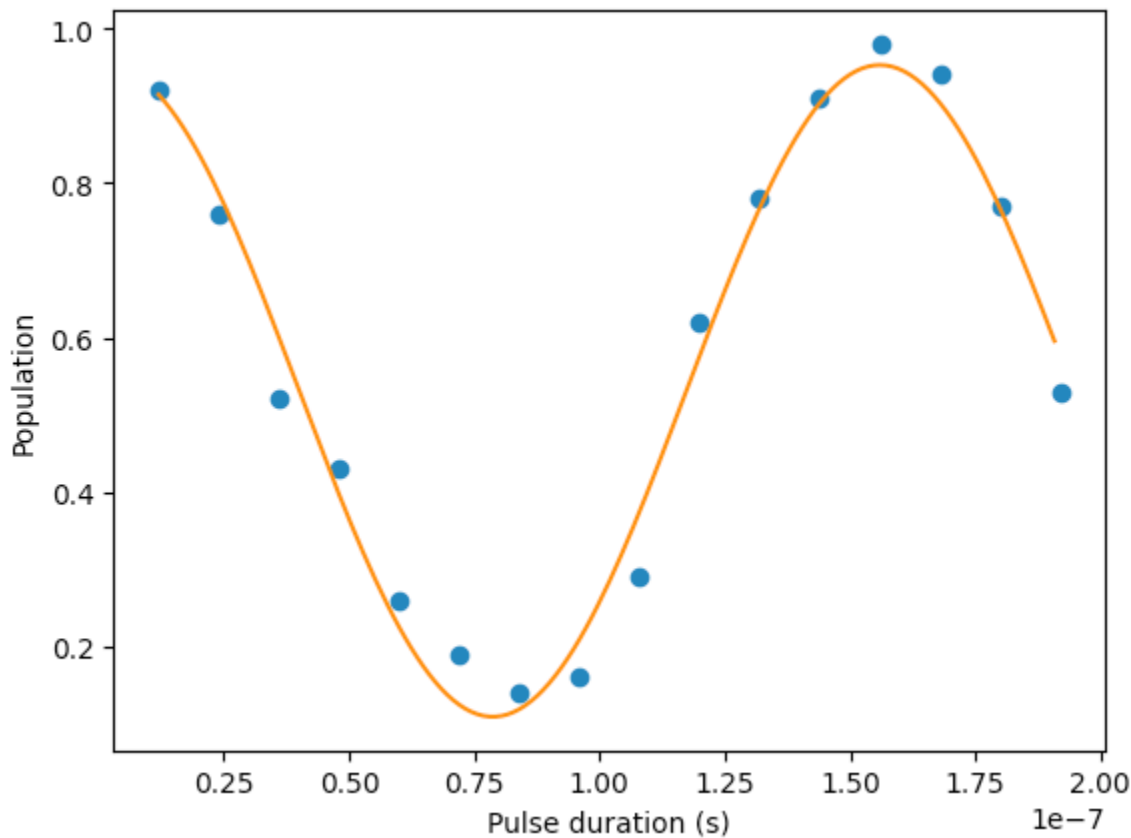
Scannez une plage de durées d'impulsion et soumettez-les au QPU. Avant d'exécuter les séquences d'impulsions sur un QPU, liez la valeur des paramètres libres.

```
start_length = 12e-9
end_length = 2e-7
lengths = np.arange(start_length, end_length, 12e-9)
N_shots = 100

tasks = [
    device.run(pulse_sequence(length=length), shots=N_shots)
    for length in lengths
]

probability_of_zero = [
    task.result().measurement_counts['0']/N_shots
    for task in tasks
]
```

Les statistiques de la mesure du qubit montrent la dynamique oscillatoire du qubit qui oscille entre l'état 0 et l'état 1. À partir des données de mesure, vous pouvez extraire la fréquence Rabi et affiner la durée de l'impulsion pour implémenter une porte à 1 qubit particulière. Par exemple, d'après les données de la figure ci-dessous, la périodicité est d'environ 154 ns. Ainsi, une porte de rotation $\pi/2$ correspondrait à la séquence d'impulsions d'une longueur = 38,5 ns.



Hello Pulse utilise OpenPulse

[OpenPulse](#) est un langage permettant de spécifier le contrôle au niveau des impulsions d'un dispositif quantique général et fait partie de la spécification OpenQASM 3.0. Amazon Braket prend en charge la programmation directe OpenPulse d'impulsions à l'aide de la représentation OpenQASM 3.0.

Braket utilise OpenPulse comme représentation intermédiaire sous-jacente pour exprimer des impulsions dans des instructions natives. OpenPulse permet d'ajouter des étalonnages d'instructions sous la forme de déclarations `defcal` (abréviation de « définir l'étalonnage »). Avec ces déclarations, vous pouvez spécifier une implémentation d'une instruction de porte dans une grammaire de contrôle de niveau inférieur.

Vous pouvez visualiser le OpenPulse programme d'un Braket à `PulseSequence` l'aide de la commande suivante.

```
print(pulse_sequence.to_ir())
```

Vous pouvez également créer un OpenPulse programme directement.

```
from braket.ir.openqasm import Program

openpulse_script = """
OPENQASM 3.0;
cal {
    bit[1] psb;
    waveform my_waveform = gaussian(12.0ns, 3.0ns, 0.2, false);
    play(Transmon_25_charge_tx, my_waveform);
    psb[0] = capture_v0(Transmon_25_readout_rx);
}
"""
```

Créez un Program objet à l'aide de votre script. Soumettez ensuite le programme à un QPU.

```
from braket.aws import AwsDevice
from braket.devices import Devices
from braket.ir.openqasm import Program

program = Program(source=openpulse_script)

device = AwsDevice(Devices.Rigetti.Ankaa3)
task = device.run(program, shots=100)
```

Accès aux portes natives à l'aide d'impulsions

Les chercheurs ont souvent besoin de savoir exactement comment les portes natives prises en charge par un QPU particulier sont implémentées sous forme d'impulsions. Les séquences d'impulsions sont soigneusement calibrées par les fournisseurs de matériel, mais l'accès à celles-ci permet aux chercheurs de concevoir de meilleures portes ou d'explorer des protocoles pour atténuer les erreurs, tels que l'extrapolation sans bruit en étirant les impulsions de portes spécifiques.

Amazon Braket prend en charge l'accès programmatique aux portails natifs de Rigetti.

```
import math
from braket.aws import AwsDevice
from braket.circuits import Circuit, GateCalibrations, QubitSet
from braket.circuits.gates import Rx

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

calibrations = device.gate_calibrations
```

```
print(f"Downloaded {len(calibrations)} calibrations.")
```

Note

Les fournisseurs de matériel étalonnent régulièrement le QPU, souvent plus d'une fois par jour. Le SDK Braket vous permet d'obtenir les derniers étalonnages de portail.

```
device.refresh_gate_calibrations()
```

Pour récupérer une porte native donnée, telle que la porte RX ou XY, vous devez transmettre l'Gateobjet et les qubits qui vous intéressent. Par exemple, vous pouvez inspecter l'implémentation par impulsion du RX ($\pi/2$) appliqué à qubit 0.

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

pulse_sequence_rx_pi_2_q0 = calibrations.pulse_sequences[rx_pi_2_q0]
```

Vous pouvez créer un ensemble filtré d'étalonnages à l'aide de cette `filter` fonction. Vous passez une liste de portes ou une liste de `QubitSet`. Le code suivant crée deux ensembles contenant tous les étalonnages pour RX ($\pi/2$) et pour 0. qubit

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

Vous pouvez désormais fournir ou modifier l'action des portes natives en joignant un ensemble de calibrage personnalisé. Par exemple, considérez le circuit suivant.

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

Vous pouvez l'exécuter avec un calibrage de porte personnalisé pour la rx porte qubit 0 en transmettant un dictionnaire d'`PulseSequence`objets à l'argument du `gate_definitions`

mot clé. Vous pouvez créer un dictionnaire à partir de l'attribut `pulse_sequences` de l'`GateCalibrations` objet. Toutes les portes non spécifiées sont remplacées par l'étalonnage des impulsions du fournisseur de matériel quantique.

```
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)
```

Simulation hamiltonienne analogique

La [simulation hamiltonienne analogique](#) (AHS) est un nouveau paradigme de l'informatique quantique qui diffère considérablement du modèle de circuit quantique traditionnel. Au lieu d'une séquence de portes, où chaque circuit n'agit que sur quelques qubits à la fois. Un programme AHS est défini par les paramètres dépendants du temps et de l'espace de l'hamiltonien en question. L'[hamiltonien d'un système](#) code ses niveaux d'énergie et les effets des forces extérieures, qui, ensemble, régissent l'évolution temporelle de ses états. Pour un système à N qubits, l'hamiltonien peut être représenté par une matrice carrée de $2^N \times 2^N$ de nombres complexes.

Les dispositifs quantiques capables d'exécuter l'AHS sont conçus pour se rapprocher étroitement de l'évolution temporelle d'un système quantique selon un hamiltonien personnalisé en ajustant soigneusement leurs paramètres de contrôle interne. Par exemple, le réglage de l'amplitude et les paramètres de désaccordage d'un champ de conduite cohérent. Le paradigme AHS est bien adapté pour simuler les propriétés statiques et dynamiques de systèmes quantiques comportant de nombreuses particules en interaction, comme dans le cas de la physique de la matière condensée ou de la chimie quantique. Les unités de traitement quantique spécialement conçues (QPUs), comme le [dispositif Aquila](#) de QuEra, ont été développées pour utiliser la puissance de l'AHS et résoudre des problèmes hors de portée des approches informatiques quantiques numériques conventionnelles de manière innovante.

Dans cette section :

- [Bonjour AHS : Exécutez votre première simulation hamiltonienne analogique](#)
- [Soumettre un programme analogique à l'aide d' QuEra Aquila](#)

Bonjour AHS : Exécutez votre première simulation hamiltonienne analogique

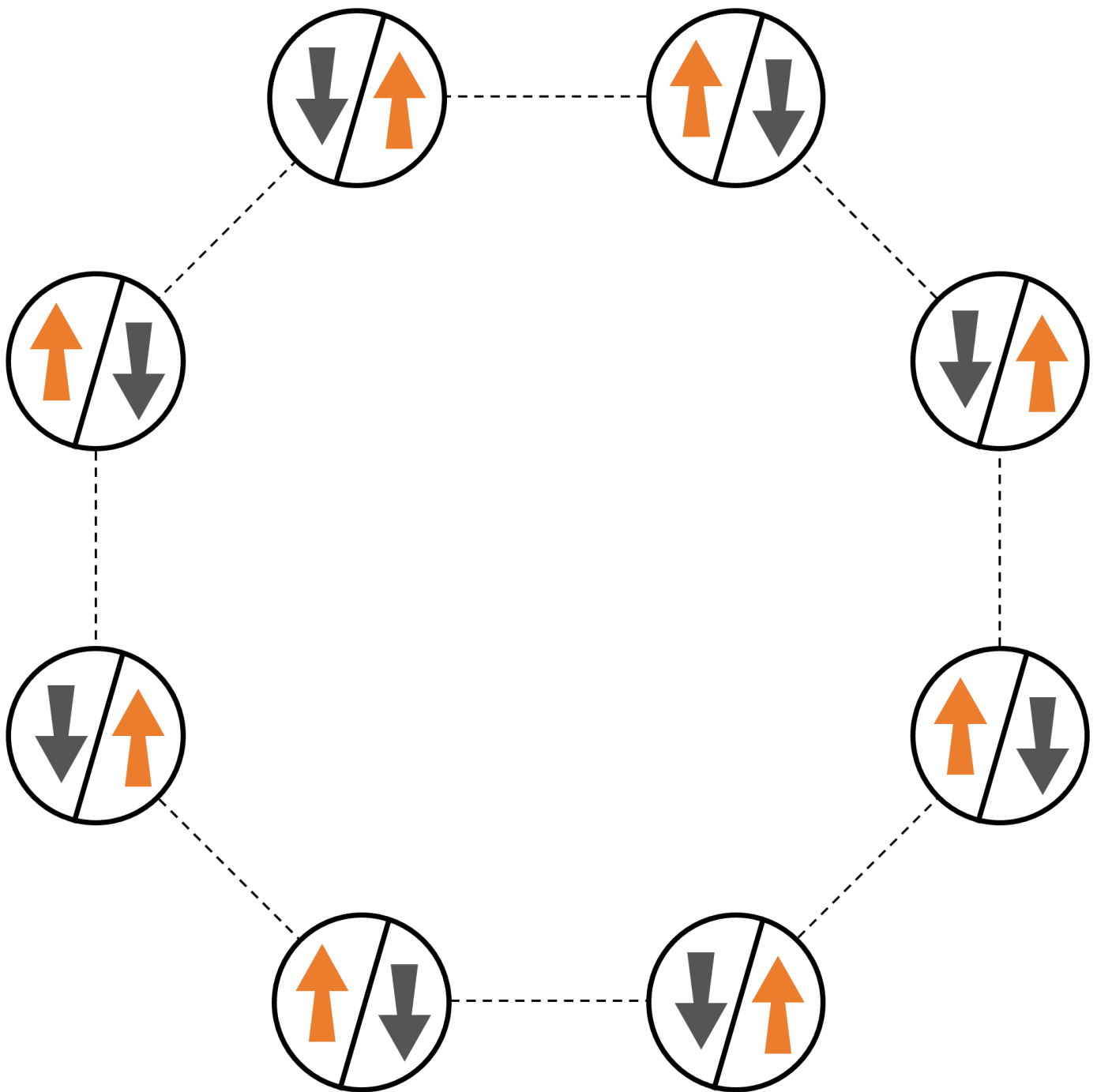
Cette section fournit des informations sur l'exécution de votre première simulation hamiltonienne analogique.

Dans cette section :

- [Chaîne de spin en interaction](#)
- [Arrangement](#)
- [Interaction](#)
- [Champ de conduite](#)
- [Programme AHS](#)
- [Exécution sur un simulateur local](#)
- [Analyse des résultats du simulateur](#)
- [Running QuEra On's Aquila QPU](#)
- [Analyse des résultats du QPU](#)
- [Étapes suivantes](#)

Chaîne de spin en interaction

Pour un exemple canonique d'un système de nombreuses particules en interaction, considérons un anneau de huit spins (dont chacun peut être dans les états « haut » $\uparrow$ et « bas » $\downarrow$). Bien que petit, ce système modèle présente déjà une poignée de phénomènes intéressants liés aux matériaux magnétiques naturels. Dans cet exemple, nous allons montrer comment préparer un ordre dit antiferromagnétique, dans lequel des spins consécutifs pointent dans des directions opposées.



Arrangement

Nous utiliserons un atome neutre pour représenter chaque spin, et les états de spin « haut » et « bas » seront codés dans l'état de Rydberg excité et dans l'état fondamental des atomes, respectivement. Tout d'abord, nous créons l'arrangement 2D. Nous pouvons programmer l'anneau de tours ci-dessus avec le code suivant.

Prérequis : Vous devez installer le SDK [Braket](#) en un clin d'œil. (Si vous utilisez une instance de bloc-notes hébergée par Braket, ce SDK est préinstallé avec les blocs-notes.) Pour reproduire les tracés, vous devez également installer matplotlib séparément à l'aide de la commande shell. `pip install matplotlib`

```
from braket.ahs.atom_arrangement import AtomArrangement
import numpy as np
import matplotlib.pyplot as plt # Required for plotting

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

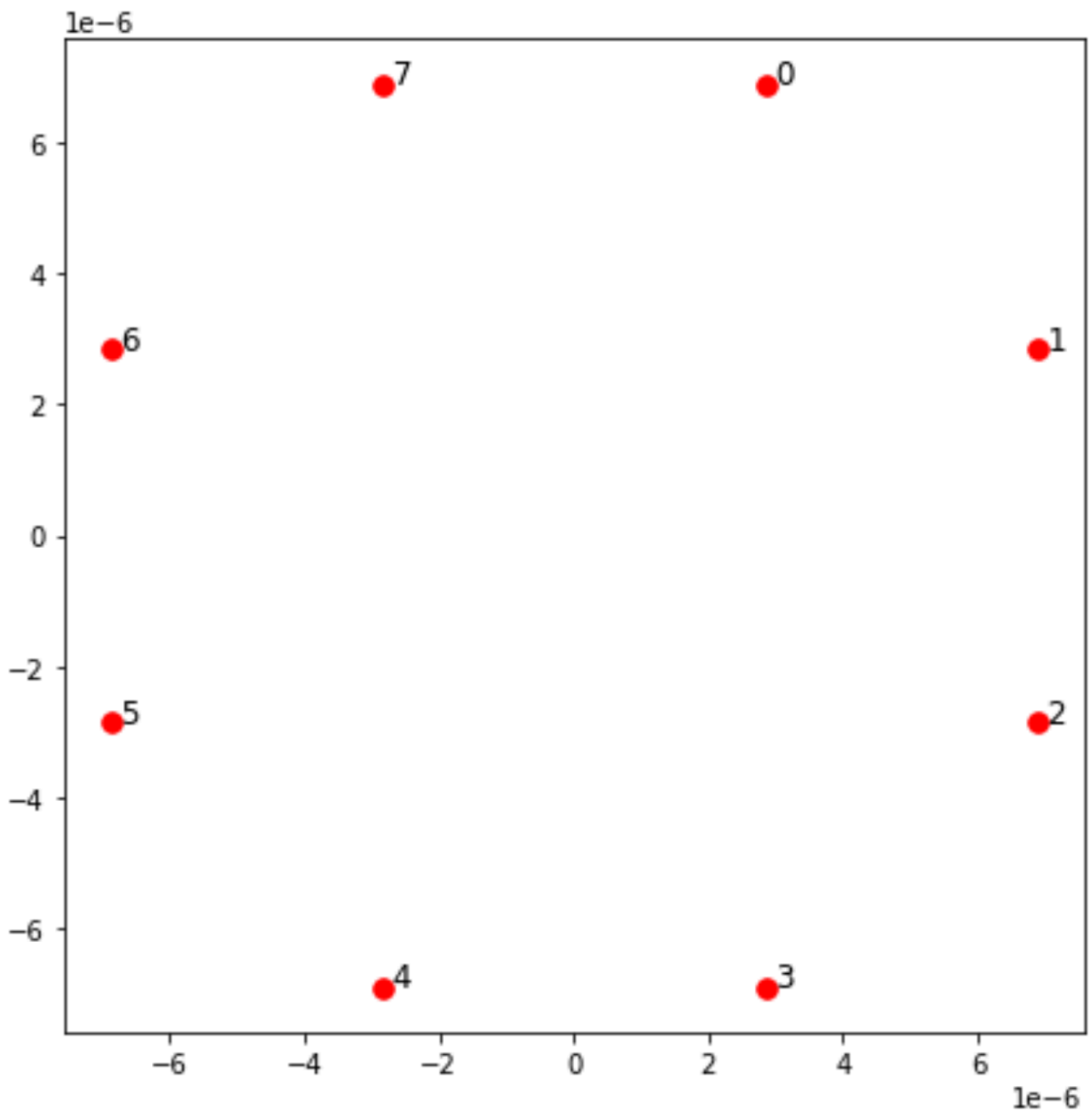
register = AtomArrangement()
register.add(np.array([0.5, 0.5 + 1/np.sqrt(2)]) * a)
register.add(np.array([0.5 + 1/np.sqrt(2), 0.5]) * a)
register.add(np.array([0.5 + 1/np.sqrt(2), - 0.5]) * a)
register.add(np.array([0.5, - 0.5 - 1/np.sqrt(2)]) * a)
register.add(np.array([-0.5, - 0.5 - 1/np.sqrt(2)])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

avec lequel nous pouvons également compléter

```
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
```



Interaction

Pour préparer la phase antiferromagnétique, nous devons induire des interactions entre des spins voisins. Pour cela, nous utilisons l'[interaction de van der Waals](#), qui est implémentée nativement par des appareils à atomes neutres (tels que le Aquila périphérique deQuEra). En utilisant la

représentation du spin, le terme hamiltonien pour cette interaction peut être exprimé sous la forme d'une somme de toutes les paires de spins (j, k).

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

Ici, $n_j = \uparrow_j \cdot \uparrow_j$ est $\uparrow_j$ un opérateur qui prend la valeur 1 uniquement si le spin j est à l'état « up », et 0 dans le cas contraire. La force est $V_{j,k} = C_6 / (d_{j,k})^6$, où C_6 est le coefficient fixe, et $d_{j,k}$ est la distance euclidienne entre les spins j et k. L'effet immédiat de ce terme d'interaction est que tout état dans lequel le spin j et le spin k sont à la fois « en hausse » produit une énergie élevée (de la quantité $V_{j,k}$). En concevant soigneusement le reste du programme AHS, cette interaction empêchera les spins voisins d'être tous deux à l'état « actif », un effet communément appelé « blocus de Rydberg ».

Champ de conduite

Au début du programme AHS, tous les spins (par défaut) démarrent dans leur état « down », ils sont dans une phase dite ferromagnétique. En gardant un œil sur notre objectif de préparer la phase antiferromagnétique, nous avons spécifié un champ de commande cohérent dépendant du temps qui fait passer en douceur les spins de cet état à un état à plusieurs corps où les états « haut » sont préférés. L'hamiltonien correspondant peut être écrit comme

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

où $\Omega(t)$, $\phi(t)$, $\Delta(t)$ sont l'amplitude globale (ou [fréquence de Rabi](#)), la phase et le désaccord du champ moteur en fonction du temps, affectant tous les spins de manière uniforme. Ici, $S_{-,k} = \downarrow_k \cdot \uparrow_k$ et $S_{+,k} = (S_{-,k})^\dagger = \uparrow_k \cdot \downarrow_k$ sont respectivement les opérateurs d'abaissement et d'augmentation du spin k, et $n_k = \uparrow_k \cdot \uparrow_k$. La partie Ω du champ moteur couple de manière cohérente les états « bas » et « haut » de tous les spins simultanément, tandis que la partie Δ contrôle la récompense énergétique pour les états « haut ».

Pour programmer une transition en douceur de la phase ferromagnétique à la phase antiferromagnétique, nous indiquons le champ de commande avec le code suivant.

```
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
)

```

Nous pouvons visualiser les séries chronologiques du champ de conduite avec le script suivant.

```

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]')

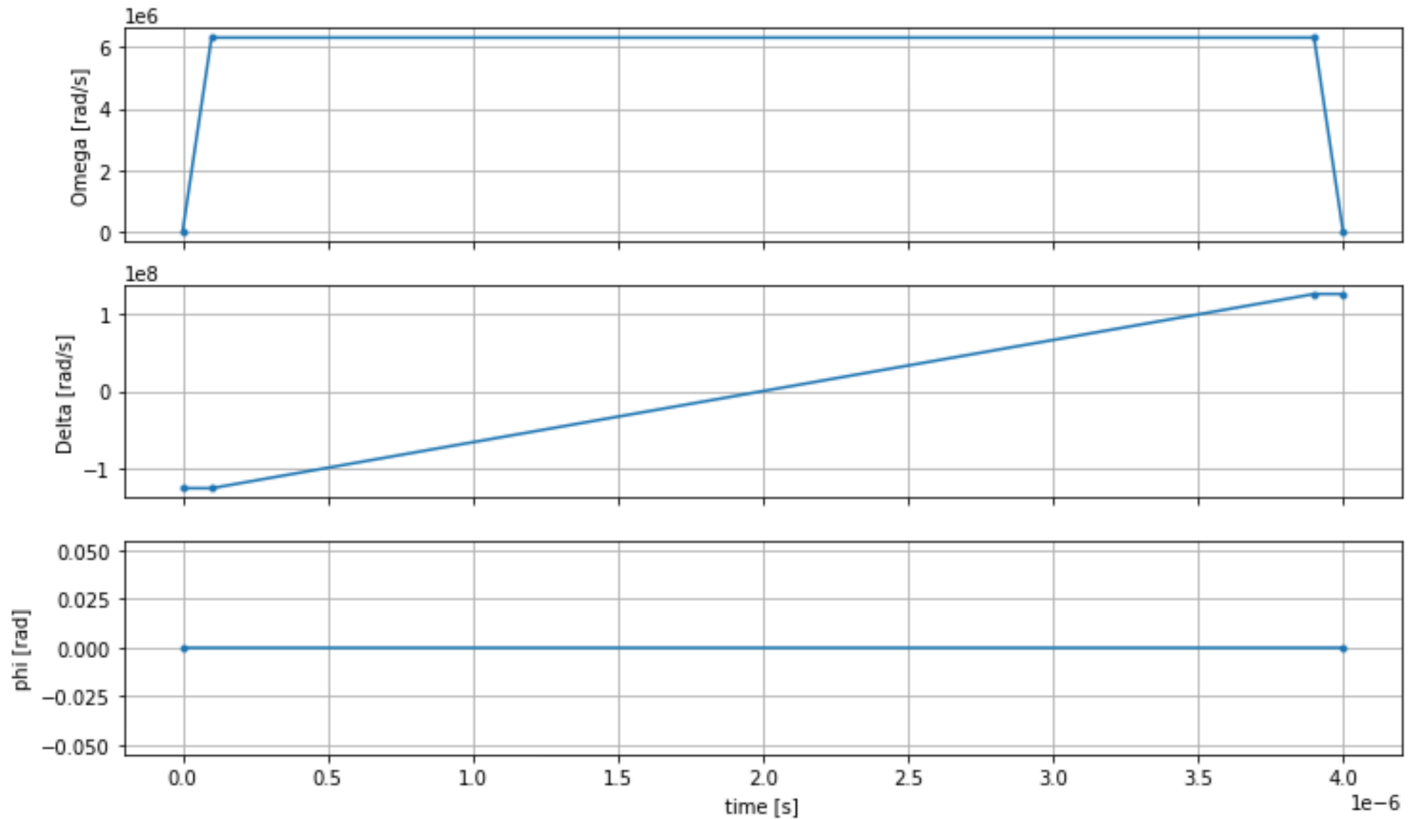
ax = axes[1]
time_series = drive.detuning.time_series
ax.plot(time_series.times(), time_series.values(), '-.')
ax.grid()
ax.set_ylabel('Delta [rad/s]')

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
```



Programme AHS

Le registre, le champ de conduite (et les interactions implicites de van der Waals) constituent le programme de simulation hamiltonienne analogique. `ahs_program`

```
from braket.ahs.analog_hamiltonian_simulation import AnalogHamiltonianSimulation

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

Exécution sur un simulateur local

Comme cet exemple est petit (moins de 15 tours), avant de l'exécuter sur un QPU compatible AHS, nous pouvons l'exécuter sur le simulateur AHS local fourni avec le SDK Braket. Le simulateur local étant disponible gratuitement avec le SDK Braket, il s'agit d'une bonne pratique pour garantir que notre code puisse s'exécuter correctement.

Ici, nous pouvons régler le nombre de prises à une valeur élevée (disons, 1 million) car le simulateur local suit l'évolution temporelle de l'état quantique et prélève des échantillons à partir de l'état final, augmentant ainsi le nombre de prises de vue, tout en n'augmentant que légèrement le temps d'exécution total.

```
from braket.devices import LocalSimulator

device = LocalSimulator("braket_ahs")

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

Analyse des résultats du simulateur

Nous pouvons agréger les résultats des tirs à l'aide de la fonction suivante qui déduit l'état de chaque rotation (qui peut être « d » pour « vers le bas », « u » pour « haut » ou « e » pour un site vide) et compte le nombre de fois où chaque configuration s'est produite sur les plans.

```
from collections import Counter

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

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

    Args:
        result
    (braket.tasks.analog_hamiltonian_simulation_quantum_task_result.AnalogHamiltonianSimulationQuantumTaskResult)"""
```

Returns

dict: number of times each state configuration is measured

"""

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

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

[Output]

```
{'ddddddddd': 5, 'dddddddu': 12, 'ddddddud': 15, ...}
```

countsVoici un dictionnaire qui compte le nombre de fois que chaque configuration d'état est observée sur les plans. Nous pouvons également les visualiser avec le code suivant.

```
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 = []
```



```

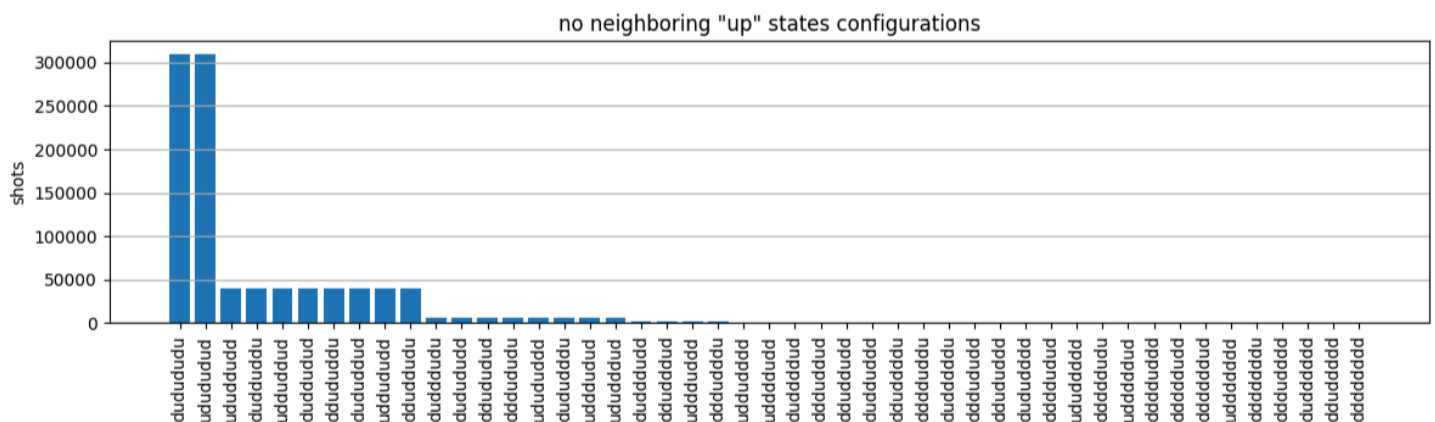
for state, count in counts.items():
    if not has_neighboring_up_states(state):
        collection = non_blockaded
    else:
        collection = blocked
    collection.append((state, count, number_of_up_states(state)))

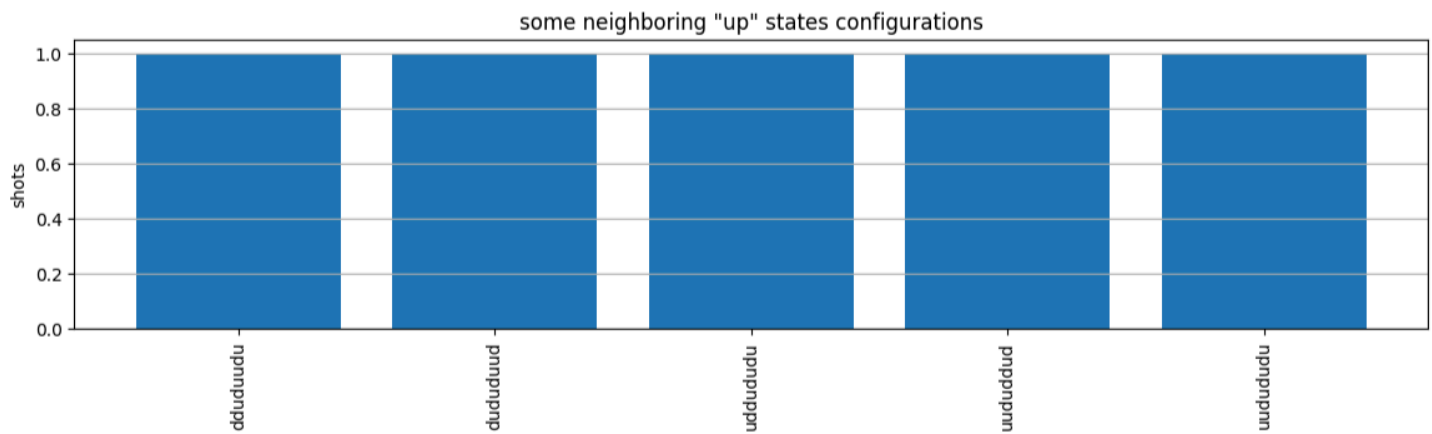
blockaded.sort(key=lambda _: _[1], reverse=True)
non_blockaded.sort(key=lambda _: _[1], reverse=True)

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)
```





À partir des graphiques, nous pouvons lire les observations suivantes pour vérifier que nous avons bien préparé la phase antiferromagnétique.

1. En général, les états non bloqués (où aucun spin voisin n'est dans l'état « haut ») sont plus courants que les états où au moins une paire de spins voisins sont tous deux dans l'état « haut ».
2. En général, les états avec plus d'excitations « ascendantes » sont privilégiés, sauf si la configuration est bloquée.
3. Les états les plus courants sont en effet les états "dudududu" antiferromagnétiques parfaits et "udududud".
4. Les deuxièmes états les plus courants sont ceux où il n'y a que 3 excitations « ascendantes » avec des séparations consécutives de 1, 2, 2. Cela montre que l'interaction de van der Waals a également un effet (bien que beaucoup plus faible) sur les voisins les plus proches.

Running QuEra On's Aquila QPU

Conditions préalables : [Outre l'installation du SDK Braket par pip, si vous utilisez Amazon Braket pour la première fois, assurez-vous d'avoir suivi les étapes de démarrage nécessaires.](#)

Note

Si vous utilisez une instance de bloc-notes hébergée par Braket, le SDK Braket est préinstallé avec l'instance.

Une fois toutes les dépendances installées, nous pouvons nous connecter au Aquila QPU.

```
from braket.aws import AwsDevice
```

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

Pour que notre programme AHS soit adapté à la QuEra machine, nous devons arrondir toutes les valeurs afin de respecter les niveaux de précision autorisés par le Aquila QPU. (Ces exigences sont régies par les paramètres de l'appareil dont le nom contient « Résolution ». Nous pouvons les voir en les exécutant `aquila_qpu.properties.dict()` dans un carnet. Pour plus de détails sur les fonctionnalités et les exigences d'Aquila, consultez le bloc-notes [Introduction à Aquila](#).) Nous pouvons le faire en appelant la `discretize` méthode.

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

Nous pouvons maintenant exécuter le programme (seulement 100 prises de vue pour le moment) sur le Aquila QPU.

Note

L'exécution de ce programme sur le Aquila processeur entraînera un coût. Le SDK Amazon Braket inclut un outil de [suivi des coûts](#) qui permet aux clients de définir des limites de coûts et de suivre leurs coûts en temps quasi réel.

```
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}")
```

[Output]

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

En raison de la grande variation de la durée d'exécution d'une tâche quantique (en fonction des fenêtres de disponibilité et de l'utilisation du QPU), il est conseillé de noter l'ARN de la tâche

quantique, afin que nous puissions vérifier son statut ultérieurement à l'aide de l'extrait de code suivant.

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

Une fois le statut TERMINÉ (qui peut également être vérifié depuis la page des tâches quantiques de la [console](#) Amazon Braket), nous pouvons interroger les résultats avec :

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

Analyse des résultats du QPU

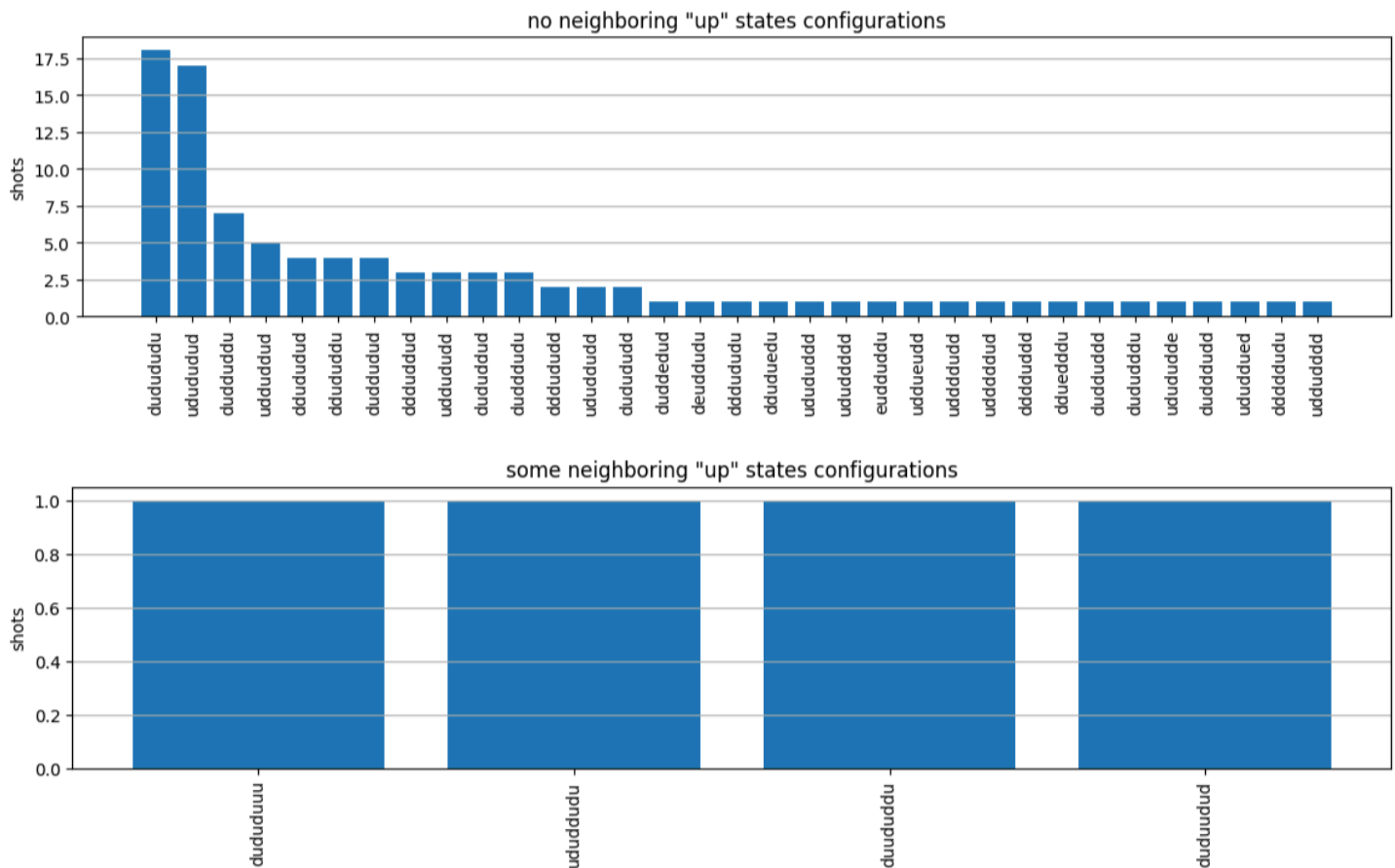
En utilisant les mêmes `get_counts` fonctions que précédemment, nous pouvons calculer les nombres :

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

```
*[Output]*
{'dddududd': 2, 'dudududu': 18, 'ddududud': 4, ...}
```

et tracez-les avec `plot_counts` :

```
plot_counts(counts_aquila)
```



Notez qu'une petite partie des prises de vue comportent des sites vides (marqués d'un « e »). Cela est dû à une imperfection de préparation de 1 à 2 % par atome du Aquila QPU. En outre, les résultats correspondent à la simulation dans les limites des fluctuations statistiques attendues en raison du faible nombre de prises de vue.

Étapes suivantes

Félicitations, vous venez d'exécuter votre première charge de travail AHS sur Amazon Braket à l'aide du simulateur AHS local et du Aquila QPU.

[Pour en savoir plus sur la physique de Rydberg, la simulation hamiltonienne analogique et le Aquila dispositif, consultez nos exemples de carnets de notes.](#)

Soumettre un programme analogique à l'aide d' QuEra Aquila

Cette page fournit une documentation complète sur les fonctionnalités de la Aquila machine à partir de QuEra. Les détails abordés ici sont les suivants :

1. L'hamiltonien paramétré simulé par Aquila
2. Paramètres du programme AHS
3. Contenu du résultat AHS
4. Aquila paramètre de capacités

Dans cette section :

- [hamiltonien](#)
- [Schéma du programme Braket AHS](#)
- [Schéma des résultats des tâches Braket AHS](#)
- [QuEra schéma des propriétés de l'appareil](#)

hamiltonien

La Aquila machine de QuEra simule nativement l'hamiltonien suivant (en fonction du temps) :

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

Note

L'accès au désaccordage local est une [fonctionnalité expérimentale](#) et est disponible sur demande via Braket Direct.

où

- $H_{\text{drive},k}(t) = (\frac{1}{2}\Omega(t) e^{i\varphi(t)} S_{-,k} + \frac{1}{2}\Omega(t) e^{-i\varphi(t)} S_{+,k}) + (-\Delta_{\text{global}}(t, k) n)_{,k}$
- $\Omega(t)$ est l'amplitude de conduite globale en fonction du temps (également appelée fréquence de Rabi), en unités de (rad/s)
- $\varphi(t)$ est la phase globale dépendante du temps, mesurée en radians

- $S_{-,k}$ et $S_{+,k}$ sont les opérateurs d'abaissement et d'augmentation du spin de l'atome k (dans la base $|\downarrow\rangle = |g\rangle$, $|\uparrow\rangle = |r\rangle$, ils sont $S_{-} = |g\rangle\langle r|$ et $S_{+} = |r\rangle\langle g|$)
- $\Delta_{\text{global}}(t)$ est le déréglage global dépendant du temps
- n_k est l'opérateur de projection sur l'état de Rydberg de l'atome k (c'est-à-dire, $n_k = |r\rangle\langle r|$)
- $H_{\text{local detuning},k}(t) = -\Delta_{\text{local}}(t) h_k n_k$
 - $\Delta_{\text{local}}(t)$ est le facteur dépendant du temps du décalage de fréquence local, en unités de (rad/s)
 - h_k est le facteur dépendant du site, un nombre sans dimension compris entre 0,0 et 1,0
- $V_{\text{vdw},k,l} = C_6/(d_{k,l})^6 n_k n_l$
 - C_6 est le coefficient de van der Waals, en unités de (rad/s) * (m) ^6
 - $d_{k,l}$ est la distance euclidienne entre l'atome k et l , mesurée en mètres.

Les utilisateurs peuvent contrôler les paramètres suivants via le schéma du programme Braket AHS.

- Arrangement bidimensionnel des atomes (x_k coordonnées x_k et y_k de chaque atome k , en unités de μm), qui contrôle les distances atomiques par paires $d_{k,l}$ avec $k, l=1,2,\dots, N$
- $\Omega(t)$, fréquence globale de Rabi dépendante du temps, en unités de (rad/s)
- $\varphi(t)$, phase globale dépendante du temps, en unités de (rad)
- $\Delta_{\text{global}}(t)$, le désaccordage global dépendant du temps, en unités de (rad/s)
- $\Delta_{\text{local}}(t)$, le facteur (global) dépendant du temps de l'ampleur du déréglage local, en unités de (rad/s)
- h_k , le facteur (statique) dépendant du site de l'ampleur du déréglage local, un nombre adimensionnel compris entre 0,0 et 1,0

Note

L'utilisateur ne peut pas contrôler les niveaux concernés (c'est-à-dire que les opérateurs S_{-} , S_{+} , n sont fixes) ni la force du coefficient d'interaction Rydberg-Rydberg (C_6).

Schéma du programme Braket AHS

Objet Braket.IR.AHS.Program_V1.program (exemple)

Note

Si la fonctionnalité de [désaccordage local](#) n'est pas activée pour votre compte, utilisez-la `localDetuning=[]` dans l'exemple suivant.

```
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('0.000003')]
                    ),
                    pattern='uniform'
                ),
                detuning=PhysicalField(
                    time_series=TimeSeries(
```



```

        values=[Decimal('-54000000.0'), Decimal('54000000.0')],
        times=[Decimal('0'), Decimal('0.000003')]
    ),
    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 (exemple)

Note

Si la fonctionnalité de [désaccordage local](#) n'est pas activée pour votre compte, utilisez-la "localDetuning": [] dans l'exemple suivant.

```

{
  "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, 0.000003000]
                },
                "pattern": "uniform"
            },
            "detuning": {
                "time_series": {
                    "values": [-54000000.0, 54000000.0],
                    "times": [0E-9, 0.000003000]
                },
                "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]
            }
        }
    ]
}
}

```

Principaux domaines

Champ du programme	type	description
setup.ahs_register.sites	Liste [Liste [Décimal]]	Liste des coordonnées bidimensionnelles où la pince à épiler piège les atomes
setup.ahs_register.filling	Liste [int]	Marque les atomes qui occupent les sites piégés avec 1 et les sites vides avec 0
Hamiltonian.DrivingFields [] .amplitude.time_series.times	Liste [décimale]	points temporels de l'amplitude de conduite, Omega (t)
Hamiltonian.DrivingFields [] .amplitude.time_series.values	Liste [décimale]	valeurs de l'amplitude de de conduite, Omega (t)
Hamiltonian.DrivingFields [] .amplitude.pattern	str	schéma spatial de l'amplitude de conduite, Omega (t) ; doit être « uniforme »
Hamiltonian.DrivingFields [] .phase.time_series.times	Liste [décimale]	points temporels de la phase de conduite, phi (t)
Hamiltonian.DrivingFields [] .phase.time_series.values	Liste [décimale]	valeurs de la phase de conduite, phi (t)

Champ du programme	type	description
Hamiltonian.DrivingFields [] .phase.pattern	str	schéma spatial de la phase de conduite, $\phi(t)$; doit être « uniforme »
Hamiltonian.DrivingFields [] .detuning.time_series.times	Liste [décimale]	points temporels du désaccordage de conduite, $\Delta_{\text{global}}(t)$
Hamiltonian.DrivingFields [] .detuning.time_series.values	Liste [décimale]	valeurs du désaccordage de conduite, $\Delta_{\text{global}}(t)$
Hamiltonian.DrivingFields [] .detuning.pattern	str	schéma spatial de désaccordage de conduite, $\Delta_{\text{global}}(t)$; doit être « uniforme »
Hamiltonian.LocalDeTuning [] .magnitude.time_series.times	Liste [décimale]	points temporels du facteur dépendant du temps de l'amplitude de désaccordage locale, $\Delta_{\text{local}}(t)$
Hamiltonian.LocalDeTuning [] .magnitude.time_series.values	Liste [décimale]	valeurs du facteur dépendant du temps de l'amplitude de dérégulation locale, $\Delta_{\text{local}}(t)$

Champ du programme	type	description
Hamiltonian.LocalDeTuning [] .magnitude.pattern	Liste [décimale]	facteur dépendant du site de l'amplitude de du dérèglement local, h_k (les valeurs correspondent aux sites dans setup.ahs_register .sites)

Champs de métadonnées

Champ du programme	type	description
braketSchemaHeader.nom	str	nom du schéma ; doit être « braket.ir .ahs.program »
braketSchemaHeader.version	str	version du schéma

Schéma des résultats des tâches Braket AHS

braket.tasks.analog_hamiltonian_simulation_quantum_task_result.

AnalogHamiltonianSimulationQuantumTaskResult(exemple)

```

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 (exemple)

```

{
  "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
    }
}
}

```

Principaux domaines

Champ de résultat de la tâche	type	description
mesures [].ShotResult.PreSequence	Liste [int]	Bits de mesure de pré-séquence (un pour chaque site atomique) pour chaque tir : 0 si le site est vide, 1 si le site est plein, mesurés avant les séquences d'impulsions qui exécutent l'évolution quantique
mesures [].ShotResult.PostSequence	Liste [int]	Bits de mesure post-séquence pour chaque tir : 0 si l'atome est dans l'état de Rydberg ou si le site est vide, 1 si l'atome est dans

Champ de résultat de la tâche	type	description
		l'état fondamental, mesurés à la fin des séquences d'impulsions qui exécutent l'évolution quantique

Champs de métadonnées

Champ de résultat de la tâche	type	description
braketSchemaHeader.nom	str	nom du schéma ; doit être « braket.task_result.analog_hamiltonian_simulation_task_result »
braketSchemaHeader.version	str	version du schéma
Métadonnées des tâches. braketSchemaHeader.nom	str	nom du schéma ; doit être « braket.task_metadata »
Métadonnées des tâches. braketSchemaHeader.version	str	version du schéma
TaskMetadata.id	str	L'identifiant de la tâche quantique . Pour les

Champ de résultat de la tâche	type	description
		tâches AWS quantiques, il s'agit de l'ARN des tâches quantiques.
TaskMetadata.Shots	int	Le nombre de tirs pour la tâche quantique
TaskMetadata.Shots.DeviceID	str	L'identifiant de l'appareil sur lequel la tâche quantique s'est exécutée. Pour les AWS appareils, il s'agit de l'ARN de l'appareil.
TaskMetadata.Shots.Créé à	str	Horodatage de la création ; le format doit être au format RFC3339 ISO-8601/string format:MM:SS.SSSZ.YYYY-MM-DDTHH La valeur par défaut est None.

Champ de résultat de la tâche	type	description
TaskMetadata.shots.Endedat	str	Horodatage de la fin de la tâche quantique ; le format doit être au format RFC3339 ISO-8601/string format:MM:SS.SSSZ.YYYY-MM-DDTHH La valeur par défaut est None.
TaskMetadata.Shots.Status	str	État de la tâche quantique (CREATED, QUEUED, RUNNING, COMPLETED, FAILED). La valeur par défaut est None.

Champ de résultat de la tâche	type	description
TaskMetadata.Shots.Raison de l'échec	str	La raison de l'échec de la tâche quantique . La valeur par défaut est None.
Métadonnées.Action supplémentaires	Braket.IR.AHS.Program_V1.program	(Voir la section sur le schéma du programme Braket AHS)
Métadonnées supplémentaires. Action. braketSchemaHeader.querametadata.name	str	nom du schéma ; doit être « braket.task_result .quera_metadata »
Métadonnées supplémentaires. Action. braketSchemaHeader.Quera Metadata. Version	str	version du schéma
Métadonnées supplémentaires. Action. numSuccessfulShots	int	nombre de prises complètement réussies ; doit être égal au nombre de prises demandé

Champ de résultat de la tâche	type	description
mesures [].shotMetadata.shotStatus	int	Le statut du tir (succès, succès partiel, échec) doit être « Succès »

QuEra schéma des propriétés de l'appareil

braket.device_schema.quera.quera_device_capabilities_v1. QueraDeviceCapabilities(exemple)

```
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 (exemple)

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

Champs des propriétés du service

Champ des propriétés du service	type	description
Service.ExecutionWindows [].ExecutionDay	ExecutionDay	Jours de la fenêtre d'exécution ; doivent être

Champ des propriétés du service	type	description
		« Tous les jours », « Jours de semaine », « Week-end », « lundi », « mardi », « mercredi », « jeudi », « vendredi », « samedi » ou « dimanche »
Service.ExecutionWindows []. windowStartHour	datetime.heure	Format UTC 24 heures de l'heure de début de la fenêtre d'exécution
Service.ExecutionWindows []. windowEndHour	datetime.heure	Format UTC de 24 heures indiquant l'heure de fin de la fenêtre d'exécution
service.qpu_capabilities.service.shotsRange	Tuple [int, int]	Nombre minimum et maximum de prises de vue pour l'appareil
service.qpu_capabilities.service.DeviceCost.Price	float	Prix de l'appareil en dollars américains
service.qpu_capabilities.service.DeviceCost.unit	str	unité pour facturer le prix, par exemple : « minute », « heure », « shot », « task »

Champs de métadonnées

Champ de métadonnées	type	description
action [].version	str	version du schéma du programme AHS
action [].ActionType	ActionType	Nom du schéma du programme AHS ;

Champ de métadonnées	type	description
		doit être « braket.ir .ahs.program »
service. braketSchemaHeader.nom	str	nom du schéma ; doit être « braket.de vice_schema.device _service_properties »
service. braketSchemaHeader.version	str	version du schéma
Service.DeviceDocumentation.URL de l'image	str	URL de l'image de l'appareil
Service.DeviceDocumentation.Résumé	str	brève description de l'appareil
Service.DeviceDocumentation. externalDocumentat ionUrl	str	URL de documenta tion externe
Service.Emplacement de l'appareil	str	emplacement géographique de l'appareil
Service. Mis à jour à	datetime	heure à laquelle les propriétés de l'appareil ont été mises à jour pour la dernière fois

Travailler avec AWS Boto3

Boto3 est le AWS SDK pour Python. Avec Boto3, les développeurs Python peuvent créer, configurer et gérer Services AWS, comme Amazon Braket. Boto3 fournit un accès orienté objet API ainsi qu'un accès de bas niveau à Amazon Braket.

Suivez les instructions du [guide de démarrage rapide du Boto3](#) pour savoir comment installer et configurer Boto3.

Boto3 fournit les fonctionnalités de base qui fonctionnent avec le SDK Amazon Braket Python pour vous aider à configurer et à exécuter vos tâches quantiques. Les clients de Python doivent toujours installer Boto3, car il s'agit de l'implémentation principale. Si vous souhaitez utiliser des méthodes d'assistance supplémentaires, vous devez également installer le SDK Amazon Braket.

Par exemple, lorsque vous appelez `CreateQuantumTask`, le SDK Amazon Braket envoie la demande à Boto3, qui appelle ensuite le AWS API

Dans cette section :

- [Activez le client Amazon Braket Boto3](#)
- [Configuration AWS CLI des profils pour Boto3 et le SDK Braket](#)

Activez le client Amazon Braket Boto3

Pour utiliser Boto3 avec Amazon Braket, vous devez importer Boto3, puis définir un client que vous utiliserez pour vous connecter à Amazon Braket. API Dans l'exemple suivant, le client Boto3 est nommé `braket`

```
import boto3
import botocore

braket = boto3.client("braket")
```

Note

[Supports de support IPv6](#). Si vous utilisez un réseau IPv6 uniquement ou si vous souhaitez vous assurer que votre charge de travail utilise IPv6 du trafic, utilisez les points de terminaison à double pile, comme indiqué dans le guide des points de terminaison [à double pile et FIPS](#).

Maintenant que vous avez un `braket` client établi, vous pouvez faire des demandes et traiter les réponses depuis le service Amazon Braket. Vous pouvez obtenir plus de détails sur les données de demande et de réponse dans la [référence de l'API](#).

Les exemples suivants montrent comment travailler avec des appareils et des tâches quantiques.

- [Rechercher des appareils](#)

- [Récupérez un appareil](#)
- [Créez une tâche quantique](#)
- [Récupérez une tâche quantique](#)
- [Recherche de tâches quantiques](#)
- [Annuler une tâche quantique](#)

Rechercher des appareils

- `search_devices(**kwargs)`

Recherchez des appareils à l'aide des filtres spécifiés.

```
# 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'])
```

Récupérez un appareil

- `get_device(deviceArn)`

Récupérez les appareils disponibles sur Amazon Braket.

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

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

Créez une tâche quantique

- `create_quantum_task(**kwargs)`

Créez une tâche quantique.

```
# 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")
```

Récupérez une tâche quantique

- `get_quantum_task(quantumTaskArn)`

Récupérez la tâche quantique spécifiée.

```
# 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'])
```

Recherche de tâches quantiques

- `search_quantum_tasks(**kwargs)`

Recherchez les tâches quantiques qui correspondent aux valeurs de filtre spécifiées.

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

Annuler une tâche quantique

- `cancel_quantum_task(quantumTaskArn)`

Annulez la tâche quantique spécifiée.

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

Configuration AWS CLI des profils pour Boto3 et le SDK Braket

Le SDK Amazon Braket repose sur les informations d'AWS CLI d'identification par défaut, sauf indication contraire explicite de votre part. Nous vous recommandons de conserver la valeur par

défaut lorsque vous exécutez sur un bloc-notes Amazon Braket géré, car vous devez fournir un rôle IAM autorisé à lancer l'instance de bloc-notes.

Facultativement, si vous exécutez votre code localement (sur une instance Amazon EC2, par exemple), vous pouvez établir des profils nommés AWS CLI . Vous pouvez attribuer à chaque profil un ensemble d'autorisations différent, plutôt que de remplacer régulièrement le profil par défaut.

Cette section explique brièvement comment configurer une telle CLI `profile` et comment intégrer ce profil dans Amazon Braket afin que les API appels soient effectués avec les autorisations de ce profil.

Dans cette section :

- [Étape 1 : Configuration d'une AWS CLI locale `profile`](#)
- [Étape 2 : établir un objet de session Boto3](#)
- [Étape 3 : Incorporez la session Boto3 dans le Braket `AwsSession`](#)

Étape 1 : Configuration d'une AWS CLI locale **profile**

Il n'entre pas dans le cadre de ce document d'expliquer comment créer un utilisateur et comment configurer un profil autre que celui par défaut. Pour plus d'informations sur ces sujets, voir :

- [Premiers pas](#)
- [Configuration du AWS CLI à utiliser AWS IAM Identity Center](#)

Pour utiliser Amazon Braket, vous devez fournir à cet utilisateur (et à la CLI associée `profile`) les autorisations Braket nécessaires. Par exemple, vous pouvez joindre la `AmazonBraketFullAccess` politique.

Étape 2 : établir un objet de session Boto3

Pour établir un objet de session Boto3, utilisez l'exemple de code suivant.

```
from boto3 import Session

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

 Note

Si les API appels attendus comportent des restrictions basées sur la région qui ne sont pas alignées sur votre région profile par défaut, vous pouvez spécifier une région pour la session Boto3 comme indiqué dans l'exemple suivant.

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

Pour l'argument désigné commeregion, remplacez une valeur qui correspond à l'une des valeurs Régions AWS dans lesquelles Amazon Braket est disponible us-east-1us-west-1, par exemple, etc.

Étape 3 : Incorporez la session Boto3 dans le Braket AwsSession

L'exemple suivant montre comment initialiser une session Boto3 Braket et instancier un appareil dans cette session.

```
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)
```

Une fois cette configuration terminée, vous pouvez soumettre des tâches quantiques à cet AwsDevice objet instancié (en appelant la `device.run(...)` commande par exemple). Tous les API appels effectués par cet appareil peuvent utiliser les informations d'identification IAM associées au profil CLI que vous avez précédemment désignéprofile.

Testez vos tâches quantiques avec Amazon Braket

Amazon Braket propose une variété de simulateurs de circuits quantiques à hautes performances pour vous aider à tester et à valider vos algorithmes quantiques avant de les exécuter sur du matériel quantique réel. Ces simulateurs gèrent le logiciel et l'infrastructure sous-jacents complexes, ainsi que les clusters Amazon Elastic Compute Cloud (Amazon EC2) pour alléger le fardeau de la simulation de circuits quantiques sur une infrastructure de calcul haute performance (HPC) classique. Ces ressources vous permettent de vous concentrer sur le développement et l'optimisation de vos applications quantiques.

Avec les simulateurs de Braket, vous pouvez tester de manière approfondie vos circuits et algorithmes quantiques sans les contraintes et les limites des dispositifs quantiques physiques. Cela vous permet d'explorer un large éventail de concepts informatiques quantiques, des portes et circuits quantiques de base aux algorithmes quantiques plus avancés et aux techniques d'atténuation des erreurs.

Le SDK Braket simplifie la soumission de vos tâches quantiques aux simulateurs, vous permettant de contrôler les paramètres de simulation, tels que le nombre de prises de vue et le modèle de bruit, afin de mieux comprendre le comportement de vos algorithmes quantiques. Vous pouvez également utiliser les fonctionnalités d'Amazon Braket Hybrid Job pour combiner des éléments informatiques classiques et quantiques, élargissant ainsi la portée de vos tests et de vos validations.

En testant minutieusement vos tâches quantiques sur les simulateurs de Braket, vous pouvez obtenir des informations précieuses, affiner vos algorithmes et vous assurer de leur exactitude avant de les déployer sur du matériel quantique réel. Cela permet de réduire le temps de développement, de minimiser les erreurs et, en fin de compte, d'accélérer vos progrès dans le domaine de l'informatique quantique.

Dans cette section :

- [Soumission de tâches quantiques à des simulateurs](#)
- [Émulateur de dispositif quantique local](#)

Soumission de tâches quantiques à des simulateurs

Amazon Braket donne accès à plusieurs simulateurs qui peuvent tester vos tâches quantiques. Vous pouvez soumettre des tâches quantiques individuellement ou [exécuter plusieurs programmes](#).

Simulateurs

- Simulateur de matrice de densité, DM1 : `arn:aws:braket:::device/quantum-simulator/amazon/dm1`
- Simulateur de vecteurs d'état, SV1 : `arn:aws:braket:::device/quantum-simulator/amazon/sv1`
- Simulateur de réseau Tensor, TN1 : `arn:aws:braket:::device/quantum-simulator/amazon/tn1`
- Le simulateur local : `LocalSimulator()`

Note

Vous pouvez annuler des tâches quantiques telles que celles pour CREATED les simulateurs QPUs ou à la demande. Vous pouvez annuler les tâches quantiques dans l'QUEUED état dans la mesure du possible pour les simulateurs à la demande et. QPUs Notez qu'il est peu probable que les tâches QUEUED quantiques QPU soient annulées avec succès pendant les fenêtres de disponibilité des QPU.

Dans cette section :

- [Simulateur vectoriel d'état local \(braket_sv\)](#)
- [Simulateur de matrice de densité locale \(braket_dm\)](#)
- [Simulateur AHS local \(braket_ahs\)](#)
- [Simulateur de vecteurs d'état \(SV1\)](#)
- [Simulateur de matrice de densité \(DM1\)](#)
- [Simulateur de réseau Tensor \(\) TN1](#)
- [À propos des simulateurs embarqués](#)
- [Comparez les simulateurs Amazon Braket](#)
- [Exemples de tâches quantiques sur Amazon Braket](#)
- [Tester une tâche quantique avec le simulateur local](#)

Simulateur vectoriel d'état local (**braket_sv**)

Le simulateur vectoriel d'état local (`braket_sv`) fait partie du SDK Amazon Braket qui s'exécute localement dans votre environnement. Il convient parfaitement au prototypage rapide sur de petits circuits (jusqu'à 25 qubits) en fonction des spécifications matérielles de votre instance d'ordinateur portable Braket ou de votre environnement local.

Le simulateur local prend en charge toutes les portes du SDK Amazon Braket, mais les appareils QPU en prennent en charge un sous-ensemble plus restreint. Vous pouvez trouver les portes compatibles d'un appareil dans les propriétés de l'appareil.

Note

Le simulateur local prend en charge les fonctionnalités avancées d'OpenQASM qui peuvent ne pas être prises en charge sur les appareils QPU ou d'autres simulateurs. Pour plus d'informations sur les fonctionnalités prises en charge, consultez les exemples fournis dans le bloc-notes [OpenQASM Local Simulator](#).

Pour plus d'informations sur l'utilisation des simulateurs, consultez [les exemples d'Amazon Braket](#).

Simulateur de matrice de densité locale (**braket_dm**)

Le simulateur de matrice de densité locale (`braket_dm`) fait partie du SDK Amazon Braket qui s'exécute localement dans votre environnement. Il convient parfaitement au prototypage rapide sur de petits circuits avec du bruit (jusqu'à 12 qubits) en fonction des spécifications matérielles de votre instance d'ordinateur portable Braket ou de votre environnement local.

Vous pouvez créer des circuits bruyants courants à partir de zéro en utilisant des opérations de bruit de grille telles que le retournement de bits et l'erreur de dépolarisation. Vous pouvez également appliquer des opérations antibruit à des portes spécifiques qubits et à des portes de circuits existants conçus pour fonctionner à la fois avec ou sans bruit.

Le simulateur `braket_dm` local peut fournir les résultats suivants, compte tenu du nombre spécifié de shots :

- Matrice de densité réduite : Shots = 0

Note

Le simulateur local prend en charge les fonctionnalités avancées d'OpenQASM, qui peuvent ne pas être prises en charge sur les appareils QPU ou d'autres simulateurs. Pour plus d'informations sur les fonctionnalités prises en charge, consultez les exemples fournis dans le bloc-notes [OpenQASM Local Simulator](#).

Pour en savoir plus sur le simulateur de matrice de densité locale, consultez [l'exemple de simulateur de bruit d'introduction de Braket](#).

Simulateur AHS local (**braket_ahs**)

Le simulateur AHS (Analog Hamiltonian Simulation) local (**braket_ahs**) fait partie du SDK Amazon Braket qui s'exécute localement dans votre environnement. Il peut être utilisé pour simuler les résultats d'un programme AHS. Il convient parfaitement au prototypage sur de petits registres (jusqu'à 10 à 12 atomes) en fonction des spécifications matérielles de votre instance d'ordinateur portable Braket ou de votre environnement local.

Le simulateur local prend en charge les programmes AHS avec un champ de conduite uniforme, un champ variable (non uniforme) et des arrangements atomiques arbitraires. Pour plus de détails, reportez-vous à la [classe Braket AHS](#) et au schéma du [programme Braket AHS](#).

Pour en savoir plus sur le simulateur AHS local, consultez la page [Hello AHS : Exécutez votre première simulation hamiltonienne analogique et les carnets d'exemples de simulation hamiltonienne analogique](#).

Simulateur de vecteurs d'état (SV1)

SV1 est un simulateur vectoriel d'état universel, performant et à la demande. Il peut simuler des circuits allant jusqu'à 34 qubits. Vous pouvez vous attendre à ce qu'un 34-qubit circuit dense et carré (profondeur du circuit = 34) prenne environ 1 à 2 heures, selon le type de portes utilisées et d'autres facteurs. Les circuits avec all-to-all portes sont bien adaptés pour SV1. Il renvoie les résultats sous des formes telles qu'un vecteur d'état complet ou un tableau d'amplitudes.

SV1 a une autonomie maximale de 6 heures. Il comporte par défaut 35 tâches quantiques simultanées et un maximum de 100 (50 dans us-west-1 et eu-west-2) tâches quantiques simultanées.

SV1 résultats

SV1 peut fournir les résultats suivants, compte tenu du nombre spécifié de shots :

- Échantillon : Shots > 0
- Espérance : Shots >= 0
- Écart : Shots >= 0
- Probabilité : Shots > 0
- Amplitude : Shots = 0
- Gradient adjoint : Shots = 0

Pour en savoir plus sur les résultats, consultez la section [Types de résultats](#).

SV1 est toujours disponible, il fait fonctionner vos circuits à la demande et peut faire fonctionner plusieurs circuits en parallèle. Le temps d'exécution évolue de manière linéaire avec le nombre d'opérations et de manière exponentielle avec le nombre de qubits. Le nombre de shots a un faible impact sur le temps d'exécution. Pour en savoir plus, consultez l'article [Comparer les simulateurs](#).

Les simulateurs prennent en charge toutes les portes du SDK Braket, mais les appareils QPU en prennent en charge un sous-ensemble plus restreint. Vous pouvez trouver les portes compatibles d'un appareil dans les propriétés de l'appareil.

Simulateur de matrice de densité (DM1)

DM1 est un simulateur de matrice de densité à haute performance à la demande. Il peut simuler des circuits allant jusqu'à 17 qubits.

DM1 a une durée d'exécution maximale de 6 heures, une valeur par défaut de 35 tâches quantiques simultanées et un maximum de 50 tâches quantiques simultanées.

DM1 résultats

DM1 peut fournir les résultats suivants, compte tenu du nombre spécifié de shots :

- Échantillon : Shots > 0
- Espérance : Shots >= 0
- Écart : Shots >= 0
- Probabilité : Shots > 0

- Matrice de densité réduite : Shots = 0, jusqu'à 8 au maximum qubits

Pour plus d'informations sur les résultats, consultez la section [Types de résultats](#).

DM1 est toujours disponible, il fait fonctionner vos circuits à la demande et peut faire fonctionner plusieurs circuits en parallèle. Le temps d'exécution évolue de manière linéaire avec le nombre d'opérations et de manière exponentielle avec le nombre de qubits. Le nombre de shots a un faible impact sur le temps d'exécution. Pour en savoir plus, consultez la section [Comparaison de simulateurs](#).

Barrières antibruit et limites

```
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]
```

Simulateur de réseau Tensor () TN1

TN1 est un simulateur de réseau tensoriel performant et à la demande. TN1 peut simuler certains types de circuits avec un maximum de 50 qubits et une profondeur de circuit inférieure ou égale à 1 000. TN1 est particulièrement puissant pour les circuits clairsemés, les circuits dotés de portes locales et les autres circuits dotés d'une structure spéciale, tels que les circuits à transformée de Fourier quantique (QFT). TN1 fonctionne en deux phases. Tout d'abord, la phase de répétition tente

d'identifier un chemin de calcul efficace pour votre circuit, TN1 afin d'estimer le temps d'exécution de l'étape suivante, appelée phase de contraction. Si le temps de contraction estimé dépasse la limite d'exécution de la TN1 simulation, TN1 ne tente pas de contraction.

TN1 a une durée d'exécution maximale de 6 heures. Il est limité à un maximum de 10 (5 dans eu-west-2) tâches quantiques simultanées.

TN1 résultats

La phase de contraction consiste en une série de multiplications matricielles. La série de multiplications continue jusqu'à ce qu'un résultat soit atteint ou jusqu'à ce qu'il soit déterminé qu'un résultat ne peut pas être atteint.

Remarque : Shots doit être > 0 .

Les types de résultats incluent :

- Exemple
- Espérance
- Variance

Pour en savoir plus sur les résultats, consultez la section [Types de résultats](#).

TN1 est toujours disponible, il fait fonctionner vos circuits à la demande et peut faire fonctionner plusieurs circuits en parallèle. Pour en savoir plus, consultez la section [Comparaison de simulateurs](#).

Les simulateurs prennent en charge toutes les portes du SDK Braket, mais les appareils QPU en prennent en charge un sous-ensemble plus restreint. Vous pouvez trouver les portes compatibles d'un appareil dans les propriétés de l'appareil.

Consultez le GitHub référentiel Amazon Braket pour obtenir un [TN1 exemple de bloc-notes](#) qui vous aidera à TN1 démarrer.

Bonnes pratiques pour travailler avec TN1

- Évitez les all-to-all circuits.
- Testez un nouveau circuit ou une nouvelle classe de circuits avec un petit nombre de shots, pour connaître la « dureté » du circuit pour TN1.
- Répartissez de grandes shot simulations sur plusieurs tâches quantiques.

À propos des simulateurs embarqués

Les simulateurs intégrés fonctionnent en incorporant la simulation directement dans le code de l'algorithme. En outre, il est contenu dans le même conteneur et exécute la simulation directement sur l'instance de tâche hybride. Cette approche est utile pour éliminer les goulots d'étranglement généralement associés à la communication entre la simulation et un appareil distant. En conservant tous les calculs dans un environnement unique et cohérent, les simulateurs intégrés peuvent réduire considérablement les besoins en mémoire et le nombre d'exécutions de circuits nécessaires pour atteindre un résultat cible. Cela peut entraîner des améliorations de performances substantielles, souvent multipliées par dix ou plus, par rapport aux configurations traditionnelles qui reposent sur la simulation à distance. Pour plus d'informations sur la manière dont les simulateurs intégrés améliorent les performances et permettent de rationaliser les tâches hybrides, consultez la page de [documentation Exécuter une tâche hybride avec Amazon Braket Hybrid Jobs](#).

PennyLane des simulateurs de foudre

Vous pouvez utiliser les simulateurs PennyLane de foudre comme simulateurs intégrés sur Braket. Avec les simulateurs PennyLane de foudre, vous pouvez utiliser des méthodes avancées de calcul des dégradés, telles que la [différenciation adjointe](#), pour évaluer les dégradés plus rapidement. Le [simulateur lightning.qubit](#) est disponible en tant que périphérique via Braket NBIs et en tant que simulateur intégré, tandis que le simulateur lightning.gpu doit être exécuté en tant que simulateur intégré avec une instance de GPU. Consultez les [simulateurs intégrés du bloc-notes Braket Hybrid Jobs](#) pour un exemple d'utilisation de lightning.gpu.

Comparez les simulateurs Amazon Braket

Cette section vous aide à sélectionner le simulateur Amazon Braket le mieux adapté à votre tâche quantique, en décrivant certains concepts, limitations et cas d'utilisation.

Choix entre des simulateurs locaux et des simulateurs à la demande (SV1,,TN1) DM1

Les performances des simulateurs locaux dépendent du matériel qui héberge l'environnement local, tel qu'une instance de bloc-notes Braket, utilisée pour exécuter votre simulateur. Les simulateurs à la demande s'exécutent dans le AWS cloud et sont conçus pour évoluer au-delà des environnements locaux classiques. Les simulateurs à la demande sont optimisés pour les circuits de plus grande taille, mais ajoutent un certain temps de latence par tâche quantique ou par lot de tâches quantiques. Cela peut impliquer un compromis si de nombreuses tâches quantiques sont impliquées. Compte tenu de ces caractéristiques de performance générales, les conseils suivants peuvent vous aider à choisir le mode d'exécution des simulations, y compris celles impliquant du bruit.

Pour les simulations :

- Si vous employez moins de 18 qubits, utilisez un simulateur local.
- Si vous employez entre 18 et 24 qubits, choisissez un simulateur en fonction de la charge de travail.
- Si vous employez plus de 24 qubits, utilisez un simulateur à la demande.

Pour les simulations de bruit :

- Si vous en employez moins de 9 qubits, utilisez un simulateur local.
- Lorsque vous employez 9 à 12 qubits, choisissez un simulateur en fonction de la charge de travail.
- Lorsque vous en employez plus de 12 qubits, utilisez DM1.

Qu'est-ce qu'un simulateur de vecteurs d'état ?

SV1 est un simulateur de vecteur d'état universel. Il mémorise la fonction pleine onde de l'état quantique et applique séquentiellement des opérations de porte à l'état. Il stocke toutes les possibilités, même les plus improbables. La durée de fonctionnement du SV1 simulateur pour une tâche quantique augmente de façon linéaire avec le nombre de portes du circuit.

Qu'est-ce qu'un simulateur de matrice de densité ?

DM1 simule des circuits quantiques avec du bruit. Il stocke la matrice de densité complète du système et applique séquentiellement les portes et les opérations de bruit du circuit. La matrice de densité finale contient des informations complètes sur l'état quantique après le fonctionnement du circuit. Le temps d'exécution évolue généralement de manière linéaire avec le nombre d'opérations et de manière exponentielle avec le nombre de qubits.

Qu'est-ce qu'un simulateur de réseau tensoriel ?

TN1 code des circuits quantiques dans un graphe structuré.

- Les nœuds du graphe sont constitués de portes quantiques, ou qubits.
- Les arêtes du graphique représentent les connexions entre les portes.

Grâce à cette structure, TN1 on peut trouver des solutions simulées pour des circuits quantiques relativement grands et complexes.

TN1 nécessite deux phases

TN1 fonctionne généralement selon une approche en deux phases pour simuler le calcul quantique.

- La phase de répétition : Au cours de cette TN1 phase, trouvez un moyen de parcourir le graphique de manière efficace, ce qui implique de visiter chaque nœud afin d'obtenir la mesure souhaitée. En tant que client, vous ne voyez pas cette phase car elle TN1 exécute les deux phases ensemble pour vous. Il termine la première phase et détermine s'il convient d'exécuter la deuxième phase seule en fonction de contraintes pratiques. Vous n'avez aucune influence sur cette décision une fois que la simulation a commencé.
- La phase de contraction : Cette phase est analogue à la phase d'exécution d'un calcul dans un ordinateur classique. La phase consiste en une série de multiplications matricielles. L'ordre de ces multiplications a une grande influence sur la difficulté du calcul. Par conséquent, la phase de répétition est d'abord effectuée pour trouver les chemins de calcul les plus efficaces sur le graphique. Une fois qu'il a trouvé le chemin de contraction pendant la phase de répétition, il TN1 contracte les portes de votre circuit pour produire les résultats de la simulation.

TN1 les graphes sont analogues à une carte

Métaphoriquement, vous pouvez comparer le TN1 graphique sous-jacent aux rues d'une ville. Dans une ville dotée d'une grille planifiée, il est facile de trouver un itinéraire vers votre destination à l'aide d'une carte. Dans une ville avec des rues non planifiées, des noms de rue dupliqués, etc., il peut être difficile de trouver un itinéraire vers votre destination en consultant une carte.

Si vous TN1 n'avez pas effectué la phase de répétition, ce serait comme marcher dans les rues de la ville pour trouver votre destination, au lieu de regarder d'abord une carte. Il peut être très rentable en termes de temps de marche de passer plus de temps à regarder la carte. De même, la phase de répétition fournit des informations précieuses.

On pourrait dire qu'il TN1 a une certaine « conscience » de la structure du circuit sous-jacent qu'il traverse. Il acquiert cette prise de conscience au cours de la phase de répétition.

Types de problèmes les mieux adaptés à chacun de ces types de simulateurs

SV1 est bien adapté à tous les types de problèmes qui reposent principalement sur la présence d'un certain nombre de portes qubits et. Généralement, le temps requis augmente linéairement avec le nombre de portes, alors qu'il ne dépend pas du nombre de shots. SV1 est généralement plus rapide que TN1 pour les circuits de moins de 28 ansqubits.

SV1 peut être plus lent pour des qubit nombres plus élevés car il simule en fait toutes les possibilités, même les plus improbables. Il n'a aucun moyen de déterminer quels résultats sont probables. Ainsi, pour une 30-qubit évaluation, SV1 il faut calculer 2^{30} configurations. La limite de 34 qubits pour le SV1 simulateur Amazon Braket est une contrainte pratique en raison des limites de mémoire et de stockage. Vous pouvez y penser comme ceci : chaque fois que vous ajoutez un qubit à SV1, le problème devient deux fois plus difficile.

Pour de nombreuses catégories de problèmes, il est TN1 possible d'évaluer des circuits beaucoup plus grands en temps TN1 réel qu'SV1 en tirant parti de la structure du graphe. Il suit essentiellement l'évolution des solutions depuis leur point de départ et ne retient que les configurations qui contribuent à une traversée efficace. Autrement dit, il enregistre les configurations pour créer un ordre de multiplication matricielle qui simplifie le processus d'évaluation.

En TN1 effet, le nombre de portes qubits et est important, mais la structure du graphe est beaucoup plus importante. Par exemple, TN1 est très bon pour évaluer les circuits (graphes) dans lesquels les portes sont à courte portée (c'est-à-dire que chacune qubit est connectée par des portes uniquement à son voisin le plus proche qubits) et les circuits (graphes) dans lesquels les connexions (ou portes) ont une portée similaire. Une fourchette typique TN1 consiste à ce que chacun ne qubit parle qu'à d'autres qubits personnes situées qubits à 5. Si la majeure partie de la structure peut être décomposée en relations plus simples telles que celles-ci, qui peuvent être représentées dans des matrices plus nombreuses, plus petites ou plus uniformes, l'évaluation est TN1 réalisée de manière efficace.

Limites de TN1

TN1 peut être plus lent qu'SV1 en fonction de la complexité structurelle du graphique. Pour certains graphiques, TN1 met fin à la simulation après la phase de répétition et affiche un statut de FAILED, pour l'une des deux raisons suivantes :

- Impossible de trouver un chemin — Si le graphe est trop complexe, il est trop difficile de trouver un bon chemin de traversée et le simulateur abandonne le calcul. TN1 Impossible d'effectuer la contraction. Un message d'erreur similaire à celui-ci peut s'afficher : `No viable contraction path found`.
- La phase de contraction est trop difficile : dans certains graphiques, il est TN1 possible de trouver un chemin de traversée, mais son évaluation est très longue et prend énormément de temps. Dans ce cas, la contraction est si coûteuse qu'elle serait prohibitive et qu'elle se terminerait après la TN1 phase de répétition. Un message d'erreur similaire à celui-ci peut s'afficher : `Predicted runtime based on best contraction path found exceeds TN1 limit`.

 Note

La phase de répétition vous est facturée TN1 même si aucune contraction n'est effectuée et que vous voyez un FAILED statut.

Le temps d'exécution prévu dépend également du shot nombre. Dans le pire des cas, le temps de TN1 contraction dépend linéairement du nombre. shot Le circuit peut être contractable avec moins shots. Par exemple, vous pouvez soumettre une tâche quantique avec 100shots, qui TN1 décide qu'elle n'est pas contractualisable, mais si vous la soumettez à nouveau avec seulement 10, la contraction se poursuit. Dans ce cas, pour obtenir 100 échantillons, vous pouvez soumettre 10 tâches quantiques de 10 shots pour le même circuit et combiner les résultats au final.

À titre de bonne pratique, nous vous recommandons de toujours tester votre circuit ou votre classe de circuit avec quelques-uns shots (par exemple, 10) pour déterminer la dureté de votre circuitTN1, avant de passer à un nombre plus élevé deshots.

 Note

La série de multiplications qui forme la phase de contraction commence par de petites matrices $N \times N$. Par exemple, une 2-qubit porte nécessite une matrice 4×4 . Les matrices intermédiaires nécessaires lors d'une contraction jugée trop difficile sont gigantesques. Un tel calcul nécessiterait des jours. C'est pourquoi Amazon Braket ne tente pas de contractions extrêmement complexes.

Simultanéité

Tous les simulateurs Braket vous permettent d'exécuter plusieurs circuits simultanément. Les limites de simultanéité varient selon le simulateur et la région. Pour plus d'informations sur les limites de simultanéité, consultez la page [Quotas](#).

Exemples de tâches quantiques sur Amazon Braket

Cette section décrit les étapes de l'exécution d'un exemple de tâche quantique, de la sélection de l'appareil à l'affichage du résultat. Comme bonne pratique pour Amazon Braket, nous vous recommandons de commencer par exécuter le circuit sur un simulateur, tel que. SV1

Dans cette section :

- [Spécifiez l'appareil](#)
- [Soumettre un exemple de tâche quantique](#)
- [Soumettre une tâche paramétrée](#)
- [Spécifiez shots](#)
- [Sondage pour les résultats](#)
- [Voir les exemples de résultats](#)

Spécifiez l'appareil

Tout d'abord, sélectionnez et spécifiez l'appareil pour votre tâche quantique. Cet exemple montre comment choisir le simulateur, SV1.

```
from braket.aws import AwsDevice

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

Vous pouvez consulter certaines des propriétés de cet appareil comme suit :

```
print(device.name)
for iter in device.properties.action['braket.ir.jaqcd.program']:
    print(iter)
```

```
SV1
('version', ['1.0', '1.1'])
('actionType', 'braket.ir.jaqcd.program')
('supportedOperations', ['ccnot', 'cnot', 'cphaseshift', 'cphaseshift00',
'cphaseshift01', 'cphaseshift10', 'cswap', 'cy', 'cz', 'ecr', '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)])
('disabledQubitRewiringSupported', None)
```

Soumettre un exemple de tâche quantique

Soumettez un exemple de tâche quantique à exécuter sur le simulateur à la demande.

```
from braket.circuits import Circuit, Observable

# 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-s3-demo-bucket" # 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()
```

La `device.run()` commande crée une tâche quantique via l'`CreateQuantumTaskAPI`. Après une courte période d'initialisation, la tâche quantique est mise en file d'attente jusqu'à ce qu'il soit possible d'exécuter la tâche quantique sur un appareil. Dans ce cas, l'appareil est SV1. Une fois que l'appareil a terminé le calcul, Amazon Braket écrit les résultats sur l'emplacement Amazon S3 spécifié lors de l'appel. L'argument positionnel `s3_location` est obligatoire pour tous les appareils, à l'exception du simulateur local.

Note

La taille de l'action de tâche quantique Braket est limitée à 3 Mo.

Soumettre une tâche paramétrée

Amazon Braket propose des simulateurs locaux et à la demande et prend QPUs également en charge la spécification de valeurs de paramètres libres lors de la soumission des tâches. Vous pouvez le faire en utilisant l'argument `device.run()`, comme indiqué dans l'exemple suivant. `inputs` doit s'agir d'un dictionnaire de paires chaîne-flottante, où les clés sont les noms des paramètres.

La compilation paramétrique peut améliorer les performances d'exécution de circuits paramétriques sur certains circuits. QPUs Lorsque vous soumettez un circuit paramétrique en tant que tâche quantique à un QPU compatible, Braket compilera le circuit une fois et mettra le résultat en cache. Aucune recompilation n'est nécessaire pour les mises à jour ultérieures des paramètres du même circuit, ce qui se traduit par des temps d'exécution plus rapides pour les tâches utilisant le même circuit. Braket utilise automatiquement les données d'étalonnage mises à jour fournies par le fournisseur du matériel lors de la compilation de votre circuit afin de garantir des résultats de la plus haute qualité.

Note

La compilation paramétrique est prise en charge sur tous les modèles supraconducteurs basés sur des portes, QPUs à l'exception des programmes de niveau d'impulsion.

```
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}, shots=100)
```

Spécifiez shots

L'argument `shots` fait référence au nombre de mesures souhaitées. Les simulateurs tels que celui-ci SV1 prennent en charge deux modes de simulation.

- Pour `shots = 0`, le simulateur effectue une simulation exacte, renvoyant les valeurs vraies pour tous les types de résultats. (Non disponible sur TN1.)
- Pour les valeurs non nulles de `shots`, le simulateur prélève des échantillons à partir de la distribution de sortie pour émuler le bruit réel. Les appareils QPU autorisent uniquement `shots > 0`.

Pour plus d'informations sur le nombre maximum de tirs par tâche quantique, reportez-vous à [Braket Quotas](#).

Sondage pour les résultats

Lors de l'exécution `my_task.result()`, le SDK commence à rechercher un résultat avec les paramètres que vous définissez lors de la création de la tâche quantique :

- `poll_timeout_second` est le nombre de secondes nécessaires pour interroger la tâche quantique avant son expiration lors de l'exécution de la tâche quantique sur le simulateur à la demande et/ou sur des appareils QPU. La valeur par défaut est 432 000 secondes, soit 5 jours.
- Remarque : pour les appareils QPU tels que Rigetti et IonQ, nous vous recommandons d'attendre quelques jours. Si le délai d'attente de votre sondage est trop court, les résultats risquent de ne pas être renvoyés dans le délai imparti. Par exemple, lorsqu'un QPU n'est pas disponible, une erreur de temporisation locale est renvoyée.
- `poll_interval_second` est la fréquence à laquelle la tâche quantique est interrogée. Il indique à quelle fréquence vous appelez le Braket API pour obtenir le statut lorsque la tâche quantique est exécutée sur le simulateur à la demande et sur les appareils QPU. La valeur par défaut est de 1 seconde.

Cette exécution asynchrone facilite l'interaction avec les périphériques QPU qui ne sont pas toujours disponibles. Par exemple, un appareil peut être indisponible pendant une période de maintenance normale.

Le résultat renvoyé contient une série de métadonnées associées à la tâche quantique. Vous pouvez vérifier le résultat de la mesure à l'aide des commandes suivantes :

```
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]
 [0 0 0]
 ...
 [0 0 0]
 [0 0 0]
 [1 0 1]]
```

Counts for collapsed states:

```
Counter({'000': 766, '101': 220, '010': 11, '111': 3})
```

Probabilities for collapsed states:

```
{'101': 0.22, '000': 0.766, '010': 0.011, '111': 0.003}
```

Voir les exemples de résultats

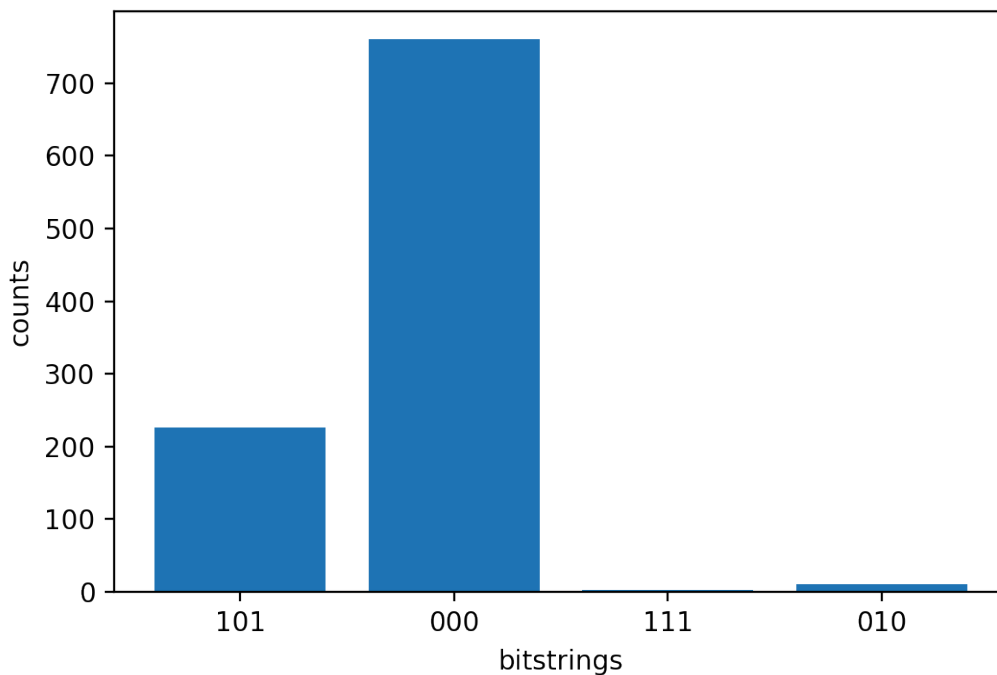
Comme vous avez également spécifié `leResultType`, vous pouvez consulter les résultats renvoyés. Les types de résultats apparaissent dans l'ordre dans lequel ils ont été ajoutés au circuit.

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

# 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=Variance(observable=['z'], targets=[0], type=<Type.variance:
'variance'>), value=0.693084), ResultTypeValue(type=Probability(targets=[0, 2],
type=<Type.probability: 'probability'>), value=array([0.777, 0.    , 0.    , 0.223]))]
Variance= 0.693084
Probability= [0.777 0.    0.    0.223]
Text(0, 0.5, 'counts')
```

Tester une tâche quantique avec le simulateur local

Vous pouvez envoyer des tâches quantiques directement à un simulateur local pour un prototypage et des tests rapides. Ce simulateur s'exécute dans votre environnement local, vous n'avez donc pas besoin de spécifier un emplacement Amazon S3. Les résultats sont calculés directement dans votre session. Pour exécuter une tâche quantique sur le simulateur local, vous devez uniquement spécifier le shots paramètre.

Note

La vitesse d'exécution et qubits le nombre maximum que le simulateur local peut traiter dépendent du type d'instance du bloc-notes Amazon Braket ou de vos spécifications matérielles locales.

Les commandes suivantes sont toutes identiques et instancient le simulateur local à vecteur d'état (sans bruit).

```
# 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")
```

Exécutez ensuite une tâche quantique avec les instructions suivantes.

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

Pour instancier le simulateur de matrice de densité locale (bruit), les clients modifient le backend comme suit.

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

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

Mesurer des qubits spécifiques sur le simulateur local

Le simulateur de vecteur d'état local et le simulateur de matrice de densité locale permettent de faire fonctionner des circuits dans lesquels un sous-ensemble des qubits du circuit peut être mesuré, ce que l'on appelle souvent mesure partielle.

Par exemple, dans le code suivant, vous pouvez créer un circuit à deux qubits et mesurer uniquement le premier qubit en ajoutant une mesure instruction avec les qubits cibles à la fin du circuit.

```
# 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)
```

Émulateur de dispositif quantique local

Avec l'outil d'émulation local d'Amazon Braket, vous pouvez émuler vos programmes quantiques textuels localement avant de les exécuter sur du matériel quantique réel. L'émulateur utilise les données d'étalonnage de l'appareil pour valider les circuits mot pour mot, ce qui vous permet de détecter les problèmes de compatibilité plus tôt.

En outre, l'émulateur local simule le bruit matériel quantique selon le processus suivant :

- Utilisation des données d'étalonnage de l'appareil pour construire le modèle de bruit
- Appliquer un bruit dépolarisant à chaque porte de votre circuit
- Appliquer une erreur de lecture à la fin de votre circuit
- Simulation du circuit bruyant avec un simulateur de matrice de densité locale

Pour plus d'informations sur l'utilisation d'un émulateur local, consultez la section [Émulation locale pour les circuits verbatim sur Amazon Braket](#) dans le référentiel. `amazon-braket-examples` GitHub

Dans cette section :

- [Avantages de l'émulation locale](#)
- [Création d'un émulateur local](#)

Avantages de l'émulation locale

- Validez mot pour mot les circuits en fonction des contraintes de l'appareil à l'aide de données d'étalonnage en temps réel ou historiques.
- Déboguez les problèmes avant de soumettre des tâches au matériel quantique.
- Comparez les émulations silencieuses et bruyantes avec les résultats matériels pour comprendre les effets du bruit.
- Simplifiez le flux de travail de développement d'algorithmes quantiques sensibles au bruit.

Création d'un émulateur local

Un émulateur de dispositif quantique local peut être créé directement à partir d'un dispositif quantique ou d'un ensemble de propriétés de dispositif. Lors de l'émulation directe d'un périphérique, l'émulateur utilise les données d'étalonnage les plus récentes du périphérique instancié. L'exemple de code suivant montre comment émuler directement Rigetti's Ankaa-3 un appareil.

```
from braket.aws.aws_device import AwsDevice

ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_emulator = ankaa3.emulator()
```

L'exemple suivant montre la création d'un émulateur de périphérique local à partir d'un ensemble de propriétés de Ankaa-3 périphérique au format JSON.

```
from braket.aws import AwsDevice
from braket.emulation.local_emulator import LocalEmulator
import json

# Instantiate the device
ankaa3 = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3")
ankaa3_properties = ankaa3.properties

# Put the Ankaa-3 properties in a file named ankaa3_device_properties.json
with open("ankaa3_device_properties.json", "w") as f:
    json.dump(ankaa3_properties.json(), f)

# Load the json into the ankaa3_data_json variable
with open("ankaa3_device_properties.json", "r") as json_file:
    ankaa3_data_json = json.load(json_file)

# Create the Ankaa-3 local emulator from the json file you created
ankaa3_emulator = LocalEmulator.from_json(ankaa3_data_json)
```

Vous pouvez personnaliser l'exemple en :

- Utilisation des propriétés d'un autre périphérique QPU
- Spécification d'un fichier JSON différent pour l'émulateur
- Modification des valeurs des propriétés de l'appareil avant d'instancier l'émulateur

Exécution de vos tâches quantiques avec Amazon Braket

Braket fournit un accès sécurisé à la demande à différents types d'ordinateurs quantiques. Vous avez accès à des ordinateurs quantiques basés sur des portes depuis AQT, IonQ, et IQM Rigetti, ainsi qu'à un simulateur hamiltonien analogique de QuEra. Vous n'avez également aucun engagement initial et vous n'avez pas besoin de vous procurer un accès auprès de fournisseurs individuels.

- La [console Amazon Braket](#) fournit des informations et l'état de l'appareil pour vous aider à créer, gérer et surveiller vos ressources et vos tâches quantiques.
- Soumettez et exécutez des tâches quantiques via le [SDK Amazon Braket Python](#), ainsi que via la console. Le SDK est accessible via des blocs-notes Amazon Braket préconfigurés.
- L'[API Amazon Braket](#) est accessible via le SDK Python Amazon Braket et les carnets de notes. Vous pouvez passer des appels directement au API si vous créez des applications qui fonctionnent avec l'informatique quantique de manière programmatique.

Les exemples présentés dans cette section montrent comment utiliser Amazon Braket API directement à l'aide du SDK Python Amazon Braket et du SDK Python pour Braket ([AWS Boto3](#)).

En savoir plus sur le Amazon SDK Braket Python

Pour utiliser le SDK Python Amazon Braket, installez d'abord le SDK AWS Python pour Braket (Boto3) afin de pouvoir communiquer avec le. AWS API. Vous pouvez considérer le SDK Amazon Braket Python comme une solution pratique pour les clients du secteur quantique autour de Boto3.

- Boto3 contient des interfaces sur lesquelles vous devez accéder. AWS API (Notez que Boto3 est un grand SDK Python qui communique avec. AWS API. La plupart Services AWS supportent une interface Boto3.)
- Le SDK Amazon Braket Python contient des modules logiciels pour les circuits, les portes, les appareils, les types de résultats et d'autres parties d'une tâche quantique. Chaque fois que vous créez un programme, vous importez les modules dont vous avez besoin pour cette tâche quantique.
- Le SDK Amazon Braket Python est accessible via des blocs-notes, qui sont préchargés avec tous les modules et dépendances dont vous avez besoin pour exécuter des tâches quantiques.
- Vous pouvez importer des modules depuis le SDK Amazon Braket Python dans n'importe quel script Python si vous ne souhaitez pas travailler avec des blocs-notes.

Après avoir [installé Boto3](#), voici un aperçu des étapes de création d'une tâche quantique via le SDK Amazon Braket Python :

1. (Facultatif) Ouvrez votre bloc-notes.
2. Importez les modules SDK dont vous avez besoin pour vos circuits.
3. Spécifiez un QPU ou un simulateur.
4. Instanciez le circuit.
5. Lancez le circuit.
6. Collectez les résultats.

Les exemples présentés dans cette section présentent les détails de chaque étape.

Pour plus d'exemples, consultez le référentiel [Amazon Braket Examples sur](#) GitHub

Dans cette section :

- [Soumission de tâches quantiques à QPUs](#)
- [Exécution de plusieurs programmes](#)
- [Quand s'exécutera ma tâche quantique ?](#)
- [Travailler avec les réservations](#)
- [Techniques d'atténuation des erreurs](#)

Soumission de tâches quantiques à QPUs

Amazon Braket donne accès à plusieurs appareils capables d'exécuter des tâches quantiques. Vous pouvez soumettre des tâches quantiques individuellement ou configurer le traitement par [lots de tâches quantiques](#).

Unités de traitement quantique (QPUs)

Vous pouvez soumettre des tâches quantiques QPUs à tout moment, mais la tâche s'exécute dans certaines fenêtres de disponibilité affichées sur la page Appareils de la console Amazon Braket. Vous pouvez récupérer les résultats de la tâche quantique à l'aide de l'identifiant de tâche quantique, présenté dans la section suivante.

- AQT IBEX-Q1 : `arn:aws:braket:eu-north-1::device/qpu/aqt/Ibex-Q1`
- IonQ Aria-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Aria-1`

- IonQ Forte-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-1`
- IonQ Forte-Enterprise-1 : `arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-1`
- IQM Garnet : `arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet`
- IQM Emerald : `arn:aws:braket:eu-north-1::device/qpu/iqm/Emerald`
- QuEra Aquila : `arn:aws:braket:us-east-1::device/qpu/quera/Aquila`
- Rigetti Ankaa-3 : `arn:aws:braket:us-west-1::device/qpu/rigetti/Ankaa-3`

Note

Vous pouvez annuler des tâches quantiques telles que celles pour CREATED les simulateurs QPUs ou à la demande. Vous pouvez annuler les tâches quantiques dans l'QUEUED état dans la mesure du possible pour les simulateurs à la demande et QPUs. Notez qu'il est peu probable que les tâches QUEUED quantiques QPU soient annulées avec succès pendant les fenêtres de disponibilité des QPU.

Dans cette section :

- [AQT](#)
- [IonQ](#)
- [IQM](#)
- [Rigetti](#)
- [QuEra](#)
- [Exemple : Soumission d'une tâche quantique à un QPU](#)
- [Inspection des circuits compilés](#)

AQT

AQT Le QPU IBEX-Q1 est basé sur un cristal de $^{40}\text{Ca}^{+}$ ions dans un piège à radiofréquences macroscopique situé dans une chambre à très haut vide. L'appareil fonctionne à température ambiante et s'insère dans deux racks de 19 pouces compatibles avec les centres de données.

Les portes haute fidélité sont rendues possibles par les faibles taux de chauffage du piège et par l'utilisation d'une transition optique directe pour la rotation des qubits. La transition du qubit est

entraînée par un laser à faible largeur de ligne avec une stabilité de fréquence relative très élevée. Les qubits permettent également une préparation et une lecture efficaces de l'état grâce à des étagères optiques. All-to-all connectivité est obtenue grâce à l'interaction de Coulomb à longue portée dans le cristal ionique. L'adressage et la lecture à ions uniques sont réalisés à l'aide d'un objectif à ouverture numérique élevée.

L'AQTappareil prend en charge les portes quantiques suivantes.

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

Avec la compilation textuelle, l'AQTappareil prend en charge les portes natives suivantes.

```
'prx', 'xx', 'rz'
```

Note

Ce qui suit décrit les passerelles équivalentes entre les portes AQT natives et Amazon Braket :

- La porte AQT Mølmer-Sørensen (MS ou RXX) correspond à la porte de Braket 'xx'
- La porte AQT R correspond à la porte de 'prx' Braket
- Le nom de la 'rz' porte est le même

IonQ

IonQpropose une technologie basée sur une porte QPUs basée sur la technologie du piège à ions. IonQ's QPUs les ions piégés sont construits sur une chaîne d'ions $^{171}\text{Yb}^+$ piégés qui sont confinés dans l'espace au moyen d'un piège à électrodes de surface microfabriqué dans une chambre à vide.

IonQles appareils supportent les portes quantiques suivantes.

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


Avec la compilation textuelle, les IonQ QPUs supportent les portes natives suivantes.

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

Si vous ne spécifiez que deux paramètres de phase lorsque vous utilisez la porte MS native, une porte MS entièrement enchevêtrée s'exécute. Une porte MS totalement enchevêtrée effectue toujours une rotation de $\pi/2$. Pour spécifier un angle différent et exécuter une porte MS partiellement enchevêtrée, vous devez spécifier l'angle souhaité en ajoutant un troisième paramètre. Pour plus d'informations, consultez le module braket.circuits.gate.

Ces portes natives ne peuvent être utilisées qu'avec une compilation textuelle. Pour en savoir plus sur la compilation textuelle, consultez la section Compilation [textuelle](#).

IQM

Les processeurs quantiques sont des dispositifs de type portail universel basés sur des qubits de transmission supraconducteurs. IQM GarnetII s'agit d'un appareil de 20 qubits, tandis qu'IQM Emerald s'agit d'un appareil de 54 qubits. Ces deux appareils utilisent une topologie à réseau carré, également connue sous le nom de topologie à réseau cristallin.

Les IQM appareils supportent les portes quantiques suivantes.

```
"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"
```

Avec la compilation textuelle, les IQM appareils prennent en charge les portes natives suivantes.

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

Rigetti

Les processeurs quantiques sont des machines universelles de type portail basées sur des systèmes supraconducteurs entièrement réglables. qubits

- Le Ankaa-3 système est un appareil de 84 qubits qui utilise une technologie multipuce évolutive.

L'appareil Rigetti prend en charge les portes quantiques suivantes.

```
'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'
```

Avec la compilation textuelle, Ankaa-3 prend en charge les portes natives suivantes.

```
'rx', 'rz', 'iswap'
```

Rigettiles processeurs quantiques supraconducteurs peuvent exécuter la porte « rx » uniquement avec les angles de $\pm\pi/2$ ou $\pm\pi$.

Le contrôle du niveau d'impulsion est disponible sur les appareils Rigetti, qui prennent en charge un ensemble de cadres prédéfinis des types suivants pour le système. Ankaa-3

```
`flux_tx`, `charge_tx`, `readout_rx`, `readout_tx`
```

QuEra

QuEra propose des dispositifs basés sur des atomes neutres capables d'exécuter des tâches quantiques de simulation hamiltonienne analogique (AHS). Ces dispositifs spéciaux reproduisent fidèlement la dynamique quantique dépendante du temps de centaines de qubits interagissant simultanément.

On peut programmer ces appareils selon le paradigme de la simulation hamiltonienne analogique en prescrivant la disposition du registre des qubits et la dépendance temporelle et spatiale des champs de manipulation. Amazon Braket fournit des utilitaires permettant de créer de tels programmes via le module AHS du SDK Python, `.braket.ahs`

Pour plus d'informations, consultez les [carnets d'exemples de simulation hamiltonienne analogique](#) ou la page [Soumettre un programme analogique à l'aide QuEra](#) d'Aquila.

Exemple : Soumission d'une tâche quantique à un QPU

Amazon Braket vous permet d'exécuter un circuit quantique sur un appareil QPU. L'exemple suivant montre comment soumettre une tâche quantique à Rigetti ou à IonQ des appareils.

Choisissez l'Rigetti Ankaa-3 appareil, puis regardez le graphique de connectivité associé

```
# import the QPU module
```

```

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

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

```

```

{'fullyConnected': False,
 'connectivityGraph': {'0': ['1', '7'],
   '1': ['0', '2', '8'],
   '2': ['1', '3', '9'],
   '3': ['2', '4', '10'],
   '4': ['3', '5', '11'],
   '5': ['4', '6', '12'],
   '6': ['5', '13'],
   '7': ['0', '8', '14'],
   '8': ['1', '7', '9', '15'],
   '9': ['2', '8', '10', '16'],
   '10': ['3', '9', '11', '17'],
   '11': ['4', '10', '12', '18'],
   '12': ['5', '11', '13', '19'],
   '13': ['6', '12', '20'],
   '14': ['7', '15', '21'],
   '15': ['8', '14', '22'],
   '16': ['9', '17', '23'],
   '17': ['10', '16', '18', '24'],
   '18': ['11', '17', '19', '25'],
   '19': ['12', '18', '20', '26'],
   '20': ['13', '19', '27'],
   '21': ['14', '22', '28'],
   '22': ['15', '21', '23', '29'],
   '23': ['16', '22', '24', '30'],
   '24': ['17', '23', '25', '31'],
   '25': ['18', '24', '26', '32'],
   '26': ['19', '25', '33'],
   '27': ['20', '34'],
   '28': ['21', '29', '35'],
   '29': ['22', '28', '30', '36'],
   '30': ['23', '29', '31', '37'],
   '31': ['24', '30', '32', '38'],
   '32': ['25', '31', '33', '39'],
   '33': ['26', '32', '34', '40'],
   '34': ['27', '33', '41'],

```

```
'35': ['28', '36', '42'],
'36': ['29', '35', '37', '43'],
'37': ['30', '36', '38', '44'],
'38': ['31', '37', '39', '45'],
'39': ['32', '38', '40', '46'],
'40': ['33', '39', '41', '47'],
'41': ['34', '40', '48'],
'42': ['35', '43', '49'],
'43': ['36', '42', '44', '50'],
'44': ['37', '43', '45', '51'],
'45': ['38', '44', '46', '52'],
'46': ['39', '45', '47', '53'],
'47': ['40', '46', '48', '54'],
'48': ['41', '47', '55'],
'49': ['42', '56'],
'50': ['43', '51', '57'],
'51': ['44', '50', '52', '58'],
'52': ['45', '51', '53', '59'],
'53': ['46', '52', '54'],
'54': ['47', '53', '55', '61'],
'55': ['48', '54', '62'],
'56': ['49', '57', '63'],
'57': ['50', '56', '58', '64'],
'58': ['51', '57', '59', '65'],
'59': ['52', '58', '60', '66'],
'60': ['59'],
'61': ['54', '62', '68'],
'62': ['55', '61', '69'],
'63': ['56', '64', '70'],
'64': ['57', '63', '65', '71'],
'65': ['58', '64', '66', '72'],
'66': ['59', '65', '67'],
'67': ['66', '68'],
'68': ['61', '67', '69', '75'],
'69': ['62', '68', '76'],
'70': ['63', '71', '77'],
'71': ['64', '70', '72', '78'],
'72': ['65', '71', '73', '79'],
'73': ['72', '80'],
'75': ['68', '76', '82'],
'76': ['69', '75', '83'],
'77': ['70', '78'],
'78': ['71', '77', '79'],
'79': ['72', '78', '80'],
```

```
'80': ['73', '79', '81'],
'81': ['80', '82'],
'82': ['75', '81', '83'],
'83': ['76', '82']}]}
```

Le dictionnaire précédent `connectivityGraph` répertorie les qubits voisins pour chaque qubit de l'`Rigetti` appareil.

Choisissez l'`IonQ Aria-1` appareil

Pour l'`IonQ Aria-1` appareil, le `connectivityGraph` est vide, comme indiqué dans l'exemple suivant, car l'appareil offre une all-to-all connectivité. Par conséquent, un détail `connectivityGraph` est pas nécessaire.

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

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

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

Comme indiqué dans l'exemple suivant, vous avez la possibilité d'ajuster le `shots` (par défaut = 1000), le `poll_timeout_seconds` (par défaut = 432000 = 5 jours), le `poll_interval_seconds` (par défaut = 1) et l'emplacement du compartiment S3 (`s3_location`) où vos résultats seront stockés si vous choisissez de spécifier un emplacement autre que le compartiment par défaut.

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

Les `Rigetti` appareils `IonQ` et compilent automatiquement le circuit fourni dans leurs ensembles de portes natifs respectifs, et ils mappent les qubit indices abstraits aux indices physiques qubits sur le QPU correspondant.

Note

Les appareils QPU ont une capacité limitée. Vous pouvez vous attendre à des temps d'attente plus longs lorsque la capacité sera atteinte.

Amazon Braket peut exécuter des tâches quantiques QPU dans certaines fenêtres de disponibilité, mais vous pouvez toujours soumettre des tâches quantiques à tout moment (24 heures sur 24, 7 jours sur 7) car toutes les données et métadonnées correspondantes sont stockées de manière fiable dans le compartiment S3 approprié. Comme indiqué dans la section suivante, vous pouvez récupérer votre tâche quantique à l'aide `AwsQuantumTask` de votre identifiant unique de tâche quantique.

Inspection des circuits compilés

Lorsqu'un circuit quantique doit être exécuté sur un périphérique matériel, tel qu'une unité de traitement quantique (QPU), le circuit doit d'abord être compilé dans un format acceptable que le périphérique peut comprendre et traiter. Par exemple, transpiler le circuit quantique de haut niveau vers les portes natives spécifiques prises en charge par le matériel QPU cible. L'inspection de la sortie compilée réelle du circuit quantique peut être extrêmement utile à des fins de débogage et d'optimisation. Ces connaissances peuvent aider à identifier les problèmes potentiels, les goulets d'étranglement ou les opportunités d'amélioration des performances et de l'efficacité de l'application quantique. Vous pouvez visualiser et analyser la sortie compilée de vos circuits quantiques pour les deux Rigetti et pour les appareils informatiques IQM quantiques à l'aide du code fourni ci-dessous.

```
task = AwsQuantumTask(arn=task_id, aws_session=session)
# After the task has finished running
task_result = task.result()
compiled_circuit = task_result.get_compiled_circuit()
```

Note

Actuellement, l'affichage de la sortie du circuit compilé pour les IonQ appareils n'est pas pris en charge.

Exécution de plusieurs programmes

Amazon Braket propose deux approches pour exécuter efficacement plusieurs programmes quantiques : les ensembles de programmes et le traitement par lots de tâches quantiques.

Les ensembles de programmes constituent la méthode préférée pour exécuter des charges de travail avec plusieurs programmes. Ils vous permettent de regrouper plusieurs programmes dans une seule tâche quantique Amazon Braket. Les ensembles de programmes permettent d'[améliorer les](#)

[performances](#) et de réaliser des économies par rapport à la soumission individuelle de programmes, en particulier lorsque le nombre d'exécutions de programmes approche les 100.

Actuellement, IQM et les Rigetti appareils prennent en charge les ensembles de programmes. Avant de soumettre des ensembles de programmes à QPUs, il est recommandé de les [tester d'abord sur le simulateur local Amazon Braket](#). [Pour vérifier si un appareil prend en charge les ensembles de programmes, vous pouvez consulter les propriétés de l'appareil à l'aide du SDK Amazon Braket ou consulter la page de l'appareil dans la console Amazon Braket](#).

L'exemple suivant montre comment exécuter un ensemble de programmes.

```
from math import pi
from braket.devices import LocalSimulator
from braket.program_sets import ProgramSet
from braket.circuits import Circuit

program_set = ProgramSet([
    Circuit().h(0).cnot(0,1),
    Circuit().rx(0, pi/4).ry(1, pi/8).cnot(1,0),
    Circuit().t(0).t(1).cz(0,1).s(0).cz(1,2).s(1).s(2),
])

device = LocalSimulator()
result = device.run(program_set, shots=300).result()
print(result[0][0].counts) # The result of the first program in the program set
```

Pour en savoir plus sur les différentes manières de créer un ensemble de programmes (par exemple, créer un ensemble de programmes à partir de nombreux observables ou paramètres avec un seul programme) et de récupérer les résultats d'un ensemble de programmes, consultez la section sur les [ensembles de programmes](#) du guide du développeur Amazon Braket et le dossier des [ensembles de programmes dans le](#) référentiel Github des exemples de Braket.

Le traitement par lots de tâches quantiques est disponible sur tous les appareils Amazon Braket. Le traitement par lots est particulièrement utile pour les tâches quantiques que vous exécutez sur les simulateurs à la demande (DM1ouTN1)SV1, car ils peuvent traiter plusieurs tâches quantiques en parallèle. Le traitement par lots vous permet de lancer des tâches quantiques en parallèle. Par exemple, si vous souhaitez effectuer un calcul qui nécessite 10 tâches quantiques et que les programmes de ces tâches quantiques sont indépendants les uns des autres, il est recommandé d'utiliser le traitement par lots de tâches. Utilisez le traitement par lots de tâches quantiques lorsque

vous exécutez des charges de travail avec plusieurs programmes sur un appareil qui ne prend pas en charge les ensembles de programmes.

L'exemple suivant montre comment exécuter un lot de tâches quantiques.

```
from braket.circuits import Circuit
from braket.devices import LocalSimulator

bell = Circuit().h(0).cnot(0, 1)
circuits = [bell for _ in range(5)]

device = LocalSimulator()
batch = device.run_batch(circuits, shots=100)
print(batch.results()[0].measurement_counts) # The result of the first quantum task in
the batch
```

Pour des informations plus spécifiques sur le traitement par lots, consultez les exemples [d'Amazon Braket](#) sur GitHub

Dans cette section :

- [À propos de l'ensemble du programme et des coûts](#)
- [À propos du traitement par lots de tâches quantiques et des coûts](#)
- [Traitement par lots de tâches quantiques et PennyLane](#)
- [Traitement par lots de tâches et circuits paramétrés](#)

À propos de l'ensemble du programme et des coûts

Les ensembles de programmes exécutent efficacement plusieurs programmes quantiques en regroupant jusqu'à 100 programmes ou ensembles de paramètres dans une seule tâche quantique. Avec les ensembles de programmes, vous ne payez qu'un seul frais par tâche, plus les frais par plan basés sur le nombre total de prises de vue pour tous les programmes, ce qui réduit considérablement les coûts par rapport à la soumission de programmes individuels. Cette approche est particulièrement avantageuse pour les charges de travail impliquant de nombreux programmes et un faible nombre de prises par programme. Les ensembles de programmes sont actuellement pris en charge sur Rigetti les appareils IQM et sur le simulateur local Amazon Braket.

Pour plus d'informations, consultez la section [Ensembles de programmes](#) pour connaître les étapes de mise en œuvre détaillées, les meilleures pratiques et des exemples de code.

À propos du traitement par lots de tâches quantiques et des coûts

Quelques mises en garde à l'esprit concernant le traitement par lots de tâches quantiques et les coûts de facturation :

- Par défaut, le traitement par lots de tâches quantiques recommence à tout moment ou échoue 3 fois aux tâches quantiques.
- Un lot de tâches quantiques de longue durée, telles que 34 qubits pourSV1, peut entraîner des coûts importants. Assurez-vous de bien vérifier les valeurs `run_batch` d'assignation avant de commencer un lot de tâches quantiques. Nous ne recommandons pas l'utilisation TN1 avec `run_batch`.
- TN1 peut entraîner des coûts en cas d'échec des tâches de la phase de répétition (voir [la TN1 description](#) pour plus d'informations). Les nouvelles tentatives automatiques peuvent augmenter le coût. Nous vous recommandons donc de définir le nombre de « `max_retries` » sur 0 lors du traitement par lots lors de l'utilisation TN1 (voir [Quantum Task Batching](#), ligne 186).

Traitement par lots de tâches quantiques et PennyLane

Tirez parti du traitement par lots lorsque vous l'utilisez PennyLane sur Amazon Braket en `parallel = True` définissant le moment où vous instanciez un appareil Amazon Braket, comme indiqué dans l'exemple suivant.

```
import pennylane as qml

# Define the number of wires (qubits) you want to use
wires = 2 # For example, using 2 qubits

# Define your S3 bucket
my_bucket = "amazon-braket-s3-demo-bucket"
my_prefix = "pennylane-batch-output"
s3_folder = (my_bucket, my_prefix)

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)
```

Pour plus d'informations sur le traitement par lots avec PennyLane, consultez [Optimisation parallélisée des](#) circuits quantiques.

Traitement par lots de tâches et circuits paramétrés

Lorsque vous soumettez un lot de tâches quantiques contenant des circuits paramétrés, vous pouvez soit fournir un `inputs` dictionnaire, qui est utilisé pour toutes les tâches quantiques du lot, soit un dictionnaire `list` d'entrée, auquel cas le `i` -ème dictionnaire est associé à la `i` -ème tâche, comme illustré dans l'exemple suivant.

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

# Define your quantum device
device = AwsDevice("arn:aws:braket:::device/quantum-simulator/amazon/sv1")

# 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)
```

Vous pouvez également préparer une liste de dictionnaires d'entrée pour un seul circuit paramétrique et les soumettre sous forme de lot de tâches quantiques. Si la liste contient `N` dictionnaires d'entrée, le lot contient `N` tâches quantiques. La `i` -ème tâche quantique correspond au circuit exécuté avec le `i` -ème dictionnaire d'entrée.

```
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, shots=100)
```

Quand s'exécutera ma tâche quantique ?

Lorsque vous soumettez un circuit, Amazon Braket l'envoie à l'appareil que vous spécifiez. Les tâches quantiques de l'unité de traitement quantique (QPU) et du simulateur à la demande sont mises en file d'attente et traitées dans l'ordre de leur réception. Le temps nécessaire au traitement de votre tâche quantique une fois que vous l'avez soumise varie en fonction du nombre et de la complexité des tâches soumises par d'autres clients d'Amazon Braket et de la disponibilité du QPU sélectionné.

Dans cette section :

- [Fenêtres de disponibilité et état du QPU](#)
- [Visibilité de la file](#)
- [Configurer les notifications par e-mail ou par SMS](#)

Fenêtres de disponibilité et état du QPU

La disponibilité du QPU varie d'un appareil à l'autre.

Sur la page Appareils de la console Amazon Braket, vous pouvez consulter les fenêtres de disponibilité actuelles et à venir ainsi que l'état des appareils. De plus, chaque page de périphérique indique la profondeur de file d'attente individuelle pour les tâches quantiques et les tâches hybrides.

Un appareil est considéré comme hors ligne s'il n'est pas disponible pour les clients, quelle que soit la fenêtre de disponibilité. Par exemple, il peut être hors ligne en raison d'une maintenance planifiée, de mises à niveau ou de problèmes opérationnels.

Visibilité de la file

Avant de soumettre une tâche quantique ou une tâche hybride, vous pouvez voir combien de tâches quantiques ou de tâches hybrides vous attendent en vérifiant la profondeur de la file d'attente des appareils.

Profondeur de file d'attente

Queue depth fait référence au nombre de tâches quantiques et de tâches hybrides mises en file d'attente pour un appareil donné. Les tâches quantiques et le nombre de files d'attente de tâches hybrides d'un appareil sont accessibles via le Braket Software Development Kit (SDK) bloc opératoire Amazon Braket Management Console.

1. La profondeur de la file d'attente des tâches fait référence au nombre total de tâches quantiques qui attendent actuellement d'être exécutées en priorité normale.
2. La profondeur de la file d'attente des tâches prioritaires fait référence au nombre total de tâches quantiques soumises en attente d'exécution Amazon Braket Hybrid Jobs. Ces tâches s'exécutent avant les tâches autonomes.
3. La profondeur de la file d'attente des tâches hybrides fait référence au nombre total de tâches hybrides actuellement en file d'attente sur un appareil. Quantum tasks soumis dans le cadre d'une tâche hybride sont prioritaires et sont agrégés dans le Priority Task Queue.

Les clients qui souhaitent visualiser la profondeur de la file d'attente via le Braket SDK peuvent modifier l'extrait de code suivant pour obtenir la position dans la file d'attente de leur tâche quantique ou de leur tâche hybride :

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

# 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'
```

La soumission d'une tâche quantique ou d'une tâche hybride à un QPU peut entraîner une baisse de votre charge de travail QUEUED. Amazon Braket fournit aux clients une visibilité sur leur position dans la file d'attente des tâches quantiques et hybrides.

Position de la file

Queue position fait référence à la position actuelle de votre tâche quantique ou de votre tâche hybride dans une file d'attente d'appareils correspondante. Il peut être obtenu pour des tâches quantiques ou des tâches hybrides par le biais du Braket Software Development Kit (SDK) ou Amazon Braket Management Console.

Les clients qui souhaitent consulter la position de la file d'attente via le Braket SDK peuvent modifier l'extrait de code suivant pour obtenir la position dans la file d'attente de leur tâche quantique ou de leur tâche hybride :

```
# choose the device to run your circuit
device = AwsDevice("arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet")

#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:eu-north-1::device/qpu/iqm/Garnet",
    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
```

Configurer les notifications par e-mail ou par SMS

Amazon Braket envoie des événements à Amazon EventBridge lorsque la disponibilité d'un QPU change ou lorsque l'état de votre tâche quantique change. Suivez ces étapes pour recevoir des notifications de modification de l'état de l'appareil et des tâches quantiques par e-mail ou par SMS :

1. Créez une rubrique Amazon SNS et un abonnement par e-mail ou SMS. La disponibilité des e-mails ou des SMS dépend de votre région. Pour plus d'informations, consultez [Commencer à utiliser Amazon SNS](#) et [Envoyer des SMS](#).
2. Créez une règle EventBridge qui déclenche les notifications adressées à votre rubrique SNS. Pour plus d'informations, consultez la section [Surveillance d'Amazon Braket avec Amazon EventBridge](#).

(Facultatif) Configurer les notifications SNS

Vous pouvez configurer des notifications via le service Amazon Simple Notification Service (SNS) afin de recevoir une alerte lorsque votre tâche quantique Amazon Braket est terminée. Les notifications actives sont utiles si vous prévoyez un long délai d'attente, par exemple lorsque vous soumettez une tâche quantique importante ou lorsque vous soumettez une tâche quantique en dehors de la fenêtre de disponibilité d'un appareil. Si vous ne souhaitez pas attendre la fin de la tâche quantique, vous pouvez configurer une notification SNS.

Un carnet Amazon Braket vous guide à travers les étapes de configuration. Pour plus d'informations, consultez [les exemples d'Amazon Braket GitHub et, en particulier, l'exemple de bloc-notes pour configurer les notifications](#).

Travailler avec les réservations

Les réservations vous donnent un accès exclusif à l'appareil quantique de votre choix. Vous pouvez planifier une réservation à votre convenance afin de savoir exactement quand votre charge de travail commence et se termine. Les réservations sont disponibles par tranches d'une heure pour tous les appareils Braket et peuvent être annulées jusqu'à 48 heures à l'avance, sans frais supplémentaires. Nous vous recommandons de mettre en file d'attente les tâches quantiques et les tâches hybrides pour une prochaine réservation à l'avance, en utilisant votre ARN de réservation Braket Direct ou en soumettant des charges de travail lors de votre réservation.

Le coût de l'accès à un appareil dédié est basé sur la durée de votre réservation, quel que soit le nombre de tâches quantiques et de tâches hybrides que vous exécutez sur l'unité de traitement

quantique (QPU). Une liste actualisée des ordinateurs quantiques disponibles pour les réservations est disponible sur notre [page de tarification](#) ou via la console de [gestion Amazon Braket](#).

Note

Dans le cas d'une réservation, il n'y a pas de [limite d'accès](#). En outre, pour les IonQ appareils, le nombre minimum de tirs pour les tâches [d'atténuation des erreurs](#) est réduit à 500 (contre 2 500 pour les tâches à la demande).

Quand utiliser une réservation

Tirer parti de l'accès aux réservations vous offre la commodité et la prévisibilité de savoir exactement quand votre charge de travail quantique commence et se termine son exécution. Par rapport à la soumission de tâches et de tâches hybrides à la demande, vous n'avez pas à attendre dans une file d'attente pour d'autres tâches clients. Comme vous avez un accès exclusif à l'appareil lors de votre réservation, seules vos charges de travail sont exécutées sur l'appareil pendant toute la durée de la réservation.

Nous vous recommandons d'utiliser l'accès à la demande pour la phase de conception et de prototypage de votre recherche, afin de permettre une itération rapide et rentable de vos algorithmes. Une fois que vous serez prêt à produire les résultats finaux de l'expérience, pensez à planifier une réservation d'appareil à votre convenance afin de respecter les délais de projet ou de publication. Nous vous recommandons également d'utiliser les réservations lorsque vous souhaitez exécuter des tâches à des moments précis, par exemple lorsque vous organisez une démonstration en direct ou un atelier sur un ordinateur quantique.

Dans cette section :

- [Comment créer une réservation](#)
- [Exécution de tâches quantiques lors d'une réservation](#)
- [Exécution de tâches hybrides lors d'une réservation](#)
- [Que se passe-t-il à la fin de votre réservation](#)
- [Annuler ou replanifier une réservation existante](#)

Comment créer une réservation

Pour créer une réservation, contactez l'équipe Braket en suivant ces étapes :

1. Ouvrez la console Amazon Braket.
2. Choisissez Braket Direct dans le volet de gauche, puis dans la section Réservations, choisissez Réserver un appareil.
3. Sélectionnez l'appareil que vous souhaitez réserver.
4. Fournissez vos informations de contact, y compris votre nom et votre e-mail. Assurez-vous de fournir une adresse e-mail valide que vous consultez régulièrement.
5. Sous Parlez-nous de votre charge de travail, fournissez des informations sur la charge de travail à exécuter en utilisant votre réservation. Par exemple, la durée de réservation souhaitée, les contraintes pertinentes ou le calendrier souhaité.

Après avoir soumis le formulaire, vous recevez un e-mail de l'équipe Braket avec les prochaines étapes. Une fois votre réservation confirmée, vous recevrez l'ARN de réservation par e-mail. Vous devez utiliser l'ARN de réservation pour créer des tâches de réservation. Les tâches créées sans l'ARN de réservation seront soumises à la file d'attente à la demande normale et ne seront PAS exécutées pendant votre réservation.

Note

Votre réservation n'est confirmée qu'une fois que vous avez reçu l'ARN de réservation.

Les réservations sont disponibles par tranches d'au moins une heure et certains appareils peuvent être soumis à des contraintes de durée de réservation supplémentaires (y compris les durées de réservation minimales et maximales). L'équipe Braket partage toutes les informations pertinentes avec vous avant de confirmer la réservation.

L'équipe Braket vous contactera par e-mail pour organiser une session de 30 minutes avec un expert de Braket.

Exécution de tâches quantiques lors d'une réservation

Après avoir obtenu un ARN de réservation valide auprès de [Create a reservation](#), vous pouvez créer des tâches quantiques à exécuter pendant la réservation. Les tâches quantiques et hybrides soumises avec un ARN de réservation n'apparaîtront pas dans une file d'attente d'appareils. Les tâches soumises avant l'heure de début de la réservation resteront QUEUED inchangées jusqu'au début de votre réservation.

 Note

Les réservations sont spécifiques au AWS compte et à l'appareil. Seul le AWS compte qui a créé la réservation peut utiliser votre ARN de réservation.

Lors d'une réservation, il est possible de créer à la fois des tâches de réservation et des tâches régulières. Pour vérifier qu'une tâche quantique Braket créée est associée à une réservation, vérifiez le champ « Reservation ARN » sur la page de la tâche quantique dans la console Braket, ou interrogez le même champ dans les métadonnées de la tâche à l'aide du SDK. Le reste de cette page explique comment spécifier les tâches associées à la réservation.

Vous pouvez créer des tâches quantiques en Python SDKs utilisant [Braket](#),,, [CUDA-QPennyLaneQiskit](#), ou directement avec boto3 ([Working with Boto3](#)). Pour utiliser les réservations, vous devez disposer de la version [v1.79.0](#) ou supérieure d'Amazon [Braket](#). Python SDK Vous pouvez effectuer la mise à jour vers le dernier SDK, Qiskit fournisseur et PennyLane plugin Braket à l'aide du code suivant.

```
pip install --upgrade amazon-braket-sdk amazon-braket-pennylane-plugin qiskit-braket-provider
```

Exécuter des tâches avec le gestionnaire de **DirectReservation** contexte

La méthode recommandée pour exécuter une tâche dans le cadre de votre réservation planifiée consiste à utiliser le gestionnaire de `DirectReservation` contexte. En spécifiant votre appareil cible et votre ARN de réservation, le gestionnaire de contexte garantit que toutes les tâches créées dans l'`withinstruction` Python sont exécutées avec un accès exclusif à l'appareil.

Définissez d'abord un circuit quantique et le dispositif. Utilisez ensuite le contexte de réservation et exécutez la tâche. Assurez-vous que l'intégralité de votre charge de travail est gérée à l'intérieur du **with** bloc ; tout ce qui se trouve en dehors du **with** bloc ne sera pas associé à votre réservation !

```
from braket.aws import AwsDevice, DirectReservation
from braket.circuits import Circuit
from braket.devices import Devices

bell = Circuit().h(0).cnot(0, 1)
device = AwsDevice(Devices.IonQ.Aria1)
```

```
# run the circuit in a reservation
with DirectReservation(device, reservation_arn="<my_reservation_arn>"):
    task = device.run(bell, shots=100)
```

Vous pouvez créer des tâches quantiques dans une réservation en utilisant CUDA-QPennyLane, et des Qiskit plugins, à condition que le `DirectReservation` contexte soit actif lors de la création de tâches quantiques. Par exemple, avec le Qiskit-Braket fournisseur, vous pouvez exécuter des tâches comme suit.

```
from braket.devices import Devices
from braket.aws import DirectReservation
from qiskit import QuantumCircuit
from qiskit_braket_provider import BraketProvider

qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)

aria = BraketProvider().get_backend("Aria 1")

# run the circuit in a reservation
with DirectReservation(Devices.IonQ.Aria1, reservation_arn="<my_reservation_arn>"):
    aria_task = aria.run(qc, shots=10)
```

De même, le code suivant exécute un circuit lors d'une réservation à l'aide du Braket-PennyLane plugin.

```
from braket.devices import Devices
from braket.aws import DirectReservation
import pennylane as qml

dev = qml.device("braket.aws.qubit", device_arn=Devices.IonQ.Aria1.value, wires=2,
    shots=10)

@qml.qnode(dev)
def bell_state():
    qml.Hadamard(wires=0)
    qml.CNOT(wires=[0, 1])
    return qml.probs(wires=[0, 1])
```

```
# run the circuit in a reservation
with DirectReservation(Devices.IonQ.Aria1, reservation_arn="<my_reservation_arn>"):
    probs = bell_state()
```

Configuration manuelle du contexte de réservation

Vous pouvez également définir manuellement le contexte de réservation à l'aide du code suivant.

```
# set reservation context
reservation_context = DirectReservation(device,
    reservation_arn="<my_reservation_arn>").start()

# run circuit during reservation
task = device.run(bell, shots=100)
```

C'est idéal pour les blocs-notes Jupyter où le contexte peut être exécuté dans la première cellule et où toutes les tâches suivantes seront exécutées dans la réservation.

Note

La cellule contenant l'.start() appel ne doit être exécutée qu'une seule fois.

Pour revenir au mode à la demande : redémarrez le bloc-notes Jupyter ou appelez le numéro suivant pour revenir au mode à la demande.

```
reservation_context.stop() # unset reservation context
```

Note

Les réservations ont une heure de début et de fin prédéterminées (voir [Création d'une réservation](#)). Les reservation_context.stop() méthodes reservation_context.start() et ne commencent ni ne terminent une réservation. Au lieu de cela, lorsque le contexte est actif, toutes les tâches quantiques que vous créez seront associées à votre réservation et ne seront exécutées que pendant votre réservation planifiée. Le contexte de réservation n'a aucun effet sur l'heure de réservation prévue.

Transmettez explicitement l'ARN de réservation lors de la création de la tâche

Une autre méthode pour créer des tâches lors d'une réservation consiste à transmettre explicitement l'ARN de la réservation lors de l'appel `device.run()`.

```
task = device.run(bell, shots=100, reservation_arn="<my_reservation_arn>")
```

Cette méthode associe directement la tâche quantique à l'ARN de réservation, garantissant ainsi son exécution pendant la période réservée. Pour cette option, ajoutez l'ARN de réservation à chaque tâche que vous prévoyez d'exécuter lors d'une réservation. Notez toutefois que lorsque vous utilisez des bibliothèques tierces telles que Qiskit ou PennyLane, il peut être difficile de s'assurer que les tâches soumises utilisent le bon ARN de réservation. C'est pourquoi il est recommandé d'utiliser le gestionnaire de DirectReservation contexte.

Lorsque vous utilisez directement boto3, transmettez l'ARN de réservation sous forme d'association lors de la création d'une tâche.

```
import boto3

braket_client = boto3.client("braket")

kwargs["associations"] = [
    {
        "arn": "<my_reservation_arn>",
        "type": "RESERVATION_TIME_WINDOW_ARN",
    }
]

response = braket_client.create_quantum_task(**kwargs)
```

Exécution de tâches hybrides lors d'une réservation

Une fois que vous avez une fonction Python à exécuter en tant que tâche hybride, vous pouvez exécuter la tâche hybride dans une réservation en transmettant l'argument du `reservation_arn` mot clé. Toutes les tâches de la tâche hybride utilisent l'ARN de réservation. Il est important de noter que le travail hybride `reservation_arn` n'active le calcul classique qu'une fois que votre réservation commence.

 Note

Une tâche hybride exécutée pendant une réservation n'exécute avec succès que des tâches quantiques sur l'appareil réservé. Toute tentative d'utilisation d'un autre appareil Braket à la demande entraînera une erreur. Si vous devez exécuter des tâches à la fois sur un simulateur à la demande et sur l'appareil réservé dans le cadre d'une même tâche hybride, `DirectReservation` utilisez-les plutôt.

Le code suivant montre comment exécuter une tâche hybride lors d'une réservation.

```
from braket.aws import AwsDevice
from braket.devices import Devices
from braket.jobs import get_job_device_arn, hybrid_job

@hybrid_job(device=Devices.IonQ.Aria1, reservation_arn="<my_reservation_arn>")
def example_hybrid_job():
    # declare AwsDevice within the hybrid job
    device = AwsDevice(get_job_device_arn())
    bell = Circuit().h(0).cnot(0, 1)

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

Pour les tâches hybrides qui utilisent un script Python (voir la section sur la [création de votre premier Job hybride](#) dans le guide du développeur), vous pouvez les exécuter dans le cadre de la réservation en transmettant l'argument du `reservation_arn` mot-clé lors de la création de la tâche.

```
from braket.aws import AwsQuantumJob
from braket.devices import Devices

job = AwsQuantumJob.create(
    Devices.IonQ.Aria1,
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    reservation_arn="<my_reservation_arn>"
)
```

Que se passe-t-il à la fin de votre réservation

Une fois votre réservation terminée, vous ne disposez plus d'un accès dédié à l'appareil. Toutes les charges de travail restantes mises en file d'attente avec cette réservation sont automatiquement annulées.

Note

Toute tâche dont le RUNNING statut était en cours à la fin de la réservation est annulée. Nous vous recommandons d'utiliser des [points de contrôle pour enregistrer et redémarrer les tâches](#) à votre convenance.

Une réservation en cours, par exemple après le début et avant la fin de la réservation, ne peut pas être prolongée car chaque réservation représente un accès autonome à un appareil dédié. Par exemple, deux back-to-back réservations sont considérées comme distinctes et toutes les tâches en attente depuis la première réservation sont automatiquement annulées. Ils ne reprennent pas lors de la deuxième réservation.

Note

Les réservations représentent un accès à un appareil dédié à votre AWS compte. Même si l'appareil reste inactif, aucun autre client ne peut l'utiliser. Par conséquent, vous êtes facturé pour la durée du temps réservé, quel que soit le temps utilisé.

Annuler ou replanifier une réservation existante

Vous pouvez annuler votre réservation au moins 48 heures avant l'heure de début prévue de la réservation. Pour annuler, répondez à l'e-mail de confirmation de réservation que vous avez reçu avec votre demande d'annulation.

Pour replanifier, vous devez annuler votre réservation existante, puis en créer une nouvelle.

Techniques d'atténuation des erreurs

L'atténuation des erreurs quantiques est un ensemble de techniques visant à réduire les effets des erreurs dans les ordinateurs quantiques.

Les appareils quantiques sont soumis au bruit ambiant qui dégrade la qualité des calculs effectués. Alors que l'informatique quantique tolérante aux pannes promet une solution à ce problème, les dispositifs quantiques actuels sont limités par le nombre de qubits et les taux d'erreur relativement élevés. Pour y remédier à court terme, les chercheurs étudient des méthodes permettant d'améliorer la précision du calcul quantique bruité. Cette approche, connue sous le nom d'atténuation des erreurs quantiques, implique l'utilisation de diverses techniques pour extraire le meilleur signal des données de mesure bruyantes.

Dans cette section :

- [Techniques d'atténuation des erreurs sur IonQ les appareils](#)

Techniques d'atténuation des erreurs sur IonQ les appareils

L'atténuation des erreurs implique l'exécution de plusieurs circuits physiques et la combinaison de leurs mesures pour obtenir un meilleur résultat.

Note

Pour tous les IonQ appareils : lorsque vous utilisez un modèle à la demande, il existe une limite d'un million de prises [de vue](#) et un minimum de 2 500 prises de vue pour les tâches de [réduction des erreurs](#). Pour une réservation directe, il n'y a pas de limite de prises de vue, et un minimum de 500 prises de vue est requis pour les tâches de réduction des erreurs.

Débiaiser

IonQles appareils disposent d'une méthode d'atténuation des erreurs appelée debiasing.

Le débiais permet de cartographier un circuit en plusieurs variantes qui agissent sur différentes permutations de qubits ou avec différentes décompositions de portes. Cela réduit l'effet des erreurs systématiques telles que les surrotations des portes ou un seul qubit défectueux en utilisant différentes implémentations d'un circuit qui pourraient autrement biaiser les résultats de mesure. Cela se fait au détriment des frais supplémentaires liés au calibrage de plusieurs qubits et portes.

Pour plus d'informations sur le débiais, voir [Améliorer les performances des ordinateurs quantiques grâce à la symétrisation](#).

 Note

L'utilisation du débiaising nécessite un minimum de 2 500 prises de vue.

Vous pouvez exécuter une tâche quantique avec débiais sur un IonQ appareil à l'aide du code suivant :

```
from braket.aws import AwsDevice
from braket.circuits import Circuit
from braket.error_mitigation import Debias

# choose an IonQ device
device = AwsDevice("arn:aws:braket:us-east-1::device/qpu/ionq/Forte-Enterprise-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
```

Lorsque la tâche quantique est terminée, vous pouvez voir les probabilités de mesure et tous les types de résultats de la tâche quantique. Les probabilités de mesure et les dénombrements de toutes les variantes sont agrégés en une seule distribution. Tous les types de résultats spécifiés dans le circuit, tels que les valeurs attendues, sont calculés à l'aide des nombres de mesures agrégés.

Netteté

Vous pouvez également accéder aux probabilités de mesure calculées à l'aide d'une autre stratégie de post-traitement appelée affûtage. La netteté compare les résultats de chaque variante et élimine les prises de vue incohérentes, favorisant ainsi le résultat de mesure le plus probable pour toutes les variantes. Pour plus d'informations, voir [Améliorer les performances des ordinateurs quantiques grâce à la symétrisation](#).

Il est important de noter que l'affinement suppose que la forme de la distribution de sortie est clairesmée, avec peu d'états à probabilité élevée et de nombreux états à probabilité nulle. Cela peut fausser la distribution de probabilité si cette hypothèse n'est pas valide.

Vous pouvez accéder aux probabilités à partir d'une distribution affinée dans le `additional_metadata` champ du SDK `GateModelTaskResult` Braket Python. Notez que

l'affûtage ne renvoie pas le nombre de mesures, mais renvoie une distribution de probabilité normalisée à nouveau. L'extrait de code suivant montre comment accéder à la distribution après l'affinage.

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

Travailler avec Amazon Braket Hybrid Jobs

Amazon Braket Hybrid Jobs vous permet d'exécuter des algorithmes hybrides quantiques-classiques nécessitant à la fois des AWS ressources classiques et des unités de traitement quantiques (). QPUs Hybrid Jobs est conçu pour générer les ressources classiques demandées, exécuter votre algorithme et libérer les instances une fois celles-ci terminées, afin que vous ne payiez que pour ce que vous utilisez.

Hybrid Jobs est idéal pour les algorithmes itératifs de longue durée qui impliquent l'utilisation de ressources informatiques classiques et de ressources informatiques quantiques. Avec Hybrid Jobs, après avoir soumis votre algorithme à exécuter, Braket exécutera votre algorithme dans un environnement conteneurisé évolutif. Une fois l'algorithme terminé, vous pouvez récupérer les résultats.

De plus, les tâches quantiques créées à partir d'une tâche hybride bénéficient d'une mise en file d'attente plus prioritaire vers le périphérique QPU cible. Cette hiérarchisation garantit que vos calculs quantiques sont traités et exécutés avant les autres tâches en attente. Cela est particulièrement avantageux pour les algorithmes hybrides itératifs, dans lesquels les résultats d'une tâche quantique dépendent des résultats de tâches quantiques antérieures. [Des exemples de tels algorithmes incluent l'algorithme d'optimisation approximative quantique \(QAOA\), le solveur quantique variationnel ou l'apprentissage automatique quantique.](#) Vous pouvez également suivre la progression de votre algorithme en temps quasi réel, ce qui vous permet de suivre les coûts, le budget ou des indicateurs personnalisés tels que les pertes d'entraînement ou les valeurs attendues.

Vous pouvez accéder aux jobs hybrides dans Braket en utilisant :

- Le [SDK Amazon Braket Python](#).
- La [console Amazon Braket](#).
- L'Amazon BraketAPI.

Dans cette section :

- [Quand utiliser Amazon Braket Hybrid Jobs](#)
- [Exécution d'une tâche hybride avec Amazon Braket Hybrid Jobs](#)
- [Concepts clés pour les emplois hybrides](#)
- [Conditions préalables](#)

- [Créez un Job hybride](#)
- [Annuler un Job hybride](#)
- [Personnalisation de votre Hybrid Job](#)
- [Utilisation PennyLane avec Amazon Braket](#)
- [Utilisation de CUDA-Q avec Amazon Braket](#)

Quand utiliser Amazon Braket Hybrid Jobs

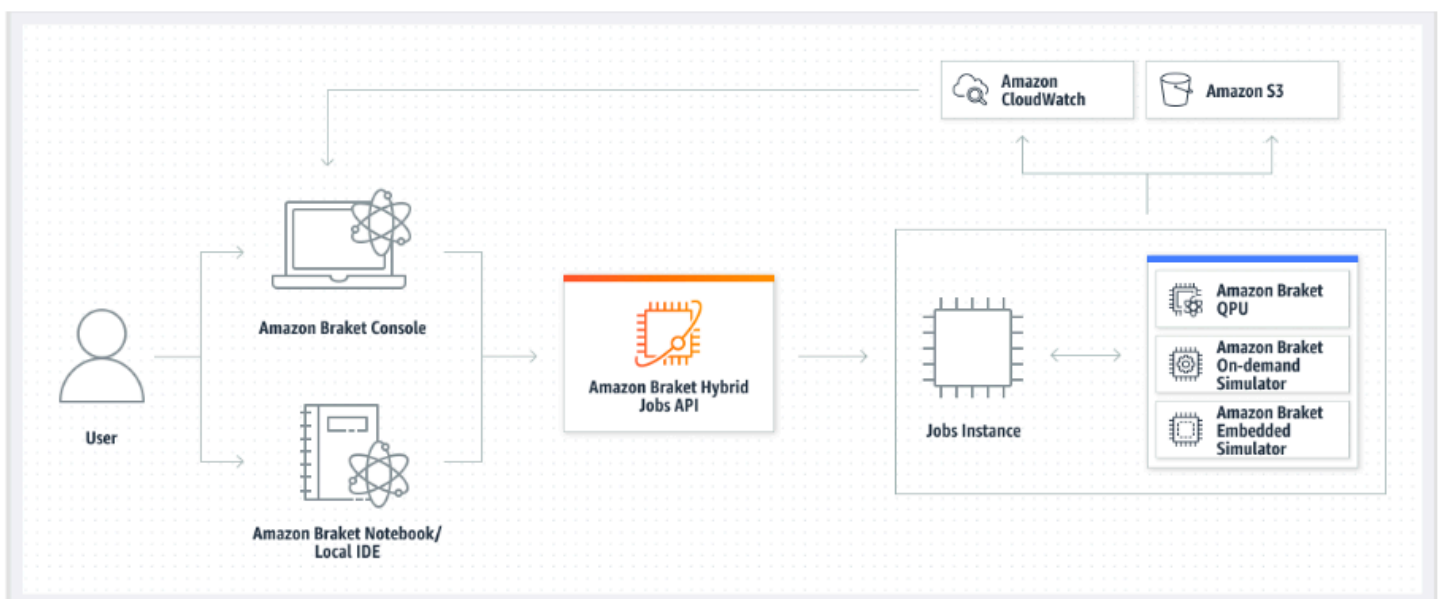
Amazon Braket Hybrid Jobs vous permet d'exécuter des algorithmes hybrides classiques et quantiques, tels que le variational Quantum Eigensolver (VQE) et l'algorithme d'optimisation quantique approximatif (QAOA), qui combinent des ressources de calcul classiques avec des dispositifs informatiques quantiques afin d'optimiser les performances des systèmes quantiques actuels. Amazon Braket Hybrid Jobs offre trois avantages principaux :

1. **Performances** : Amazon Braket Hybrid Jobs offre de meilleures performances que l'exécution d'algorithmes hybrides depuis votre propre environnement. Pendant que votre tâche est en cours d'exécution, elle dispose d'un accès prioritaire au QPU cible sélectionné. Les tâches de votre travail sont exécutées avant les autres tâches mises en file d'attente sur l'appareil. Cela se traduit par des temps d'exécution plus courts et plus prévisibles pour les algorithmes hybrides. Amazon Braket Hybrid Jobs prend également en charge la compilation paramétrique. Vous pouvez soumettre un circuit en utilisant des paramètres libres et Braket le compile une seule fois, sans qu'il soit nécessaire de le recompiler pour les mises à jour ultérieures des paramètres du même circuit, ce qui se traduit par des temps d'exécution encore plus rapides.
2. **Commodité** : Amazon Braket Hybrid Jobs simplifie la configuration et la gestion de votre environnement informatique et le fait fonctionner pendant que votre algorithme hybride s'exécute. Il vous suffit de fournir votre script d'algorithme et de sélectionner un dispositif quantique (une unité de traitement quantique ou un simulateur) sur lequel exécuter. Amazon Braket attend que l'appareil cible soit disponible, active les ressources classiques, exécute la charge de travail dans des environnements de conteneurs prédéfinis, renvoie les résultats à Amazon Simple Storage Service (Amazon S3) et libère les ressources de calcul.
3. **Métriques** : Amazon Braket Hybrid Jobs fournit des on-the-fly informations sur les algorithmes en cours d'exécution et fournit des métriques d'algorithmes personnalisables en temps quasi réel à Amazon CloudWatch et à la console Amazon Braket afin que vous puissiez suivre la progression de vos algorithmes.

Exécution d'une tâche hybride avec Amazon Braket Hybrid Jobs

Pour exécuter une tâche hybride avec Amazon Braket Hybrid Jobs, vous devez d'abord définir votre algorithme. Vous pouvez le définir en écrivant le script de l'algorithme et, éventuellement, d'autres fichiers de dépendance à l'aide du [SDK Amazon Braket Python](#) ou [PennyLane](#). Si vous souhaitez utiliser d'autres bibliothèques (open source ou propriétaires), vous pouvez définir votre propre image de conteneur personnalisée à l'aide de Docker, qui inclut ces bibliothèques. Pour plus d'informations, consultez [Bring your own container \(BYOC\)](#).

Dans les deux cas, vous créez ensuite une tâche hybride à l'aide d'Amazon BraketAPI, dans laquelle vous fournissez votre script d'algorithme ou votre conteneur, sélectionnez le dispositif quantique cible que la tâche hybride doit utiliser, puis choisissez parmi une variété de paramètres facultatifs. Les valeurs par défaut fournies pour ces paramètres facultatifs fonctionnent dans la majorité des cas d'utilisation. Pour que l'appareil cible exécute votre Hybrid Job, vous avez le choix entre un QPU, un simulateur à la demande (tel queSV1, DM1 ouTN1) ou l'instance de tâche hybride classique elle-même. Avec un simulateur à la demande ou un QPU, votre conteneur de tâches hybride envoie des appels d'API à un appareil distant. Avec les simulateurs intégrés, le simulateur est intégré dans le même conteneur que votre script d'algorithme. Les [simulateurs de foudre](#) PennyLane sont intégrés au conteneur de tâches hybrides prédéfini par défaut que vous pouvez utiliser. Si vous exécutez votre code à l'aide d'un PennyLane simulateur intégré ou d'un simulateur personnalisé, vous pouvez spécifier un type d'instance ainsi que le nombre d'instances que vous souhaitez utiliser. Consultez la [page de tarification d'Amazon Braket](#) pour connaître les coûts associés à chaque choix.



Si votre appareil cible est un simulateur intégré ou à la demande, Amazon Braket commence immédiatement à exécuter la tâche hybride. Il lance l'instance de travail hybride (vous pouvez personnaliser le type d'instance dans l'API `call`), exécute votre algorithme, écrit les résultats sur Amazon S3 et libère vos ressources. Cette mise à disposition des ressources garantit que vous ne payez que pour ce que vous utilisez.

Le nombre total de tâches hybrides simultanées par unité de traitement quantique (QPU) est limité. Aujourd'hui, une seule tâche hybride peut être exécutée sur un QPU à la fois. Les files d'attente sont utilisées pour contrôler le nombre de tâches hybrides autorisées à s'exécuter afin de ne pas dépasser la limite autorisée. Si votre équipement cible est un QPU, votre tâche hybride entre d'abord dans la file d'attente des tâches du QPU sélectionné. Amazon Braket lance l'instance de tâche hybride requise et exécute votre tâche hybride sur l'appareil. Pendant toute la durée de votre algorithme, votre tâche hybride bénéficie d'un accès prioritaire, ce qui signifie que les tâches quantiques de votre tâche hybride devancent les autres tâches quantiques Braket mises en file d'attente sur l'appareil, à condition que les tâches quantiques de la tâche soient soumises au QPU une fois toutes les quelques minutes. Une fois votre projet hybride terminé, les ressources sont libérées, ce qui signifie que vous ne payez que pour ce que vous utilisez.

Note

Les appareils sont régionaux et votre tâche hybride s'exécute de la même manière Région AWS que votre appareil principal.

Dans les scénarios cibles du simulateur et du QPU, vous avez la possibilité de définir des métriques d'algorithme personnalisées, telles que l'énergie de votre hamiltonien, dans le cadre de votre algorithme. Ces statistiques sont automatiquement communiquées à Amazon CloudWatch et, à partir de là, elles s'affichent en temps quasi réel dans la console Amazon Braket.

Note

Si vous souhaitez utiliser une instance basée sur un GPU, veuillez à utiliser l'un des simulateurs basés sur le GPU disponibles avec les simulateurs intégrés sur Braket (par exemple, `lightning.gpu`). Si vous choisissez l'un des simulateurs intégrés basés sur le processeur (par exemple, `oubraket:default-simulator`), `lightning.qubit`, le GPU ne sera pas utilisé et vous risquez d'encourir des coûts inutiles.

Concepts clés pour les emplois hybrides

Cette section explique les concepts clés de la `AwsQuantumJob.create` fonction fournie par le SDK Amazon Braket Python et le mappage à la structure du fichier conteneur.

Outre le ou les fichiers qui constituent votre script d'algorithme complet, votre tâche hybride peut comporter des entrées et des sorties supplémentaires. Lorsque votre tâche hybride démarre, Amazon Braket copie les entrées fournies dans le cadre de la création de la tâche hybride dans le conteneur qui exécute le script d'algorithme. Lorsque la tâche hybride est terminée, toutes les sorties définies au cours de l'algorithme sont copiées vers l'emplacement Amazon S3 spécifié.

Note

Les métriques de l'algorithme sont rapportées en temps réel et ne suivent pas cette procédure de sortie.

Amazon Braket fournit également plusieurs variables d'environnement et fonctions d'assistance pour simplifier les interactions avec les entrées et sorties des conteneurs. Pour plus d'informations, consultez le [package `braket.jobs` dans le SDK](#) Amazon Braket.

Dans cette section :

- [Inputs](#)
- [Sorties](#)
- [Variables d'environnement](#)
- [Fonctions d'assistance](#)

Inputs

Données d'entrée : Les données d'entrée peuvent être fournies à l'algorithme hybride en spécifiant le fichier de données d'entrée, qui est configuré sous forme de dictionnaire, avec l'`input_data` argument. L'utilisateur définit l'`input_data` argument au sein de la `AwsQuantumJob.create` fonction dans le SDK. Cela copie les données d'entrée dans le système de fichiers conteneur à l'emplacement indiqué par la variable d'environnement "`AMZN_BRAKET_INPUT_DIR`". Pour quelques exemples de la façon dont les données d'entrée sont utilisées dans un algorithme hybride, consultez le [QAOA avec Amazon Braket](#)

[Hybrid Jobs PennyLane et l'apprentissage automatique quantique dans les blocs-notes Jupyter d'Amazon Braket](#) Hybrid Jobs.

Note

Lorsque les données d'entrée sont volumineuses (> 1 Go), le temps d'attente est long avant que la tâche hybride ne soit soumise. Cela est dû au fait que les données d'entrée locales seront d'abord téléchargées dans un compartiment S3, puis le chemin S3 sera ajouté à la demande de travail hybride et, enfin, la demande de travail hybride est soumise au service Braket.

Hyperparamètres : si vous les transmettez `hyperparameters`, ils sont disponibles sous la variable `"AMZN_BRAKET_HP_FILE"` d'environnement.

Note

[Pour plus d'informations sur la façon de créer des hyperparamètres et des données d'entrée, puis de transmettre ces informations au script de tâche hybride, consultez la section Utiliser les hyperparamètres et cette page github.](#)

Points de contrôle : pour spécifier `job-arn` le point de contrôle que vous souhaitez utiliser dans une nouvelle tâche hybride, utilisez la `copy_checkpoints_from_job` commande. Cette commande copie les données du point `checkpoint_configs3Uri` de contrôle vers la nouvelle tâche hybride, les rendant disponibles sur le chemin indiqué par la variable d'environnement `AMZN_BRAKET_CHECKPOINT_DIR` pendant l'exécution de la tâche. La valeur par défaut est `None`, ce qui signifie que les données de point de contrôle d'une autre tâche hybride ne seront pas utilisées dans la nouvelle tâche hybride.

Sorties

Tâches quantiques : les résultats des tâches quantiques sont stockés dans l'emplacement S3 `s3://amazon-braket-<region>-<accountID>/jobs/<job-name>/tasks`.

Résultats du job : tout ce que votre script d'algorithme enregistre dans le répertoire indiqué par la variable d'environnement `"AMZN_BRAKET_JOB_RESULTS_DIR"` est copié vers l'emplacement S3 spécifié dans `output_data_config`. Si la valeur n'est pas spécifiée, elle est définie par défaut

surs3://amazon-braket-<region>-<accountID>/jobs/<job-name>/<timestamp>/data. Nous fournissons la fonction d'assistance du SDK **save_job_result**, que vous pouvez utiliser pour stocker facilement les résultats sous forme de dictionnaire lorsque vous les appelez à partir de votre script d'algorithme.

Points de contrôle : Si vous souhaitez utiliser des points de contrôle, vous pouvez les enregistrer dans le répertoire indiqué par la variable d'environnement. "AMZN_BRAKET_CHECKPOINT_DIR" Vous pouvez également utiliser la fonction d'assistance du SDK à la place. `save_job_checkpoint`

Métriques d'algorithme : vous pouvez définir des métriques d'algorithme dans le cadre de votre script d'algorithme qui sont envoyées à Amazon CloudWatch et affichées en temps réel dans la console Amazon Braket pendant l'exécution de votre tâche hybride. Pour un exemple d'utilisation des métriques d'algorithmes, consultez [Utiliser les tâches hybrides Amazon Braket pour exécuter un algorithme QAOA](#).

Pour plus d'informations sur la sauvegarde des résultats de vos tâches, consultez la section [Enregistrer vos résultats](#) dans la documentation relative aux tâches hybrides.

Variables d'environnement

Amazon Braket fournit plusieurs variables d'environnement pour simplifier les interactions avec les entrées et sorties des conteneurs. Le code suivant répertorie les variables environnementales utilisées par Braket.

- `AMZN_BRAKET_INPUT_DIR`— Le répertoire des données d'entrée `opt/braket/input/data`.
- `AMZN_BRAKET_JOB_RESULTS_DIR`— Le répertoire de sortie dans lequel `opt/braket/model` écrit les résultats de la tâche.
- `AMZN_BRAKET_JOB_NAME`— Le nom de la tâche.
- `AMZN_BRAKET_CHECKPOINT_DIR`— Le répertoire des points de contrôle.
- `AMZN_BRAKET_HP_FILE`— Le fichier contenant les hyperparamètres.
- `AMZN_BRAKET_DEVICE_ARN`— L'ARN (nom de AWS ressource) de l'appareil.
- `AMZN_BRAKET_OUT_S3_BUCKET`— Le compartiment Amazon S3 de sortie, tel que spécifié dans la `CreateJob` `demandeOutputDataConfig`.
- `AMZN_BRAKET_SCRIPT_ENTRY_POINT`— Le point d'entrée tel que spécifié dans la `CreateJob` `demandeScriptModeConfig`.
- `AMZN_BRAKET_SCRIPT_COMPRESSION_TYPE`— Le type de compression tel que spécifié dans la `CreateJob` `demandeScriptModeConfig`.

- `AMZN_BRAKET_SCRIPT_S3_URI`— L'emplacement du script de l'utilisateur sur Amazon S3, tel que spécifié dans la `CreateJob` demande `ScriptModeConfig`.
- `AMZN_BRAKET_TASK_RESULTS_S3_URI`— L'emplacement Amazon S3 où le SDK stockerait les résultats de la tâche quantique par défaut pour la tâche.
- `AMZN_BRAKET_JOB_RESULTS_S3_PATH`— L'emplacement Amazon S3 où les résultats de la tâche seraient stockés, comme indiqué dans les `CreateJob` demandes `OutputDataConfig`.
- `AMZN_BRAKET_JOB_TOKEN`— Chaîne à transmettre au `jobToken` paramètre pour `CreateQuantumTask` les tâches quantiques créées dans le conteneur de tâches.

Fonctions d'assistance

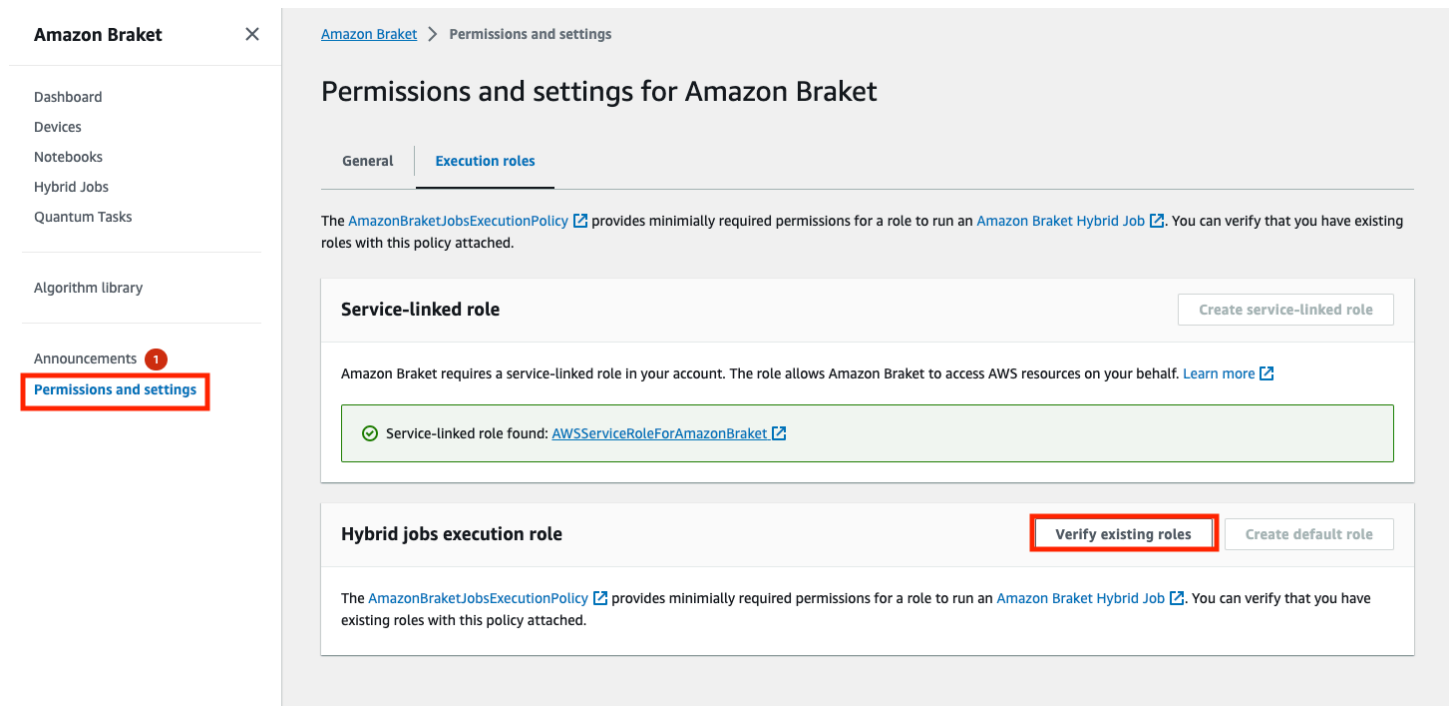
Amazon Braket propose plusieurs fonctions d'assistance pour simplifier les interactions avec les entrées et sorties des conteneurs. Ces fonctions d'assistance seraient appelées depuis le script d'algorithme utilisé pour exécuter votre Hybrid Job. L'exemple suivant montre comment les utiliser.

```
from braket.jobs import get_checkpoint_dir, get_hyperparameters, get_input_data_dir,
    get_job_device_arn, get_job_name, get_results_dir, save_job_result,
    save_job_checkpoint, load_job_checkpoint

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(result_data='data') # Save hybrid job results
save_job_checkpoint(checkpoint_data={'key': 'value'}) # Save a checkpoint
load_job_checkpoint() # Load a previously saved checkpoint
```

Conditions préalables

Avant d'exécuter votre première tâche hybride, vous devez vous assurer que vous disposez des autorisations suffisantes pour effectuer cette tâche. Pour déterminer si vous disposez des autorisations appropriées, sélectionnez **Autorisations** dans le menu situé sur le côté gauche de la console Braket. La page de gestion des autorisations pour Amazon Braket vous permet de vérifier si l'un de vos rôles existants dispose d'autorisations suffisantes pour exécuter votre tâche hybride ou vous guide dans la création d'un rôle par défaut pouvant être utilisé pour exécuter votre tâche hybride si vous ne possédez pas déjà un tel rôle.



Amazon Braket ×

Dashboard
Devices
Notebooks
Hybrid Jobs
Quantum Tasks

Algorithm library

Announcements 1
Permissions and settings

Amazon Braket > Permissions and settings

Permissions and settings for Amazon Braket

General | **Execution roles**

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

Service-linked role [Create service-linked role](#)

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

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

Hybrid jobs execution role [Verify existing roles](#) [Create default role](#)

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

Pour vérifier que vous disposez de rôles dotés d'autorisations suffisantes pour exécuter une tâche hybride, cliquez sur le bouton Vérifier le rôle existant. Si vous le faites, vous recevez un message indiquant que les rôles ont été trouvés. Pour voir les noms des rôles et leur rôle ARNs, cliquez sur le bouton Afficher les rôles.

Amazon Braket

×

Dashboard

Devices

Notebooks

Hybrid Jobs

Quantum Tasks

Algorithm library

Announcements 1

Permissions and settings

Amazon Braket > Permissions and settings

Permissions and settings for Amazon Braket

General | Execution roles

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

Service-linked role

Create service-linked role

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

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

Hybrid jobs execution role

Verify existing roles

Create default role

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

✓ Roles were found with sufficient permissions to execute hybrid jobs.

▼ Show roles

Role name	Role ARN
AmazonBraketJobsExecutionRole	arn:aws:iam::260818742045:role/service-role/AmazonBraketJobsExecutionRole

Si vous ne disposez pas d'un rôle doté des autorisations suffisantes pour exécuter une tâche hybride, vous recevez un message indiquant qu'aucun rôle de ce type n'a été trouvé. Cliquez sur le bouton Créer un rôle par défaut pour obtenir un rôle avec des autorisations suffisantes.

Amazon Braket ✕

- Dashboard
- Devices
- Notebooks
- Hybrid Jobs
- Quantum Tasks

- Algorithm library

Announcements 1

Permissions and settings

[Amazon Braket](#) > Permissions and settings

Permissions and settings for Amazon Braket

General | **Execution roles**

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

Service-linked role Create service-linked role

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

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

Hybrid jobs execution role Verify existing roles Create default role

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

❗ No roles found with the AmazonBraketJobsExecutionPolicy attached and braket.amazonaws.com as a trusted entity in IAM.

Si le rôle a été créé avec succès, vous recevez un message le confirmant.

Amazon Braket ✕

- Dashboard
- Devices
- Notebooks
- Hybrid Jobs
- Quantum Tasks

- Algorithm library

Announcements 1

Permissions and settings

[Amazon Braket](#) > Permissions and settings

Permissions and settings for Amazon Braket

General | **Execution roles**

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

Service-linked role Create service-linked role

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

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

Hybrid jobs execution role Verify existing roles Create default role

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

✔ Created [AmazonBraketJobsExecutionRole](#) successfully.

Si vous n'êtes pas autorisé à effectuer cette demande, l'accès vous sera refusé. Dans ce cas, contactez votre AWS administrateur interne.

The screenshot shows the 'Permissions management for Amazon Braket' page. At the top, there's a breadcrumb 'Amazon Braket > Permissions'. Below the title, a paragraph explains that resources like notebooks or jobs can be configured with IAM roles. There are two buttons: 'Verify existing roles' and 'Create default role'. A section titled 'Jobs' contains a paragraph stating that Amazon Braket jobs require the 'AmazonBraketJobsExecutionPolicy' role. Below this, a red-bordered box highlights an 'AccessDenied' error message: 'User: arn:aws:sts::012345678912:assumed-role/SampleRoleName/username is not authorized to perform: iam:ListAttachedRolePolicies on resource: role AmazonBraketJobsExecutionRole with an explicit deny'.

Créez un Job hybride

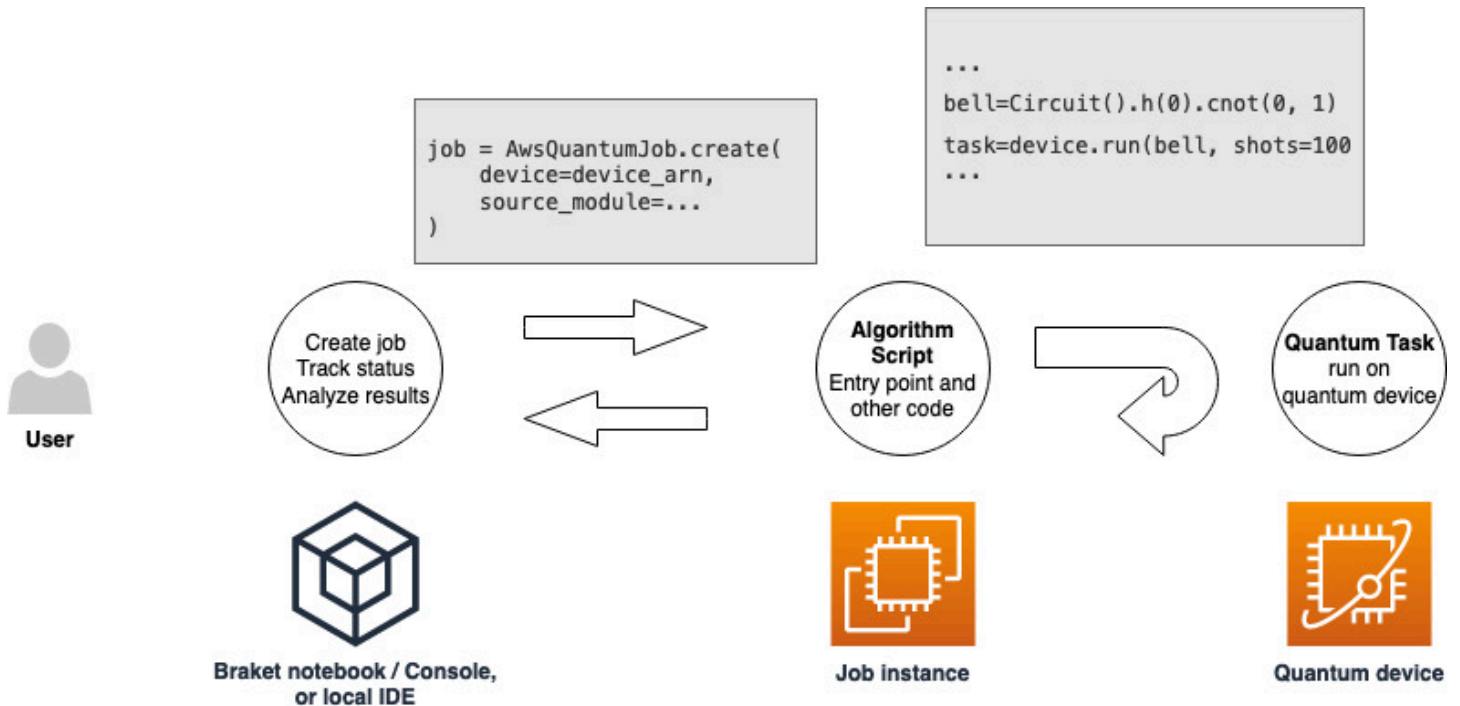
Cette section explique comment créer un Job hybride à l'aide d'un script Python. Vous pouvez également créer une tâche hybride à partir du code Python local, tel que votre environnement de développement intégré (IDE) préféré ou un bloc-notes Braket, voir [Exécutez votre code local en tant que tâche hybride](#).

Dans cette section :

- [Créez et exécutez](#)
- [Surveillez vos résultats](#)
- [Enregistrez vos résultats](#)
- [Utilisation des points de contrôle](#)
- [Exécutez votre code local en tant que tâche hybride](#)
- [Utilisation de l'API avec des tâches hybrides](#)
- [Création et débogage d'une tâche hybride en mode local](#)

Créez et exécutez

Une fois que vous avez un rôle autorisé à exécuter une tâche hybride, vous êtes prêt à continuer. L'élément clé de votre première tâche hybride Braket est le script d'algorithme. Il définit l'algorithme que vous souhaitez exécuter et contient les tâches logiques et quantiques classiques qui font partie de votre algorithme. Outre votre script d'algorithme, vous pouvez fournir d'autres fichiers de dépendance. Le script d'algorithme ainsi que ses dépendances sont appelés module source. Le point d'entrée définit le premier fichier ou la première fonction à exécuter dans votre module source au démarrage de la tâche hybride.



Tout d'abord, considérez l'exemple de base suivant d'un script d'algorithme qui crée cinq états de cloche et imprime les résultats de mesure correspondants.

```

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!")
```

Enregistrez ce fichier sous le nom `algorithm_script.py` dans votre répertoire de travail actuel sur votre bloc-notes Braket ou dans votre environnement local. Le fichier `algorithm_script.py` a `start_here()` pour point d'entrée prévu.

Créez ensuite un fichier Python ou un bloc-notes Python dans le même répertoire que le fichier `algorithm_script.py`. Ce script lance la tâche hybride et gère tout traitement asynchrone, tel que l'impression du statut ou des principaux résultats qui nous intéressent. Ce script doit au minimum spécifier votre script de tâche hybride et votre appareil principal.

Note

Pour plus d'informations sur la façon de créer un bloc-notes Braket ou de télécharger un fichier, tel que le fichier `algorithm_script.py`, dans le même répertoire que les blocs-notes, consultez [Exécuter votre premier circuit à l'aide du SDK Amazon Braket Python](#)

Dans ce premier cas de base, vous ciblez un simulateur. Quel que soit le type d'appareil quantique que vous ciblez, qu'il s'agisse d'un simulateur ou d'une véritable unité de traitement quantique (QPU), le périphérique que vous spécifiez `device` dans le script suivant est utilisé pour planifier la tâche hybride et est disponible pour les scripts d'algorithme en tant que variable `AMZN_BRAKET_DEVICE_ARN` d'environnement.

Note

Vous ne pouvez utiliser que les appareils disponibles dans le cadre Région AWS de votre tâche hybride. Le SDK Amazon Braket le sélectionne automatiquement. Région AWS Par exemple, une tâche hybride dans `us-east-1` peut lonQ utiliser `SV1`, `TN1` et des appareils `DM1`, mais pas des appareils. Rigetti

Si vous choisissez un ordinateur quantique plutôt qu'un simulateur, Braket planifie vos tâches hybrides pour exécuter toutes leurs tâches quantiques avec un accès prioritaire.

```
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
)
```

Le paramètre `wait_until_complete=True` définit un mode détaillé afin que votre tâche imprime le résultat de la tâche réelle pendant son exécution. Vous devriez voir une sortie similaire à l'exemple suivant.

```
Initializing Braket Job: arn:aws:braket:us-west-2:111122223333:job/braket-job-
default-123456789012
Job queue position: 1
Job queue position: 1
Job queue position: 1
.....
.
.
.
Beginning Setup
Checking for Additional Requirements
Additional Requirements Check Finished
Running Code As Process
Test job started!
Counter({'00': 58, '11': 42})
Counter({'00': 55, '11': 45})
Counter({'11': 51, '00': 49})
Counter({'00': 56, '11': 44})
Counter({'11': 56, '00': 44})
Test job completed!
Code Run Finished
2025-09-24 23:13:40,962 sagemaker-training-toolkit INFO      Reporting training SUCCESS
```


Note

Vous pouvez également utiliser votre module personnalisé avec la méthode [AwsQuantumJob.create](#) en transmettant son emplacement (soit le chemin d'accès à un répertoire ou à un fichier local, soit l'URI S3 d'un fichier tar.gz). [Pour un exemple pratique, consultez le fichier Parallelize_Training_for_qml.ipynb dans le dossier des tâches hybrides du référentiel Github des exemples Amazon Braket.](#)

Surveillez vos résultats

Vous pouvez également accéder à la sortie du journal depuis Amazon CloudWatch. Pour ce faire, accédez à l'onglet Groupes de journaux dans le menu de gauche de la page détaillée de la tâche, sélectionnez le groupe de journaux `aws/braket/jobs`, puis choisissez le flux de journaux contenant le nom de la tâche. Dans l'exemple ci-dessus, il s'agit de `braket-job-default-1631915042705/algo-1-1631915190`.

The screenshot shows the Amazon CloudWatch console interface. On the left sidebar, the 'Log groups' option is highlighted. The main panel displays the breadcrumb navigation: 'CloudWatch > Log groups > /aws/braket/jobs > JobTest-autograd-1636588595/algo-1-1636588740'. Below this, the 'Log events' section is visible, showing a list of log entries with columns for 'Timestamp' and 'Message'. The messages are truncated and point to various test files within the Braket SDK, such as `test_gates.py`, `test_instruction.py`, `test_moments.py`, `test_noise.py`, `test_noise_helpers.py`, `test_observable.py`, `test_observables.py`, `test_quantum_operator.py`, `test_quantum_operator_helpers.py`, `test_qubit.py`, `test_qubit_set.py`, `test_result_type.py`, `test_result_types.py`, `test_devices.py`, `test_local_simulator.py`, `test_jobs.py`, and `test_local_job.py`.

Vous pouvez également consulter l'état de la tâche hybride dans la console en sélectionnant la page Tâches hybrides, puis en choisissant Paramètres.

Amazon Braket

Dashboard
Devices
Notebooks
Hybrid Jobs
Quantum Tasks
Algorithm library
Announcements 1
Permissions and settings

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

braket-job-default-1693508892180

Summary

Status
COMPLETED

Runtime
00:01:21

Hybrid job logs
[View in CloudWatch](#)

Settings

Events

Monitor

Quantum Tasks

Tags

Details

Hybrid job name
braket-job-default-1693508892180

Device
arn:aws:braket::device/quantum-simulator/amazon/sv1

Status reason
—

Hybrid job ARN
arn:aws:braket:us-west-2:260818742045:job/braket-job-default-1693508892180

Execution role
arn:aws:iam::260818742045:role/service-role/AmazonBraketJobsExecutionRole

Event times

Created at
Aug 31, 2023 19:08 (UTC)

Started at
Aug 31, 2023 19:09 (UTC)

Ended at
Aug 31, 2023 19:10 (UTC)

Source code and instance configuration

Entry point
job_test_script:start_here

Instance type
ml.m5.large

Stopping conditions

Max runtime (seconds)
432000

Votre tâche hybride produit des artefacts dans Amazon S3 pendant son exécution. Le nom du compartiment S3 par défaut est `amazon-braket-<region>-<accountid>` et le contenu se trouve dans le `jobs/<jobname>/<timestamp>` répertoire. Vous pouvez configurer les emplacements S3 où ces artefacts sont stockés en spécifiant une autre `code_location` date de création de la tâche hybride avec le SDK Braket Python.

Note

Ce compartiment S3 doit se trouver au même endroit Région AWS que votre script de tâche.

Le `jobs/<jobname>/<timestamp>` répertoire contient un sous-dossier contenant le résultat du script du point d'entrée dans un `model.tar.gz` fichier. Il existe également un répertoire appelé `script` qui contient les artefacts de votre script d'algorithme dans un `source.tar.gz` fichier. Les résultats de vos tâches quantiques réelles se trouvent dans le répertoire nommé `jobs/<jobname>/tasks`.

Surveillez vos résultats

218

Enregistrez vos résultats

Vous pouvez enregistrer les résultats générés par le script d'algorithme afin qu'ils soient disponibles depuis l'objet de tâche hybride dans le script de tâche hybride ainsi que depuis le dossier de sortie dans Amazon S3 (dans un fichier compressé nommé `model.tar.gz`).

La sortie doit être enregistrée dans un fichier au format JSON (JavaScript Object Notation). Si les données ne peuvent pas être facilement sérialisées en texte, comme dans le cas d'un tableau numpy, vous pouvez transmettre une option de sérialisation à l'aide d'un format de données décapé. Consultez le module [braket.jobs.data_persistence](#) pour plus de détails.

Pour enregistrer les résultats des tâches hybrides, ajoutez les lignes suivantes commentées avec `#ADD` dans le fichier `algorithm_script.py`.

```
import os

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!")
```

Vous pouvez ensuite afficher les résultats de la tâche à partir de votre script de tâche en ajoutant la ligne `print(job.result())` commentée avec `#ADD`.

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
    device="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
```

Dans cet exemple, nous avons supprimé `wait_until_complete=True` pour supprimer la sortie détaillée. Vous pouvez le rajouter pour le débogage. Lorsque vous exécutez cette tâche hybride, elle affiche l'identifiant et le `job-arn`, suivis de l'état de la tâche hybride toutes les 10 secondes jusqu'à ce que la tâche hybride soit `COMPLETED` terminée, après quoi elle vous montre les résultats du circuit en cloche. Consultez l'exemple suivant.

```
arn:aws:braket:us-west-2:111122223333:job/braket-job-default-123456789012
INITIALIZED
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
RUNNING
...
RUNNING
RUNNING
COMPLETED
{'measurement_counts': [{'11': 53, '00': 47}, ..., {'00': 51, '11': 49}]}
```

Utilisation des points de contrôle

Vous pouvez enregistrer les itérations intermédiaires de vos tâches hybrides à l'aide de points de contrôle. Dans l'exemple de script d'algorithme de la section précédente, vous devez ajouter les lignes suivantes commentées avec `#ADD` pour créer des fichiers de points de contrôle.

```
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!")
```

Lorsque vous exécutez la tâche hybride, elle crée le fichier `-checkpoint-1.json <jobname>` dans les artefacts de votre tâche hybride dans le répertoire des points de contrôle avec un chemin par défaut. `/opt/jobs/checkpoints` Le script de tâche hybride reste inchangé, sauf si vous souhaitez modifier ce chemin par défaut.

Si vous souhaitez charger une tâche hybride à partir d'un point de contrôle généré par une tâche hybride précédente, le script d'algorithme utilise `from braket.jobs import load_job_checkpoint`. La logique à charger dans votre script d'algorithme est la suivante.

```
from braket.jobs import load_job_checkpoint
```

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

Après avoir chargé ce point de contrôle, vous pouvez poursuivre votre logique en fonction du contenu chargé dans `checkpoint-1`

Note

Le `checkpoint_file_suffix` doit correspondre au suffixe précédemment spécifié lors de la création du point de contrôle.

Votre script d'orchestration doit spécifier celui `job-arn` de la tâche hybride précédente avec la ligne commentée avec `#ADD`.

```
from braket.aws import AwsQuantumJob  
  
job = AwsQuantumJob.create(  
    source_module="source_dir",  
    entry_point="source_dir.algorithm_script:start_here",  
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",  
    copy_checkpoints_from_job="<previous-job-ARN>", #ADD  
)
```

Exécutez votre code local en tant que tâche hybride

Amazon Braket Hybrid Jobs fournit une orchestration entièrement gérée d'algorithmes hybrides quantiques-classiques, combinant les ressources de calcul Amazon EC2 avec l'accès à l'unité de traitement quantique (QPU) Amazon Braket. Les tâches quantiques créées dans le cadre d'une tâche hybride sont mises en file d'attente en priorité par rapport aux tâches quantiques individuelles afin que vos algorithmes ne soient pas interrompus par les fluctuations de la file d'attente des tâches quantiques. Chaque QPU gère une file d'attente de tâches hybrides distincte, ce qui garantit qu'une seule tâche hybride peut être exécutée à la fois.

Dans cette section :

- [Création d'une tâche hybride à partir du code Python local](#)
- [Installation de packages Python et de code source supplémentaires](#)

- [Enregistrer et charger des données dans une instance de tâche hybride](#)
- [Bonnes pratiques pour les décorateurs hybrides](#)

Création d'une tâche hybride à partir du code Python local

Vous pouvez exécuter votre code Python local sous forme d'Amazon Braket Hybrid Job. Vous pouvez le faire en annotant votre code à l'aide d'un `@hybrid_job` décorateur, comme illustré dans l'exemple de code suivant. Pour les environnements personnalisés, vous pouvez choisir d'[utiliser un conteneur personnalisé d'Amazon Elastic Container Registry \(ECR\)](#).

Note

Seul Python 3.12 est pris en charge par défaut.

Vous pouvez utiliser le `@hybrid_job` décorateur pour annoter une fonction. [Braket transforme le code contenu dans le décorateur en un script d'algorithme de travail hybride Braket](#). La tâche hybride invoque ensuite la fonction dans le décorateur sur une instance Amazon EC2. Vous pouvez suivre la progression de la tâche avec `job.state()` ou avec la console Braket. L'exemple de code suivant montre comment exécuter une séquence de cinq états sur le 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}
```

Vous créez la tâche hybride en invoquant la fonction comme vous le feriez pour les fonctions Python normales. Cependant, la fonction de décorateur renvoie le descripteur de tâche hybride plutôt que le résultat de la fonction. Pour récupérer les résultats une fois l'opération terminée, utilisez `job.result()`.

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

L'argument `device` dans le `@hybrid_job` décorateur indique le périphérique auquel la tâche hybride a un accès prioritaire, dans ce cas, le SV1 simulateur. Pour obtenir la priorité QPU, vous devez vous assurer que l'ARN du périphérique utilisé dans la fonction correspond à celui spécifié dans le décorateur. Pour plus de commodité, vous pouvez utiliser la fonction d'assistance `get_job_device_arn()` pour capturer l'ARN du périphérique déclaré dans `@hybrid_job`.

Note

Chaque tâche hybride a un temps de démarrage d'au moins une minute car elle crée un environnement conteneurisé sur Amazon EC2. Ainsi, pour des charges de travail très courtes, telles qu'un circuit unique ou un lot de circuits, il peut suffire d'utiliser des tâches quantiques.

Hyperparamètres

La `run_hybrid_job()` fonction utilise l'argument `num_tasks` pour contrôler le nombre de tâches quantiques créées. La tâche hybride le capture automatiquement sous forme d'[hyperparamètre](#).

 Note

Les hyperparamètres sont affichés dans la console Braket sous forme de chaînes limitées à 2 500 caractères.

Métriques et journalisation

Dans la `run_hybrid_job()` fonction, les métriques issues d'algorithmes itératifs sont enregistrées avec `log_metrics`. Les métriques sont automatiquement tracées sur la page de la console Braket, sous l'onglet Job hybride. Vous pouvez utiliser des métriques pour suivre les coûts quantiques des tâches en temps quasi réel pendant l'exécution d'une tâche hybride avec le système de suivi des [coûts Braket](#). L'exemple ci-dessus utilise le nom de métrique « probabilité » qui enregistre la première probabilité du [type de résultat](#).

Récupération des résultats

Une fois la tâche hybride terminée, vous pouvez `job.result()` récupérer les résultats des tâches hybrides. Tous les objets figurant dans le relevé de retour sont automatiquement capturés par Braket. Notez que les objets renvoyés par la fonction doivent être un tuple, chaque élément étant sérialisable. Par exemple, le code suivant montre un exemple de fonctionnement et un exemple d'échec.

```
import numpy as np

# Working example
@hybrid_job(device=Devices.Amazon.SV1)
def passing():
    np_array = np.random.rand(5)
    return np_array # Serializable

# # Failing example
# @hybrid_job(device=Devices.Amazon.SV1)
# def failing():
#     return MyObject() # Not serializable
```

Nom du job

Par défaut, le nom de cette tâche hybride est déduit du nom de la fonction. Vous pouvez également spécifier un nom personnalisé d'une longueur maximale de 50 caractères. Par exemple, dans le code suivant, le nom de la tâche est « my-job-name ».

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

Mode local

Les [tâches locales](#) sont créées en ajoutant l'argument `local=True` au décorateur. Cela exécute la tâche hybride dans un environnement conteneurisé sur votre environnement informatique local, tel que votre ordinateur portable. Les emplois locaux ne sont pas prioritaires dans les files d'attente pour les tâches quantiques. Dans les cas avancés tels que le multi-nœuds ou le MPI, les tâches locales peuvent avoir accès aux variables d'environnement Braket requises. Le code suivant crée une tâche hybride locale avec le périphérique comme SV1 simulateur.

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

Toutes les autres options d'emploi hybrides sont prises en charge. Pour une liste des options, consultez le module [braket.jobs.quantum_job_creation](#).

Installation de packages Python et de code source supplémentaires

Vous pouvez personnaliser votre environnement d'exécution pour utiliser vos packages Python préférés. Vous pouvez utiliser un `requirements.txt` fichier, une liste de noms de paquets ou [apporter votre propre conteneur \(BYOC\)](#). Par exemple, le `requirements.txt` fichier peut inclure d'autres packages à installer.

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

Pour personnaliser un environnement d'exécution à l'aide d'un `requirements.txt` fichier, reportez-vous à l'exemple de code suivant.

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

Vous pouvez également fournir les noms des packages sous forme de liste Python comme suit.

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

Le code source supplémentaire peut être spécifié sous la forme d'une liste de modules ou d'un seul module, comme dans l'exemple de code suivant.

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

Enregistrer et charger des données dans une instance de tâche hybride

Spécification des données d'entraînement en entrée

Lorsque vous créez une tâche hybride, vous pouvez fournir des ensembles de données de formation en entrée en spécifiant un bucket Amazon Simple Storage Service (Amazon S3). Vous pouvez également spécifier un chemin local, puis Braket télécharge automatiquement les données vers Amazon S3 à l'adresse. `s3://<default_bucket_name>/jobs/<job_name>/<timestamp>/data/<channel_name>` Si vous spécifiez un chemin local, le nom du canal est par défaut « input ». Le code suivant montre un fichier numpy à partir du chemin `data/file.npy` local.

```
import numpy as np

@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 ...
```

Pour S3, vous devez utiliser la fonction `get_input_data_dir()` d'assistance.

```
import numpy as np
from braket.jobs import get_input_data_dir

s3_path = "s3://amazon-braket-us-east-1-123456789012/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")
```

Vous pouvez spécifier plusieurs sources de données d'entrée en fournissant un dictionnaire des valeurs de canal et des chemins S3 URIs ou locaux.

```
import numpy as np
from braket.jobs import get_input_data_dir

input_data = {
    "input": "data/file.npy",
    "input_2": "s3://amzn-s3-demo-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

Lorsque les données d'entrée sont volumineuses (> 1 Go), le temps d'attente est long avant que la tâche ne soit créée. Cela est dû aux données d'entrée locales lorsqu'elles sont téléchargées pour la première fois dans un compartiment S3, puis le chemin S3 est ajouté à la demande de travail. Enfin, la demande d'emploi est soumise au service Braket.

Sauvegarde des résultats dans S3

Pour enregistrer les résultats non inclus dans l'instruction de retour de la fonction décorée, vous devez ajouter le répertoire approprié à toutes les opérations d'écriture de fichiers. L'exemple suivant montre la sauvegarde d'un tableau numpy et d'une figure matplotlib.

```
import matplotlib.pyplot as plt
import numpy as np
```

```
@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 ...
```

Tous les résultats sont compressés dans un fichier nommé `model.tar.gz`. Vous pouvez télécharger les résultats à l'aide de la fonction `job.result()` Python ou en accédant au dossier des résultats depuis la page des tâches hybrides dans la console de gestion Braket.

Sauvegarde et reprise depuis les points de contrôle

Pour les tâches hybrides de longue durée, il est recommandé de sauvegarder régulièrement l'état intermédiaire de l'algorithme. Vous pouvez utiliser la fonction `save_job_checkpoint()` assistance intégrée ou enregistrer des fichiers dans le `AMZN_BRAKET_JOB_RESULTS_DIR` chemin. Ce dernier est disponible avec la fonction d'assistance `get_job_results_dir()`

Voici un exemple pratique minimal pour enregistrer et charger des points de contrôle avec un décorateur de tâches hybride :

```
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()
```

Dans le premier job hybride, `save_job_checkpoint()` il est appelé avec un dictionnaire contenant les données que nous voulons enregistrer. Par défaut, chaque valeur doit être sérialisable sous forme de texte. Pour le pointage d'objets Python plus complexes, tels que des tableaux numpy, vous pouvez définir `data_format = PersistedJobDataFormat.PICKLED_V4`. Ce code crée et remplace un fichier de point de contrôle portant le nom par défaut `<jobname>.json` dans vos artefacts de tâches hybrides dans un sous-dossier appelé « points de contrôle ».

Pour créer une nouvelle tâche hybride afin de continuer depuis le point de contrôle, nous devons indiquer `copy_checkpoints_from_job=job_arn` où `job_arn` trouve l'ARN de la tâche hybride de la tâche précédente. Ensuite, nous `load_job_checkpoint(job_name)` chargeons depuis le point de contrôle.

Bonnes pratiques pour les décorateurs hybrides

Adoptez l'asynchronicité

Les tâches hybrides créées avec l'annotation du décorateur sont asynchrones : elles s'exécutent une fois que les ressources classiques et quantiques sont disponibles. Vous surveillez la progression de l'algorithme à l'aide de Braket Management Console ou Amazon CloudWatch. Lorsque vous soumettez votre algorithme à exécuter, Braket l'exécute dans un environnement conteneurisé évolutif et les résultats sont récupérés lorsque l'algorithme est terminé.

Exécuter des algorithmes variationnels itératifs

Les tâches hybrides vous fournissent les outils nécessaires pour exécuter des algorithmes classiques quantiques itératifs. Pour les problèmes purement quantiques, utilisez [des tâches quantiques](#) ou un [lot de tâches quantiques](#). L'accès prioritaire à certains QPUs est particulièrement avantageux pour les algorithmes variationnels de longue durée nécessitant plusieurs appels itératifs QPUs avec un traitement classique entre les deux.

Déboguer en mode local

Avant d'exécuter une tâche hybride sur un QPU, il est recommandé de l'exécuter d'abord sur le simulateur SV1 pour vérifier qu'elle fonctionne comme prévu. Pour les tests à petite échelle, vous pouvez les exécuter en mode local pour une itération et un débogage rapides.

Améliorez la reproductibilité avec [Bring your own container \(BYOC\)](#)

Créez une expérience reproductible en encapsulant votre logiciel et ses dépendances dans un environnement conteneurisé. En regroupant l'ensemble de votre code, de vos dépendances et de vos paramètres dans un conteneur, vous évitez les conflits potentiels et les problèmes de version.

Simulateurs distribués multi-instances

Pour exécuter un grand nombre de circuits, pensez à utiliser le support MPI intégré pour exécuter des simulateurs locaux sur plusieurs instances dans le cadre d'une seule tâche hybride. Pour plus d'informations, consultez la section [Simulateurs intégrés](#).

Utiliser des circuits paramétriques

Les circuits paramétriques que vous soumettez à partir d'une tâche hybride sont automatiquement compilés sur certains à QPUs l'aide d'une [compilation paramétrique](#) afin d'améliorer les temps d'exécution de vos algorithmes.

Point de contrôle périodique

Pour les tâches hybrides de longue durée, il est recommandé de sauvegarder régulièrement l'état intermédiaire de l'algorithme.

Pour plus d'exemples, de cas d'utilisation et de bonnes pratiques, consultez les exemples [d'Amazon GitHub Braket](#).

Utilisation de l'API avec des tâches hybrides

Vous pouvez accéder et interagir avec Amazon Braket Hybrid Jobs directement à l'aide du API. Cependant, les valeurs par défaut et les méthodes pratiques ne sont pas disponibles lors de l'utilisation API directe du.

Note

Nous vous recommandons vivement d'interagir avec Amazon Braket Hybrid Jobs à l'aide du SDK Amazon [Braket Python](#). Il propose des paramètres par défaut et des protections pratiques qui contribuent au bon fonctionnement de vos tâches hybrides.

Cette rubrique décrit les principes de base de l'utilisation du API. Si vous choisissez d'utiliser l'API, n'oubliez pas que cette approche peut être plus complexe et qu'elle doit être prête à plusieurs itérations pour exécuter votre tâche hybride.

Pour utiliser l'API, votre compte doit avoir un rôle dans la politique AmazonBraketFullAccess gérée.

Note

Pour plus d'informations sur la manière d'obtenir un rôle avec la politique AmazonBraketFullAccess gérée, consultez la [page Activer Amazon Braket](#).

De plus, vous avez besoin d'un rôle d'exécution. Ce rôle sera transmis au service. Vous pouvez créer le rôle à l'aide de la console Amazon Braket. Utilisez l'onglet Rôles d'exécution de la page Autorisations et paramètres pour créer un rôle par défaut pour les tâches hybrides.

Vous devez spécifier tous les paramètres requis pour la tâche hybride. CreateJob API Pour utiliser Python, compressez les fichiers de script de votre algorithme dans un bundle tar, tel qu'un fichier input.tar.gz, et exécutez le script suivant. Mettez à jour les parties du code entre crochets (<>) pour qu'elles correspondent aux informations de votre compte et au point d'entrée qui spécifient le chemin, le fichier et la méthode par lesquels votre tâche hybride commence.

```
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/Ankaa-3"
    ]
}
},
hyperParameters={
    "hyperparameter key you wish to pass": "<hyperparameter value you wish to
pass>",
},
stoppingCondition={
    "maxRuntimeInSeconds": 1200,
    "maximumTaskLimit": 10
},
)

```

Une fois que vous avez créé votre tâche hybride, vous pouvez accéder aux détails de la tâche hybride via la console GetJob API ou la console. Pour obtenir les détails de la tâche hybride à partir de la session Python dans laquelle vous avez exécuté le `createJob` code comme dans l'exemple précédent, utilisez la commande Python suivante.

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

Pour annuler une tâche hybride, appelez le `CancelJob` API with the Amazon Resource Name of the job ('JobArn').

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

Vous pouvez spécifier des points de contrôle dans le cadre de l'utilisation du `createJobAPI` du `checkpointConfig` paramètre.

```

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

```

Note

Le `LocalPath` de `checkpointConfig` ne peut démarrer par aucun des chemins réservés suivants : `/opt/ml`, `/opt/braket/tmp`, ou `/usr/local/nvidia`.

Création et débogage d'une tâche hybride en mode local

Lorsque vous créez un nouvel algorithme hybride, le mode local vous permet de déboguer et de tester votre script d'algorithme. Le mode local est une fonctionnalité qui vous permet d'exécuter le code que vous prévoyez d'utiliser dans Amazon Braket Hybrid Jobs, mais sans avoir besoin de Braket pour gérer l'infrastructure nécessaire à l'exécution de la tâche hybride. Exécutez plutôt des tâches hybrides localement sur votre instance Amazon Braket Notebook ou sur un client préféré, tel qu'un ordinateur portable ou de bureau.

En mode local, vous pouvez toujours envoyer des tâches quantiques à des appareils réels, mais vous ne bénéficiez pas des avantages en termes de performances lorsque vous les exécutez contre une unité de traitement quantique (QPU) réelle en mode local.

Pour utiliser le mode local, `AwsQuantumJob` modifiez-le à `LocalQuantumJob` endroit où il apparaît dans votre programme. Par exemple, pour exécuter l'exemple de [Create your first hybrid job](#), modifiez le script de job hybride dans le code comme suit.

```
from braket.jobs.local import LocalQuantumJob

job = LocalQuantumJob.create(
    device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
    source_module="algorithm_script.py",
    entry_point="algorithm_script:start_here",
)
```

Note

Docker, qui est déjà préinstallé dans les blocs-notes Amazon Braket, doit être installé dans votre environnement local pour utiliser cette fonctionnalité. Les instructions d'installation de Docker se trouvent sur la page [Get Docker](#). De plus, tous les paramètres ne sont pas pris en charge en mode local.

Annuler un Job hybride

Il se peut que vous deviez annuler une tâche hybride dans un état non terminal. Cela peut être fait dans la console ou avec du code.

Pour annuler votre tâche hybride dans la console, sélectionnez la tâche hybride à annuler sur la page Tâches hybrides, puis sélectionnez Annuler la tâche hybride dans le menu déroulant Actions.

The screenshot shows the Amazon Braket console interface. On the left is a sidebar with navigation links: Dashboard, Devices, Notebooks, Hybrid Jobs (selected), Quantum Tasks, Algorithm library, Announcements (1), and Permissions and settings. The main content area is titled 'Hybrid Jobs (4)' and contains a table with columns: Hybrid job name, Status, Device, and a timestamp. The table lists four jobs. The second job, 'braket-job-default-1693600353661', is selected. An 'Actions' dropdown menu is open for this job, showing options: View hybrid job, Cancel hybrid job (highlighted with a red box), and Manage tags. A 'Create hybrid job' button is also visible.

	Hybrid job name	Status	Device	
☐	braket-job-default-1693603871840	✖ CANCELLED	arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2	Sep 01, 2023 21:31 (UTC)
☒	braket-job-default-1693600353661	⌚ QUEUED	arn:aws:braket:us-east-1::device/qpu/ionq/Aria-2	Sep 01, 2023 20:32 (UTC)
☐	test-job-example	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Jun 02, 2022 22:26 (UTC)
☐	Test-ashlhans	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	May 25, 2022 19:50 (UTC)

Pour confirmer l'annulation, saisissez Annuler dans le champ de saisie lorsque vous y êtes invité, puis sélectionnez OK.

The screenshot shows a confirmation dialog box titled 'Cancel Job "JobTest-autograd-1637034526"?'. It contains a warning icon and a list of bullet points explaining the consequences of cancellation. Below the list, it instructs the user to enter 'cancel' in a text input field to confirm. The 'cancel' text is already entered in the field. At the bottom right, there are 'Cancel' and 'Ok' buttons, with the 'Ok' button highlighted by a red box.

Cancel Job "JobTest-autograd-1637034526"?

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

To confirm cancellation, enter *cancel* in the text input field.

Cancel **Ok**

Pour annuler votre tâche hybride à l'aide du code du SDK Braket Python, utilisez le `job_arn` pour identifier la tâche hybride, puis appelez la `cancel` commande correspondante comme indiqué dans le code suivant.

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

```
job.cancel()
```

La `cancel` commande met immédiatement fin au conteneur de tâches hybrides classique et fait de son mieux pour annuler toutes les tâches quantiques associées qui sont toujours dans un état non terminal.

Personnalisation de votre Hybrid Job

Amazon Braket propose plusieurs méthodes pour personnaliser le fonctionnement de vos tâches hybrides, ce qui vous permet d'adapter l'environnement à vos besoins spécifiques. Cette section explore les options permettant de personnaliser les tâches hybrides, qu'il s'agisse de définir l'environnement de script d'algorithme ou d'apporter votre propre conteneur. Vous apprendrez à optimiser votre flux de travail à l'aide d'hyperparamètres, à configurer des instances de travail et à tirer parti de la compilation paramétrique pour améliorer les performances. Ces techniques de personnalisation vous aident à optimiser le potentiel de vos calculs quantiques hybrides sur Amazon Braket.

Dans cette section :

- [Définissez l'environnement de votre script d'algorithme](#)
- [Utilisation d'hyperparamètres](#)
- [Configurez votre instance de tâche hybride](#)
- [Utilisation de la compilation paramétrique pour accélérer les tâches hybrides](#)

Définissez l'environnement de votre script d'algorithme

Amazon Braket prend en charge les environnements définis par des conteneurs pour votre script d'algorithme :

- Un conteneur de base (par défaut, si aucun conteneur `image_uri` est spécifié)
- Un conteneur avec CUDA-Q
- Un conteneur avec Tensorflow et PennyLane
- Un conteneur avec PyTorch PennyLane, et CUDA-Q

Le tableau suivant fournit des informations détaillées sur les conteneurs et les bibliothèques qu'ils incluent.

Conteneurs Amazon Braket

Type	Base	CUDA-Q	TensorFlow	PyTorch
URI de l'image	292282985 366.dkr. ecr.us- west-2.amaz onaws.com /:latest amazon-braket- base-jobs	292282985 366.dkr. ecr.us- west-2.amaz onaws.com /:latest amazon-braket- cudaq-jobs	292282985 366.dkr. ecr.us- east-1.amaz onaws.com /:latest amazon-braket- tensorflow-jobs	292282985 366.dkr. ecr.us- west-2.amaz onaws.com /:latest amazon-braket- pytorch-jobs
Bibliothèques héritées		• amazon-braket-default-simulator • amazon-braket-pennylane-plugin • amazon-braket-schemas • amazon-braket-sdk • awscli • botocore • boto3 • dask • matplotlib • numpy • pandas • PennyLane • PennyLane-Éclair • qiskit-braket-provider • demandes	• awscli • numpy • pandas • scipy	• awscli • numpy • pandas • scipy

Type	Base	CUDA-Q	TensorFlow	PyTorch
		- formation de sagemaker - scikit-learn - scipy		
Bibliothèques supplémentaires	- amazon-braket-default-simulator - amazon-braket-pennylane-plugin - amazon-braket-schemas - amazon-braket-sdk - awscli - boto3 - ipykernel - matplotlib - réseaux - numpy - openbabel - pandas - PennyLane - protobuf - psi4 - RSA - scipy	- cudaq - cudaq-qec - Solveurs Cudaq	- amazon-braket-default-simulator - amazon-braket-pennylane-plugin - amazon-braket-schemas - amazon-braket-sdk - ipykernel - keras - matplotlib - réseaux - openbabel - PennyLane - protobuf - psi4 - RSA - PennyLane-GPU Lightning - CuQuantum	- amazon-braket-default-simulator - amazon-braket-pennylane-plugin - amazon-braket-schemas - amazon-braket-sdk - ipykernel - keras - matplotlib - réseaux - openbabel - PennyLane - protobuf - psi4 - RSA - PennyLane-GPU Lightning - CuQuantum - cudaq - cudaq-qec - Solveurs Cudaq

Vous pouvez consulter et accéder aux définitions de conteneurs open source sur [aws/ amazon-braket-containers](#). Choisissez le contenant qui correspond le mieux à votre cas d'utilisation. Vous pouvez utiliser n'importe laquelle des AWS régions disponibles dans Braket (us-east-1, us-west-1, us-west-2, eu-north-1, eu-west-2), mais la région du conteneur doit correspondre à la région correspondant à votre tâche hybride. Spécifiez l'image du conteneur lorsque vous créez une tâche hybride en ajoutant l'un des trois arguments suivants à votre `create(...)` appel dans le script de tâche hybride. Vous pouvez installer des dépendances supplémentaires dans le conteneur que vous choisissez lors de l'exécution (au prix du démarrage ou de l'exécution) car les conteneurs Amazon Braket sont connectés à Internet. L'exemple suivant concerne la région us-west-2.

- Image de base : `image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest » amazon-braket-base-jobs`
- Image CUDA-Q : `image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest » amazon-braket-cudaq-jobs`
- Image de Tensorflow : `image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest » amazon-braket-tensorflow-jobs`
- PyTorch image : `image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com /:latest » amazon-braket-pytorch-jobs`

Ils `image-uris` peuvent également être récupérés à l'aide de la `retrieve_image()` fonction du SDK Amazon Braket. L'exemple suivant montre comment les récupérer depuis l'us-west-2 Région AWS.

```
from braket.jobs.image_uris import retrieve_image, Framework

image_uri_base = retrieve_image(Framework.BASE, "us-west-2")
image_uri_cudaq = retrieve_image(Framework.CUDAQ, "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")
```

Apport de votre propre conteneur

Amazon Braket Hybrid Jobs fournit trois conteneurs prédéfinis pour exécuter du code dans différents environnements. Si l'un de ces conteneurs prend en charge votre cas d'utilisation, il vous suffit de fournir votre script d'algorithme lorsque vous créez une tâche hybride. Les dépendances manquantes mineures peuvent être ajoutées à partir de votre script d'algorithme ou d'un `requirements.txt` fichier à l'aide de `pip`.

Si aucun de ces conteneurs ne correspond à votre cas d'utilisation, ou si vous souhaitez les développer, Braket Hybrid Jobs prend en charge l'exécution de tâches hybrides avec votre propre image de Docker conteneur personnalisée, ou le Bring Your Own Container (BYOC). Assurez-vous qu'il s'agit de la fonctionnalité adaptée à votre cas d'utilisation.

Dans cette section :

- [Quand est-ce qu'apporter mon propre contenant est la bonne décision ?](#)
- [Recette pour apporter votre propre contenant](#)
- [Exécution de tâches hybrides avec Braket dans votre propre conteneur](#)

Quand est-ce qu'apporter mon propre contenant est la bonne décision ?

Bringing your own container (BYOC) to Braket Hybrid Jobs offre la flexibilité d'utiliser votre propre logiciel en l'installant dans un environnement packagé. En fonction de vos besoins spécifiques, il existe peut-être des moyens d'obtenir la même flexibilité sans avoir à passer par le cycle complet de BYOC Docker compilation (téléchargement Amazon ECR) et d'URI d'image personnalisée.

 Note

Le BYOC n'est peut-être pas le bon choix si vous souhaitez ajouter un petit nombre de packages Python supplémentaires (généralement moins de 10) accessibles au public. Par exemple, si vous utilisez PyPi.

Dans ce cas, vous pouvez utiliser l'une des images Braket prédéfinies, puis inclure un `requirements.txt` fichier dans votre répertoire source lors de la soumission de la tâche. Le fichier est automatiquement lu et `pip` installera les packages avec les versions spécifiées comme d'habitude. Si vous installez un grand nombre de packages, le temps d'exécution de vos tâches peut être considérablement augmenté. Vérifiez la version Python et, le cas échéant, la version CUDA du conteneur prédéfini que vous souhaitez utiliser pour tester le fonctionnement de votre logiciel.

Le BYOC est nécessaire lorsque vous souhaitez utiliser un langage autre que Python (comme C++ ou Rust) pour votre script de travail, ou si vous souhaitez utiliser une version de Python non disponible dans les conteneurs prédéfinis de Braket. C'est également un bon choix si :

- Vous utilisez un logiciel doté d'une clé de licence, et vous devez authentifier cette clé auprès d'un serveur de licences pour exécuter le logiciel. Avec BYOC, vous pouvez intégrer la clé de licence dans votre Docker image et inclure du code pour l'authentifier.

- Vous utilisez un logiciel qui n'est pas accessible au public. Par exemple, le logiciel est hébergé sur un dépôt privé GitLab ou un GitHub dépôt auquel vous avez besoin d'une clé SSH spécifique pour accéder.
- Vous devez installer une vaste suite de logiciels qui n'est pas incluse dans les conteneurs fournis par Braket. Le BYOC vous permettra d'éliminer les longs délais de démarrage de vos conteneurs de tâches hybrides en raison de l'installation de logiciels.

Le BYOC vous permet également de mettre votre SDK ou algorithme personnalisé à la disposition des clients en créant un Docker conteneur avec votre logiciel et en le mettant à la disposition de vos utilisateurs. Vous pouvez le faire en définissant les autorisations appropriées dans Amazon ECR.

 Note

Vous devez respecter toutes les licences logicielles applicables.

Recette pour apporter votre propre contenant

Dans cette section, nous fournissons un step-by-step guide expliquant ce dont vous avez besoin bring your own container (BYOC) pour Braket Hybrid Jobs : les scripts, les fichiers et les étapes à suivre pour les combiner afin de les utiliser avec vos Docker images personnalisées. Les recettes pour deux cas courants :

1. Installez des logiciels supplémentaires dans une Docker image et utilisez uniquement des scripts d'algorithme Python dans vos tâches.
2. Utilisez des scripts d'algorithme écrits dans un langage autre que Python avec Hybrid Jobs, ou une architecture de processeur autre que x86.

La définition du script de saisie du conteneur est plus complexe dans le cas 2.

Lorsque Braket exécute votre Hybrid Job, il lance le nombre et le type d'instances Amazon EC2 demandés, puis exécute Docker l'image spécifiée par l'URI de l'image pour créer la tâche sur celles-ci. Lorsque vous utilisez la fonctionnalité BYOC, vous spécifiez un URI d'image hébergé dans un [référentiel Amazon ECR privé](#) auquel vous avez accès en lecture. Braket Hybrid Jobs utilise cette image personnalisée pour exécuter le job.

Les composants spécifiques dont vous avez besoin pour créer une Docker image utilisable avec Hybrid Jobs. [Si vous n'êtes pas familiarisé avec l'écriture et la construction Dockerfiles, reportez-vous à la documentation Dockerfile et à la Amazon ECR CLI documentation.](#)

Prérequis:

- [Une image de base pour votre Dockerfile](#)
- [\(Facultatif\) Script de point d'entrée du conteneur modifié](#)
- [Installez le logiciel et le script de conteneur nécessaires avec Dockerfile](#)

Une image de base pour votre Dockerfile

Si vous utilisez Python et que vous souhaitez installer un logiciel en plus de ce qui est fourni dans les conteneurs fournis par Braket, une option pour une image de base est l'une des images de conteneur Braket, hébergées dans notre [GitHub dépôt et sur Amazon ECR](#). Vous devrez vous [authentifier auprès d'Amazon ECR](#) pour extraire l'image et créer dessus. Par exemple, la première ligne de votre Docker fichier BYOC pourrait être : `FROM [IMAGE_URI_HERE]`

Ensuite, remplissez le reste Dockerfile pour installer et configurer le logiciel que vous souhaitez ajouter au conteneur. Les images Braket prédéfinies contiennent déjà le script de point d'entrée du conteneur approprié, vous n'avez donc pas à vous soucier de l'inclure.

Si vous souhaitez utiliser un langage autre que Python, tel que C++, Rust ou Julia, ou si vous souhaitez créer une image pour une architecture de processeur autre que x86, comme ARM, vous devrez peut-être créer à partir d'une image publique simple. Vous trouverez de nombreuses images de ce type dans la [galerie publique d'Amazon Elastic Container Registry](#). Assurez-vous d'en choisir un qui convient à l'architecture du processeur et, si nécessaire, au processeur graphique que vous souhaitez utiliser.

(Facultatif) Script de point d'entrée du conteneur modifié

Note

Si vous ajoutez uniquement un logiciel supplémentaire à une image Braket prédéfinie, vous pouvez ignorer cette section.

Pour exécuter du code autre que Python dans le cadre de votre tâche hybride, modifiez le script Python qui définit le point d'entrée du conteneur. Par exemple, le [script `braket\_container.py`](#)

[python sur le Github d'Amazon Braket](#). Il s'agit du script que les images précréées par Braket utilisent pour lancer votre script d'algorithme et définir les variables d'environnement appropriées. Le script du point d'entrée du conteneur lui-même doit être en Python, mais il peut lancer des scripts autres que Python. Dans l'exemple prédéfini, vous pouvez voir que les scripts d'algorithme Python sont lancés soit en tant que [sous-processus Python](#), soit en tant que [processus entièrement nouveau](#). En modifiant cette logique, vous pouvez permettre au script du point d'entrée de lancer des scripts d'algorithmes autres que Python. Par exemple, vous pouvez modifier la [thekick_off_customer_script\(\)](#) fonction pour lancer des processus Rust en fonction de la fin de l'extension du fichier.

Vous pouvez également choisir d'en écrire un tout nouveau `braket_container.py`. Il doit copier les données d'entrée, les archives sources et les autres fichiers nécessaires depuis Amazon S3 dans le conteneur, et définir les variables d'environnement appropriées.

Installez le logiciel et le script de conteneur nécessaires avec **Dockerfile**

 Note

Si vous utilisez une image Braket prédéfinie comme image de Docker base, le script de conteneur est déjà présent.

Si vous avez créé un script de conteneur modifié à l'étape précédente, vous devez le copier dans le conteneur et définir la variable d'environnement `SAGEMAKER_PROGRAM` to `braket_container.py`, ou le nom que vous avez donné à votre nouveau script de point d'entrée de conteneur.

Voici un exemple `Dockerfile` qui vous permet d'utiliser Julia sur des instances de Jobs accélérées par GPU :

```
FROM nvidia/cuda:12.2.0-devel-ubuntu22.04

ARG DEBIAN_FRONTEND=noninteractive
ARG JULIA_RELEASE=1.8
ARG JULIA_VERSION=1.8.3

ARG PYTHON=python3.11
ARG PYTHON_PIP=python3-pip
ARG PIP=pip
```

```
ARG JULIA_URL = https://julialang-s3.julialang.org/bin/linux/x64/${JULIA_RELEASE}/
ARG TAR_NAME = julia-${JULIA_VERSION}-linux-x86_64.tar.gz

ARG PYTHON_PKGS = # list your Python packages and versions here

RUN curl -s -L ${JULIA_URL}/${TAR_NAME} | tar -C /usr/local -x -z --strip-components=1
-f -

RUN apt-get update \

    && apt-get install -y --no-install-recommends \

    build-essential \

    tzdata \

    openssh-client \

    openssh-server \

    ca-certificates \

    curl \

    git \

    libtemplate-perl \

    libssl1.1 \

    openssl \

    unzip \

    wget \

    zlib1g-dev \

    ${PYTHON_PIP} \
```

```

    ${PYTHON}-dev \

RUN ${PIP} install --no-cache --upgrade ${PYTHON_PKGS}

RUN ${PIP} install --no-cache --upgrade sagemaker-training==4.1.3

# Add EFA and SMDDP to LD library path
ENV LD_LIBRARY_PATH="/opt/conda/lib/python${PYTHON_SHORT_VERSION}/site-packages/
smdistributed/dataparallel/lib:$LD_LIBRARY_PATH"
ENV LD_LIBRARY_PATH=/opt/amazon/efa/lib:$LD_LIBRARY_PATH

# Julia specific installation instructions
COPY Project.toml /usr/local/share/julia/environments/v${JULIA_RELEASE}/
RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using Pkg; Pkg.instantiate(); Pkg.API.precompile()'
# generate the device runtime library for all known and supported devices
RUN JULIA_DEPOT_PATH=/usr/local/share/julia \

    julia -e 'using CUDA; CUDA.precompile_runtime()'

# Open source compliance scripts
RUN HOME_DIR=/root \

    && curl -o ${HOME_DIR}/oss_compliance.zip https://aws-dlinfra-
utilities.s3.amazonaws.com/oss_compliance.zip \

    && unzip ${HOME_DIR}/oss_compliance.zip -d ${HOME_DIR}/ \

    && cp ${HOME_DIR}/oss_compliance/test/testOSSCompliance /usr/local/bin/
testOSSCompliance \

    && chmod +x /usr/local/bin/testOSSCompliance \

    && chmod +x ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh \

    && ${HOME_DIR}/oss_compliance/generate_oss_compliance.sh ${HOME_DIR} ${PYTHON} \

```

```
&& rm -rf ${HOME_DIR}/oss_compliance*

# Copying the container entry point script
COPY braket_container.py /opt/ml/code/braket_container.py
ENV SAGEMAKER_PROGRAM braket_container.py
```

Cet exemple télécharge et exécute des scripts fournis par AWS pour garantir la conformité avec toutes les licences Open Source pertinentes. Par exemple, en attribuant correctement tout code installé régi par un MIT license.

Si vous devez inclure du code non public, par exemple du code hébergé dans un dépôt privé GitHub ou dans un GitLab dépôt, n'intégrez pas de clés SSH dans l'image Docker pour y accéder. Utilisez-le plutôt Docker Compose lorsque vous créez pour autoriser l'accès Docker à SSH sur la machine hôte sur laquelle il est construit. Pour plus d'informations, consultez le guide [Utilisation sécurisée des clés SSH dans Docker pour accéder aux référentiels Github privés](#).

Création et téléchargement de votre image Docker

Une fois correctement défini `Dockerfile`, vous êtes maintenant prêt à suivre les étapes pour [créer un référentiel Amazon ECR privé](#), s'il n'en existe pas déjà un. Vous pouvez également créer, étiqueter et télécharger votre image de conteneur dans le référentiel.

Vous êtes prêt à créer, étiqueter et publier l'image. Consultez la [documentation de compilation de Docker](#) pour une explication complète des options `docker build` et quelques exemples.

Pour le fichier d'exemple défini ci-dessus, vous pouvez exécuter :

```
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
```

Attribution des autorisations Amazon ECR appropriées

Braket Hybrid Jobs Docker les images doivent être hébergées dans des référentiels Amazon ECR privés. Par défaut, un dépôt Amazon ECR privé ne fournit pas d'accès en lecture aux utilisateurs qui souhaitent utiliser votre image, tels qu'un collaborateur Braket Hybrid Jobs IAM role ou un étudiant.

Vous devez [définir une politique de dépôt](#) pour accorder les autorisations appropriées. En général, autorisez uniquement les utilisateurs et les IAM rôles spécifiques auxquels vous souhaitez accéder à vos images, plutôt que de permettre image URI à toute personne possédant le droit de les récupérer.

Exécution de tâches hybrides avec Braket dans votre propre conteneur

Pour créer une tâche hybride avec votre propre conteneur, appelez `AwsQuantumJob.create()` avec l'argument `image_uri` spécifié. Vous pouvez utiliser un QPU, un simulateur à la demande ou exécuter votre code localement sur le processeur classique disponible avec Braket Hybrid Jobs. Nous vous recommandons de tester votre code sur un simulateur SV1 DM1, ou TN1 avant de l'exécuter sur un véritable QPU.

Pour exécuter votre code sur le processeur classique, spécifiez le `instanceType` et `instanceCount` que vous utilisez en mettant à jour le `InstanceConfig`. Notez que si vous spécifiez un `instance_count > 1`, vous devez vous assurer que votre code peut être exécuté sur plusieurs hôtes. La limite supérieure du nombre d'instances que vous pouvez choisir est de 5. Par exemple :

```
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.g4dn.xlarge", instanceCount=3),
    device="local:braket/braket.local.qubit",
    # ...)
```

Note

Utilisez l'ARN de l'appareil pour suivre le simulateur que vous avez utilisé comme métadonnées de tâches hybrides. Les valeurs acceptables doivent respecter le format `device = "local:<provider>/<simulator_name>"`. N'oubliez pas cela `<provider>` et ne `<simulator_name>` devez être composé que de lettres_, de chiffres-, et.. La chaîne est limitée à 256 caractères.

Si vous envisagez d'utiliser le BYOC et que vous n'utilisez pas le SDK Braket pour créer des tâches quantiques, vous devez transmettre la valeur de la variable environnementale `AMZN_BRAKET_JOB_TOKEN` au `jobToken` paramètre de la demande. `CreateQuantumTask` Si vous ne le faites pas, les tâches quantiques ne sont pas prioritaires et sont facturées comme des tâches quantiques autonomes ordinaires.

Utilisation d'hyperparamètres

Vous pouvez définir les hyperparamètres nécessaires à votre algorithme, tels que le taux d'apprentissage ou la taille des étapes, lorsque vous créez une tâche hybride. Les valeurs des hyperparamètres sont généralement utilisées pour contrôler différents aspects de l'algorithme et peuvent souvent être ajustées pour optimiser les performances de l'algorithme. Pour utiliser des hyperparamètres dans une tâche hybride Braket, vous devez spécifier leurs noms et leurs valeurs de manière explicite sous forme de dictionnaire. Spécifiez les valeurs des hyperparamètres à tester lors de la recherche de l'ensemble de valeurs optimal. La première étape de l'utilisation des hyperparamètres consiste à configurer et à définir les hyperparamètres sous forme de dictionnaire, comme le montre le code suivant.

```
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

hyperparameters = {"shots": 1_000}
```

Passez ensuite les hyperparamètres définis dans l'extrait de code ci-dessus à utiliser dans l'algorithme de votre choix. Pour exécuter l'exemple de code suivant, créez un répertoire nommé « src » dans le même chemin que votre fichier d'hyperparamètres. [Dans le répertoire « src », ajoutez les fichiers de code 0_Getting_Started_PaperMill.ipynb, notebook_runner.py et requirements.txt.](#)

```
import time
from braket.aws import AwsQuantumJob

job = AwsQuantumJob.create(
    device=device_arn,
    source_module="src",
    entry_point="src.notebook_runner:run_notebook",
    input_data="src/0_Getting_started_papermill.ipynb",
    hyperparameters=hyperparameters,
    job_name=f"papermill-job-demo-{int(time.time())}",
)

# Print job to record the ARN
print(job)
```

Pour accéder à vos hyperparamètres depuis votre script de tâche hybride, consultez la `load_jobs_hyperparams()` fonction dans le fichier python [notebook_runner.py](#). Pour accéder à vos hyperparamètres en dehors de votre script de tâche hybride, exécutez le code suivant.

```
from braket.aws import AwsQuantumJob

# Get the job using the ARN
job_arn = "arn:aws:braket:us-east-1:111122223333:job/5eabb790-d3ff-47cc-98ed-
b4025e9e296f" # Replace with your job ARN
job = AwsQuantumJob(arn=job_arn)

# Access the hyperparameters
job_metadata = job.metadata()
hyperparameters = job_metadata.get("hyperParameters", {})
print(hyperparameters)
```

Pour plus d'informations sur l'apprentissage de l'utilisation des hyperparamètres, consultez les didacticiels [QAOA avec Amazon Braket Hybrid Jobs PennyLane et Quantum Machine Learning dans Amazon Braket Hybrid Jobs](#).

Configurez votre instance de tâche hybride

En fonction de votre algorithme, vous pouvez avoir des exigences différentes. Par défaut, Amazon Braket exécute votre script d'algorithme sur une `m1.m5.large` instance. Toutefois, vous pouvez personnaliser ce type d'instance lorsque vous créez une tâche hybride à l'aide de l'argument d'importation et de configuration suivant.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instance_type="m1.g4dn.xlarge"), # Use NVIDIA T4
    instance_with_4_gpus.
    ...
),
```

Si vous exécutez une simulation intégrée et que vous avez spécifié un périphérique local dans la configuration de l'appareil, vous pouvez également demander plusieurs instances dans le `InstanceConfig` spécifiant le `instanceCount` et en le définissant pour qu'il soit supérieur à un. La limite supérieure est de 5. Par exemple, vous pouvez choisir 3 instances comme suit.

```
from braket.jobs.config import InstanceConfig
job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge", instanceCount=3), #
    Use 3 NVIDIA T4 instances
    ...
),
```

Lorsque vous utilisez plusieurs instances, pensez à distribuer votre tâche hybride à l'aide de la fonctionnalité data parallel. Consultez l'exemple de bloc-notes suivant pour plus de détails sur la façon de voir cet exemple de [formation Parallelize](#) pour QML.

Les trois tableaux suivants répertorient les types d'instances et les spécifications disponibles pour les instances standard, hautes performances et accélérées par GPU.

 Note

Pour consulter les quotas d'instances de calcul classiques par défaut pour les tâches hybrides, consultez la page [Amazon Braket Quotas](#).

Instances standard	vCPU	Mémoire (Gio)
ml.m5.large (par défaut)	4	16
ml.m5.xlarge	4	16
ml.m5.2xlarge	8	32
ml.m5.4xlarge	16	64
ml.m5.12xlarge	48	192
ml.m5.24xlarge	96	384

Instances à hautes performances	vCPU	Mémoire (Gio)
ml.c5.xlarge	4	8
ml.c5.2xlarge	8	16
ml.c5.4xlarge	16	32
ml.c5.9xlarge	36	72
ml.c5.18xlarge	72	144
ml.c5n.xlarge	4	10,5
ml.c5n.2xlarge	8	21
ml.c5n.4xlarge	16	32
ml.c5n.9xlarge	36	72
ml.c5n.18xlarge	72	192

Instances accélérées par GPU	GPUs	vCPU	Mémoire (Gio)	Mémoire GPU (Gio)
ml.p4d.24xlarge	8	96	1 152	320
ml.g4dn.xlarge	1	4	16	16
ml.g4dn.2xlarge	1	8	32	16
ml.g4dn.4xlarge	1	16	64	16
ml.g4dn.8xlarge	1	32	128	16
ml.g4dn.12xlarge	4	48	192	64

Instances accélérées par GPU	GPUs	vCPU	Mémoire (Gio)	Mémoire GPU (Gio)
ml.g4dn.16xlarge	1	64	256	16

Chaque instance utilise une configuration de stockage de données (SSD) par défaut de 30 Go. Mais vous pouvez régler le stockage de la même manière que vous configurez le `instanceType`. L'exemple suivant montre comment augmenter le stockage total à 50 Go.

```
from braket.jobs.config import InstanceConfig

job = AwsQuantumJob.create(
    ...
    instance_config=InstanceConfig(
        instance_type="ml.g4dn.xlarge",
        volume_size_in_gb=50,
    ),
    ...
),
```

Configurez le bucket par défaut dans **AwsSession**

L'utilisation de votre propre `AwsSession` instance vous offre une flexibilité accrue, notamment la possibilité de spécifier un emplacement personnalisé pour votre compartiment Amazon S3 par défaut. Par défaut, un `AwsSession` possède un emplacement de compartiment Amazon S3 préconfiguré de `"amazon-braket-{id}-{region}"`. Cependant, vous avez la possibilité de remplacer l'emplacement par défaut du compartiment Amazon S3 lors de la création d'un `AwsSession`. Les utilisateurs peuvent éventuellement transmettre un `AwsSession` objet dans la `AwsQuantumJob.create()` méthode, en fournissant le `aws_session` paramètre comme illustré dans l'exemple de code suivant.

```
aws_session = AwsSession(default_bucket="amazon-braket-s3-demo-bucket")

# Then you can use that AwsSession when creating a hybrid job
job = AwsQuantumJob.create(
    ...
    aws_session=aws_session
)
```

Utilisation de la compilation paramétrique pour accélérer les tâches hybrides

Amazon Braket prend en charge la compilation paramétrique sur certains QPUs. Cela vous permet de réduire la surcharge associée à l'étape de compilation coûteuse en calculant un circuit une seule fois et non pour chaque itération de votre algorithme hybride. Cela peut améliorer considérablement les temps d'exécution des tâches hybrides, car vous évitez d'avoir à recompiler votre circuit à chaque étape. Soumettez simplement des circuits paramétrés à l'un de nos Job hybrides pris en charge en QPUs tant que Braket. Pour les travaux hybrides de longue durée, Braket utilise automatiquement les données d'étalonnage mises à jour fournies par le fournisseur de matériel lors de la compilation de votre circuit afin de garantir des résultats de la plus haute qualité.

Pour créer un circuit paramétrique, vous devez d'abord fournir des paramètres en tant qu'entrées dans votre script d'algorithme. Dans cet exemple, nous utilisons un petit circuit paramétrique et ignorons tout traitement classique entre chaque itération. Pour les charges de travail classiques, vous devez soumettre de nombreux circuits par lots et effectuer un traitement classique tel que la mise à jour des paramètres à chaque itération.

```
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.")
```

Vous pouvez soumettre le script d'algorithme à exécuter en tant que Job hybride avec le script de travail suivant. Lorsque vous exécutez le Job hybride sur un QPU qui prend en charge la compilation

paramétrique, le circuit est compilé uniquement lors de la première exécution. Lors des exécutions suivantes, le circuit compilé est réutilisé, ce qui augmente les performances d'exécution du Hybrid Job sans aucune ligne de code supplémentaire.

```
from braket.aws import AwsQuantumJob

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

Note

La compilation paramétrique est prise en charge sur tous les modèles supraconducteurs basés sur des portes, QPUs à l'Rigetti Computing exception des programmes de niveau d'impulsion.

Utilisation PennyLane avec Amazon Braket

Les algorithmes hybrides sont des algorithmes qui contiennent à la fois des instructions classiques et quantiques. Les instructions classiques sont exécutées sur du matériel classique (une instance EC2 ou votre ordinateur portable), et les instructions quantiques sont exécutées sur un simulateur ou sur un ordinateur quantique. Nous vous recommandons d'exécuter des algorithmes hybrides à l'aide de la fonctionnalité Hybrid Jobs. Pour plus d'informations, consultez [Quand utiliser Amazon Braket Jobs](#).

Amazon Braket vous permet de configurer et d'exécuter des algorithmes quantiques hybrides à l'aide du plugin Amazon Braket ou du SDK Python Amazon PennyLane Braket et de référentiels d'exemples de blocs-notes. Des exemples de blocs-notes Amazon Braket, basés sur le SDK, vous permettent de configurer et d'exécuter certains algorithmes hybrides sans le plug-in. PennyLane Cependant, nous le recommandons PennyLane car cela offre une expérience plus riche.

À propos des algorithmes quantiques hybrides

Les algorithmes quantiques hybrides sont importants pour l'industrie aujourd'hui, car les dispositifs informatiques quantiques contemporains produisent généralement du bruit et, par conséquent, des erreurs. Chaque porte quantique ajoutée à un calcul augmente le risque d'ajouter du bruit ; par conséquent, les algorithmes de longue durée peuvent être submergés par le bruit, ce qui entraîne des erreurs de calcul.

Les algorithmes quantiques purs tels que ceux de Shor ([exemple d'estimation de phase quantique](#)) ou de Grover ([exemple de Grover](#)) nécessitent des milliers, voire des millions, d'opérations. Pour cette raison, ils peuvent être peu pratiques pour les dispositifs quantiques existants, généralement appelés dispositifs quantiques bruyants à échelle intermédiaire (NISQ).

Dans les algorithmes quantiques hybrides, les unités de traitement quantique (QPUs) fonctionnent comme des coprocesseurs pour les algorithmes classiques CPUs, notamment pour accélérer certains calculs dans un algorithme classique. Les exécutions des circuits deviennent beaucoup plus courtes, à la portée des capacités des appareils actuels.

Dans cette section :

- [Amazon Braket avec PennyLane](#)
- [Exemples d'algorithmes hybrides dans les carnets de notes Amazon Braket](#)
- [Algorithmes hybrides avec PennyLane simulateurs intégrés](#)
- [Dégradé adjoint activé PennyLane avec les simulateurs Amazon Braket](#)
- [Utilisation de tâches hybrides et exécution PennyLane d'un algorithme QAOA](#)
- [Exécutez des charges de travail hybrides avec des simulateurs PennyLane intégrés](#)

Amazon Braket avec PennyLane

Amazon Braket fournit un support pour [PennyLane](#), un framework logiciel open source construit autour du concept de programmation différentiable quantique. Vous pouvez utiliser ce cadre pour entraîner des circuits quantiques de la même manière que vous entraîneriez un réseau neuronal afin de trouver des solutions à des problèmes de calcul en chimie quantique, en apprentissage automatique quantique et en optimisation.

La PennyLane bibliothèque fournit des interfaces vers des outils d'apprentissage automatique familiers, notamment PyTorch et TensorFlow pour rendre l'apprentissage des circuits quantiques rapide et intuitif.

- La PennyLane bibliothèque — PennyLane est préinstallée dans les blocs-notes Amazon Braket. Pour accéder aux appareils Amazon Braket depuis PennyLane, ouvrez un bloc-notes et importez la PennyLane bibliothèque à l'aide de la commande suivante.

```
import pennylane as qml
```


Les carnets de didacticiels vous aident à démarrer rapidement. Vous pouvez également l'utiliser PennyLane sur Amazon Braket à partir de l'IDE de votre choix.

- Le PennyLane plugin Amazon Braket — Pour utiliser votre propre IDE, vous pouvez installer le PennyLane plugin Amazon Braket manuellement. Le plugin se connecte PennyLane au [SDK Amazon Braket Python](#), afin que vous puissiez exécuter des circuits PennyLane sur Amazon les appareils Braket. Pour installer le PennyLane plugin, utilisez la commande suivante.

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

L'exemple suivant montre comment configurer l'accès aux appareils Amazon Braket dans PennyLane :

```
# to use SV1
import pennylane as qml
sv1 = qml.device("braket.aws.qubit", device_arn="arn:aws:braket:::device/quantum-
simulator/amazon/sv1", wires=2)

# to run a circuit:
@qml.qnode(sv1)
def circuit(x):
    qml.RZ(x, wires=0)
    qml.CNOT(wires=[0,1])
    qml.RY(x, wires=1)
    return qml.expval(qml.PauliZ(1))

result = circuit(0.543)

#To use the local sim:
local = qml.device("braket.local.qubit", wires=2)
```

Pour des exemples de didacticiels et plus d'informations à ce sujet PennyLane, consultez le [référentiel d'exemples Amazon Braket](#).

Le PennyLane plugin Amazon Braket vous permet de basculer entre Amazon Braket QPU et les dispositifs de simulation intégrés en PennyLane une seule ligne de code. Il propose deux appareils quantiques Amazon Braket avec PennyLane lesquels travailler :

- `braket.aws.qubit` pour fonctionner avec les appareils quantiques Amazon du service Braket, y compris QPUs les simulateurs
- `braket.local.qubit` pour fonctionner avec le Amazon simulateur local du SDK Braket

Le PennyLane plugin Amazon Braket est open source. Vous pouvez l'installer depuis le [GitHub dépôt des PennyLane plugins](#).

Pour plus d'informations PennyLane, consultez la documentation sur le [PennyLane site Web](#).

Exemples d'algorithmes hybrides dans les carnets de notes Amazon Braket

Amazon Braket fournit une variété d'exemples de blocs-notes qui ne s'appuient pas sur le PennyLane plugin pour exécuter des algorithmes hybrides. Vous pouvez commencer avec n'importe lequel de ces [carnets hybrides Amazon Braket](#) illustrant des méthodes variationnelles, comme l'algorithme d'optimisation approximative quantique (QAOA) ou le solveur quantique variationnel (VQE).

Les carnets d'exemple Amazon Braket s'appuient sur le SDK Amazon [Braket](#) Python. Le SDK fournit un cadre permettant d'interagir avec les dispositifs matériels informatiques quantiques via Amazon Braket. Il s'agit d'une bibliothèque open source conçue pour vous aider à gérer la partie quantique de votre flux de travail hybride.

Vous pouvez explorer Amazon Braket plus en détail avec nos [exemples de carnets](#) de notes.

Algorithmes hybrides avec PennyLane simulateurs intégrés

Amazon Braket Hybrid Jobs est désormais livré avec des simulateurs intégrés hautes performances basés sur le processeur et le GPU de. [PennyLane](#) Cette famille de simulateurs intégrés peut être intégrée directement dans votre conteneur de tâches hybrides et inclut le `lightning.qubit` simulateur vectoriel d'état rapide, le simulateur accéléré à l'`lightning.gpu` aide de la [bibliothèque cuQuantum](#) de NVIDIA, etc. Ces simulateurs intégrés sont parfaitement adaptés aux algorithmes variationnels tels que l'apprentissage automatique quantique, qui peuvent bénéficier de méthodes avancées telles que la méthode de [différenciation adjointe](#). Vous pouvez exécuter ces simulateurs intégrés sur une ou plusieurs instances de CPU ou de GPU.

Avec Hybrid Jobs, vous pouvez désormais exécuter le code de votre algorithme variationnel en combinant un coprocesseur classique et un QPU, un simulateur à la demande Amazon Braket tel que SV1, ou directement en utilisant le simulateur intégré de PennyLane

Le simulateur intégré est déjà disponible avec le conteneur Hybrid Jobs, vous devez décorer votre fonction Python principale avec le `@hybrid_job` décorateur. Pour utiliser le PennyLane `lightning.gpu` simulateur, vous devez également spécifier une instance de GPU dans le, `InstanceConfig` comme indiqué dans l'extrait de code suivant :

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

@hybrid_job(device="local:pennylane/lightning.gpu",
            instance_config=InstanceConfig(instance_type="ml.g4dn.xlarge"))
def function(wires):
    dev = qml.device("lightning.gpu", wires=wires)
    ...
```

Reportez-vous à l'[exemple de bloc-notes](#) pour commencer à utiliser un simulateur PennyLane intégré avec Hybrid Jobs.

Dégradé adjoint activé PennyLane avec les simulateurs Amazon Braket

Avec le PennyLane plugin pour Amazon Braket, vous pouvez calculer les dégradés à l'aide de la méthode de différenciation adjointe lorsque vous l'exécutez sur le simulateur de vecteur d'état local ou. SV1

Remarque : Pour utiliser la méthode de différenciation adjointe, vous devez spécifier `diff_method='device'` dans votre `qnode` et `nondiff_method='adjoint'`. Consultez l'exemple suivant.

```
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

Actuellement, je PennyLane vais calculer des indices de regroupement pour les hamiltoniens QAOA et les utiliser pour diviser l'hamiltonien en plusieurs valeurs attendues. Si vous souhaitez utiliser SV1 la fonctionnalité de différenciation adjointe lorsque vous exécutez QAOA à partir de PennyLane, vous devez reconstruire l'hamiltonien des coûts en supprimant les indices de regroupement, comme suit :

```
cost_h, mixer_h = qml.qaoa.max_clique(g, constrained=False) cost_h = qml.Hamiltonian(cost_h.coeffs, cost_h.ops)
```

Utilisation de tâches hybrides et exécution PennyLane d'un algorithme QAOA

Dans cette section, vous utiliserez ce que vous avez appris pour écrire un véritable programme hybride à l'aide de PennyLane et d'une compilation paramétrique. Vous utilisez le script d'algorithme pour résoudre un problème d'algorithme d'optimisation approximative quantique (QAOA). Le programme crée une fonction de coût correspondant à un problème d'optimisation Max Cut classique, spécifie un circuit quantique paramétré et utilise une méthode de descente par gradient pour optimiser les paramètres afin de minimiser la fonction de coût. Dans cet exemple, nous générons le graphe du problème dans le script de l'algorithme pour des raisons de simplicité, mais pour les cas d'utilisation les plus courants, la meilleure pratique consiste à fournir la spécification du problème via un canal dédié dans la configuration des données d'entrée. L'indicateur `parametrize_differentiable` défini par défaut pour `True` que vous puissiez bénéficier automatiquement des avantages d'une amélioration des performances d'exécution grâce à la compilation paramétrique si elle est prise en charge. QPUs

```
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

La compilation paramétrique est prise en charge sur tous les modèles supraconducteurs basés sur des portes, QPUs à l'Rigetti Computing exception des programmes de niveau d'impulsion.

Exécutez des charges de travail hybrides avec des simulateurs PennyLane intégrés

Voyons comment utiliser les simulateurs intégrés disponibles PennyLane sur Amazon Braket Hybrid Jobs pour exécuter des charges de travail hybrides. Le simulateur intégré basé sur le GPU de PennyLane utilise la [bibliothèque Nvidia CuQuantum pour accélérer](#) les simulations de circuits. `lightning.gpu` Le simulateur GPU intégré est préconfiguré dans tous les [conteneurs de tâches](#) Braket que les utilisateurs peuvent utiliser immédiatement. Dans cette page, nous vous montrons comment l'utiliser `lightning.gpu` pour accélérer vos charges de travail hybrides.

Utilisation **lightning.gpu** pour les charges de travail QAOA

[Examinez les exemples d'algorithmes d'optimisation approximative quantique \(QAOA\)](#)

[présentés dans ce bloc-notes](#). Pour sélectionner un simulateur intégré, vous devez spécifier que l'`device` argument doit être une chaîne de la forme : `"local:<provider>/<simulator_name>"`. Par exemple, vous devez définir `"local:pennylane/lightning.gpu"` pour `lightning.gpu`. La chaîne de périphérique que vous donnez au Job hybride lorsque vous le lancez est transmise au job en tant que variable d'environnement `"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)
```

Dans cette page, comparez les deux simulateurs de vecteurs PennyLane d'état intégrés `lightning.qubit` (basés sur le processeur) et `lightning.gpu` (basés sur le GPU). Fournissez aux simulateurs des décompositions de portes personnalisées pour calculer différents dégradés.

Vous êtes maintenant prêt à préparer le script de lancement de la tâche hybride. Exécutez l'algorithme QAOA à l'aide de deux types d'instances : `m1.m5.2xlarge` et `m1.g4dn.xlarge`. Le type d'instance `m1.m5.2xlarge` est comparable à celui d'un ordinateur portable de développeur standard. `m1.g4dn.xlarge` s'agit d'une instance de calcul accéléré dotée d'un seul GPU NVIDIA T4 avec 16 Go de mémoire.

Pour exécuter le GPU, nous devons d'abord spécifier une image compatible et l'instance correcte (qui est par défaut une `m1.m5.2xlarge` instance).

```
from braket.aws import AwsSession
from braket.jobs.image_uris import Framework, retrieve_image

image_uri = retrieve_image(Framework.PL_PYTORCH, AwsSession().region)
instance_config = InstanceConfig(instanceType="m1.g4dn.xlarge")
```

Nous devons ensuite les saisir dans le décorateur de tâches hybrides, ainsi que les paramètres de périphérique mis à jour dans les arguments du système et des tâches hybrides.

```
@hybrid_job(
    device="local:pennylane/lightning.gpu",
    input_data=input_file_path,
    image_uri=image_uri,
    instance_config=instance_config)
def run_qaoa_hybrid_job_gpu(p=1, steps=10):
    params = np.random.rand(2, p)

    braket_task_tracker = Tracker()

    graph = nx.read_adjlist(input_file_path, nodetype=int)
    wires = list(graph.nodes)
    cost_h, _mixer_h = qaoa.maxcut(graph)
```



```
device_string = os.environ["AMZN_BRAKET_DEVICE_ARN"]
prefix, device_name = device_string.split("/")
dev= qml.device(simulator_name, wires=len(wires))
...
```

Note

Si vous spécifiez le `instance_config` comme utilisant une instance basée sur le GPU, mais que vous choisissez le device simulateur intégré basé sur le processeur (`lightning.qubit`), le GPU ne sera pas utilisé. Assurez-vous d'utiliser le simulateur intégré basé sur le GPU si vous souhaitez cibler le GPU !

Le temps d'itération moyen pour l'`m5.2xlarge` instance est d'environ 73 secondes, tandis que pour l'`m1.g4dn.xlarge` instance, il est d'environ 0,6 seconde. Pour ce flux de travail à 21 qubits, l'instance GPU nous offre une accélération 100 fois supérieure. Si vous consultez la [page de tarification](#) d'Amazon Braket Hybrid Jobs, vous pouvez constater que le coût par minute pour une `m5.2xlarge` instance est de 0,00768 USD, alors que pour l'instance, il est de 0,01227 `m1.g4dn.xlarge` USD. Dans ce cas, il est plus rapide et moins coûteux de l'exécuter sur l'instance GPU.

Apprentissage automatique quantique et parallélisme des données

Si votre type de charge de travail est l'apprentissage automatique quantique (QML) qui s'entraîne sur des ensembles de données, vous pouvez encore accélérer votre charge de travail grâce au parallélisme des données. Dans QML, le modèle contient un ou plusieurs circuits quantiques. Le modèle peut également contenir ou non des réseaux neuronaux classiques. Lors de l'entraînement du modèle avec le jeu de données, les paramètres du modèle sont mis à jour afin de minimiser la fonction de perte. Une fonction de perte est généralement définie pour un seul point de données, et la perte totale est définie pour la perte moyenne sur l'ensemble de données. En QML, les pertes sont généralement calculées en série avant d'être moyennées par rapport à la perte totale pour les calculs de gradient. Cette procédure prend beaucoup de temps, en particulier lorsqu'il existe des centaines de points de données.

Comme la perte d'un point de données ne dépend pas des autres points de données, les pertes peuvent être évaluées en parallèle ! Les pertes et les gradients associés à différents points de données peuvent être évalués en même temps. C'est ce que l'on appelle le parallélisme des données. Grâce à SageMaker sa bibliothèque de données parallèles distribuées, Amazon Braket

Hybrid Jobs vous permet d'utiliser plus facilement le parallélisme des données pour accélérer votre entraînement.

Considérez la charge de travail QML suivante pour le parallélisme des données, qui utilise le jeu de [données Sonar](#) du célèbre référentiel UCI comme exemple de classification binaire. L'ensemble de données Sonar comprend 208 points de données, chacun avec 60 caractéristiques collectées à partir des signaux du sonar rebondissant sur les matériaux. Chaque point de données est étiqueté « M » pour les mines ou « R » pour les roches. Notre modèle QML se compose d'une couche d'entrée, d'un circuit quantique en tant que couche cachée et d'une couche de sortie. Les couches d'entrée et de sortie sont des réseaux neuronaux classiques implémentés dans PyTorch. Le circuit quantique est intégré PyTorch aux réseaux neuronaux à l'aide PennyLane du module `qml.qnn`. Consultez nos [exemples de blocs-notes](#) pour plus de détails sur la charge de travail. Comme dans l'exemple QAOA ci-dessus, vous pouvez exploiter la puissance du GPU en utilisant des simulateurs intégrés basés sur le GPU tels que PennyLane ceux `lightning.gpu` pour améliorer les performances par rapport aux simulateurs intégrés basés sur le processeur.

Pour créer une tâche hybride, vous pouvez appeler `AwsQuantumJob.create` et spécifier le script d'algorithme, le périphérique et d'autres configurations via ses arguments de mots clés.

```
instance_config = InstanceConfig(instanceType='ml.g4dn.xlarge')

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,
    ...
)
```

Pour utiliser le parallélisme des données, vous devez modifier quelques lignes de code dans le script d'algorithme de la bibliothèque SageMaker distribuée afin de paralléliser correctement l'apprentissage. Tout d'abord, vous importez le `smdistributed` package qui effectue le plus gros du travail pour répartir vos charges de travail sur de multiples instances GPUs. Ce package est préconfiguré dans le Braket PyTorch et TensorFlow les conteneurs. Le `dist` module indique à

notre script d'algorithme le nombre total de points GPUs pour l'entraînement (`world_size`) ainsi que la fin `rank` `local_rank` d'un cœur de GPU. `rank` est l'indice absolu d'un GPU pour toutes les instances, tandis que `local_rank` est l'indice d'un GPU au sein d'une instance. Par exemple, s'il y a quatre instances dont huit chacune est GPUs allouée à la formation, les `rank` valeurs sont comprises entre 0 et 31 et `local_rank` entre 0 et 7.

```
import smdistributed.dataparallel.torch.distributed as dist

dp_info = {
    "world_size": dist.get_world_size(),
    "rank": dist.get_rank(),
    "local_rank": dist.get_local_rank(),
}
batch_size //= dp_info["world_size"] // 8
batch_size = max(batch_size, 1)
```

Ensuite, vous définissez un `DistributedSampler` en fonction du `world_size` `rank` et puis vous le transmettez au chargeur de données. Cet échantillonneur évite d' GPUs accéder à la même tranche d'un ensemble de données.

```
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,
)
```

Ensuite, vous utilisez la `DistributedDataParallel` classe pour activer le parallélisme des données.

```
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"])
```

Les modifications ci-dessus sont nécessaires pour utiliser le parallélisme des données. Dans QML, vous souhaitez souvent enregistrer les résultats et imprimer la progression de l'entraînement. Si chaque GPU exécute la commande d'enregistrement et d'impression, le journal sera inondé d'informations répétées et les résultats se remplaceront mutuellement. Pour éviter cela, vous ne pouvez enregistrer et imprimer qu'à partir du processeur graphique dont la valeur est `rank 0`.

```
if dp_info["rank"]==0:
    print('elapsed time: ', elapsed)
    torch.save(model.state_dict(), f"{output_dir}/test_local.pt")
    save_job_result({"last loss": loss_before})
```

Amazon Braket Hybrid Jobs prend en charge les types `m1.g4dn.12xlarge` instances pour la bibliothèque parallèle de données SageMaker distribuées. Vous configurez le type d'instance via l'`InstanceConfig` argument dans Hybrid Jobs. Pour que la bibliothèque de données parallèles SageMaker distribuées sache que le parallélisme des données est activé, vous devez ajouter deux hyperparamètres supplémentaires, définis `"true"` et `"sagemaker_instance_type"` définis `"sagemaker_distributed_dataparallel_enabled"` en fonction du type d'instance que vous utilisez. Ces deux hyperparamètres sont utilisés par le `smdistributed` package. Votre script d'algorithme n'a pas besoin de les utiliser explicitement. Dans le SDK Amazon Braket, il fournit un argument de mot clé pratique. `distribution="data_parallel"` Dans le cadre de la création de tâches hybrides, le SDK Amazon Braket insère automatiquement les deux hyperparamètres pour vous. Si vous utilisez l'API Amazon Braket, vous devez inclure ces deux hyperparamètres.

Une fois le parallélisme des instances et des données configuré, vous pouvez désormais soumettre votre tâche hybride. Il y en a 4 GPUs dans une `m1.g4dn.12xlarge` instance. Lorsque vous définissez `instanceCount=1`, la charge de travail est répartie sur les 8 GPUs de l'instance. Lorsque vous définissez une `instanceCount` valeur supérieure à un, la charge de travail est répartie sur toutes les instances GPUs disponibles. Lorsque vous utilisez plusieurs instances, chaque instance est facturée en fonction de la durée pendant laquelle vous l'utilisez. Par exemple, lorsque vous utilisez quatre instances, le temps facturable est quatre fois supérieur au temps d'exécution par instance, car quatre instances exécutent vos charges de travail en même temps.

```
instance_config = InstanceConfig(instanceType='m1.g4dn.12xlarge',
```

```
        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

Dans la création de tâches hybrides ci-dessus, `train_dp.py` se trouve le script d'algorithme modifié pour utiliser le parallélisme des données. N'oubliez pas que le parallélisme des données ne fonctionne correctement que lorsque vous modifiez votre script d'algorithme conformément à la section ci-dessus. Si l'option de parallélisme des données est activée sans qu'un script d'algorithme soit correctement modifié, la tâche hybride peut générer des erreurs ou chaque GPU peut traiter à plusieurs reprises la même tranche de données, ce qui est inefficace.

Si elle est utilisée correctement, l'utilisation de plusieurs instances peut entraîner une réduction de plusieurs ordres de grandeur en termes de temps et de coûts. Consultez l'[exemple de bloc-notes pour plus de détails](#).

Utilisation de CUDA-Q avec Amazon Braket

NVIDIA's CUDA-Q est une bibliothèque logicielle conçue pour programmer des algorithmes quantiques hybrides combinant CPUs, GPUs, et des unités de traitement quantiques (QPUs). Il fournit un modèle de programmation unifié, permettant aux développeurs d'exprimer des instructions classiques et quantiques dans un seul programme, rationalisant ainsi les flux de travail. CUDA-Q accélère la simulation et l'exécution de programmes quantiques grâce à ses simulateurs CPU et

GPU intégrés. CUDA-Qest disponible avec les instances natives de bloc-notes Braket (NBIs) et Amazon Braket Hybrid Jobs.

Dans cette section :

- [CUDA-Q en NBIs](#)
- [CUDA-Q dans les emplois hybrides](#)

CUDA-Q en NBIs

CUDA-Qest installé par défaut dans l'environnement Braket NBI. Vous pouvez ouvrir un CUDA-Q exemple de bloc-notes en accédant à la page du lanceur Jupyter et en sélectionnant la CUDA-Q vignette et Braket. Cela ouvre le bloc-notes d'exemple `0_hello_cudaq_jobs.ipynb` dans la fenêtre principale. Pour plus d'CUDA-Qexemples, consultez le panneau de gauche du `nvidia_cuda_q/` répertoire.

Vous pouvez également vérifier la version de CUDA-Q ou de tout autre package tiers installé dans votre NBI. Par exemple, vous pouvez exécuter la commande suivante dans une cellule de code de bloc-notes pour vérifier les versions des CUDA-Q packages Qiskit et Braket installées dans l'environnement. PennyLane

```
%pip freeze | grep -i -e cudaq -e qiskit -e pennylane -e braket
```

CUDA-Q dans les emplois hybrides

L'utilisation CUDA-Q d'[Amazon Braket Hybrid Jobs](#) offre un environnement informatique flexible et à la demande. Les instances de calcul ne s'exécutent que pendant la durée de votre charge de travail, ce qui garantit que vous ne payez que pour ce que vous utilisez. Amazon Braket Hybrid Jobs propose également une expérience évolutive. Les utilisateurs peuvent commencer par des instances plus petites pour le prototypage et les tests, puis passer à des instances plus grandes capables de gérer des charges de travail plus importantes pour des expériences complètes.

Le support Amazon Braket Hybrid Jobs GPUs est essentiel pour optimiser le potentielCUDA-Q. GPUs accélère considérablement les simulations de programmes quantiques par rapport aux simulateurs basés sur un processeur, en particulier lorsque vous travaillez avec des circuits à nombre de bits quantiques élevé. La parallélisation devient simple lorsque vous utilisez Amazon Braket CUDA-Q Hybrid Jobs. Hybrid Jobs simplifie la distribution de l'échantillonnage des circuits et des évaluations observables sur plusieurs nœuds de calcul. Cette parallélisation fluide des CUDA-Q

charges de travail permet aux utilisateurs de se concentrer davantage sur le développement de leurs charges de travail plutôt que sur la mise en place d'une infrastructure pour des expériences à grande échelle.

Pour commencer, consultez l'[exemple de CUDA-Q démarrage](#) sur le Github d'Amazon Braket pour utiliser un conteneur de tâches CUDA-Q hybrides fourni par Braket.

L'extrait de code suivant est un hello-world exemple d'exécution d'un CUDA-Q programme avec Amazon Braket Hybrid Jobs.

```
image_uri = retrieve_image(Framework.CUDAQ, AwsSession().region)

@hybrid_job(device='local:nvidia/qpp-cpu', image_uri=image_uri)
def hello_quantum():
    import cudaq

    # define the backend
    device=get_job_device_arn()
    cudaq.set_target(device.split('/')[1])

    # define the Bell circuit
    kernel = cudaq.make_kernel()
    qubits = kernel.qalloc(2)
    kernel.h(qubits[0])
    kernel.cx(qubits[0], qubits[1])

    # sample the Bell circuit
    result = cudaq.sample(kernel, shots_count=1000)
    measurement_probabilities = dict(result.items())

    return measurement_probabilities
```

L'exemple ci-dessus simule un circuit Bell sur un simulateur de processeur. Cet exemple s'exécute localement sur votre ordinateur portable ou sur votre bloc-notes Braket Jupyter. En raison de ce `local=True` paramètre, lorsque vous exécutez ce script, un conteneur démarre dans votre environnement local pour exécuter le CUDA-Q programme à des fins de test et de débogage. Une fois le test terminé, vous pouvez supprimer le `local=True` drapeau et exécuter votre tâche AWS. Pour en savoir plus, consultez l'article [Travailler avec Amazon Braket Hybrid Jobs](#).

Si vos charges de travail comportent un nombre de qubits élevé, un grand nombre de circuits ou un grand nombre d'itérations, vous pouvez utiliser des ressources de calcul CPU plus puissantes en

spécifiant le paramètre. `instance_config` L'extrait de code suivant montre comment configurer le `instance_config` paramètre dans le `hybrid_job` décorateur. Pour plus d'informations sur les types d'instances pris en charge, consultez [Configurer votre instance de tâche hybride](#). Pour obtenir la liste des types d'instances, consultez la section Types d'[instances Amazon EC2](#).

```
@hybrid_job(
    device="local:nvidia/qpp-cpu",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.c5.2xlarge"),
)
def my_job_script():
    ...
```

Pour les charges de travail plus exigeantes, vous pouvez exécuter vos charges de travail sur un simulateur de CUDA-Q GPU. Pour activer un simulateur de GPU, utilisez le nom `nvidia` du backend. Le `nvidia` backend fonctionne comme un simulateur de CUDA-Q GPU. Ensuite, sélectionnez un type d'instance Amazon EC2 qui prend en charge un NVIDIA GPU. L'extrait de code suivant montre le décorateur configuré par `GPUhybrid_job`.

```
@hybrid_job(
    device="local:nvidia/nvidia",
    image_uri=image_uri,
    instance_config=InstanceConfig(instanceType="ml.g4dn.xlarge"),
)
def my_job_script():
    ...
```

Amazon Braket Hybrid Jobs prend en charge les simulations de GPU NBLs parallèles avec. CUDA-Q Vous pouvez paralléliser l'évaluation de plusieurs observables ou de plusieurs circuits pour améliorer les performances de votre charge de travail. Pour paralléliser plusieurs observables, apportez les modifications suivantes à votre script d'algorithme.

Définissez `lmgpuoption` du `nvidia` backend. Cela est nécessaire pour paralléliser les observables. La parallélisation utilise le MPI pour la communication entre les deux GPUs. Le MPI doit donc être initialisé avant l'exécution et finalisé après celle-ci.

Ensuite, spécifiez le mode d'exécution en réglant `execution=cudaq.parallel.mpi`. L'extrait de code suivant montre ces modifications.

```
cudaq.set_target("nvidia", option="mqpu")
```



```
cudaq.mpi.initialize()
result = cudaq.observe(
    kernel, hamiltonian, shots_count=n_shots, execution=cudaq.parallel.mpi
)
cudaq.mpi.finalize()
```

Dans le `hybrid_job` décorateur, spécifiez un type d'instance qui héberge plusieurs instances GPUs , comme indiqué dans l'extrait de code suivant.

```
@hybrid_job(
    device="local:nvidia/nvidia-mqpu",
    instance_config=InstanceConfig(instanceType="ml.g4dn.12xlarge", instanceCount=1),
    image_uri=image_uri,
)
def parallel_observables_gpu_job(sagemaker_mpi_enabled=True):
    ...
```

Le [carnet de simulations parallèles](#) du Github consacré aux exemples d'Amazon Braket fournit des end-to-end exemples qui montrent comment exécuter des simulations de programmes quantiques sur des backends GPU et effectuer des simulations parallèles d'observables et de lots de circuits.

Exécution de vos charges de travail sur des ordinateurs quantiques

Après avoir terminé les tests du simulateur, vous pouvez passer à l'exécution d'expériences sur QPUs. Il suffit de passer de la cible à un QPU Amazon Braket, tel que les appareils IQM IonQ, ou. Rigetti L'extrait de code suivant montre comment définir la cible sur le périphérique. IQM Garnet Pour obtenir la liste des produits disponibles QPUs, consultez la [console Amazon Braket](#).

```
device_arn = "arn:aws:braket:eu-north-1::device/qpu/iqm/Garnet"
cudaq.set_target("braket", machine=device_arn)
```

Pour plus d'informations sur les jobs hybrides, consultez la section [Working with Amazon Braket Hybrid Jobs](#) dans le guide du développeur. Pour en savoir plus sur CUDA-Q, consultez [Documentation NVIDIA CUDA-Q](#).

Résolution des problèmes liés à Amazon Braket

Utilisez les informations de dépannage et les solutions de cette section pour résoudre les problèmes liés à Amazon Braket.

Dans cette section :

- [AccessDeniedException](#)
- [Une erreur s'est produite \(ValidationException\) lors de l'appel de l' CreateQuantumTask opération](#)
- [Une fonctionnalité du SDK ne fonctionne pas](#)
- [Une tâche hybride échoue en raison de ServiceQuotaExceededException](#)
- [Les composants ont cessé de fonctionner dans une instance de bloc-notes](#)
- [Résolution des problèmes liés à la mise à niveau de Python 3.12](#)
- [Résolution des problèmes liés à OpenQASM](#)

AccessDeniedException

Si vous recevez un message AccessDeniedException lors de l'activation ou de l'utilisation de Braket, vous tentez probablement d'activer ou d'utiliser Braket dans une région à laquelle votre rôle restreint n'y a pas accès.

Dans ce cas, contactez votre AWS administrateur interne pour savoir laquelle des conditions suivantes s'applique :

- Si des restrictions de rôles empêchent l'accès à une région.
- Si le rôle que vous essayez d'utiliser est autorisé à utiliser Braket.

Si votre rôle n'a pas accès à une région donnée lors de l'utilisation de Braket, vous ne pourrez pas utiliser d'appareils dans cette région en particulier.

Une erreur s'est produite (ValidationException) lors de l'appel de l' CreateQuantumTask opération

Si vous recevez un message d'erreur similaire à : `An error occurred (ValidationException) when calling the CreateQuantumTask operation: Caller`

doesn't have access to amazon-braket-... Vérifiez que vous faites référence à un dossier s3_existant. Braket ne crée pas automatiquement de nouveaux compartiments et préfixes Amazon S3 pour vous.

Si vous y accédez API directement et que vous recevez un message d'erreur similaire à : `Failed to create quantum task: Caller doesn't have access to s3://MY_BUCKET` Vérifiez que vous n'êtes pas inclus `s3://` dans le chemin du compartiment Amazon S3.

Une fonctionnalité du SDK ne fonctionne pas

Votre version de Python doit être 3.10 ou supérieure. Pour les tâches hybrides Amazon Braket, nous recommandons Python 3.12.

Vérifiez que votre SDK et vos schémas le sont. up-to-date Pour mettre à jour le SDK depuis le bloc-notes ou votre éditeur Python, exécutez la commande suivante :

```
pip install amazon-braket-sdk --upgrade --upgrade-strategy eager
```

Pour mettre à jour les schémas, exécutez la commande suivante :

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

Si vous accédez à Amazon Braket depuis votre propre client, vérifiez que votre [AWS région est définie sur une région](#) prise en charge par Amazon Braket.

Une tâche hybride échoue en raison de ServiceQuotaExceededException

Une tâche hybride exécutant des tâches quantiques sur les simulateurs Amazon Braket peut ne pas être créée si vous dépassez la limite de tâches quantiques simultanées pour le simulateur que vous ciblez. Pour plus d'informations sur les limites de service, consultez la rubrique [Quotas](#).

Si vous exécutez des tâches simultanées sur un simulateur dans le cadre de plusieurs tâches hybrides à partir de votre compte, cette erreur peut se produire.

Pour voir le nombre de tâches quantiques simultanées par rapport à un dispositif de simulation spécifique, utilisez le `search-quantum-tasksAPI`, comme indiqué dans l'exemple de code suivant.

```
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
```

Vous pouvez également consulter les tâches quantiques créées par rapport à un appareil à l'aide CloudWatch des métriques Amazon : Braket > Par appareil.

Pour éviter de rencontrer ces erreurs :

1. Demandez une augmentation du quota de service pour le nombre de tâches quantiques simultanées pour le simulateur. Cela ne s'applique qu'à l'SV1 appareil.
2. Gérez les ServiceQuotaExceeded exceptions dans votre code et réessayez.

Les composants ont cessé de fonctionner dans une instance de bloc-notes

Si certains composants de votre bloc-notes ne fonctionnent plus, essayez ce qui suit :

1. Téléchargez tous les blocs-notes que vous avez créés ou modifiés sur un disque local.
2. Arrêtez votre instance de bloc-notes.
3. Supprimez votre instance de bloc-notes.
4. Créez une nouvelle instance de bloc-notes avec un nom différent.
5. Téléchargez les blocs-notes sur la nouvelle instance.

Résolution des problèmes liés à la mise à niveau de Python 3.12

Date d'entrée en vigueur : 21 janvier 2026

Présentation de

À compter du 21 janvier 2026, Amazon Braket met à niveau le runtime Python de la version 3.10 à la version 3.12 pour [toutes les instances Notebook et les images de conteneurs gérés](#) (Base, TensorFlow CUDA-Q et). PyTorch Ce guide propose des solutions aux problèmes de compatibilité courants.

Dans cette section :

- [Messages d'erreur courants](#)
- [Ordinateurs portables gérés par Braket](#)
- [Job Decorator Hybrid](#)
- [Bring-Your-Own-Container \(BYOC\)](#)
- [Mise à niveau de l'instance Braket Notebook](#)

Messages d'erreur courants

Erreur de non-concordance entre les versions Python du SDK

Erreur :

```
RuntimeError: Python version must match between local environment and container. Client is running Python 3.10 locally, but container uses Python 3.12.
```

Cause : Le SDK Braket a détecté que votre bloc-notes exécutait Python 3.10 alors que le conteneur Hybrid Job exécutait Python 3.12.

Solution : [mettez à niveau votre bloc-notes vers Python 3.12 ou insérez-le dans des conteneurs Python 3.10](#).

Erreur de sérialisation Cloudpickle

Erreur :

```
TypeError: code() argument 13 must be str, not int
```

Cause : Si la validation du SDK est ignorée, cloudpickle ne parvient pas à sérialiser le code entre Python 3.10 et 3.12 en raison d'un CodeType changement de constructeur dans Python 3.12.

Solution : assurez-vous que votre bloc-notes et votre conteneur utilisent la même version de Python.

Ordinateurs portables gérés par Braket

Si vous exécutez une instance Braket Notebook sur Python 3.10 et que vous soumettez des tâches hybrides, vous rencontrerez des erreurs d'incompatibilité de version car les conteneurs de tâches utilisent désormais Python 3.12 par défaut.

Vous avez deux options :

1. [Recommandé] Créez une nouvelle instance de bloc-notes avec Python 3.12 - voir Mise à niveau de l'instance de [bloc-notes de Braket](#)
2. Épinglez des conteneurs Python 3.10 - voir [Hybrid Job Decorator](#)

Job Decorator Hybrid

Pour utiliser le `@hybrid_job` décorateur, la version Python de votre environnement doit correspondre à la version Python du conteneur.

Option 1 : utiliser des conteneurs Python 3.12 (recommandé)

Si vous avez mis à niveau votre environnement vers Python 3.12, il utilise la dernière balise (comportement par défaut).

Option 2 : utiliser des conteneurs Python 3.10

Si vous devez rester sur Python 3.10, spécifiez explicitement le `image_uri` paramètre dans le `@hybrid_job` décorateur.

Balises de conteneur Python 3.10 :

Nom de l'image	Balise
Base	1.0-cpu-py310-ubuntu22.04
CUDA-Q	0.12.0-cpu-py310-0.12.0
PyTorch	2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow	2.14.1-gpu-py310-cu118-ubuntu20.04

L'exemple suivant concerne la région us-west-2.

Image complète URIs :

```
Base:          292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-py310-ubuntu22.04
CUDA-Q:        292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-py310-0.12.0
PyTorch:       292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:2.2.0-gpu-py310-cu121-ubuntu20.04
TensorFlow:    292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:2.14.1-gpu-py310-cu118-ubuntu20.04
```

Exemple :

```
from braket.jobs.hybrid_job import hybrid_job
from braket.devices import Devices

device_arn = Devices.Amazon.SV1

@hybrid_job(
    device=device_arn,
    image_uri="292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-py310-ubuntu22.04"
)
def my_job():
    pass
```

Note

- Les conteneurs Python 3.10 resteront disponibles mais ne recevront pas de mises à jour.
- Consultez la section [Définition de l'environnement pour votre script d'algorithme](#).

Bring-Your-Own-Container (BYOC)

Si votre Dockerfile utilise une image gérée par Braket avec la dernière balise, la reconstruction après le 21 janvier 2026 extraira les images compatibles avec Python 3.12.

Pour continuer à utiliser les images gérées par Braket compatibles avec Python 3.10, mettez à jour votre Dockerfile :

Avant (obtient Python 3.12 après la mise à niveau) :

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:latest
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:latest
```

Après (reste sur Python 3.10) :

```
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-base-jobs:1.0-cpu-
py310-ubuntu22.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-cudaq-jobs:0.12.0-cpu-
py310-0.12.0
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-pytorch-jobs:2.2.0-gpu-
py310-cu121-ubuntu20.04
FROM 292282985366.dkr.ecr.us-west-2.amazonaws.com/amazon-braket-tensorflow-jobs:2.14.1-
gpu-py310-cu118-ubuntu20.04
```

Mise à niveau de l'instance Braket Notebook

Procédez comme suit pour effectuer la mise à niveau vers Python 3.12 :

Important

Avant de supprimer votre instance de bloc-notes, assurez-vous d'avoir téléchargé tous les blocs-notes et fichiers que vous souhaitez conserver. Ces données ne peuvent pas être récupérées après leur suppression.

1. Téléchargez tous les blocs-notes que vous avez créés ou modifiés sur un disque local.
2. Arrêtez votre instance de bloc-notes.
3. Supprimez votre instance de bloc-notes.
4. Créez une nouvelle instance de bloc-notes avec un nom différent.
5. Téléchargez vos blocs-notes sur la nouvelle instance.

Résolution des problèmes liés à OpenQASM

Cette section fournit des conseils de dépannage qui peuvent être utiles en cas d'erreur lors de l'utilisation d'OpenQASM 3.0.

Dans cette section :

- [Inclure une erreur de déclaration](#)
- [Erreur non contiguë qubits](#)
- [Mélange d'erreur physique qubits et d'qubitserreur virtuelle](#)
- [Erreur lors de la demande de types de résultats et de la mesure qubits dans le même programme](#)
- [Erreur de dépassement des limites classiques et des limites de qubit registre](#)
- [Boîte non précédée d'une erreur de pragma textuelle](#)
- [Erreur concernant les portes natives manquantes dans les boîtes Verbatim](#)
- [Erreur physique manquante dans les boîtes verbatim qubits](#)
- [Il manque une erreur « bracket » dans le pragma textuel](#)
- [Une seule qubits erreur ne peut pas être indexée](#)
- [L'erreur physique qubits dans une qubit porte à deux portes n'est pas connectée](#)
- [Avertissement de support du simulateur local](#)

Inclure une erreur de déclaration

Braket n'a actuellement pas de fichier de bibliothèque de portail standard à inclure dans les programmes OpenQASM. Par exemple, l'exemple suivant génère une erreur d'analyseur.

```
OPENQASM 3;  
include "standardlib.inc";
```

Ce code génère le message d'erreur : `No terminal matches ''' in the current parser context, at line 2 col 17.`

Erreur non contiguë qubits

L'utilisation de la méthode non contiguë qubits sur les appareils dont la fonctionnalité est définie `requiresContiguousQubitIndices` comme telle entraîne une erreur. `true`

Lorsque vous exécutez des tâches quantiques sur des simulateurs IonQ, le programme suivant déclenche l'erreur.

```
OPENQASM 3;

qubit[4] q;

h q[0];
cnot q[0], q[2];
cnot q[0], q[3];
```

Ce code génère le message d'erreur : `Device requires contiguous qubits. Qubit register q has unused qubits q[1], q[4].`

Mélange d'erreur physique qubits et d'erreur virtuelle

Le mélange du physique qubits et du virtuel qubits dans le même programme n'est pas autorisé et entraîne une erreur. Le code suivant génère l'erreur.

```
OPENQASM 3;

qubit[2] q;
cnot q[0], $1;
```

Ce code génère le message d'erreur : `[line 4] mixes physical qubits and qubits registers.`

Erreur lors de la demande de types de résultats et de la mesure qubits dans le même programme

La demande de types de résultats qubits mesurés explicitement dans le même programme entraîne une erreur. Le code suivant génère l'erreur.

```
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])
```

Ce code génère le message d'erreur : `Qubits should not be explicitly measured when result types are requested.`

Erreur de dépassement des limites classiques et des limites de qubit registre

Un seul registre classique et un seul qubit registre sont autorisés. Le code suivant génère l'erreur.

```
OPENQASM 3;  
  
qubit[2] q0;  
qubit[2] q1;
```

Ce code génère le message d'erreur : `[line 4] cannot declare a qubit register. Only 1 qubit register is supported.`

Boîte non précédée d'une erreur de pragma textuelle

Toutes les cases doivent être précédées d'un verbatim pragma. Le code suivant génère l'erreur.

```
box{  
  rx(0.5) $0;  
}
```

Ce code génère le message d'erreur : `In verbatim boxes, native gates are required. x is not a device native gate.`

Erreur concernant les portes natives manquantes dans les boîtes Verbatim

Les boîtes verbatim doivent avoir des portes natives et physiques. qubits Le code suivant génère l'erreur `native gates.`

```
#pragma braket verbatim  
box{  
  x $0;  
}
```

Ce code génère le message d'erreur : `In verbatim boxes, native gates are required. x is not a device native gate.`

Erreur physique manquante dans les boîtes verbatim qubits

Les boîtes verbatim doivent être physiques. qubits Le code suivant génère l'qubit erreur physique manquante.

```
qubit[2] q;

#pragma braket verbatim
box{
  rx(0.1) q[0];
}
```

Ce code génère le message d'erreur : `Physical qubits are required in verbatim box.`

Il manque une erreur « braket » dans le pragma textuel

Vous devez inclure le mot « braket » dans le verbatim pragma. Le code suivant génère l'erreur.

```
#pragma braket verbatim      // Correct
#pragma verbatim              // wrong
```

Ce code génère le message d'erreur : `You must include "braket" in the verbatim pragma`

Une seule qubits erreur ne peut pas être indexée

Le single qubits ne peut pas être indexé. Le code suivant génère l'erreur.

```
OPENQASM 3;

qubit q;
h q[0];
```

Ce code génère l'erreur suivante : `[line 4] single qubit cannot be indexed.`

Cependant, les qubit tableaux individuels peuvent être indexés comme suit :

```
OPENQASM 3;

qubit[1] q;
h q[0];    // This is valid
```

L'erreur physique qubits dans une qubit porte à deux portes n'est pas connectée

Pour utiliser le physique qubits, vérifiez d'abord que le périphérique utilise le physique qubits en cochant,

```
device.properties.action[DeviceActionType.OPENQASM].supportPhysicalQubits
puis vérifiez le graphique de connectivité en cochant
device.properties.paradigm.connectivity.connectivityGraph
ou device.properties.paradigm.connectivity.fullyConnected.
```

```
OPENQASM 3;

cnot $0, $14;
```

Ce code génère le message d'erreur : `[line 3] has disconnected qubits 0 and 14`

Avertissement de support du simulateur local

Il `LocalSimulator` prend en charge les fonctionnalités avancées d'OpenQASM qui peuvent ne pas être disponibles sur les simulateurs QPUs ou à la demande. Si votre programme contient des fonctionnalités linguistiques spécifiques uniquement à `LocalSimulator`, comme le montre l'exemple suivant, vous recevrez un avertissement.

```
qasm_string = """
qubit[2] q;

h q[0];
ctrl @ x q[0], q[1];
"""
qasm_program = Program(source=qasm_string)
```

Ce code génère l'avertissement suivant : `Ce programme utilise les fonctionnalités du langage OpenQASM uniquement prises en charge dans le. LocalSimulator Certaines de ces fonctionnalités peuvent ne pas être prises en charge sur les simulateurs QPUs ou à la demande.

Pour plus d'informations sur les fonctionnalités prises en charge par OpenQASM, consultez la page [Support des fonctionnalités avancées d'OpenQASM](#) sur le simulateur local.

Sécurité dans Amazon Braket

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez de centres de données et d'architectures réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le [modèle de responsabilité partagée](#) décrit ceci comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de [AWS conformité Programmes](#) de de conformité. Pour en savoir plus sur les programmes de conformité qui s'appliquent à Amazon Braket, consultez la section [AWS Services concernés par programme de conformitéAWS](#) .
- Sécurité dans le cloud — Votre responsabilité est déterminée par le AWS service que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris la sensibilité de vos données, les exigences de votre entreprise, ainsi que la législation et la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de Braket. Les rubriques suivantes expliquent comment configurer Braket pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres AWS services qui vous aident à surveiller et à sécuriser vos ressources Braket.

Dans cette section :

- [Responsabilité partagée en matière de sécurité](#)
- [Protection des données](#)
- [Conservation des données](#)
- [Gérer l'accès à Amazon Braket](#)
- [Rôle lié au service Amazon Braket](#)
- [Validation de conformité pour Amazon Braket](#)
- [Sécurité de l'infrastructure dans Amazon Braket](#)
- [Sécurité des fournisseurs de matériel Amazon Braket](#)
- [Points de terminaison Amazon VPC pour Amazon Braket](#)

Responsabilité partagée en matière de sécurité

La sécurité est une responsabilité partagée entre vous AWS et vous. Le [modèle de responsabilité partagée](#) décrit cette notion par les termes sécurité du cloud et sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui s'exécute Services AWS dans le AWS Cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des [programmes de conformitéAWS](#). Pour en savoir plus sur les programmes de conformité qui s'appliquent à Amazon Braket, consultez la section [AWS Services concernés par programme de conformité](#).
- Sécurité dans le cloud : vous êtes responsable de garder le contrôle sur votre contenu hébergé sur cette AWS infrastructure. Ce contenu inclut la configuration de la sécurité et les tâches de gestion pour le Services AWS produit que vous utilisez.

Protection des données

Le [modèle de responsabilité AWS partagée](#) s'applique à la protection des données dans Amazon Braket. Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez [Questions fréquentes \(FAQ\) sur la confidentialité des données](#). Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog [Modèle de responsabilité partagée d'AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.

- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section [Utilisation des CloudTrail sentiers](#) dans le guide de AWS CloudTrail l'utilisateur.
- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez [Norme FIPS \(Federal Information Processing Standard\) 140-3](#).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec Amazon Braket ou une autre entreprise à Services AWS l'aide de la console, de l'API ou. AWS CLI AWS SDKs Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Conservation des données

Au bout de 90 jours, Amazon Braket supprime automatiquement toutes les tâches quantiques IDs et autres métadonnées associées à vos tâches quantiques. En raison de cette politique de conservation des données, ces tâches et résultats ne peuvent plus être récupérés par recherche depuis la console Amazon Braket, bien qu'ils restent stockés dans votre compartiment S3.

Si vous avez besoin d'accéder aux tâches quantiques historiques et aux résultats stockés dans votre compartiment S3 pendant plus de 90 jours, vous devez conserver un enregistrement séparé de votre identifiant de tâche et des autres métadonnées associées à ces données. Assurez-vous de sauvegarder les informations avant 90 jours. Vous pouvez utiliser ces informations enregistrées pour récupérer les données historiques.

Gérer l'accès à Amazon Braket

Ce chapitre décrit les autorisations requises pour exécuter Amazon Braket ou pour restreindre l'accès à des utilisateurs et à des rôles spécifiques. Vous pouvez accorder (ou refuser) les autorisations requises à n'importe quel utilisateur ou rôle de votre compte. Pour ce faire, associez la politique Amazon Braket appropriée à cet utilisateur ou à ce rôle dans votre compte, comme décrit dans les sections suivantes.

Comme condition préalable, vous devez [activer Amazon Braket](#). Pour activer Braket, assurez-vous de vous connecter en tant qu'utilisateur ou en tant que rôle disposant (1) des autorisations d'administrateur ou (2) de la `AmazonBraketFullAccess` politique et des autorisations nécessaires pour créer des buckets Amazon Simple Storage Service (Amazon S3).

Dans cette section :

- [Ressources Amazon Braket](#)
- [Carnets de notes et rôles](#)
- [AWS politiques gérées pour Amazon Braket](#)
- [Restreindre l'accès des utilisateurs à certains appareils](#)
- [Restreindre l'accès des utilisateurs à certaines instances de blocs-notes](#)
- [Restreindre l'accès des utilisateurs à certains compartiments S3](#)

Ressources Amazon Braket

Braket crée un type de ressource : la ressource de tâches quantiques. Le nom de AWS ressource (ARN) pour ce type de ressource est le suivant :

- Nom de la ressource AWS : `::Service::Braket`
- ARN Regex : `arn : $ {Partition} :braket : $ {Region} :$ {Account} :quantum-task/$ {} RandomId`

Carnets de notes et rôles

Vous pouvez utiliser le type de ressource bloc-notes dans Braket. Un carnet est une ressource Amazon SageMaker AI que Braket est capable de partager. Pour utiliser un bloc-notes avec Braket, vous devez spécifier un rôle IAM dont le nom commence par `AmazonBraketServiceSageMakerNotebook`

Pour créer un bloc-notes, vous devez utiliser un rôle doté d'autorisations d'administrateur ou associé à la politique intégrée suivante.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CreateTheRole",
      "Effect": "Allow",
      "Action": "iam:CreateRole",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
    },
    {
      "Sid": "CreateThePolicy",
      "Effect": "Allow",
      "Action": "iam:CreatePolicy",
      "Resource": [
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
        "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
      ]
    },
    {
      "Sid": "AttachTheRolePolicy",
      "Effect": "Allow",
      "Action": "iam:AttachRolePolicy",
      "Resource": "arn:aws:iam::*:role/service-role/
AmazonBraketServiceSageMakerNotebookRole*",
      "Condition": {
        "ArnLike": {
          "iam:PolicyARN": [
            "arn:aws:iam::aws:policy/AmazonBraketFullAccess",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookAccess*",
            "arn:aws:iam::*:policy/service-role/
AmazonBraketServiceSageMakerNotebookRole*"
          ]
        }
      }
    }
  ]
}
```

```
}  
}  
]  
}
```

Pour créer le rôle, suivez les étapes indiquées sur la page [Créer un bloc-notes](#) ou demandez à votre administrateur de le créer pour vous. Assurez-vous que la `AmazonBraketFullAccess` politique est jointe.

Après avoir créé le rôle, vous pouvez le réutiliser pour tous les blocs-notes que vous lancerez dans le futur.

AWS politiques gérées pour Amazon Braket

Une politique AWS gérée est une politique autonome créée et administrée par AWS. Les politiques gérées sont conçues pour fournir des autorisations pour de nombreux cas d'utilisation courants afin que vous puissiez commencer à attribuer des autorisations aux utilisateurs, aux groupes et aux rôles.

N'oubliez pas que les politiques AWS gérées peuvent ne pas accorder d'autorisations de moindre privilège pour vos cas d'utilisation spécifiques, car elles sont accessibles à tous les AWS clients. Nous vous recommandons de réduire encore les autorisations en définissant des [politiques gérées par le client](#) qui sont propres à vos cas d'utilisation.

Vous ne pouvez pas modifier les autorisations définies dans les politiques AWS gérées. Si les autorisations définies dans une politique AWS gérée sont mises à jour, la mise à jour affecte toutes les identités principales (utilisateurs, groupes et rôles) auxquelles la politique est attachée. AWS est le plus susceptible de mettre à jour une politique AWS gérée lorsqu'une nouvelle politique Service AWS est lancée ou lorsque de nouvelles opérations d'API sont disponibles pour les services existants.

Pour plus d'informations, consultez [Politiques gérées par AWS](#) dans le Guide de l'utilisateur IAM.

Rubriques

- [AWS politique gérée : AmazonBraketFullAccess](#)
- [AWS politique gérée : AmazonBraketJobsExecutionPolicy](#)
- [AWS politique gérée : AmazonBraketServiceRolePolicy](#)
- [Amazon Braket met à jour les politiques gérées AWS](#)

AWS politique gérée : AmazonBraketFullAccess

La AmazonBraketFullAccesspolitique accorde des autorisations pour les opérations d'Amazon Braket, y compris des autorisations pour les tâches suivantes :

- Téléchargez des conteneurs depuis Amazon Elastic Container Registry : pour lire et télécharger des images de conteneurs utilisées pour la fonctionnalité Amazon Braket Hybrid Jobs. Les conteneurs doivent être conformes au format « `arn:aws:ecr : ::repository/amazon-braket` ».
- Conservez les AWS CloudTrail journaux : pour toutes les actions de description, d'obtention et de liste, en plus du démarrage et de l'arrêt des requêtes, du test des filtres de mesures et du filtrage des événements des journaux. Le fichier AWS CloudTrail journal contient un enregistrement de toutes les API activités Amazon Braket effectuées sur votre compte.
- Utiliser les rôles pour contrôler les ressources : pour créer un rôle lié à un service dans votre compte. Le rôle lié au service a accès aux AWS ressources en votre nom. Il ne peut être utilisé que par le service Amazon Braket. Il s'agit également de transmettre des rôles IAM à Amazon CreateJob API Braket, de créer un rôle et d'associer une politique définie AmazonBraketFullAccess à ce rôle.
- Créez des groupes de journaux, des événements et des groupes de journaux de requêtes afin de gérer les fichiers journaux d'utilisation de votre compte. Pour créer, stocker et consulter les informations de journalisation relatives à l'utilisation d'Amazon Braket sur votre compte. Interrogez les métriques sur les groupes de journaux de tâches hybrides. Indiquez le chemin de braket approprié et autorisez l'enregistrement des données du journal. Insérez des données métriques CloudWatch.
- Créez et stockez des données dans des compartiments Amazon S3, et listez tous les compartiments. Pour créer des compartiments S3, listez les compartiments S3 de votre compte, placez des objets dans et récupérez des objets depuis n'importe quel compartiment de votre compte dont le nom commence par amazon-braket-. Ces autorisations sont requises pour que Braket puisse placer des fichiers contenant les résultats de tâches quantiques traitées dans le compartiment et les récupérer depuis le compartiment.
- Transmettre des rôles IAM — Pour transmettre des rôles IAM au. CreateJob API
- Amazon SageMaker AI Notebook — Pour créer et gérer des instances de SageMaker bloc-notes associées à la ressource « `arn:aws:sagemaker : ::notebook-instance/amazon-braket-` ».
- Validez les quotas de service : pour créer des blocs-notes basés sur l' SageMaker IA et des tâches Amazon Braket Hybrid, le nombre de vos ressources ne peut pas [dépasser les quotas de](#) votre compte.

- Consultez les prix des produits — Passez en revue et planifiez les coûts du matériel quantique avant de soumettre vos charges de travail.

Pour consulter les autorisations associées à cette politique, consultez le [AmazonBraketFullAccess](#) document AWS Managed Policy Reference.

AWS politique gérée : AmazonBraketJobsExecutionPolicy

La AmazonBraketJobsExecutionPolicy politique accorde des autorisations pour les rôles d'exécution utilisés dans Amazon Braket Hybrid Jobs comme suit :

- Téléchargez des conteneurs depuis Amazon Elastic Container Registry : autorisations de lecture et de téléchargement d'images de conteneurs utilisées pour la fonctionnalité Amazon Braket Hybrid Jobs. Les conteneurs doivent être conformes au format « `arn:aws:ecr : *: *: repository/amazon-braket*` ».
- Créez des groupes de journaux, enregistrez les événements et interrogez des groupes de journaux afin de gérer les fichiers journaux d'utilisation de votre compte : créez, stockez et consultez les informations de journalisation relatives à l'utilisation d'Amazon Braket sur votre compte. Interrogez les métriques sur les groupes de journaux de tâches hybrides. Indiquez le chemin de braket approprié et autorisez l'enregistrement des données du journal. Insérez des données métriques CloudWatch.
- Stockez les données dans des compartiments Amazon S3 : listez les compartiments S3 de votre compte, insérez des objets et récupérez des objets depuis n'importe quel compartiment de votre compte dont le nom commence par `amazon-braket-`. Ces autorisations sont requises pour que Braket puisse placer des fichiers contenant les résultats de tâches quantiques traitées dans le compartiment et les récupérer depuis le compartiment.
- Transmettre des rôles IAM — Transmettre des rôles IAM au `CreateJob` API Les rôles doivent être conformes au format `arn:aws:iam : :* *: :role/service-role/AmazonBraketJobsExecutionRole`

Pour consulter les autorisations associées à cette politique, consultez le [AmazonBraketJobsExecutionPolicy](#) document AWS Managed Policy Reference.

AWS politique gérée : AmazonBraketServiceRolePolicy

La AmazonBraketServiceRolePolicy politique accorde des autorisations pour les opérations d'Amazon Braket, y compris des autorisations pour les tâches suivantes :

- Amazon S3 : autorisations permettant de répertorier les compartiments de votre compte, d'y placer des objets et d'en récupérer des objets depuis n'importe quel compartiment de votre compte avec un nom commençant par `amazon-braket-`
- Amazon CloudWatch Logs : autorisations permettant de répertorier et de créer des groupes de journaux, de créer les flux de journaux associés et de placer des événements dans le groupe de journaux créé pour Amazon Braket.

Pour plus d'informations sur les rôles liés à un service, consultez la section [Rôle lié à un service Amazon Braket](#).

Pour consulter les autorisations associées à cette politique, consultez le [AmazonBraketServiceRolePolicy](#) document AWS Managed Policy Reference.

Amazon Braket met à jour les politiques gérées AWS

Le tableau suivant fournit des informations sur les mises à jour des politiques AWS gérées pour Amazon Braket depuis le moment où ce service a commencé à suivre ces modifications.

Modification	Description	Date
AmazonBraketServiceRolePolicy - Politique de gestion des ressources	La portée de la condition « <code>aws : ResourceAccount</code> » : « <code>\$ {aws :PrincipalAccount}</code> » a été ajoutée à Amazon S3 et CloudWatch les actions enregistrées.	11 juillet 2025
AmazonBraketFullAccess - Politique d'accès complet à Braket	L'action « tarification : <code>GetProducts</code> » a été ajoutée.	14 avril 2025
AmazonBraketFullAccess - Politique d'accès complet à Braket	La portée de la condition <code>ResourceAccount</code> « <code>aws : </code> » : « <code>\$ {aws :PrincipalAccount}</code> » a été ajoutée aux actions S3.	07 mars 2025
AmazonBraketFullAccess - Politique d'accès complet à Braket	Les actions <code>servicequotas : GetServiceQuota</code> et <code>cloudwatch :</code> ont été ajoutées. <code>GetMetricData</code>	24 mars 2023

Modification	Description	Date
AmazonBraketFullAccess - Politique d'accès complet à Braket	Ajout des ListAllMyBuckets autorisations s3 : permettant d'afficher et d'inspecter les buckets Amazon S3 utilisés.	31 mars 2022
AmazonBraketFullAccess - Politique d'accès complet à Braket	iam ajusté au crochet : PassRole autorisations AmazonBraketFullAccess pour inclure le service-role/ chemin.	29 novembre 2021
AmazonBraketJobsExecutionPolicy - Politique d'exécution des tâches hybrides pour Amazon Braket Hybrid Jobs	Braket a mis à jour l'ARN du rôle d'exécution des tâches hybrides pour inclure le service-role/ chemin.	29 novembre 2021
Braket a commencé à suivre les modifications	Braket a commencé à suivre les modifications apportées à ses politiques AWS gérées.	29 novembre 2021

Restreindre l'accès des utilisateurs à certains appareils

Pour restreindre l'accès des utilisateurs à certains appareils Braket, vous pouvez ajouter une politique de refus d'autorisations à un IAM rôle spécifique.

Les actions suivantes peuvent être restreintes :

- `CreateQuantumTask`- pour refuser la création de tâches quantiques sur des appareils spécifiques.
- `CreateJob`- pour empêcher la création d'emplois hybrides sur certains appareils.
- `GetDevice`- pour refuser d'obtenir les détails des appareils spécifiés.

L'exemple suivant restreint l'accès à tous QPUs pour le Compte AWS 123456789012.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "braket:CreateQuantumTask",
        "braket:CreateJob",
        "braket:GetDevice"
      ],
      "Resource": [
        "arn:aws:braket:*:*:device/qpu/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "123456789012"
        }
      }
    }
  ]
}
```

Note

Excluez l'action `braket:GetDevice` de la politique pour permettre à l'utilisateur d'accéder en lecture aux propriétés de l'appareil, telles que la disponibilité de l'appareil, les données d'étalonnage et les prix via la console Braket.

Pour adapter ce code, remplacez le numéro de Amazon ressource (ARN) du périphérique restreint par la chaîne illustrée dans l'exemple précédent. Cette chaîne fournit la valeur de la ressource. Dans Braket, un appareil représente un QPU ou un simulateur que vous pouvez appeler pour exécuter des tâches quantiques. Les appareils disponibles sont répertoriés sur la [page Appareils](#). Deux schémas sont utilisés pour spécifier l'accès à ces appareils :

- `arn:aws:braket:<region>:*:device/qpu/<provider>/<device_id>`
- `arn:aws:braket:<region>:*:device/quantum-simulator/<provider>/<device_id>`

Voici des exemples de différents types d'accès aux appareils

- Pour tout sélectionner QPUs dans toutes les régions : `arn:aws:braket:*:*:device/qpu/*`
- Pour tout sélectionner uniquement QPUs dans la région us-west-2 : `arn:aws:braket:us-west-2:*:device/qpu/*`
- De manière équivalente, pour tout sélectionner UNIQUEMENT QPUs dans la région us-west-2 (puisque les appareils sont une ressource de service et non une ressource client) : `arn:aws:braket:us-west-2:*:device/qpu/*`
- Pour restreindre l'accès à tous les appareils de simulation à la demande, procédez comme suit : `arn:aws:braket:*:*:device/quantum-simulator/*`
- Pour restreindre l'accès aux appareils d'un certain fournisseur (par exemple, aux Rigetti QPU appareils) : `arn:aws:braket:*:*:device/qpu/rigetti/*`
- Pour restreindre l'accès à l'Amazon TN1 appareil : `arn:aws:braket:*:*:device/quantum-simulator/amazon/tn1`
- Pour restreindre l'accès à toutes les Create actions : `braket:Create*`

Restreindre l'accès des utilisateurs à certaines instances de blocs-notes

Pour restreindre l'accès de certains utilisateurs à des instances spécifiques du bloc-notes Braket, vous pouvez ajouter une politique de refus d'autorisations pour un rôle, un utilisateur ou un groupe spécifique.

L'exemple suivant utilise des [variables de politique](#) pour restreindre efficacement les autorisations de démarrage, d'arrêt et d'accès à des instances de bloc-notes spécifiques dans le Compte AWS 123456789012, qui sont nommées en fonction de l'utilisateur qui doit y avoir accès (par exemple, l'utilisateur Alice aurait accès à une instance de bloc-notes nommée `amazon-braket-Alice`).

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCreateDeleteUpdateNotebookInstances",
      "Effect": "Deny",
      "Action": [
```

```

        "sagemaker:CreateNotebookInstance",
        "sagemaker>DeleteNotebookInstance",
        "sagemaker:UpdateNotebookInstance",
        "sagemaker:CreateNotebookInstanceLifecycleConfig",
        "sagemaker>DeleteNotebookInstanceLifecycleConfig",
        "sagemaker:UpdateNotebookInstanceLifecycleConfig"
    ],
    "Resource": "*"
},
{
    "Sid": "DenyDescribeStartStopNotebookInstances",
    "Effect": "Deny",
    "Action": [
        "sagemaker:DescribeNotebookInstance",
        "sagemaker:StartNotebookInstance",
        "sagemaker:StopNotebookInstance"
    ],
    "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
        ${aws:username}"
    ]
},
{
    "Sid": "DenyNotebookInstanceUrl",
    "Effect": "Deny",
    "Action": [
        "sagemaker:CreatePresignedNotebookInstanceUrl"
    ],
    "NotResource": [
        "arn:aws:sagemaker:*:123456789012:notebook-instance/amazon-braket-
        ${aws:username}*"
    ]
}
]
}

```

Restreindre l'accès des utilisateurs à certains compartiments S3

Pour restreindre l'accès de certains utilisateurs à des compartiments Amazon S3 spécifiques, vous pouvez ajouter une politique de refus pour un rôle, un utilisateur ou un groupe spécifique.

L'exemple suivant restreint les autorisations de récupération et de placement d'objets dans un S3 compartiment spécifique (`arn:aws:s3:::amazon-braket-us-east-1-123456789012-Alice`) et restreint également la liste de ces objets.

JSON

```
{
  "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/*"
      ]
    }
  ]
}
```

Pour restreindre l'accès au compartiment pour une instance de bloc-notes donnée, vous pouvez ajouter la politique précédente au rôle d'exécution du bloc-notes.

Rôle lié au service Amazon Braket

Lorsque vous activez Amazon Braket, un rôle lié à un service est créé dans votre compte.

Un rôle lié à un service est un type unique de rôle IAM qui, dans ce cas, est directement lié à Amazon Braket. Le rôle lié au service Amazon Braket est prédéfini pour inclure toutes les autorisations dont Braket a besoin pour appeler une autre personne en votre nom. Services AWS

Un rôle lié à un service facilite la configuration d'Amazon Braket, car vous n'avez pas à ajouter les autorisations nécessaires manuellement. Amazon Braket définit les autorisations associées à ses rôles liés aux services. À moins que vous ne changiez ces définitions, seul Amazon Braket peut assumer ses rôles. Les autorisations définies incluent la politique de confiance et la politique d'autorisations. La stratégie d'autorisations ne peut pas être attachée à une autre entité IAM.

[Le rôle lié au service configuré par Amazon Braket fait partie de la fonctionnalité des rôles liés Gestion des identités et des accès AWS au service \(IAM\)](#). Pour plus d'informations sur les autres services Services AWS qui prennent en charge les rôles liés à un service, consultez la section [AWS Services qui fonctionnent avec IAM](#) et recherchez les services dont la valeur est Oui dans la colonne Rôle lié au service. Cliquez sur Oui avec un lien pour consulter la documentation relative aux rôles liés à un service pour ce service.

Pour plus d'informations sur la politique AWS gérée pour les rôles liés à un service, consultez [AmazonBraketServiceRolePolicy](#).

Validation de conformité pour Amazon Braket

Note

AWS les rapports de conformité ne couvrent pas les fournisseurs QPUs de matériel tiers qui peuvent choisir de se soumettre à leurs propres audits indépendants.

Pour savoir si un [programme Services AWS de conformité Service AWS s'inscrit dans le champ d'application de programmes de conformité](#) spécifiques, consultez Services AWS la section de conformité et sélectionnez le programme de conformité qui vous intéresse. Pour des informations générales, voir Programmes de [AWS conformité Programmes AWS](#) de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, voir [Téléchargement de rapports dans AWS Artifact](#) .

Votre responsabilité en matière de conformité lors de l'utilisation Services AWS est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise et les lois et réglementations applicables. Pour plus d'informations sur votre responsabilité en matière de conformité lors de l'utilisation Services AWS, consultez [AWS la documentation de sécurité](#).

Sécurité de l'infrastructure dans Amazon Braket

En tant que service géré, Amazon Braket est protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section [Sécurité du AWS cloud](#). Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section [Protection de l'infrastructure](#) dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à Amazon Braket via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

Vous pouvez appeler ces opérations d'API depuis n'importe quel emplacement réseau, mais Braket prend en charge les politiques d'accès basées sur les ressources, qui peuvent inclure des restrictions basées sur l'adresse IP source. Vous pouvez également utiliser les politiques de Braket pour contrôler l'accès à partir de points de terminaison Amazon Virtual Private Cloud (Amazon VPC) spécifiques ou spécifiques. VPCs En fait, cela isole l'accès réseau à une ressource Braket donnée uniquement du VPC spécifique au sein du réseau. AWS

Sécurité des fournisseurs de matériel Amazon Braket

QPU sur Amazon Braket sont hébergés par des fournisseurs de matériel tiers. Lorsque vous exécutez votre tâche quantique sur un QPU, Amazon Braket utilise le DeviceARN comme identifiant lors de l'envoi du circuit au QPU spécifié pour traitement.

Si vous utilisez Amazon Braket pour accéder à du matériel informatique quantique géré par l'un des fournisseurs de matériel tiers, votre circuit et les données associées sont traités par des fournisseurs de matériel extérieurs aux installations exploitées par celui-ci. AWS Vous trouverez des informations sur l'emplacement physique et AWS la région où chaque QPU est disponible dans la section Détails de l'appareil de la console Amazon Braket.

Votre contenu est anonymisé. Seul le contenu nécessaire au traitement du circuit est envoyé à des tiers. Compte AWS les informations ne sont pas transmises à des tiers.

Toutes les données sont cryptées au repos et en transit. Les données sont déchiffrées à des fins de traitement uniquement. Les fournisseurs tiers d'Amazon Braket ne sont pas autorisés à stocker ou à utiliser votre contenu à des fins autres que le traitement de votre circuit. Une fois le circuit terminé, les résultats sont renvoyés à Amazon Braket et stockés dans votre compartiment S3.

La sécurité des fournisseurs tiers de matériel quantique Amazon Braket fait l'objet d'audits périodiques afin de garantir le respect des normes de sécurité du réseau, de contrôle d'accès, de protection des données et de sécurité physique.

Points de terminaison Amazon VPC pour Amazon Braket

Vous pouvez établir une connexion privée entre votre VPC et Amazon Braket en créant un point de terminaison VPC d'interface. Les points de terminaison de l'interface sont alimentés par [AWS PrivateLink](#) une technologie qui permet d'accéder à Braket APIs sans passerelle Internet, appareil NAT, connexion VPN ou Direct Connect connexion. Les instances de votre VPC n'ont pas besoin d'adresses IP publiques pour communiquer avec Braket. APIs

Chaque point de terminaison d'interface est représenté par une ou plusieurs [interfaces réseau Elastic](#) dans vos sous-réseaux.

Ainsi AWS PrivateLink, le trafic entre votre VPC et Braket ne quitte pas le Amazon réseau, ce qui renforce la sécurité des données que vous partagez avec des applications basées sur le cloud, car cela réduit l'exposition de vos données à l'Internet public. Pour plus d'informations, consultez la section [Accès à un AWS service à l'aide d'un point de terminaison VPC d'interface](#) dans le guide de l'utilisateur Amazon VPC.

Dans cette section :

- [Considérations relatives aux points de terminaison Amazon Braket VPC](#)
- [Configurez Braket et PrivateLink](#)
- [Informations supplémentaires sur la création d'un point de terminaison](#)
- [Contrôlez l'accès avec les politiques relatives aux points de terminaison Amazon VPC](#)

Considérations relatives aux points de terminaison Amazon Braket VPC

Avant de configurer un point de terminaison VPC d'interface pour Braket, assurez-vous de consulter les [conditions requises pour le point de terminaison d'interface dans](#) le guide de l'utilisateur Amazon VPC.

Braket permet d'appeler toutes ses [actions d'API](#) depuis votre VPC.

Par défaut, l'accès complet à Braket est autorisé via le point de terminaison VPC. Vous pouvez contrôler l'accès si vous spécifiez des politiques de point de terminaison VPC. Pour plus d'informations, consultez [Contrôle de l'accès aux points de terminaison d'un VPC à l'aide de politiques de point de terminaison](#) dans le Guide de l'utilisateur Amazon VPC.

Configurez Braket et PrivateLink

Pour l'utiliser AWS PrivateLink avec Amazon Braket, vous devez créer un point de terminaison Amazon Virtual Private Cloud (Amazon VPC) en tant qu'interface, puis vous connecter au point de terminaison via le service Amazon Braket. API

Voici les étapes générales de ce processus, qui sont expliquées en détail dans les sections suivantes.

- Configurez et lancez un Amazon VPC pour héberger vos AWS ressources. Si vous avez déjà un VPC, vous pouvez ignorer cette étape.
- Création d'un point de terminaison Amazon VPC pour Braket
- Connectez et exécutez les tâches quantiques de Braket via votre terminal

Étape 1 : Lancez un Amazon VPC si nécessaire

N'oubliez pas que vous pouvez ignorer cette étape si un VPC est déjà actif sur votre compte.

Un VPC contrôle vos paramètres réseau, tels que la plage d'adresses IP, les sous-réseaux, les tables de routage et les passerelles réseau. Essentiellement, vous lancez vos AWS ressources dans un réseau virtuel personnalisé. Pour plus d'informations à ce sujet VPCs, consultez le guide de [l'utilisateur Amazon VPC](#).

Ouvrez la [console Amazon VPC](#) et créez un nouveau VPC avec des sous-réseaux, des groupes de sécurité et des passerelles réseau.

Étape 2 : créer un point de terminaison VPC d'interface pour Braket

Vous pouvez créer un point de terminaison VPC pour le service Braket à l'aide de la console Amazon VPC ou du (). AWS Command Line Interface AWS CLI Pour plus d'informations, consultez la section [Créer un point de terminaison VPC](#) dans le guide de l'utilisateur Amazon VPC.

Pour créer un point de terminaison VPC dans la console, ouvrez la [console Amazon VPC](#), ouvrez la page Endpoints, puis créez le nouveau point de terminaison. Notez l'ID du point de terminaison pour référence ultérieure. Il est obligatoire sur le `--endpoint-url` drapeau lorsque vous passez certains appels au BraketAPI.

Créez le point de terminaison VPC pour Braket en utilisant le nom de service suivant :

- `com.amazonaws.substitute_your_region.braket`

Pour plus d'informations, consultez la section [Accès à un AWS service à l'aide d'un point de terminaison VPC d'interface](#) dans le guide de l'utilisateur Amazon VPC.

Étape 3 : Connectez et exécutez les tâches quantiques de Braket via votre terminal

Après avoir créé un point de terminaison VPC, vous pouvez exécuter des commandes CLI qui incluent le `endpoint-url` paramètre permettant de spécifier les points de terminaison de l'interface pour le moteur d'exécution du module API d'exécution, comme dans l'exemple suivant :

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

Si vous activez les noms d'hôte DNS privés pour votre point de terminaison VPC, vous n'avez pas besoin de spécifier le point de terminaison sous forme d'URL dans vos commandes CLI. Au lieu de cela, le nom d'hôte API DNS Amazon Braket, que la CLI et le SDK Braket utilisent par défaut, correspond à votre point de terminaison VPC. Il se présente sous la forme illustrée dans l'exemple suivant :

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

Le billet de blog intitulé [Accès direct aux blocs-notes Amazon SageMaker AI depuis Amazon VPC à l'aide AWS PrivateLink d'un](#) point de terminaison fournit un exemple de configuration d'un point de terminaison pour établir des connexions sécurisées avec des ordinateurs portables, similaires SageMaker Amazon aux blocs-notes Braket.

Si vous suivez les étapes décrites dans le billet de blog, n'oubliez pas de remplacer le nom AmazonBraket par Amazon SageMaker AI. Pour Nom du service, entrez `com.amazonaws.us-east-1.braket` ou remplacez le Région AWS nom correct dans cette chaîne, si votre région n'est pas us-east-1.

Informations supplémentaires sur la création d'un point de terminaison

- Pour plus d'informations sur la création d'un VPC avec des sous-réseaux privés, consultez la section Création [d'un VPC](#) avec des sous-réseaux privés.
- Pour plus d'informations sur la création et la configuration d'un point de terminaison à l'aide de la console Amazon VPC ou du AWS CLI, consultez la section [Créer un point de terminaison VPC dans](#) le guide de l'utilisateur Amazon VPC.
- Pour plus d'informations sur la création et la configuration d'un point de terminaison à l'aide de CloudFormation, consultez la AWS VPC Endpoint ressource [::EC2::](#) dans le guide de l'CloudFormation utilisateur.

Contrôlez l'accès avec les politiques relatives aux points de terminaison Amazon VPC

Pour contrôler l'accès à la connectivité à Amazon Braket, vous pouvez associer une politique de point de terminaison Gestion des identités et des accès AWS (IAM) à votre point de terminaison Amazon VPC. La politique spécifie les informations suivantes :

- Le principal (utilisateur ou rôle) qui peut effectuer des actions.
- Les actions qui peuvent être effectuées.
- Les ressources sur lesquelles les actions peuvent être exécutées.

Pour plus d'informations, consultez [Contrôle de l'accès aux points de terminaison d'un VPC à l'aide de politiques de point de terminaison](#) dans le Guide de l'utilisateur Amazon VPC.

Exemple : politique de point de terminaison VPC pour les actions Braket

L'exemple suivant montre une politique de point de terminaison pour Braket. Lorsqu'elle est attachée à un point de terminaison, cette politique accorde l'accès aux actions Braket répertoriées à tous les principaux sur toutes les ressources.

```
{
  "Statement": [
    {
      "Principal": "*",
      "Effect": "Allow",
      "Action": [
```

```
    "braket:action-1",
    "braket:action-2",
    "braket:action-3"
  ],
  "Resource": "*"
}
```

Vous pouvez créer des règles IAM complexes en associant plusieurs politiques de point de terminaison. Pour plus d'informations et des exemples, voir :

- [Politiques relatives aux terminaux Amazon Virtual Private Cloud pour Step Functions](#)
- [Création d'autorisations IAM granulaires pour les utilisateurs non administrateurs](#)
- [Contrôlez l'accès aux points de terminaison VPC à l'aide de politiques relatives aux points de terminaison](#)

Journalisation et surveillance

Après avoir soumis une tâche quantique via le service Amazon Braket, vous pouvez suivre de près le statut et la progression de cette tâche via le SDK et la console Amazon Braket. Vous disposez ainsi d'une interface centralisée pour suivre la mise en œuvre de vos charges de travail, identifier les éventuels goulots d'étranglement ou problèmes et prendre les mesures appropriées pour optimiser les performances et la fiabilité de vos applications quantiques. Lorsque la tâche quantique est terminée, Braket enregistre les résultats dans l'emplacement Amazon S3 que vous avez spécifié. Le temps d'exécution des tâches quantiques peut varier, en particulier pour celles exécutées sur des unités de traitement quantique (QPU). Cela est dû en grande partie à la longueur de la file d'exécution, car les ressources matérielles quantiques sont partagées entre plusieurs utilisateurs.

Liste des types de statut :

- **CREATED**— Amazon Braket a reçu votre tâche quantique.
- **QUEUED**— Amazon Braket a traité votre tâche quantique et elle attend maintenant de s'exécuter sur l'appareil.
- **RUNNING**— Votre tâche quantique s'exécute sur un QPU ou un simulateur à la demande.
- **COMPLETED**— Votre tâche quantique s'est terminée sur le QPU ou sur le simulateur à la demande.
- **FAILED**— Votre tâche quantique a tenté de s'exécuter et a échoué. En fonction de la raison pour laquelle votre tâche quantique a échoué, essayez de la soumettre à nouveau.
- **CANCELLED**— Vous avez annulé la tâche quantique. La tâche quantique n'a pas été exécutée.

Dans cette section :

- [Suivi des tâches quantiques à partir du SDK Amazon Braket](#)
- [Surveillance des tâches quantiques via la console Amazon Braket](#)
- [Marquage des ressources Amazon Braket](#)
- [Surveillez vos tâches quantiques avec EventBridge](#)
- [Surveillez vos indicateurs avec CloudWatch](#)
- [Enregistrez vos tâches quantiques avec CloudTrail](#)
- [Journalisation avancée avec Amazon Braket](#)

Suivi des tâches quantiques à partir du SDK Amazon Braket

La commande `device.run(...)` définit une tâche quantique avec un identifiant de tâche quantique unique. Vous pouvez interroger et suivre le statut avec `task.state()`, comme indiqué dans l'exemple suivant.

Remarque : `task = device.run()` il s'agit d'une opération asynchrone, ce qui signifie que vous pouvez continuer à travailler pendant que le système traite votre tâche quantique en arrière-plan.

Récupérez un résultat

Lorsque vous appelez `task.result()`, le SDK commence à interroger Amazon Braket pour voir si la tâche quantique est terminée. Le SDK utilise les paramètres de sondage que vous avez définis dans `run()`. Une fois la tâche quantique terminée, le SDK extrait le résultat du compartiment S3 et le renvoie sous forme d'`QuantumTaskResult` objet.

```
# 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
```

Annuler une tâche quantique

Pour annuler une tâche quantique, appelez la `cancel()` méthode, comme indiqué dans l'exemple suivant.

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

```
Status of task: CANCELLING
```

Vérifiez les métadonnées

Vous pouvez vérifier les métadonnées de la tâche quantique terminée, comme indiqué dans l'exemple suivant.

```
# 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
```

Récupérer une tâche ou un résultat quantique

Si votre noyau meurt après avoir soumis la tâche quantique ou si vous fermez votre bloc-notes ou votre ordinateur, vous pouvez reconstruire l'objet `task` avec son ARN (identifiant de tâche quantique) unique. Vous pouvez ensuite appeler `task.result()` pour obtenir le résultat depuis le compartiment S3 dans lequel il est stocké.

```
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()
```

Surveillance des tâches quantiques via la console Amazon Braket

Amazon Braket offre un moyen pratique de surveiller la tâche quantique via la console [Amazon Braket](#). Toutes les tâches quantiques soumises sont répertoriées dans le champ Tâches quantiques, comme indiqué dans la figure suivante. Ce service est spécifique à une région, ce qui signifie que vous ne pouvez visualiser que les tâches quantiques créées dans cette région. Région AWS

Amazon Braket > Quantum Tasks

ⓘ QPUs are region specific. Select the correct device region for corresponding quantum tasks. [Learn more](#)

Quantum Tasks (10+) Actions ▼ Show quantum task details

🔍 Search

	Quantum Task ID	Status	Device ARN	Created at
○	d87730f0-414f-4a60-9de2-7fd18c20f7f2	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Sep 05, 2023 19:13 (UTC)
○	62a5b6f9-2334-4bad-af4f-a5aeebbe6032	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
○	85f05c12-c4d0-42bf-8782-b825775f057a	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Aug 31, 2023 19:11 (UTC)
○	1fa148a2-aaaa-4948-b7df-808513145a20	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
○	aee8d2ad-a396-4c11-9f13-9aa62db680b9	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
○	dfef97af-3aae-4e57-bd64-29d6f9521937	✔ COMPLETED	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Aug 31, 2023 19:11 (UTC)

Vous pouvez rechercher des tâches quantiques spécifiques dans la barre de navigation. La recherche peut être basée sur l'ARN (ID) de la tâche quantique, le statut, l'appareil et l'heure de création. Les options apparaissent automatiquement lorsque vous sélectionnez la barre de navigation, comme illustré dans l'exemple suivant.

The screenshot shows the Amazon Braket Quantum Tasks console. At the top, there is a navigation bar with "Amazon Braket" and "Quantum Tasks". Below this is a light blue banner with an information icon and the text: "QPU's are region specific. Select the correct device region for corresponding quantum tasks. [Learn more](#)".

The main section is titled "Quantum Tasks (10+)". It features a search bar with the placeholder "Search". A dropdown menu is open, showing a list of properties: "Status", "Device ARN", "Quantum task ARN", and "Created at". Below the search bar, there is a table with the following columns: "Status", "Device ARN", and "Created at". The table contains three rows of data:

Status	Device ARN	Created at
7f2	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Sep 05, 2023 19:13 (UTC)
032	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:11 (UTC)
85f05c12-c4d0-42bf-8782-b825775f057a	arn:aws:braket:::device/quantum-simulator/amazon/dm1	Aug 31, 2023 19:11 (UTC)

L'image suivante montre un exemple de recherche d'une tâche quantique basée sur son identifiant unique de tâche quantique, qui peut être obtenu en appelant `task.id`.

The screenshot shows the Amazon Braket Quantum Tasks console with a search filter applied. The search bar contains the text "Search" and "(1) matches". Below the search bar, there is a filter box with the text: "Quantum task ARN = arn:aws:braket:us-west-2:260818742045:quantum-task/4cd1a31e-61c0-469c-a9cf-a2f7b4e358". To the right of the filter box is a "Clear filters" button.

The table below shows the search results:

Quantum Task ID	Status	Device ARN	Created at
4cd1a31e-61c0-469c-a9cf-a2f7b4e358	COMPLETE	arn:aws:braket:::device/quantum-simulator/amazon/sv1	Aug 31, 2023 19:10 (UTC)

De plus, comme le montre la figure ci-dessous, l'état d'une tâche quantique peut être surveillé lorsqu'elle est dans un QUEUED état. Cliquez sur l'ID de la tâche quantique pour afficher la page de détails. Cette page affiche la position de la file d'attente dynamique pour votre tâche quantique par rapport à l'appareil sur lequel elle sera traitée.

Amazon Braket > Quantum Tasks > 3d11c509-454d-4fe2-b3b9-fad6d8eab83b

3d11c509-454d-4fe2-b3b9-fad6d8eab83b

Quantum task details			Actions
Quantum task ARN	Status	Queue position	
arn:aws:braket:us-east-1:984631112496:quantum-task/3d11c509-454d-4fe2-b3b9-fad6d8eab83b	QUEUED	3 (Normal)	
Device ARN	Created	Ended	
arn:aws:braket:us-east-1::device/gpu/long/Aria-2	Sep 08, 2023 19:22 (UTC)	—	
Shots	Results	Status reason	
100	—	—	

Les tâches quantiques soumises dans le cadre d'une tâche hybride seront prioritaires dans la file d'attente. Les tâches quantiques soumises en dehors d'une tâche hybride auront une priorité de mise en file d'attente normale.

Les clients qui souhaitent interroger le SDK Braket peuvent obtenir leurs positions de tâches quantiques et de files d'attente de tâches hybrides par le biais d'un programme. Pour plus d'informations, consultez la page [Quand ma tâche sera-t-elle exécutée ?](#)

Marquage des ressources Amazon Braket

Une balise est une étiquette d'attribut personnalisée que vous attribuez ou attribuez à une AWS ressource. AWS Une balise est une métadonnée qui en dit plus sur votre ressource. Chaque balise se compose d'une clé et d'une valeur. Ensemble, ces informations sont qualifiées de paires clé-valeur. Pour les balises que vous attribuez, vous définissez la clé et la valeur.

Dans la console Amazon Braket, vous pouvez accéder à une tâche quantique ou à un bloc-notes et consulter la liste des balises qui y sont associées. Vous pouvez ajouter un tag, supprimer un tag ou modifier un tag. Vous pouvez étiqueter une tâche quantique ou un bloc-notes lors de sa création, puis gérer les balises associées via la console AWS CLI, ou API.

En savoir plus sur AWS les tags

- Pour des informations générales sur le balisage, y compris les conventions de dénomination et d'utilisation, voir [Qu'est-ce que l'éditeur de balises ?](#) dans le guide de l'utilisateur des AWS ressources de balisage et de l'éditeur de balises.
- Pour plus d'informations sur les restrictions relatives au balisage, voir [Limites et exigences en matière de dénomination des balises](#) dans le guide de l'utilisateur AWS des ressources de balisage et de l'éditeur de balises.
- Pour connaître les meilleures pratiques et les stratégies de balisage, consultez la section [Meilleures pratiques pour le balisage des ressources AWS](#).

- Pour obtenir la liste des services qui prennent en charge l'utilisation de balises, consultez [Resource Groups Tagging API Reference](#).

Les sections suivantes fournissent des informations plus spécifiques sur les tags pour Amazon Braket.

Dans cette section :

- [Utilisation d'étiquettes](#)
- [Ressources prises en charge pour le balisage dans Amazon Braket](#)
- [Balisage avec l'Amazon Braket API](#)
- [Restrictions de balisage](#)
- [Gestion des tags dans Amazon Braket](#)
- [Exemple de AWS CLI balisage dans Amazon Braket](#)

Utilisation d'étiquettes

Les balises peuvent organiser vos ressources en catégories qui vous sont utiles. Par exemple, vous pouvez attribuer une balise « Department » pour indiquer le département propriétaire de cette ressource.

Chaque balise se compose de deux parties :

- Une clé de balise (par exemple CostCenter, Environnement ou Projet). Les touches de tag distinguent les majuscules et minuscules.
- Champ facultatif appelé valeur de balise (par exemple, 111122223333 ou Production). Omettre la valeur de balise équivaut à l'utilisation d'une chaîne vide. Les valeurs de balise sont sensibles à la casse, tout comme les clés de balise.

Les balises vous aident à effectuer les tâches suivantes :

- Identifiez et organisez vos AWS ressources. Beaucoup Services AWS prennent en charge le balisage. Vous pouvez donc attribuer le même tag aux ressources provenant de différents services pour indiquer que les ressources sont liées.
- Suivez vos AWS coûts. Vous activez ces balises sur le AWS Billing and Cost Management tableau de bord. AWS utilise les balises pour classer vos coûts et vous fournir un rapport mensuel de

répartition des coûts. Pour plus d'informations, consultez [Utilisation des balises de répartition des coûts](#) dans le [Guide de l'utilisateur AWS Billing and Cost Management](#).

- Contrôlez l'accès à vos AWS ressources. Pour plus d'informations, consultez la section [Contrôle de l'accès à l'aide de balises](#).

Ressources prises en charge pour le balisage dans Amazon Braket

Le type de ressource suivant dans Amazon Braket prend en charge le balisage :

- [quantum-task](#) ressource
- Nom de la ressource : `AWS::Service::Braket`
- ARN Regex : `arn:${Partition}:braket:${Region}:${Account}:quantum-task/${RandomId}`

Remarque : vous pouvez appliquer et gérer des balises pour vos blocs-notes Amazon Braket dans la console Amazon Braket, en utilisant la console pour accéder à la ressource du bloc-notes, bien que les blocs-notes soient en fait des ressources Amazon AI. SageMaker Pour plus d'informations, consultez la section [Métadonnées des instances de bloc-notes](#) dans la SageMaker documentation.

Balisage avec l'Amazon Braket API

- Si vous utilisez Amazon Braket API pour configurer des balises sur une ressource, appelez le [TagResourceAPI](#)

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

- Pour supprimer des balises d'une ressource, appelez le [UntagResourceAPI](#).

```
aws braket list-tags-for-resource --resource-arn $YOUR_TASK_ARN
```

- Pour répertorier toutes les balises associées à une ressource donnée, appelez le [ListTagsForResourceAPI](#).

```
aws braket tag-resource --resource-arn $YOUR_TASK_ARN --tag-keys "[\"city\\\", \"state\"]"
```

Restrictions de balisage

Les restrictions de base suivantes s'appliquent aux tags figurant sur les ressources Amazon Braket :

- Nombre maximum de balises que vous pouvez attribuer à une ressource : 50
- Longueur de clé maximale : 128 caractères Unicode
- Longueur de valeur maximale : 256 caractères Unicode
- Caractères valides pour la clé et la valeur : a-z, A-Z, 0-9, space, et ces caractères :
_ . : / = + - et @
- Les clés et les valeurs sont sensibles à la casse.
- Ne l'utilisez pas aws comme préfixe pour les clés ; il est réservé à l' AWS usage.

Gestion des tags dans Amazon Braket

Vous définissez des balises en tant que propriétés d'une ressource. Vous pouvez consulter, ajouter, modifier, répertorier et supprimer des tags via la console Amazon Braket, Amazon API Braket ou le AWS CLI Pour plus d'informations, consultez la référence de l'[API Amazon Braket](#).

Dans cette section :

- [Ajout de balises](#)
- [Affichage des balises](#)
- [Modification des balises](#)
- [Suppression de balises](#)

Ajout de balises

Vous pouvez ajouter des balises aux ressources étiquetables aux moments suivants :

- Lorsque vous créez la ressource : utilisez la console ou incluez le Tags paramètre dans l'Createopération dans l'[AWS API](#).
- Après avoir créé la ressource : utilisez la console pour accéder à la tâche quantique ou à la ressource du bloc-notes, ou appelez l'TagResourceopération dans l'[AWS API](#).

Pour ajouter des balises à une ressource lorsque vous la créez, vous devez également être autorisé à créer une ressource du type spécifié.

Affichage des balises

Vous pouvez consulter les balises de n'importe quelle ressource étiquetable dans Amazon Braket en utilisant la console pour accéder à la tâche ou à la ressource du bloc-notes, ou en appelant l'opération. `AWS ListTagsForResource API`

Vous pouvez utiliser la AWS API commande suivante pour afficher les balises d'une ressource :

- AWS API: `ListTagsForResource`

Modification des balises

Vous pouvez modifier les balises en utilisant la console pour accéder à la tâche quantique ou à la ressource du bloc-notes ou vous pouvez utiliser la commande suivante pour modifier la valeur d'une balise attachée à une ressource étiquetable. Lorsque vous spécifiez une clé de balise qui existe déjà, la valeur de cette clé est remplacée :

- AWS API: `TagResource`

Suppression de balises

Vous pouvez supprimer des balises d'une ressource en spécifiant les clés à supprimer, en utilisant la console pour accéder à la tâche quantique ou à la ressource du bloc-notes, ou lorsque vous appelez l'opération. `UntagResource`

- AWS API: `UntagResource`

Exemple de AWS CLI balisage dans Amazon Braket

Lorsque vous utilisez le AWS Command Line Interface (AWS CLI) pour interagir avec Amazon Braket, le code suivant est un exemple de commande qui montre comment créer une balise qui s'applique à une tâche quantique que vous créez. Dans cet exemple, la tâche est exécutée sur le simulateur SV1 quantique avec des paramètres définis pour l'unité de traitement Rigetti quantique (QPU). Il est important que dans l'exemple de commande, la balise soit spécifiée à la toute fin, après tous les autres paramètres requis. Dans ce cas, la balise possède une clé `state` et une valeur `deWashington`. Ces balises pourraient être utilisées pour aider à catégoriser ou à identifier cette tâche quantique particulière.

```
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\"}
```

Cet exemple montre comment appliquer des balises à vos tâches quantiques lorsque vous les exécutez dans le AWS CLI, ce qui est utile pour organiser et suivre vos ressources Braket.

Surveillez vos tâches quantiques avec EventBridge

Amazon EventBridge surveille les événements de changement de statut dans les tâches quantiques Amazon Braket. Les événements d'Amazon Braket sont diffusés sur EventBridge, presque en temps réel. Vous pouvez rédiger des règles qui indiquent les événements qui vous intéressent, y compris les actions automatisées à effectuer lorsqu'un événement correspond à une règle. Les actions automatiques qui peuvent être déclenchées sont les suivantes :

- Invoquer une fonction AWS Lambda
- Activation d'une machine à AWS Step Functions états
- Notification d'une rubrique Amazon SNS

EventBridge surveille les événements de changement de statut d'Amazon Braket suivants :

- L'état de la tâche quantique change

Amazon Braket garantit la livraison des événements de changement de statut des tâches quantiques. Ces événements sont organisés au moins une fois, mais ils peuvent être hors service.

Pour plus d'informations, consultez les [événements sur Amazon EventBridge](#).

Dans cette section :

- [Surveillez l'état des tâches quantiques avec EventBridge](#)
- [Exemple d'événement Amazon Braket EventBridge](#)

Surveillez l'état des tâches quantiques avec EventBridge

Vous pouvez ainsi créer des règles qui définissent les actions à entreprendre lorsqu'Amazon Braket envoie une notification concernant un changement de statut concernant une tâche quantique Braket. EventBridge Par exemple, vous pouvez créer une règle qui vous envoie un message électronique chaque fois que le statut d'une tâche quantique change.

1. Connectez-vous à AWS l'aide d'un compte autorisé à utiliser EventBridge Amazon Braket.
2. Ouvrez la [EventBridge console Amazon](#).
3. À l'aide des valeurs suivantes, créez une EventBridge règle :
 - Pour Type de règle, choisissez Règle avec un modèle d'événement.
 - Pour Event source (Source de l'événement), choisissez Other (Autres).
 - Dans la section Modèle d'événement, choisissez Modèles personnalisés (éditeur JSON), puis collez le modèle d'événement suivant dans la zone de texte :

```
{
  "source": [
    "aws.braket"
  ],
  "detail-type": [
    "Braket Task State Change"
  ]
}
```

Pour capturer tous les événements de Amazon Braket, excluez la `detail-type` section comme indiqué dans le code suivant :

```
{
  "source": [
```

```
"aws.braket"
]
}
```

- Pour les types de cibles, choisissez `Service AWS`, et pour Sélectionner une cible, choisissez une cible telle qu'un sujet ou AWS Lambda une fonction Amazon SNS. La cible est déclenchée lorsqu'un événement de changement d'état de tâche quantique est reçu de Amazon Braket.

Par exemple, utilisez une rubrique Amazon Simple Notification Service (SNS) pour envoyer un e-mail ou un SMS lorsqu'un événement se produit. Pour ce faire, créez d'abord une rubrique Amazon SNS à l'aide de la console Amazon SNS. Pour en savoir plus, veuillez consulter [Utilisation d'Amazon SNS pour les notifications utilisateur](#).

Pour en savoir plus sur la création de règles, consultez [la section Création de EventBridge règles Amazon qui réagissent aux événements](#).

Exemple d'événement Amazon Braket EventBridge

Pour plus d'informations sur les champs relatifs à un événement de changement de statut d'une tâche Amazon Braket Quantum, consultez la section [Événements sur Amazon](#). EventBridge

Les attributs suivants apparaissent dans le champ « détail » du JSON.

- **quantumTaskArn**(str) : tâche quantique pour laquelle cet événement a été généré.
- **status**(Facultatif [str]) : état vers lequel la tâche quantique est passée.
- **deviceArn**(str) : appareil spécifié par l'utilisateur pour lequel cette tâche quantique a été créée.
- **shots**(int) : Le nombre de requêtes shots demandées par l'utilisateur.
- **outputS3Bucket**(str) : le compartiment de sortie spécifié par l'utilisateur.
- **outputS3Directory**(str) : le préfixe de clé de sortie spécifié par l'utilisateur.
- **createdAt**(str) : Le temps de création de la tâche quantique sous forme de chaîne ISO-8601.
- **endedAt**(Facultatif [str]) : heure à laquelle la tâche quantique a atteint un état terminal. Ce champ n'est présent que lorsque la tâche quantique est passée à un état terminal.

Le code JSON suivant montre un exemple d'événement Amazon Braket Quantum Task Status Change.

```
{
```



```
{
  "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"
  }
}
```

Surveillez vos indicateurs avec CloudWatch

Vous pouvez surveiller Amazon Braket à l'aide d'Amazon CloudWatch, qui collecte les données brutes et les transforme en indicateurs lisibles en temps quasi réel. Vous pouvez consulter les informations historiques générées il y a 15 mois ou les statistiques de recherche mises à jour au cours des deux dernières semaines dans la CloudWatch console Amazon afin de mieux comprendre les performances d'Amazon Braket. Pour en savoir plus, consultez la section [Utilisation CloudWatch des métriques](#).

Note

Vous pouvez consulter les flux de CloudWatch journal des blocs-notes Amazon Braket en accédant à la page détaillée du bloc-notes sur la console Amazon AI. [SageMaker Les paramètres supplémentaires du bloc-notes Amazon Braket sont disponibles via la SageMaker console.](#)

Dans cette section :

- [Statistiques et dimensions d'Amazon Braket](#)

Statistiques et dimensions d'Amazon Braket

Les métriques sont le concept fondamental de CloudWatch. Une métrique représente un ensemble chronologique de points de données publiés sur CloudWatch. Chaque métrique est caractérisée par un ensemble de dimensions. Pour en savoir plus sur les dimensions des métriques dans CloudWatch, consultez la section [CloudWatch Dimensions](#).

Amazon Braket envoie les données métriques suivantes, spécifiques à Amazon Braket, dans les métriques Amazon : CloudWatch

Métriques de tâches quantiques

Les métriques sont disponibles si des tâches quantiques existent. Ils sont affichés sous `AWS/Braket/By Device` dans la console. CloudWatch

Métrique	Description
Nombre	Nombre de tâches quantiques.
Latence	Cette métrique est émise lorsqu'une tâche quantique est terminée. Il représente le temps total entre l'initialisation de la tâche quantique et son achèvement.

Dimensions pour les métriques de tâches quantiques

Les métriques des tâches quantiques sont publiées avec une dimension basée sur le `deviceArn` paramètre, qui a la forme `arn:aws:braket : ::device/xxx`.

Enregistrez vos tâches quantiques avec CloudTrail

Amazon Braket est intégré à AWS CloudTrail un service qui fournit un enregistrement des actions entreprises par un utilisateur, un rôle ou un Service AWS dans Amazon Braket. CloudTrail capture tous les API appels pour Amazon Braket sous forme d'événements. Les appels capturés incluent

des appels provenant de la console Amazon Braket et des appels de code relatifs aux opérations d'Amazon Braket. Si vous créez un suivi, vous pouvez activer la diffusion continue d' CloudTrail événements vers un compartiment Amazon S3, y compris des événements pour Amazon Braket. Si vous ne configurez pas de suivi, vous pouvez toujours consulter les événements les plus récents dans la CloudTrail console dans Historique des événements. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à Amazon Braket, l'adresse IP à partir de laquelle la demande a été faite, l'auteur de la demande, la date à laquelle elle a été faite, ainsi que des informations supplémentaires.

Pour en savoir plus CloudTrail, consultez le [guide de AWS CloudTrail l'utilisateur](#).

Dans cette section :

- [Informations sur Amazon Braket dans CloudTrail](#)
- [Comprendre les entrées du fichier journal Amazon Braket](#)

Informations sur Amazon Braket dans CloudTrail

CloudTrail est activé sur votre compte Compte AWS lorsque vous créez le compte. Lorsqu'une activité a lieu dans Amazon Braket, cette activité est enregistrée dans un CloudTrail événement avec d'autres Service AWS événements dans l'historique des événements. Vous pouvez consulter, rechercher et télécharger les événements récents dans votre Compte AWS. Pour plus d'informations, consultez la section [Affichage des événements à l'aide de l'historique des CloudTrail événements](#).

Pour obtenir un historique permanent des événements de votre site Compte AWS, y compris des événements pour Amazon Braket, créez un parcours. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un journal d'activité dans la console, il s'applique à toutes les régions Régions AWS. Le journal enregistre les événements de toutes les régions de la AWS partition et transmet les fichiers journaux au compartiment Amazon S3 que vous spécifiez. En outre, vous pouvez en configurer d'autres Services AWS pour analyser plus en détail les données d'événements collectées dans les CloudTrail journaux et agir en conséquence. Pour plus d'informations, consultez les ressources suivantes :

- [Vue d'ensemble de la création d'un journal d'activité](#)
- [CloudTrail Services et intégrations pris en charge](#)
- [Configuration des notifications Amazon SNS pour CloudTrail](#)
- [Réception de fichiers CloudTrail journaux de plusieurs régions](#) et [réception de fichiers CloudTrail journaux de plusieurs comptes](#)

Toutes les actions de Amazon Braket sont enregistrées par CloudTrail. Par exemple, les appels aux `GetDevice` actions `GetQuantumTask` ou génèrent des entrées dans les fichiers CloudTrail journaux.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la requête a été effectuée par un autre Service AWS.

Pour plus d'informations, consultez la section [Élément `userIdentity` CloudTrail](#) .

Comprendre les entrées du fichier journal Amazon Braket

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment Amazon S3 que vous spécifiez. CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des API appels publics, ils n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant est une entrée de journal pour l'`GetQuantumTask` action, qui permet d'obtenir les détails d'une tâche quantique.

```
{
  "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"
}

```

Ce qui suit montre une entrée du journal de l'GetDeviceaction, qui renvoie les détails d'un événement lié à un appareil.

```

{
  "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"
}
```

Journalisation avancée avec Amazon Braket

Vous pouvez enregistrer l'ensemble du processus de traitement des tâches à l'aide d'un enregistreur. Ces techniques de journalisation avancées vous permettent de voir le sondage en arrière-plan et de créer un enregistrement pour un débogage ultérieur.

Pour utiliser l'enregistreur, nous vous recommandons de modifier les `poll_interval_seconds` paramètres `poll_timeout_seconds` et afin qu'une tâche quantique puisse être longue et que l'état de la tâche quantique soit enregistré en permanence, les résultats étant enregistrés dans un fichier. Vous pouvez transférer ce code vers un script Python plutôt que vers un bloc-notes Jupyter, afin que le script puisse s'exécuter en tant que processus en arrière-plan.

Configuration de l'enregistreur

Tout d'abord, configurez l'enregistreur de manière à ce que tous les journaux soient automatiquement écrits dans un fichier texte, comme indiqué dans les exemples de lignes suivants.

```
# 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

Créez et exécutez le circuit

Vous pouvez maintenant créer un circuit, le soumettre à un appareil pour qu'il fonctionne et voir ce qui se passe, comme indiqué dans cet exemple.

```
# 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
)
```

Vérifiez le fichier journal

Vous pouvez vérifier ce qui est écrit dans le fichier en saisissant la commande suivante.

```
# 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})
```

Obtenir l'ARN à partir du fichier journal

À partir de la sortie du fichier journal renvoyée, comme indiqué dans l'exemple précédent, vous pouvez obtenir les informations ARN. Avec l'identifiant ARN, vous pouvez récupérer le résultat de la tâche quantique terminée.

```
# 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()
```


Quotas Amazon Braket

Le tableau suivant répertorie les quotas de service pour Amazon Braket. Les quotas de service, également appelés limites, représentent le nombre maximal de ressources ou d'opérations de service pour votre Compte AWS.

Certains quotas peuvent être augmentés. Pour plus d'informations, consultez [Service AWS Quotas](#).

- Les quotas de rafale ne peuvent pas être augmentés.
- L'augmentation maximale du taux pour les quotas ajustables (à l'exception de la fréquence de rafale, qui ne peut pas être ajustée) est deux fois supérieure à la limite de débit par défaut spécifiée. Par exemple, un quota par défaut de 60 peut être ajusté à un maximum de 120.
- Le quota ajustable pour les tâches quantiques simultanées SV1 (DM1) permet un maximum de 60 par Région AWS.
- Le nombre maximum autorisé d'instances de calcul pour une tâche hybride est de 1, et les quotas sont ajustables.

Ressource	Description	Restrictions	Ajustable
Taux de API demandes	Le nombre maximum de demandes par seconde que vous pouvez envoyer sur ce compte dans la région actuelle.	140	Oui
Taux de API demandes en rafale	Le nombre maximum de demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	600	Non

Ressource	Description	Restrictions	Ajustable
Taux de CreateQuantumTask demandes	Le nombre maximum de CreateQuantumTask demandes que vous pouvez envoyer par seconde sur ce compte par région.	20 par seconde	Oui
Taux de CreateQuantumTask demandes en rafale	Le nombre maximum de CreateQuantumTask demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	40	Non
Taux de SearchQuantumTasks demandes	Le nombre maximum de SearchQuantumTasks demandes que vous pouvez envoyer par seconde sur ce compte par région.	5 par seconde	Oui

Ressource	Description	Restrictions	Ajustable
Taux de SearchQuantumTasks demandes en rafale	Le nombre maximum de SearchQuantumTasks demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	50	Non
Taux de GetQuantumTask demandes	Le nombre maximum de GetQuantumTask demandes que vous pouvez envoyer par seconde sur ce compte par région.	100 par seconde	Oui
Taux de GetQuantumTask demandes en rafale	Le nombre maximum de GetQuantumTask demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	500	Non

Ressource	Description	Restrictions	Ajustable
Taux de <code>CancelQuantumTask</code> demandes	Le nombre maximum de <code>CancelQuantumTask</code> demandes que vous pouvez envoyer par seconde sur ce compte par région.	2 par seconde	Oui
Taux de <code>CancelQuantumTask</code> demandes en rafale	Le nombre maximum de <code>CancelQuantumTask</code> demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	20	Non
Taux de <code>GetDevice</code> demandes	Le nombre maximum de <code>GetDevice</code> demandes que vous pouvez envoyer par seconde sur ce compte par région.	5 par seconde	Oui
Taux de <code>GetDevice</code> demandes en rafale	Le nombre maximum de <code>GetDevice</code> demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	50	Non

Ressource	Description	Restrictions	Ajustable
Taux de SearchDevices demandes	Le nombre maximum de SearchDevices demandes que vous pouvez envoyer par seconde sur ce compte par région.	5 par seconde	Oui
Taux de SearchDevices demandes en rafale	Le nombre maximum de SearchDevices demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	50	Non
Taux de CreateJob demandes	Le nombre maximum de CreateJob demandes que vous pouvez envoyer par seconde sur ce compte par région.	1 par seconde	Oui
Taux de CreateJob demandes en rafale	Le nombre maximum de CreateJob demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	5	Non

Ressource	Description	Restrictions	Ajustable
Taux de SearchJobs demandes	Le nombre maximum de SearchJobs demandes que vous pouvez envoyer par seconde sur ce compte par région.	5 par seconde	Oui
Taux de SearchJobs demandes en rafale	Le nombre maximum de SearchJobs demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	50	Non
Taux de GetJob demandes	Le nombre maximum de GetJob demandes que vous pouvez envoyer par seconde sur ce compte par région.	5 par seconde	Oui
Taux de GetJob demandes en rafale	Le nombre maximum de GetJob demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	25	Non

Ressource	Description	Restrictions	Ajustable
Taux de <code>CancelJob</code> demandes	Le nombre maximum de <code>CancelJob</code> demandes que vous pouvez envoyer par seconde sur ce compte par région.	2 par seconde	Oui
Taux de <code>CancelJob</code> demandes en rafale	Le nombre maximum de <code>CancelJob</code> demandes supplémentaires par seconde (RPS) que vous pouvez envoyer en une seule rafale sur ce compte dans la région actuelle.	5	Non
Nombre de tâches <code>SV1</code> quantiques simultanées	Le nombre maximum de tâches quantiques simultanées exécutées sur le simulateur de vecteurs d'état (<code>SV1</code>) dans la région actuelle.	100 us-est-1, 50 US-ouest-1, 100 us-ouest-2, 50 eu-west-2	Non
Nombre de tâches <code>DM1</code> quantiques simultanées	Le nombre maximum de tâches quantiques simultanées exécutées sur le simulateur de matrice de densité (<code>DM1</code>) dans la région actuelle.	100 us-est-1, 50 US-ouest-1, 100 us-ouest-2, 50 eu-west-2	Non

Ressource	Description	Restrictions	Ajustable
Nombre de tâches TN1 quantiques simultanées	Le nombre maximum de tâches quantiques simultanées exécutées sur le simulateur de réseau tensoriel (TN1) dans la région actuelle.	10 us-est-1, 10 us-ouest-2, 5 eu-ouest-2,	Oui
Nombre d'emplois hybrides simultanés	Le nombre maximum d'emplois hybrides simultanés dans la région actuelle.	3	Oui
Limite d'exécution des tâches hybrides	Durée maximale en jours pendant laquelle une tâche hybride peut être exécutée.	5	Non

Les quotas d'instances de calcul classiques par défaut pour les tâches hybrides sont les suivants. Pour augmenter ces quotas, contactez [Support](#). En outre, les régions disponibles sont spécifiées pour chaque instance.

Ressource	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de type ml.c4.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c4.xlarge autorisé pour	5	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
	toutes les tâches hybrides Amazon Braket dans ce compte et cette région.							
Nombre maximal d'instances de ml.c4.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c4.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.c4.4xlarge pour les tâches hybrides	Le nombre maximum d'instances de ml.c4.4xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.c4.8xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c4.8xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.c5.4xlarge pour les tâches hybrides	Le nombre maximum d'instances de ml.c5.4xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	1	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5.9xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5.9xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	1	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5.18xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5.18xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5n.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5n.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5n.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5n.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.c5n.4xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5n.4xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5n.9xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5n.9xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.c5n.18xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.c5n.18xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.g4dn.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.g4dn.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.g4dn.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.g4dn.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.g4dn.xlarge pour les tâches hybrides	Le nombre maximum d'instances de ml.g4dn.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.g4dn.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.g4dn.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.g4dn.12xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.g4dn.12xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.g4dn.16xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.g4dn.16xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m4.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m4.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.m4.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m4.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.m4.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m4.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	2	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m4.10xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m4.10xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m4.16xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m4.16xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m5.large pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.large autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m5.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m5.2xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.2xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de type ml.m5.4xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.4xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	5	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m5.12xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.12xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.m5.24xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.m5.24xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Oui	Oui	Oui	Oui

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.p2.xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.p2.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Non	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximum d'instances de ml.p2.xlarge pour les tâches hybrides	Le nombre maximum d'instances de ml.p2.xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Non	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.p2.16xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.p2.16xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Non	Oui	Non	Non

Ressources	Description	Restrictions	Ajustable	us-east-1	us-west-1	us-west-2	eu-west-2	eu-north-1
Nombre maximal d'instances de ml.p4d.24xlarge pour les tâches hybrides	Le nombre maximum d'instances de type ml.p4d.24xlarge autorisé pour toutes les tâches hybrides Amazon Braket dans ce compte et cette région.	0	Oui	Oui	Non	Oui	Non	Non

Demande de mise à jour des limites

Si vous recevez une `ServiceQuotaExceeded` exception pour un type d'instance et que vous ne disposez pas d'un nombre suffisant d'instances disponibles pour ce type d'instance, vous pouvez demander une augmentation de limite depuis la page [Service Quotas](#) de la AWS console et rechercher Amazon Braket dans la section AWS Services.

Note

Si votre tâche hybride ne parvient pas à fournir la capacité de calcul ML demandée, utilisez une autre région. En outre, si aucune instance ne figure dans le tableau, elle n'est pas disponible pour les tâches hybrides.

Quotas et limites supplémentaires

- La taille de l'action de tâche quantique Amazon Braket est limitée à 3 Mo.
- En effet SV1, la durée maximale de fonctionnement est de 3 heures pour les circuits jusqu'à 31 qubits et de 11 heures pour les circuits de plus de 31 qubits.
- Le nombre maximum de prises de vue par tâche autorisé pour SV1, DM1, et par Rigetti appareil est de 50 000.
- Le nombre maximum de tirs autorisés par tâche TN1 est de 1 000.
- Sur AQT l'IBEX-Q1 appareil de For, le maximum est de 2 000 prises par tâche.
- Pour tous IonQ les appareils : le nombre minimum de prises de vue par tâche est de 100. Lorsque vous utilisez un modèle à la demande, il existe une limite d'un million de [prises de](#) vue et un minimum de 2 500 prises de vue pour les tâches [d'atténuation des erreurs](#). Pour une réservation directe, il n'y a pas de limite de prises de vue, et un minimum de 500 prises de vue est requis pour les tâches de réduction des erreurs.
- Pour QuEra l'appareil Aquila, le maximum est de 1 000 tirs par tâche.
- Pour les IQM Emerald appareils Garnet et les appareils, le maximum est de 20 000 prises de vue par tâche.
- Pour TN1 et les appareils QPU, le nombre de prises de vues par tâche doit être supérieur à 0.

Historique du document pour le guide du développeur Amazon Braket

Le tableau suivant décrit les versions de documentation pour Amazon Braket.

- Dernière mise à jour des API références : 20 novembre 2025
- Dernière mise à jour de la documentation : 14 janvier 2026

Modification	Description	Date
Mettre à jour les pages « Travailler avec les réservations »	Clarté améliorée des pages « Travailler avec les réservations »	3 février 2026
Support Python version 3.12 pour les ordinateurs portables Braket et les conteneurs gérés	Ajout de la prise en charge de Python version 3.12 pour les blocs-notes Amazon Braket et les conteneurs gérés (Base, PennyLane CUDA-Q et Tensorflow). Inclut un guide de résolution des problèmes pour la mise à niveau de Python 3.12.	21 janvier 2026
Supprimer des exemples P3	SageMaker retire sa famille d'instances <code>m1.p3</code> . Exemples remplacés par la famille d'instances <code>m1.g4dn</code> recommandée.	19 décembre 2025
Nouvelle fonctionnalité de limite de dépenses	Ajout de la prise en charge de la fonctionnalité de limite de dépenses d'Amazon Braket, qui permet de définir des plafonds budgétaires facultatifs pour les individus	20 novembre 2025

	QPU afin de valider et de rejeter automatiquement les tâches dépassant le seuil de dépenses configuré.	
Nouvel appareil Braket AQT IBEX-Q1	Ajout du support pour un AQT IBEX-Q1 appareil. Cet appareil est basé sur un cristal de $^{40}\text{Ca}^+$ dans un piège macroscopique à radiofréquences installé dans une chambre à très haut vide.	18 novembre 2025
Nouveau support natif pour CUDA-Q Amazon Braket NBIs	Ajout du support natif pour les CUDA-Q instances de bloc-notes Amazon Braket. Pour plus d'informations, voir CUDA-Q dans. NBIs	10 novembre 2025
IonQ Aria-2retrait de l'appareil	Suppression du support de l'IonQ Aria-2appareil.	27 octobre 2025
Documentation consolidée sur les tâches hybrides	Consolidation des sections relatives aux tâches hybrides pour qu'elles apparaissent sous Working with Amazon Braket Hybrid Jobs .	21 octobre 2025
Nouveau conteneur fourni par CUDA-Q Braket	Ajout de la prise en charge d'un conteneur de tâches CUDA-Q hybride fourni. Pour plus d'informations, voir Définir l'environnement de votre script d'algorithme .	2 septembre 2025

Nouvelle fonctionnalité d'émulateur de périphérique local	Ajout de la prise en charge d'un outil d' émulation de dispositifs quantiques locaux permettant d'émuler vos programmes textuels avant de les soumettre à des appareils quantiques.	25 août 2025
Pages PennyLane et CUDA-Q déplacées dans la section Build	Les pages PennyLane et CUDA-Q ont été déplacées pour qu'elles apparaissent sous la section Build de la table des matières.	15 août 2025
Nouvelle ProgramSet fonctionnalité	Ajout du support pour les ensembles de programme s , une opération permettant d'exécuter plusieurs circuits quantiques dans une seule tâche quantique.	14 août 2025
Nouvel appareil IQM Emerald	Ajout du support pour l'IQM Emerald appareil. Un appareil de 54 bits avec une topologie en réseau carré (Crystal).	21 juillet 2025
Mise à jour de la AmazonBraketServiceRolePolicy politique	AmazonBraketServiceRolePolicy fournit désormais uniquement les actions s3 : * et logs : * à aws : Principal Account. Cela limite l'accès aux compartiments et aux groupes de journaux du demandeur uniquement.	11 juillet 2025

Nouvelle fonctionnalité de capacité expérimentale : circuits dynamiques	Les opérations de mesure en milieu de circuit et d'anticipation sont disponibles sous forme de capacités expérimentales, voir Accès aux circuits dynamiques sur les appareils IQM.	26 juin 2025
Mise à jour de la AmazonBraketFullAccess politique	AmazonBraketFullAccess inclut désormais la tarification : GetProducts pour afficher les coûts du matériel sur la console.	14 avril 2025
Nouvel appareil IonQ Forte-Enterprise-1	Ajout du support pour l'appareil IonQ Forte-Enterprise-1. Un appareil de 36 bits qui utilise la technologie des ions piégés.	17 mars 2025
Autorisations relatives aux conditions S3 améliorées	Pour améliorer la sécurité, fournit AmazonBraketFullAccess désormais uniquement s3:* des actions aux aws:PrincipalAccount. Cela restreint l'accès aux compartiments du demandeur uniquement.	07 mars 2025
Nouvel appareil Rigetti Ankaa-3	Ajout du support pour l'Rigetti Ankaa-3 appareil. Un appareil de 84 bits qui utilise une technologie multipuce évolutive.	14 janvier 2025
Rigetti Ankaa-2 retrait de l'appareil	Suppression du support de l'Rigetti Ankaa-2 appareil.	14 janvier 2025

Support pour le IPv6 trafic	Amazon Braket prend désormais en charge le IPv6 trafic via le point de terminaison DualStack. <code>braket.{region}.api.aws</code>	12 décembre 2024
Support pour NVIDIA's CUDA-Q sur Amazon Braket	Les clients peuvent désormais exécuter des programmes quantiques à l'aide NVIDIA's CUDA-Q du framework de développement sur Amazon Braket.	6 décembre 2024
IonQ Forte-1 l'appareil est facilement disponible	IonQ Forte-1 l'appareil n'est plus uniquement réservé aux réservations et est désormais facilement disponible pour nos clients.	22 novembre 2024
Rigetti Aspen-M-3 retrait de l'appareil	Suppression du support de l'Aspen-M-3 appareil.	27 septembre 2024
IonQ Harmony retrait de l'appareil	Suppression du support de l'IonQ Harmony appareil.	29 août 2024
Nouvel appareil Rigetti Ankaa-2	Ajout du support pour l'Ankaa-2 appareil. Un appareil de 84 bits qui utilise une technologie multipuce évolutive.	26 août 2024
Réorganisation du guide du développeur	Le nouveau guide du développeur reprend le parcours client existant de Build, Test, Run et guide les utilisateurs sur cette voie avec Amazon Braket.	23 août 2024

OQC Lucy retrait de l'appareil	Suppression du support de l'OQC Lucy appareil.	28 juin 2024
Nouvel appareil IQM Garnet et nouvelle région Europe North 1	Ajout du support pour l'appareil I IQM Garnet . Un appareil de 20 qubits avec une topologie en treillis carré. Extension des régions prises en charge par Braket à l'Europe du Nord 1 (Stockholm).	22 mai 2024
Déréglage local publié	Les capacités expérimentales incluent désormais la fonction de désaccordage local de l'Aquila QuEra QPU.	11 avril 2024
Le gestionnaire d'inactivité des ordinateurs portables est sorti	Lorsque vous créez une instance de bloc-notes , activez le gestionnaire d'inactivité et définissez une durée d'inactivité pour réinitialiser automatiquement l'instance de bloc-notes Braket.	27 mars 2024
Refonte de la table des matières	Réorganisation de la table des matières d'Amazon Braket afin de respecter les exigences AWS du guide de style et d'améliorer le flux de contenu pour améliorer l'expérience client.	12 décembre 2023

Support à diffusion directe	Ajout de la prise en charge des fonctionnalités de Braket Direct, notamment : • Travailler avec les réservations • Obtenir des conseils d'experts • Explorez les capacités expérimentales	27 novembre 2023
Mise à jour d' Création d'une instance de bloc-notes Amazon Braket	Mise à jour de la documentation afin d'ajouter des informations permettant de créer une instance de bloc-notes pour les nouveaux clients et les clients existants d'Amazon Braket.	27 novembre 2023
Mise à jour d' Apport de votre propre conteneur	Mise à jour de la documentation pour ajouter des informations sur le moment du BYOC, la recette du BYOC et l'exécution des tâches hybrides Braket sur le conteneur.	18 octobre 2023

Publication d'un décorateur d'emplois hybride	Exécutez votre code local en tant que tâche hybride Page ajoutée. Contient des exemples : • Création d'une tâche hybride à partir du code Python local • Installation de packages Python et de code source supplémentaires • Enregistrer et charger des données dans une instance de tâche hybride • Bonnes pratiques pour les décorateurs hybrides	16 octobre 2023
Visibilité accrue des files d'attente	Mise à jour de la documentation du guide du développeur pour inclure queue depth et queue position. Mise à jour de la documentation de l'API pour refléter les nouvelles modifications apportées à l'API en matière de visibilité des files d'attente.	25 septembre 2023
Standardiser la dénomination dans la documentation	Mise à jour de la documentation pour remplacer toutes les instances de « tâche » en « tâche hybride » et de « tâche » en « tâche quantique »	11 septembre 2023
Nouvel appareil IonQ Aria 2	Support supplémentaire pour l'IonQ Aria 2 appareil	8 septembre 2023

Native Gates mis à jour	Mise à jour de la documentation pour ajouter des informations sur l'accès programmatique aux portes natives depuis Rigetti.	16 août 2023
Xanadu départ	Mise à jour de la documentation pour supprimer tous les Xanadu appareils	2 juin 2023
Nouvel appareil IonQ Aria	Support supplémentaire pour l'IonQ Aria appareil	16 mai 2023
Rigetti Appareil retiré	Support interrompu pour Rigetti Aspen-M-2	2 mai 2023
Informations de Amazon Braket Full Access politique mises à jour	Mise à jour du script qui définit le contenu de la Amazon Braket Full Access politique afin d'inclure les GetMetricData actions servicequotas : GetServiceQuota et cloudwatch : ainsi que des informations sur les limites relatives aux quotas.	19 avril 2023
Lancement de Guided Journeys	La documentation a été modifiée pour refléter la méthode plus récente et simplifiée d'intégration de Braket.	5 avril 2023
Nouvel appareil Rigetti Aspen-M-3	Support supplémentaire pour l'Rigetti Aspen-M-3 appareil	17 janvier 2023
Nouvelle fonction de dégradé adjointe	Ajout d'informations sur la fonction de dégradé adjoint proposée par SV1	7 décembre 2022

Nouvelle fonctionnalité de bibliothèque d'algorithmes	Ajout d'informations sur la bibliothèque d'algorithmes Braket, qui fournit un catalogue d'algorithmes quantiques prédéfinis	28 novembre 2022
D-Wave départ	Mise à jour de la documentation pour permettre le retrait de tous les D-Wave appareils	17 novembre 2022
Nouvel appareil QuEra Aquila	Support supplémentaire pour l'QuEra Aquila appareil	31 octobre 2022
Support pour Braket Pulse	Ajout du support pour Braket Pulse, qui permet d'utiliser le contrôle du pouls sur Rigetti les appareils OQC	20 octobre 2022
Support pour les portes natives IonQ	Ajout de la prise en charge de l'ensemble de portes natif proposé par l'appareil IonQ	13 septembre 2022
Nouveaux quotas d'instances	Mise à jour des quotas d'instance de calcul classique par défaut associés aux tâches hybrides	22 août 2022
Nouveau tableau de bord des services	Captures d'écran de console mises à jour pour inclure le tableau de bord du service	17 août 2022
Nouvel appareil Rigetti Aspen-M-2	Support supplémentaire pour l'Rigetti Aspen-M-2 appareil	12 août 2022
Nouvelles fonctionnalités d'OpenQASM	Ajout du support des fonctionnalités OpenQASM pour les simulateurs locaux (braket_sv et braket_dm)	4 août 2022

Nouvelles procédures de suivi des coûts	Ajout de la procédure permettant d'obtenir des estimations des coûts maximaux en temps quasi réel pour les simulateurs et les charges de travail matérielles	18 juillet 2022
Nouvel Xanadu Borealis appareil	Support supplémentaire pour l'Xanadu Borealis appareil	2 juin 2022
Nouvelles procédures de simplification de l'intégration	Informations supplémentaires sur le fonctionnement des nouvelles procédures d'intégration simplifiées	16 mai 2022
Nouvel appareil D-Wave Advantage_system6.1	Support supplémentaire pour l'D-Wave Advantage_system6.1 appareil	12 mai 2022
Support pour les simulateurs embarqués	Ajout de la façon d'exécuter des simulations intégrées avec des tâches hybrides et de l'utilisation du simulateur de PennyLane foudre	4 mai 2022
AmazonBraketFullAccess - Politique d'accès complet à Amazon Braket	Ajout de s3 : ListAllMyBuckets autorisations permettant aux utilisateurs de consulter et d'inspecter les buckets créés et utilisés pour Amazon Braket	31 mars 2022
Support pour OpenQASM	Ajout de la prise en charge d'OpenQASM 3.0 pour les dispositifs quantiques et les simulateurs basés sur des portes	7 mars 2022

Nouveau fournisseur de matériel quantique Oxford Quantum Circuits et nouvelle région, eu-west-2	Ajout du support pour OQC et eu-west-2	28 février 2022
Nouvel Rigetti appareil	Ajout de la prise en charge de Rigetti Aspen M-1	15 février 2022
Nouvelles limites de ressources	Le nombre maximum de SV1 tâches DM1 et de tâches simultanées a été augmenté de 55 à 100	5 janvier 2022
Nouvel Rigetti appareil	Ajout de la prise en charge de Rigetti Aspen-11	20 décembre 2021
RigettiAppareil retiré	Support interrompu pour l'Rigetti Aspen-10appareil	20 décembre 2021
Nouveau type de résultat	Type de résultat de matrice à densité réduite pris en charge par un simulateur et des DM1 dispositifs de matrice de densité locaux	20 décembre 2021
Description de la politique mise à jour	Amazon Braket a mis à jour l'ARN du rôle pour inclure le <code>servicerole/</code> chemin. Pour plus d'informations sur les mises à jour des politiques, consultez le tableau des mises à jour des politiques AWS gérées par Amazon Braket .	29 novembre 2021
Offres d'emploi chez Amazon Braket	Guide de l'utilisateur pour Amazon Braket Hybrid Jobs et ajout API	29 novembre 2021

Nouvel Rigetti appareil	Ajout de la prise en charge de Rigetti Aspen-10	20 novembre 2021
D-WaveAppareil retiré	Support interrompu pour D-Wave QPU, Advantage_system1	4 novembre 2021
Nouvel D-Wave appareil	Ajout du support pour un D-Wave QPU supplémentaire, Advantage_system4	5 octobre 2021
Nouveaux simulateurs de bruit	Ajout de la prise en charge d'un simulateur de matrice de densité (DM1), qui peut simuler des circuits allant jusqu'à 17, qubits et d'un simulateur de bruit local braket_dm	25 mai 2021
PennyLane soutien	Ajout de la prise PennyLane en charge pour Amazon Braket	8 décembre 2020
Nouveau simulateur	Ajout du support pour un simulateur de réseau Tensor (TN1), qui permet des circuits plus grands	8 décembre 2020
Regroupement des tâches	Braket prend en charge le traitement par lots des tâches des clients	24 novembre 2020
qubitAllocation manuelle	Le support permet une qubit allocation manuelle sur l'appareil Rigetti	24 novembre 2020

Quotas ajustables	Braket prend en charge des quotas ajustables en libre-service pour les ressources de vos tâches	30 octobre 2020
Support pour PrivateLink	Vous pouvez configurer des points de terminaison VPC privés pour vos tâches Braket	30 octobre 2020
Prise en charge des balises	Braket prend en charge les balises API basées sur les balises pour la ressource de tâches quantiques	30 octobre 2020
Nouvel D-Wave appareil	Ajout du support pour un D-Wave QPU supplémentaire, Advantage_system1	29 septembre 2020
Première version	Publication initiale de la documentation Amazon Braket	12 août 2020

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.