



Guide du développeur

Amazon Kinesis Data Streams



Amazon Kinesis Data Streams: Guide du développeur

Copyright © 2024 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.

Table of Contents

Qu'est-ce qu'Amazon Kinesis Data Streams ?	1
Que puis-je faire avec Kinesis Data Streams ?	1
Avantages de l'utilisation de Kinesis Data Streams	2
Services connexes	3
Terminologie et concepts	4
Passez en revue l'architecture de haut niveau de Kinesis Data Streams	4
Familiarisez-vous avec la terminologie de Kinesis Data Streams	5
Flux de données Kinesis	5
Enregistrement de données	5
Mode de capacité	5
Période de conservation	6
Producer	6
Consommateur	6
Application Amazon Kinesis Data Streams	6
Partition	6
Clé de partition	7
Numéro de séquence	7
Bibliothèque cliente Kinesis	8
Nom de l'application	8
Chiffrement côté serveur	8
Quotas et limites	10
Limites API	12
KDSAPILimites du plan de contrôle	12
KDSAPILimites du plan de données	17
Augmentation des quotas	20
Conditions préalables complètes pour configurer Amazon Kinesis Data Streams	21
Inscrivez-vous pour AWS	21
Téléchargez les bibliothèques et les outils	22
Configuration de votre environnement de développement	22
Utilisez le AWS CLI pour effectuer des opérations Amazon Kinesis Data Streams	24
Tutoriel : Installation et configuration du AWS CLI pour Kinesis Data Streams	25
Installez le AWS CLI	25
Configurez le AWS CLI	26
Tutoriel : Exécutez des opérations Kinesis Data Streams de base à l'aide du AWS CLI	26

Étape 1 : créer un stream	27
Étape 2 : Insérer un enregistrement	29
Étape 3 : Obtenir le dossier	29
Étape 4 : Nettoyer	33
Tutoriels de mise en route	34
Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 2.x	34
Exécuter les opérations prérequis	35
Création d'un flux de données	36
Création d'une IAM politique et d'un utilisateur	37
Téléchargez et compilez le code	42
Implémenter le producteur	43
Mettre en œuvre le consommateur	48
(Facultatif) Étendre le consommateur	53
Nettoyage des ressources	54
Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x	56
Exécuter les opérations prérequis	57
Création d'un flux de données	58
Création d'une IAM politique et d'un utilisateur	60
Téléchargez et compilez le code d'implémentation	65
Implémenter le producteur	66
Mettre en œuvre le consommateur	70
(Facultatif) Étendre le consommateur	75
Nettoyage des ressources	76
Tutoriel : Analyser les données boursières en temps réel à l'aide d'Amazon Managed Service pour Apache Flink	78
Prérequis	79
Étape 1 : configuration d'un compte	80
Étape 2 : configurer le AWS CLI	83
Étape 3 : Création d'une application	85
Tutoriel : Utilisation AWS Lambda avec Amazon Kinesis Data Streams	102
Utilisez la solution de AWS streaming de données pour Amazon Kinesis	102
Création et gestion de flux de données Kinesis	104
Choisissez le mode de capacité du flux de données	104
Qu'est-ce qu'un mode de capacité de flux de données ?	105
Fonctionnalités et cas d'utilisation du mode à la demande	105
Fonctionnalités et cas d'utilisation du mode provisionné	107

Basculer entre les modes de capacité	108
Créer un flux à l'aide du AWS Management Console	109
Créer un flux à l'aide du APIs	110
Créer le client Kinesis Data Streams	110
Créer le stream	111
Mettre à jour un stream	112
Utilisation de la console	113
Utilisez le API	114
Utilisez le AWS CLI	114
Afficher la liste des flux	114
Répertorier les fragments	116
Supprimer un stream	119
Revisionner un stream	119
Décidez d'une stratégie de repartage	120
Séparer un fragment	121
Fusionner deux partitions	123
Terminez l'action de repartage	125
Modifier la période de conservation des données	127
Taguez vos streams	129
Passez en revue les bases des tags	129
Suivez les coûts à l'aide du balisage	130
Comprendre les restrictions relatives aux balises	130
Étiquetez les flux à l'aide de la console Kinesis Data Streams	131
Étiquetez les flux à l'aide du AWS CLI	132
Étiquetez les flux à l'aide des Kinesis Data Streams API	132
Écrire des données dans Kinesis Data Streams	133
Développez des producteurs à l'aide de la Kinesis Producer Library (KPL)	134
Passez en revue le rôle du KPL	135
Découvrez les avantages de l'utilisation du KPL	135
Sachez quand vous ne devez pas utiliser le KPL	137
Installer le KPL	137
Transition vers les certificats Amazon Trust Services (ATS) pour KPL	137
Plateformes prises en charge par KPL	138
KPL concepts clés	139
Intégrez le KPL avec le code du producteur	141
Écrivez dans votre flux de données Kinesis à l'aide du KPL	143

Configurez le KPL	145
Mettre en œuvre la désagrégation des consommateurs	146
Utilisez le KPL avec Amazon Data Firehose	150
Utiliser le KPL avec le registre des AWS Glue schémas	150
Configuration de la configuration du KPL proxy	151
Développez les producteurs en utilisant les Kinesis Data API Streams avec AWS SDK for Java	152
Ajouter des données à un flux	153
Interagissez avec les données à l'aide du registre des AWS Glue schémas	159
Écrire sur Amazon Kinesis Data Streams à l'aide de Kinesis Agent	160
Remplissez les conditions requises pour Kinesis Agent	161
Téléchargez et installez l'agent	161
Configuration et démarrage de l'agent	162
Spécifiez les paramètres de configuration de l'agent	163
Surveillez plusieurs répertoires de fichiers et écrivez dans plusieurs flux	167
Utiliser l'agent pour prétraiter les données	168
Utiliser les CLI commandes de l'agent	172
FAQ	173
Écrivez sur Kinesis Data Streams à l'aide d'autres services AWS	174
Écrivez dans Kinesis Data Streams à l'aide de AWS Amplify	175
Écrire sur Kinesis Data Streams à l'aide d'Amazon Aurora	175
Écrivez sur Kinesis Data Streams à l'aide d'Amazon CloudFront	176
Écrivez sur Kinesis Data Streams à l'aide d'Amazon Logs CloudWatch	176
Écrivez sur Kinesis Data Streams à l'aide d'Amazon Connect	176
Écrivez dans Kinesis Data Streams à l'aide de AWS Database Migration Service	177
Écrire sur Kinesis Data Streams à l'aide d'Amazon DynamoDB	177
Écrivez sur Kinesis Data Streams à l'aide d'Amazon EventBridge	177
Écrivez dans Kinesis Data Streams à l'aide de AWS IoT Core	177
Écrivez sur Kinesis Data Streams à l'aide d'Amazon Relational Database Service	178
Écrire sur Kinesis Data Streams Pinpoint using Amazon	178
Écrivez dans Kinesis Data Streams à l'aide de la base de données Amazon Quantum Ledger (Amazon) QLDB	178
Écrivez sur Kinesis Data Streams à l'aide d'intégrations tierces	179
Apache Flink	179
Fluentd	179
Debezium	179

Oracle GoldenGate	180
Kafka Connect	180
Adobe Experience	180
Striim	180
Résoudre les problèmes liés aux producteurs de Kinesis Data Streams	180
Ma candidature de producteur s'écrit plus lentement que prévu	181
Je reçois une erreur d'autorisation de clé KMS principale non autorisée	183
Résoudre d'autres problèmes courants rencontrés par les producteurs	183
Optimisez les producteurs de Kinesis Data Streams	183
Personnaliser le KPL comportement des nouvelles tentatives et des limites de débit	183
Appliquer les meilleures pratiques à l'KPLagrégation	185
Lire des données depuis Kinesis Data Streams	186
Utiliser le visualiseur de données dans la console Kinesis	188
Interrogez vos flux de données dans la console Kinesis	189
Développez les consommateurs en utilisant AWS Lambda	189
Développez les consommateurs à l'aide du service géré pour Apache Flink	190
Développez vos clients grâce à Amazon Data Firehose	190
Utiliser la bibliothèque cliente Kinesis	190
Qu'est-ce que Kinesis Client Library ?	191
KCLversions disponibles	192
Concepts KCL	193
Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client ...	195
Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public	
Java	208
Utiliser le KCL avec le registre des AWS Glue schémas	211
Développez des consommateurs personnalisés avec un débit partagé	212
Développez des consommateurs personnalisés avec un débit partagé en utilisant KCL	212
Développez des consommateurs personnalisés avec un débit partagé à l'aide du AWS SDK	
for Java	251
Développez des consommateurs personnalisés avec un débit dédié (fan-out amélioré)	258
Développez de meilleurs fans grâce à la version 2.x KCL	260
Développez un plus grand nombre de fans grâce aux Kinesis Data Streams API	267
Gérez davantage de clients fans grâce au AWS Management Console	270
Migrer les consommateurs de la version KCL 1.x vers la version KCL 2.x	271
Migrer le processeur d'enregistrements	272
Migrer l'usine de traitement des enregistrements	276

Migrer le travailleur	278
Configuration du client Amazon Kinesis	279
Suppression des temps d'inactivité	284
Suppressions de configurations clientes	285
Utiliser d'autres AWS services pour lire les données de Kinesis Data Streams	286
Lisez les données de Kinesis Data Streams à l'aide d'Amazon EMR	286
Lisez les données de Kinesis Data Streams à l'aide d'Amazon Pipes EventBridge	287
Lisez les données de Kinesis Data Streams à l'aide de AWS Glue	287
Lisez les données de Kinesis Data Streams à l'aide d'Amazon Redshift	287
Lisez depuis Kinesis Data Streams à l'aide d'intégrations tierces	287
Apache Flink	288
Adobe Experience Platform	288
Apache Druid	288
Apache Spark	289
Databricks	289
Plateforme Confluent Kafka	289
Kinesumer	289
Talend	289
Résoudre les problèmes des utilisateurs de Kinesis Data Streams	290
Certains enregistrements Kinesis Data Streams sont ignorés lors de l'utilisation de la bibliothèque cliente Kinesis	290
Les enregistrements appartenant à la même partition sont traités par différents processeurs d'enregistrements en même temps	291
L'application destinée au consommateur lit plus lentement que prévu	291
GetRecords renvoie un tableau d'enregistrements vide même s'il y a des données dans le flux	292
L'utérateur de partition expire de façon inattendue	293
Le traitement des dossiers des consommateurs prend du retard	294
Erreur d'autorisation de clé KMS principale non autorisée	295
Résoudre d'autres problèmes courants rencontrés par les consommateurs	295
Optimisez les utilisateurs de Kinesis Data Streams	295
Améliorez le traitement à faible latence	296
Traitez les données sérialisées à l' AWS Lambda aide de la bibliothèque Kinesis Producer .	297
Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions	297
Gérer les enregistrements dupliqués	299

Gérez le démarrage, l'arrêt et la régulation	302
Surveillez Kinesis Data Streams	304
Surveillez le service Kinesis Data Streams avec CloudWatch	304
Dimensions et statistiques d'Amazon Kinesis Data Streams	305
Accédez aux CloudWatch métriques Amazon pour Kinesis Data Streams	323
Surveillez l'état de santé de l'agent Kinesis Data Streams avec CloudWatch	324
Moniteur avec CloudWatch	324
Enregistrez les appels Amazon Kinesis Data API Streams avec AWS CloudTrail	325
Informations sur Kinesis Data Streams dans CloudTrail	325
Exemple : entrées du fichier journal Kinesis Data Streams	327
Surveillez le KCL avec CloudWatch	331
Métriques et espace de noms	332
Niveaux métriques et dimensions	332
Configuration métrique	333
Liste des métriques	333
Surveillez le KPL avec CloudWatch	346
Métriques, dimensions et espaces de noms	346
Niveau métrique et granularité	347
Accès local et CloudWatch téléchargement sur Amazon	348
Liste des métriques	349
Sécurité	353
Protection des données dans Kinesis Data Streams	354
Qu'est-ce que le chiffrement côté serveur pour Kinesis Data Streams ?	354
Considérations relatives aux coûts, aux régions et aux performances	356
Comment démarrer avec le chiffrement côté serveur ?	357
Création et utilisation de clés générées par l'utilisateur KMS	358
Autorisations d'utilisation des clés générées par l'utilisateur KMS	358
Vérifier et résoudre les problèmes liés aux autorisations KMS clés	361
Utiliser Kinesis Data Streams avec les points de terminaison de l'interface VPC	361
Contrôle de l'accès aux ressources Kinesis Data Streams à l'aide de IAM	365
Syntaxe d'une politique	366
Actions pour Kinesis Data Streams	367
Amazon Resource Names (ARNs) pour Kinesis Data Streams	368
Exemples de politiques pour Kinesis Data Streams	368
Partagez votre flux de données avec un autre compte	370

Configurer une AWS Lambda fonction pour lire depuis Kinesis Data Streams dans un autre compte	376
Partagez l'accès à l'aide de politiques basées sur les ressources	376
Validation de conformité pour Kinesis Data Streams	379
Résilience dans Kinesis Data Streams	380
Reprise après sinistre dans Kinesis Data Streams	380
Sécurité de l'infrastructure dans Kinesis Data Streams	381
Bonnes pratiques en matière de sécurité pour Kinesis Data Streams	382
Implémentation d'un accès sur la base du moindre privilège	382
Utiliser IAM les rôles	382
Implémenter le chiffrement côté serveur dans les ressources dépendantes	383
CloudTrail À utiliser pour surveiller les API appels	383
Historique du document	384
.....	ccclxxxvii

Qu'est-ce qu'Amazon Kinesis Data Streams ?

Vous pouvez utiliser Amazon Kinesis Data Streams pour collecter et traiter des [flux volumineux](#) d'enregistrements de données en temps réel. Vous pouvez créer des applications de traitement de données appelées applications Kinesis Data Streams. Une application Kinesis Data Streams standard lit les données à partir d'un flux de données sous forme d'enregistrements de données. Ces applications peuvent utiliser la bibliothèque cliente Kinesis et s'exécuter sur des instances AmazonEC2. Vous pouvez envoyer les enregistrements traités à des tableaux de bord, les utiliser pour générer des alertes et modifier dynamiquement les stratégies tarifaires et publicitaires, ou envoyer des données à différents autres services AWS . Pour plus d'informations sur les fonctionnalités et les tarifs de Kinesis Data Streams, consultez la rubrique [Amazon Kinesis Data Streams](#).

[Kinesis Data Streams fait partie de la plateforme de données de streaming Kinesis, au même titre que Firehose, Kinesis Video Streams et le service géré pour Apache Flink.](#)

Pour plus d'informations sur les solutions de AWS mégadonnées, voir [Big Data sur AWS](#). Pour plus d'informations plus sur les solutions de données en flux continu AWS , consultez la rubrique [Qu'est-ce que les données en flux continu ?](#).

Rubriques

- [Que puis-je faire avec Kinesis Data Streams ?](#)
- [Avantages de l'utilisation de Kinesis Data Streams](#)
- [Services connexes](#)

Que puis-je faire avec Kinesis Data Streams ?

Vous pouvez utiliser Kinesis Data Streams pour la collecte et l'agrégation rapides et continues de données. Le type de données utilisé peut inclure des données de journaux d'infrastructure informatique, des journaux d'applications, des réseaux sociaux, des flux de données du marché et des données de flux de clics web. Comme le temps de réponse pour la récupération et le traitement des données est en temps réel, le traitement est généralement léger.

Les scénarios suivants sont typiques de l'utilisation de Kinesis Data Streams :

Extraction et traitement accélérés des journaux et des flux de données

Les applications producteur peuvent envoyer leurs données directement dans un flux. Par exemple, envoyez des journaux système et d'application ; ils sont alors disponibles pour le traitement en quelques secondes. Cela empêche la perte des données du journal en cas de défaillance du serveur frontal ou de l'application. Kinesis Data Streams permet l'accélération de la collecte des flux de données, car vous n'avez pas besoin de regrouper les données sur les serveurs avant de les soumettre à la collecte.

Métriques et création de rapports en temps réel

Vous pouvez utiliser les données collectées dans Kinesis Data Streams pour une analyse de données simple et une création de rapports en temps réel. Par exemple, votre application de traitement de données peut générer des métriques et des rapports à partir de journaux d'applications et d'événements système dès que les données ont été transmises, au lieu d'attendre qu'elles soient envoyées par lots de données.

Analyse des données en temps réel

Cette analyse combine la puissance du traitement parallèle avec la valeur des données en temps réel. Par exemple, traiter des flux de clics de site Web en temps réel, puis analyser la facilité d'utilisation du site à l'aide de plusieurs applications Kinesis Data Streams différentes exécutées en parallèle.

Traitement des flux complexes

Vous pouvez créer des graphes acycliques dirigés (DAGs) à partir d'applications et de flux de données Kinesis Data Streams. Cela implique généralement de placer des données issues de plusieurs applications Kinesis Data Streams dans un autre flux pour être traitées en aval par une application Kinesis Data Streams différente.

Avantages de l'utilisation de Kinesis Data Streams

Bien que vous puissiez utiliser Kinesis Data Streams pour résoudre divers problèmes liés aux données en flux continu, un usage courant est l'agrégation en temps réel des données, suivie du chargement de ces données agrégées dans un entrepôt des données ou un cluster map-reduce.

Les données sont placées dans des flux de données Kinesis, ce qui en garantit la durabilité et l'élasticité. Le délai entre le moment où un enregistrement est placé dans le flux et le moment où il peut être récupéré (put-to-get délai) est généralement inférieur à 1 seconde. En d'autres termes,

une application Kinesis Data Streams peut commencer à consommer les données du flux presque immédiatement après l'ajout de ces données. L'aspect de service géré de Kinesis Data Streams vous libère de la charge opérationnelle de création et d'exécution d'un pipeline d'apport de données. Vous pouvez créer des applications de streaming de type map-reduce. L'élasticité de Kinesis Data Streams vous permet de mettre à l'échelle le flux à la hausse ou à la baisse afin de ne jamais perdre d'enregistrements de données avant leur expiration.

Plusieurs applications Kinesis Data Streams peuvent consommer les données d'un flux afin que différentes actions comme l'archivage et le traitement soient effectuées simultanément et indépendamment. Par exemple, deux applications peuvent lire des données à partir du même flux. La première application calcule les regroupements en cours et met à jour une table Amazon DynamoDB, et la seconde application compresse les données et les archive dans un magasin de données comme Amazon Simple Storage Service (Amazon S3). La table DynamoDB contenant les agrégats en cours d'exécution est ensuite lue par un tableau de bord pour les rapports. up-to-the-minute

La Kinesis Client Library permet une consommation de données tolérante aux pannes à partir des flux et assure la prise en charge de la mise à l'échelle pour les applications Kinesis Data Streams.

Services connexes

[Pour plus d'informations sur l'utilisation EMR des clusters Amazon pour lire et traiter directement les flux de données Kinesis, consultez Kinesis Connector.](#)

Terminologie et concepts relatifs à Amazon Kinesis Data Streams

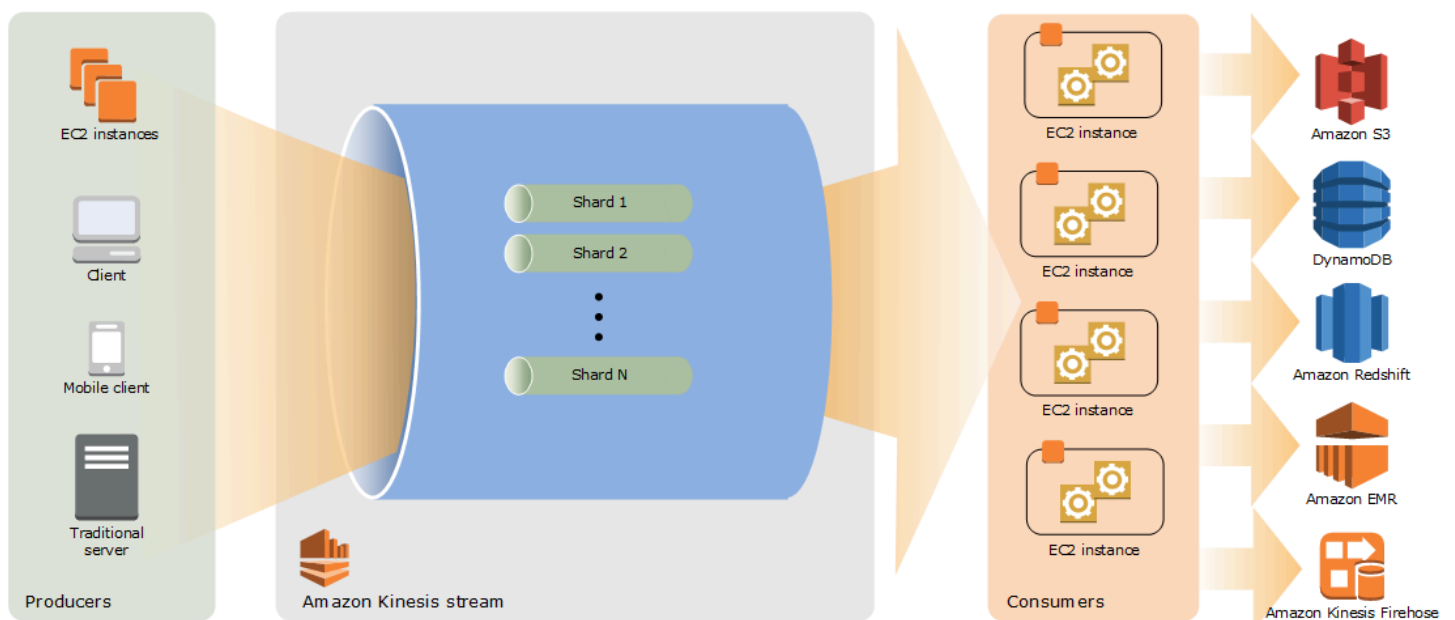
Avant de commencer à utiliser Amazon Kinesis Data Streams, découvrez son architecture et sa terminologie.

Rubriques

- [Passez en revue l'architecture de haut niveau de Kinesis Data Streams](#)
- [Familiarisez-vous avec la terminologie de Kinesis Data Streams](#)

Passez en revue l'architecture de haut niveau de Kinesis Data Streams

Le diagramme suivant illustre l'architecture de haut niveau de Kinesis Data Streams. Les applications producteur envoient (push) continuellement des données à Kinesis Data Streams et les applications consommateur traitent ces données en temps réel. Les consommateurs (tels qu'une application personnalisée exécutée sur Amazon EC2 ou un flux de diffusion Amazon Data Firehose) peuvent stocker leurs résultats à l'aide d'un AWS service tel qu'Amazon DynamoDB, Amazon Redshift ou Amazon S3.



Familiarisez-vous avec la terminologie de Kinesis Data Streams

Flux de données Kinesis

Un flux de données Kinesis est un ensemble de [partitions](#). Chaque partition comporte une séquence d'enregistrements de données. Chaque enregistrement de données dispose d'un [numéro de séquence](#) attribué par Kinesis Data Streams.

Enregistrement de données

Un enregistrement de données est l'unité de données stockée dans un [flux de données Kinesis](#). Les enregistrements de données sont composés d'un [numéro de séquence](#), d'une [clé de partition](#) et d'un blob de données, qui est une séquence immuable d'octets. Une fois que vous avez stocké les données dans l'enregistrement, Kinesis Data Streams n'inspecte pas, n'interprète pas ou ne modifie absolument pas le blob. Un blob de données peut atteindre jusqu'à 1 Mo.

Mode de capacité

Un mode de capacité de flux de données détermine comment la capacité est gérée et comment l'utilisation de votre flux de données vous est facturée. Actuellement, dans Kinesis Data Streams, vous pouvez choisir entre un mode à la demande et un mode provisionné pour vos flux de données. Pour de plus amples informations, veuillez consulter [Choisissez le mode de capacité du flux de données](#).

En mode à la demande, Kinesis Data Streams gère automatiquement les partitions afin de fournir le débit nécessaire. Vous n'êtes facturé que pour le débit réel que vous utilisez et Kinesis Data Streams répond automatiquement aux besoins de débit de vos charges de travail à mesure qu'elles augmentent ou diminuent. Pour de plus amples informations, veuillez consulter [Fonctionnalités et cas d'utilisation du mode à la demande](#).

En mode provisionné, vous devez spécifier le nombre de partitions du flux de données. La capacité totale d'un flux de données est la somme des capacités de ses partitions. Vous pouvez augmenter ou diminuer le nombre de partitions dans un flux de données selon vos besoins et le nombre de partitions vous est facturé selon un taux horaire. Pour de plus amples informations, veuillez consulter [Fonctionnalités et cas d'utilisation du mode provisionné](#).

Période de conservation

La période de conservation correspond à la durée pendant laquelle les enregistrements de données sont accessibles après avoir été ajoutés au flux. La période de conservation d'un flux a une valeur par défaut de 24 heures après la création. Vous pouvez augmenter la période de rétention jusqu'à 8760 heures (365 jours) en utilisant l'[IncreaseStreamRetentionPeriod](#) opération, et la réduire à un minimum de 24 heures en utilisant l'[DecreaseStreamRetentionPeriod](#) opération. Des frais supplémentaires s'appliquent pour les flux dont la période de conservation définie est supérieure à 24 heures. Pour en savoir plus, consultez la [Tarification Amazon Kinesis Data Streams](#).

Producer

Les producteurs placent des enregistrements dans Amazon Kinesis Data Streams. Par exemple, un serveur Web qui envoie des données de journal dans un flux est un producteur.

Consommateur

Les consommateurs récupèrent les enregistrements depuis Amazon Kinesis Data Streams et les traitent. Ces applications consommateur s'appellent [Application Amazon Kinesis Data Streams](#).

Application Amazon Kinesis Data Streams

Une application Amazon Kinesis Data Streams utilise un flux qui s'exécute généralement sur un parc EC2 d'instances.

Il existe deux types de consommateurs que vous pouvez développer : les consommateurs de diffusion partagée et les consommateurs de diffusion améliorée. Pour en savoir plus sur les différences entre ces deux types, et pour savoir comment créer chaque type de consommateur, consultez [Lire les données d'Amazon Kinesis Data Streams](#).

La sortie d'une application Kinesis Data Streams peut servir d'entrée à un autre flux, ce qui vous permet de créer des topologies complexes qui traitent des données en temps réel. Une application peut également envoyer des données à divers autres AWS services. Il peut y avoir plusieurs applications pour un seul flux, et chaque application peut utiliser des données du flux indépendamment et simultanément.

Partition

Une partition est une séquence d'enregistrements de données appartenant à un flux et identifiée de manière unique. Un flux se compose d'une ou plusieurs partitions, chacune fournissant une unité

de capacité fixe. Chaque partition peut prendre en charge jusqu'à 5 transactions par seconde pour les lectures, jusqu'à un taux total de lecture de données maximal de 2 Mo par seconde et jusqu'à 1 000 enregistrements par seconde pour les écritures, jusqu'à un taux d'écriture de données total maximal de 1 Mo par seconde (clés de partition incluses). La capacité de données de votre flux dépend du nombre de partitions que vous spécifiez pour le flux. La capacité totale du flux est la somme des capacités de ses shards.

Si votre débit de données augmente, vous pouvez augmenter ou diminuer le nombre de partitions allouées à votre flux. Pour de plus amples informations, veuillez consulter [Revisiter un stream](#).

Clé de partition

Une clé de partition sert à grouper les données par partition dans un flux. Kinesis Data Streams sépare les enregistrements de données appartenant à un flux en plusieurs partitions. Il utilise la clé de partition associée à chaque enregistrement de données pour déterminer à quelle partition un enregistrement de données spécifique appartient. Les clés de partition sont des chaînes Unicode, avec une longueur maximale de 256 caractères pour chaque clé. Une fonction de MD5 hachage est utilisée pour mapper les clés de partition à des valeurs entières de 128 bits et pour mapper les enregistrements de données associés aux partitions en utilisant les plages de clés de hachage des partitions. Lorsqu'une application place des données dans un flux, elle doit spécifier une clé de partition.

Numéro de séquence

Chaque enregistrement de données a un numéro de séquence qui est unique par clé de partition au sein de son groupe de données. Kinesis Data Streams attribue le numéro de séquence une fois que vous avez écrit dans le flux avec `client.putRecords` ou `client.putRecord`. Les numéros de séquence correspondant à une même clé de partition deviennent généralement de plus en plus longs au fil du temps. Plus le délai entre les demandes d'écriture est élevé, plus les numéros de séquence sont longs.

Note

Les numéros de séquence ne peuvent pas servir d'index aux ensembles de données d'un même flux. Pour séparer logiquement les ensembles de données, utilisez des clés de partition ou créez un flux distinct pour chaque ensemble de données.

Bibliothèque cliente Kinesis

La bibliothèque client Kinesis est compilée dans votre application pour permettre une consommation de données tolérante aux pannes à partir du flux. La bibliothèque client Kinesis garantit la présence, pour chaque partition, d'un processeur d'enregistrements qui s'exécute et traite cette partition. La bibliothèque simplifie également la lecture des données dans le flux. La bibliothèque cliente Kinesis utilise un tableau Amazon DynamoDB pour stocker les données de contrôle. Elle crée une seule table par application qui traite des données.

Il existe deux versions principales de la bibliothèque cliente Kinesis. Celle que vous utilisez dépend du type de consommateur que vous souhaitez créer. Pour de plus amples informations, veuillez consulter [Lire les données d'Amazon Kinesis Data Streams](#).

Nom de l'application

Le nom d'une application Amazon Kinesis Data Streams identifie l'application. Chacune de vos applications doit avoir un nom unique correspondant au AWS compte et à la région utilisés par l'application. Ce nom est utilisé comme nom pour la table de contrôle dans Amazon DynamoDB et comme espace de noms pour les métriques Amazon. CloudWatch

Chiffrement côté serveur

Amazon Kinesis Data Streams peut chiffrer automatiquement des données sensibles lorsqu'un producteur les saisit dans un flux. Kinesis Data Streams utilise des clés principales [AWS KMS](#) pour le chiffrement. Pour de plus amples informations, veuillez consulter [Protection des données dans Amazon Kinesis Data Streams](#).

Note

Pour lire un flux chiffré ou écrire sur un flux chiffré, les applications producteur et consommateur doivent disposer d'autorisations pour accéder à la clé principale. Pour plus d'informations sur l'octroi d'autorisations à des applications producteur et consommateur, consultez [the section called “Autorisations d'utilisation des clés générées par l'utilisateur KMS”](#).

Note

L'utilisation du chiffrement côté serveur entraîne des coûts AWS Key Management Service (AWS KMS). Pour plus d'informations, consultez [Tarification de AWS Key Management Service](#) (français non garanti).

Quotas et limites

Le tableau suivant décrit les quotas et les limites de flux et de partitions pour Amazon Kinesis Data Streams.

Quota	Mode de capacité à la demande	Mode alloué
Nombre de flux de données	Il n'y a pas de quota maximum pour le nombre de streams sur votre AWS compte. Par défaut, vous pouvez créer jusqu'à 50 flux de données en mode de capacité à la demande. Si vous avez besoin d'une augmentation de ce quota, veuillez créer un ticket d'assistance .	Il n'y a pas de quota maximum pour le nombre de flux en mode provisionné dans un compte.
Nombre de partitions	Il n'y a pas de limite supérieure. Le nombre de partitions dépend de la quantité de données ingérées et du niveau de débit requis. Kinesis Data Streams met automatiquement à l'échelle le nombre de partitions en fonction de l'évolution du volume de données et du trafic.	Il n'y a pas de limite supérieure. Le quota de partitions par défaut est de 500 partitions par AWS compte pour les AWS régions suivantes : USA Est (Virginie du Nord), USA Ouest (Oregon) et Europe (Irlande). Pour toutes les autres régions, le quota de partitions par défaut est de 200 partitions par compte AWS . Pour demander une augmentation du quota de shards-per-data diffusion, consultez la section Demande d'augmentation de quota .

Quota	Mode de capacité à la demande	Mode alloué
Débit du flux de données	<p>Par défaut, les nouveaux flux de données créés en mode de capacité à la demande ont un débit de 4 Mo/s en écriture et de 8 Mo/s en lecture. À mesure que le trafic augmente, les flux de données en mode capacité à la demande atteignent un débit de 200 Mo/s en écriture et de 400 Mo/s en lecture. Si vous avez besoin d'augmenter la capacité d'écriture à 2 Go/s et la capacité de lecture à 4 Go/s, soumettez un ticket d'assistance</p>	<p>Il n'y a pas de limite supérieure. Le débit maximal dépend du nombre de partitions provisionnées pour le flux. Chaque partition peut prendre en charge un débit d'écriture allant jusqu'à 1 Mo/s ou 1 000 enregistrements/s ou un débit de lecture allant jusqu'à 2 Mo/s ou 2 000 enregistrements/s. Si vous avez besoin d'une plus grande capacité d'ingestion, vous pouvez facilement augmenter le nombre de partitions dans le flux en utilisant le AWS Management Console ou le UpdateShardCount API</p>
Taille de la charge utile des données	<p>La taille maximale de la charge utile de données d'un enregistrement avant base64-encoding est de jusqu'à 1 Mo.</p>	
Taille de la transaction GetRecords	<p>GetRecords peut récupérer jusqu'à 10 Mo de données par appel à partir d'une seule partition, et jusqu'à 10 000 enregistrements par appel. Chaque appel à <code>GetRecords</code> est compté comme une seule transaction de lecture. Chaque partition peut prendre en charge jusqu'à cinq transactions de lecture par seconde. Chaque transaction de lecture peut fournir jusqu'à 10 000 enregistrements avec un quota supérieur de 10 Mo par transaction.</p>	

Quota	Mode de capacité à la demande	Mode alloué
Débit de lecture des données par partition	Chaque partition peut prendre en charge un taux de lecture de données total maximal de 2 Mo par seconde via GetRecords . Si un appel à <code>GetRecords</code> renvoie 10 Mo, les appels suivants effectués dans un délai de 5 secondes génèrent une exception.	
Nombre de consommateurs enregistrés par flux de données	Vous pouvez créer jusqu'à 20 consommateurs enregistrés (limite de débit amélioré Enhanced Fan-out) pour chaque flux de données.	
Basculement entre les modes provisionné et à la demande	Pour chaque flux de données de votre AWS compte, vous pouvez passer du mode à la demande au mode capacité provisionnée à deux reprises en 24 heures.	

Limites API

Comme la plupart des opérations AWS APIs, les opérations Kinesis Data API Streams sont limitées en débit. Les limites suivantes s'appliquent par AWS compte et par région. Pour plus d'informations sur Kinesis Data APIs Streams, consultez [le API Amazon Kinesis Reference](#).

KDSAPILimites du plan de contrôle

La section suivante décrit les limites du plan KDS de contrôle APIs. KDSun plan de contrôle vous APIs permet de créer et de gérer vos flux de données. Ces limites s'appliquent par compte et par région AWS .

APILimites du plan de contrôle

API	APILimite d'appels	Par compte/flux	Description
<code>AddTagsToStream</code>	5 transactions par seconde (TPS)	Par compte	50 balises par flux de données
<code>CreateStream</code>	5 TPS	Par compte	Il n'existe pas de quota supérieur au nombre de flux que

API	APLimite d'appels	Par compte/flux	Description
			<p>vous pouvez avoir dans un compte. Vous recevez une <code>LimitExceededException</code> lors d'une requête <code>CreateStream</code> lorsque vous essayez d'effectuer l'une des opérations suivantes :</p> <ul style="list-style-type: none"> • Avoir plus de cinq flux dans l'état <code>CREATING</code> à tout moment. • Créez plus de fragments que ceux autorisés pour votre compte.
<code>DecreaseStreamRetentionPeriod</code>	5 TPS	Par flux	La valeur minimale de la période de rétention d'un flux de données est de 24 heures.
<code>DeleteResourcePolicy</code>	5 TPS	Par compte	Si vous avez besoin d'une augmentation de cette limite, veuillez créer un Ticket d'assistance .
<code>DeleteStream</code>	5 TPS	Par compte	

API	APLimite d'appels	Par compte/flux	Description
DeregisterStreamConsumer	5 TPS	Par flux	
DescribeLimits	1 TPS	Par compte	
DescribeStream	10 TPS	Par compte	
DescribeStreamConsumer	20 TPS	Par flux	
DescribeStreamSummary	20 TPS	Par compte	
DisableEnhancedMonitoring	5 TPS	Par flux	
EnableEnhancedMonitoring	5 TPS	Par flux	
GetResourcePolicy	5 TPS	Par compte	Si vous avez besoin d'une augmentation de cette limite, veuillez créer un Ticket d'assistance .
IncreaseStreamRetentionPeriod	5 TPS	Par flux	La valeur maximale de la période de conservation d'un flux est de 8 760 heures (365 jours).
ListShards	1000 TPS	Par flux	
ListStreamConsumers	5 TPS	Par flux	
ListStreams	5 TPS	Par compte	
ListTagsForStream	5 TPS	Par flux	

API	APLimite d'appels	Par compte/flux	Description
MergeShards	5 TPS	Par flux	Uniquement applicable aux flux provisionnés.
PutResourcePolicy	5 TPS	Par compte	Si vous avez besoin d'une augmentation de cette limite, veuillez créer un Ticket d'assistance .
RegisterStreamConsumer	5 TPS	Par flux	Vous pouvez enregistrer jusqu'à 20 consommateurs par flux de données. Une application consommateur donnée ne peut être enregistrée qu'auprès d'un flux de données à la fois. Seuls 5 consommateurs peuvent être créés simultanément. En d'autres termes, vous ne pouvez pas avoir plus de 5 consommateurs dans un état CREATING en même temps. Enregistrement d'un 6e consommateur alors qu'il y en a 5 dans un état CREATING

API	APLimite d'appels	Par compte/flux	Description
RemoveTagsFromStream	5 TPS	Par flux	
SplitShard	5 TPS	Par flux	Uniquement applicable aux flux provisionnés
StartStreamEncryption		Par flux	Vous pouvez appliquer avec succès une nouvelle AWS KMS clé pour le chiffrement côté serveur 25 fois sur une période continue de 24 heures.
StopStreamEncryption		Par flux	Vous pouvez désactiver le chiffrement côté serveur 25 fois sur une période de 24 heures.
UpdateShardCount		Par flux	Uniquement applicable aux flux provisionnés. La limite par défaut du nombre de partitions est de 10 000. Il existe des limites supplémentaires à ce sujetAPI. Pour plus d'informations, consultez UpdateShardCount .

API	APLimite d'appels	Par compte/flux	Description
UpdateStreamMode		Par flux	Pour chaque flux de données de votre AWS compte, vous pouvez passer du mode à la demande au mode capacité provisionnée à deux reprises en 24 heures.

KDSAPILimites du plan de données

La section suivante décrit les limites du plan KDS de donnéesAPIs. KDSle plan de données vous APIs permet d'utiliser vos flux de données pour collecter et traiter des enregistrements de données en temps réel. Ces limites s'appliquent par partition dans vos flux de données.

APILimites du plan de données

API	APLimite d'appels	Limite de charge utile	Informations supplémentaires
GetRecords	5 TPS	Le nombre maximal d'enregistrements pouvant être renvoyés par appel est de 10 000. La taille maximale des données que GetRecords peut renvoyer est de 10 Mo.	Si un appel renvoie cette quantité de données, les appels ultérieurs effectués dans les 5 secondes suivantes lancent ProvisionedThroughputExceededException . Si le débit provisionné sur le flux est insuffisant, les appels

API	APLimite d'appels	Limite de charge utile	Informations supplémentaires
			ultérieurs effectués au cours de la prochaine seconde lancent <code>ProvisionedThroughputExceededException</code> .
GetShardIterator	5 TPS		Un itérateur de partition expire cinq minutes après avoir été renvoyé au demandeur. Si une <code>GetShardIterator</code> demande est faite trop souvent, vous recevez un <code>ProvisionedThroughputExceededException</code> .
PutRecord	1000 TPS	Chaque partition peut prendre en charge des écritures jusqu'à 1 000 enregistrements par seconde, jusqu'à un total d'écritures de données maximal de 1 Mo par seconde.	

API	APLimite d'appels	Limite de charge utile	Informations supplémentaires
PutRecords		<p>Chaque PutRecords demande peut prendre en charge jusqu'à 500 enregistrements. Chaque enregistrement de la demande peut atteindre 1 Mo, jusqu'à une limite de 5 Mo pour l'ensemble de la demande, y compris les clés de partition. Chaque partition peut prendre en charge des écritures jusqu'à 1 000 enregistrements par seconde, jusqu'à un total d'écritures de données maximal de 1 Mo par seconde.</p>	
SubscribeToShard	<p>Vous pouvez effectuer un appel SubscribeToShard par seconde par consommateur enregistré par partition.</p>		<p>Si vous appelez SubscribeToShard à nouveau avec le même consommateur ARN et ShardId dans les 5 secondes suivant un appel réussi, vous recevrez unResourceInUseException.</p>

Augmentation des quotas

Vous pouvez utiliser Service Quotas pour demander l'augmentation d'un quota, si ce dernier est ajustable. Certaines demandes sont automatiquement résolues, tandis que d'autres sont soumises à AWS Support. Vous pouvez suivre l'état d'une demande d'augmentation de quota envoyée à AWS Support. Les demandes d'augmentation des quotas de service ne reçoivent pas de soutien prioritaire. Si vous avez une demande urgente, contactez le AWS Support. Pour plus d'informations, consultez [Qu'est-ce que Service Quotas ?](#).

Pour demander une augmentation du quota de service, suivez la procédure décrite sous [Demande d'augmentation d'un quota](#).

Conditions préalables complètes pour configurer Amazon Kinesis Data Streams

Avant d'utiliser Amazon Kinesis Data Streams pour la première fois, effectuez les tâches suivantes pour configurer votre environnement.

Tâches

- [Inscrivez-vous pour AWS](#)
- [Téléchargez les bibliothèques et les outils](#)
- [Configuration de votre environnement de développement](#)

Inscrivez-vous pour AWS

Lorsque vous vous inscrivez à Amazon Web Services (AWS), votre AWS compte est automatiquement connecté à tous les services AWS, y compris Kinesis Data Streams. Seuls les services que vous utilisez vous sont facturés.

Si vous avez déjà un AWS compte, passez à la tâche suivante. Si vous n'avez pas de compte AWS, observez la procédure suivante pour en créer un.

Pour créer un AWS compte

1. Ouvrez l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisissez un code de vérification en utilisant le clavier numérique du téléphone.

Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des AWS services et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Téléchargez les bibliothèques et les outils

Les bibliothèques et outils suivants vous aideront à travailler avec Kinesis Data Streams :

- L'[Amazon Kinesis API Reference](#) est l'ensemble d'opérations de base pris en charge par Kinesis Data Streams. Pour plus d'informations sur l'exécution des opérations de base à l'aide du code Java, consultez les pages suivantes :
 - [Développez les producteurs en utilisant Amazon Kinesis Data API Streams avec AWS SDK for Java](#)
 - [Développez des consommateurs personnalisés avec un débit partagé à l'aide du AWS SDK for Java](#)
 - [Création et gestion de flux de données Kinesis](#)
- Le AWS SDKs pour [Go](#), [Java](#), [JavaScript](#), [.NET](#), [Node.js](#), [PHP](#), [Python](#) et [Ruby](#) incluent le support et les exemples de Kinesis Data Streams. Si votre version AWS SDK for Java ne contient pas d'exemples pour Kinesis Data Streams, vous pouvez également les télécharger depuis [GitHub](#)
- La bibliothèque cliente Kinesis (KCL) fournit un modèle de easy-to-use programmation pour le traitement des données. Ils KCL peuvent vous aider à démarrer rapidement avec Kinesis Data Streams en Java, Node.js, .NET, Python et Ruby. Pour en savoir plus, consultez [Lecture de données à partir de flux de données](#)
- La [AWS Command Line Interface](#) prend en charge Kinesis Data Streams. AWS CLI Cela vous permet de contrôler plusieurs AWS services à partir de la ligne de commande et de les automatiser par le biais de scripts.

Configuration de votre environnement de développement

Pour utiliser leKCL, assurez-vous que votre environnement de développement Java répond aux exigences suivantes :

- Java 1.7 (Java SE 7JDK) ou version ultérieure. Vous pouvez télécharger la dernière version du logiciel Java à partir de [Java SE Downloads](#) sur le site Web d'Oracle.
- Package Apache Commons (code, HTTP client et journalisation)
- JSONProcesseur Jackson

Notez que le kit [AWS SDK for Java](#) comprend Apache Commons et Jackson dans le dossier tiers. Cependant, le SDK for Java fonctionne avec Java 1.6, tandis que la bibliothèque cliente Kinesis nécessite Java 1.7.

Utilisez le AWS CLI pour effectuer des opérations Amazon Kinesis Data Streams

Cette section explique comment effectuer des opérations de base sur Amazon Kinesis Data Streams à l'aide AWS Command Line Interface du. Vous apprendrez les principes fondamentaux de la circulation des données Kinesis Data Streams et les étapes nécessaires à suivre pour placer des données dans un flux de données Kinesis ou en extraire.

Si vous ne connaissez pas Kinesis Data Streams, commencez par vous familiariser avec les concepts et la terminologie présentés dans [Terminologie et concepts relatifs à Amazon Kinesis Data Streams](#).

Rubriques

- [Tutoriel : Installation et configuration du AWS CLI pour Kinesis Data Streams](#)
- [Tutoriel : Exécutez des opérations Kinesis Data Streams de base à l'aide du AWS CLI](#)

Pour CLI y accéder, vous avez besoin d'un identifiant de clé d'accès et d'une clé d'accès secrète. Utilisation des informations d'identification temporaires au lieu des clés d'accès à long terme si possible. Les informations d'identification temporaires incluent un ID de clé d'accès, une clé d'accès secrète et un jeton de sécurité qui indique la date d'expiration des informations d'identification. Pour plus d'informations, consultez la section [Utilisation d'informations d'identification temporaires avec AWS des ressources](#) dans le Guide de IAM l'utilisateur.

Vous trouverez des instructions détaillées step-by-step IAM et de configuration des clés de sécurité sur la [page Créer un IAM utilisateur](#).

Dans cette section, les commandes spécifiques présentées sont indiquées mot pour mot, sauf lorsque des valeurs spécifiques sont nécessairement différentes pour chaque exécution. En outre, les exemples utilisent la région USA Ouest (Oregon), mais les étapes décrites dans cette section fonctionnent dans [n'importe quelle région où Kinesis Data Streams est pris en charge](#).

Tutoriel : Installation et configuration du AWS CLI pour Kinesis Data Streams

Installez le AWS CLI

Pour obtenir des instructions détaillées sur l'installation du AWS CLI pour Windows et pour les systèmes d'exploitation Linux, OS X et Unix, consultez la section [Installation du AWS CLI](#).

Utilisez la commande suivante pour répertorier les options et services disponibles :

```
aws help
```

Vous utiliserez le service Kinesis Data Streams. Vous pourrez ainsi consulter les sous-commandes associées à Kinesis Data Streams à AWS CLI l'aide de la commande suivante :

```
aws kinesis help
```

Cette commande génère une sortie qui contient les commandes Kinesis Data Streams disponibles :

AVAILABLE COMMANDS

- o add-tags-to-stream
- o create-stream
- o delete-stream
- o describe-stream
- o get-records
- o get-shard-iterator
- o help
- o list-streams
- o list-tags-for-stream
- o merge-shards

- o `put-record`
- o `put-records`
- o `remove-tags-from-stream`
- o `split-shard`
- o `wait`

Cette liste de commandes correspond aux Kinesis Data API Streams documentés dans [le Amazon API Kinesis Service Reference](#). Par exemple, la `create-stream` commande correspond à `CreateStreamAPIaction`.

Le AWS CLI est maintenant installé avec succès, mais il n'est pas configuré. Sa configuration fait l'objet de la section suivante.

Configurez le AWS CLI

Pour une utilisation générale, la `aws configure` commande est le moyen le plus rapide de configurer votre AWS CLI installation. Pour plus d'informations, consultez [Configuration de l' AWS CLI](#).

Tutoriel : Exécutez des opérations Kinesis Data Streams de base à l'aide du AWS CLI

Cette section décrit l'utilisation de base d'un flux de données Kinesis à partir de la ligne de commande à l'aide de l'interface AWS CLI. Assurez-vous de bien connaître les concepts abordés dans [Terminologie et concepts relatifs à Amazon Kinesis Data Streams](#)

Note

Une fois que vous avez créé un stream, votre compte est soumis à des frais minimes pour l'utilisation de Kinesis Data Streams, car Kinesis Data Streams n'est pas éligible au AWS niveau gratuit. Lorsque vous aurez terminé ce didacticiel, supprimez vos AWS ressources pour ne plus encourir de frais. Pour plus d'informations, consultez [Étape 4 : Nettoyer](#).

Rubriques

- [Étape 1 : créer un stream](#)
- [Étape 2 : Insérer un enregistrement](#)
- [Étape 3 : Obtenir le dossier](#)
- [Étape 4 : Nettoyer](#)

Étape 1 : créer un stream

Votre première étape consiste à créer un flux de données et à vérifier qu'il a été créé avec succès. Utilisez la commande suivante pour créer un flux nommé « Foo » :

```
aws kinesis create-stream --stream-name Foo
```

Entrez ensuite la commande suivante pour vérifier le progression de la création du flux :

```
aws kinesis describe-stream-summary --stream-name Foo
```

Vous devez obtenir une sortie similaire à l'exemple suivant :

```
{
  "StreamDescriptionSummary": {
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
    "StreamStatus": "CREATING",
    "RetentionPeriodHours": 48,
    "StreamCreationTimestamp": 1572297168.0,
    "EnhancedMonitoring": [
      {
        "ShardLevelMetrics": []
      }
    ],
    "EncryptionType": "NONE",
    "OpenShardCount": 3,
    "ConsumerCount": 0
  }
}
```

Dans cet exemple, le flux possède un statut `CREATING`, ce qui signifie qu'il n'est pas encore prêt à être utilisé. Vérifiez à nouveau après quelques instants. Vous devez voir une sortie similaire à l'exemple suivant :

```
{
  "StreamDescriptionSummary": {
    "StreamName": "Foo",
    "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
    "StreamStatus": "ACTIVE",
    "RetentionPeriodHours": 48,
    "StreamCreationTimestamp": 1572297168.0,
    "EnhancedMonitoring": [
      {
        "ShardLevelMetrics": []
      }
    ],
    "EncryptionType": "NONE",
    "OpenShardCount": 3,
    "ConsumerCount": 0
  }
}
```

Cette sortie contient des informations dont vous n'avez pas besoin pour ce didacticiel. Les informations importantes pour le moment sont `"StreamStatus": "ACTIVE"` les suivantes : elles indiquent que le flux est prêt à être utilisé, ainsi que les informations sur le fragment que vous avez demandé. Vous pouvez également vérifier l'existence de votre nouveau flux en utilisant la commande `list-streams`, comme il est illustré ici :

```
aws kinesis list-streams
```

Sortie :

```
{
  "StreamNames": [
    "Foo"
  ]
}
```

Étape 2 : Insérer un enregistrement

Maintenant que vous avez un flux actif, vous êtes prêt à placer des données. Pour ce didacticiel, vous allez utiliser la commande la plus simple possible, soit `put-record`, qui place un enregistrement de données spécifique contenant le texte « `testdata` » dans le flux :

```
aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
```

Cette commande, si elle aboutit, génère une sortie similaire à l'exemple suivant :

```
{
  "ShardId": "shardId-000000000000",
  "SequenceNumber": "49546986683135544286507457936321625675700192471156785154"
}
```

Félicitations, vous venez d'ajouter des données à un flux ! Vous allez ensuite voir comment extraire des données d'un flux.

Étape 3 : Obtenir le dossier

GetShardIterator

Avant de pouvoir obtenir des données à partir du flux, vous devez obtenir l'itérateur de partition correspondant à la partition qui vous intéresse. L'itérateur de partition représente la position du flux et de la partition à partir de laquelle l'application consommateur (dans ce cas, la commande `get-record`) lit. Vous allez utiliser la `get-shard-iterator` commande comme suit :

```
aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type
TRIM_HORIZON --stream-name Foo
```

N'oubliez pas que les `aws kinesis` commandes sont associées à Kinesis Data API Streams. Si vous êtes curieux de connaître l'un des paramètres affichés, vous pouvez en savoir plus sur eux dans [GetShardIterator](#) API la rubrique de référence. Une exécution réussie produira un résultat similaire à l'exemple suivant :

```
{
  "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWR060TZRNw9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LABk33gQweTJADBdyMwlo5r6PqcP2dzhg="
}
```

Cette longue chaîne de caractères apparemment aléatoires correspond à l'itérateur de partition (la vôtre sera différente). Vous devez copier/coller l'itérateur de partition dans la commande `get`, illustrée ci-dessous. Les itérateurs de partition ont une durée de vie valide de 300 secondes, qui doit vous suffire pour en copier/coller un dans la commande suivante. Vous devez supprimer toutes les nouvelles lignes de votre itérateur de partition avant de les coller à la commande suivante. Si vous recevez un message d'erreur indiquant que l'itérateur de partition n'est plus valide, réexécutez la `get-shard-iterator` commande.

GetRecords

La `get-records` commande extrait les données du flux et se transforme [GetRecords](#) en un appel vers Kinesis Data API Streams. L'itérateur de partition spécifie la position de la partition à partir de laquelle vous souhaitez démarrer les enregistrements de données de façon séquentielle. Si aucun enregistrement n'est disponible dans la partie de la partition vers laquelle pointe l'itérateur, `GetRecords` renvoie une liste vide. Plusieurs appels peuvent être nécessaires pour accéder à une partie de la partition contenant des enregistrements.

Dans l'exemple de `get-records` commande suivant :

```
aws kinesis get-records --shard-iterator
AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG41NR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWR06OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LABK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

Si vous exécutez ce didacticiel à partir d'un processeur de commande de type Unix tel que `bash`, vous pouvez automatiser l'acquisition de l'itérateur de partition à l'aide d'une commande imbriquée, comme celle-ci :

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000 --
shard-iterator-type TRIM_HORIZON --stream-name Foo --query 'ShardIterator')

aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

Si vous exécutez ce didacticiel à partir d'un système compatible PowerShell, vous pouvez automatiser l'acquisition de l'itérateur de partition à l'aide d'une commande telle que celle-ci :

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo).split('')
[4])
```


Si la `get-records` commande aboutit, des enregistrements seront demandés dans votre flux pour la partition que vous avez spécifiée lorsque vous avez obtenu l'itérateur de partition, comme dans l'exemple suivant :

```
{
  "Records": [ {
    "Data": "dGVzdGRhdGE=",
    "PartitionKey": "123",
    "ApproximateArrivalTimestamp": 1.441215410867E9,
    "SequenceNumber": "49544985256907370027570885864065577703022652638596431874"
  } ],
  "MillisBehindLatest": 24000,

  "NextShardIterator": "AAAAAAAAAAED0W3ugseWPE4503kqN1yN1UaodY8unE0sYs1MUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3V
}"
```

Notez que cela `get-records` est décrit ci-dessus comme une demande, ce qui signifie que vous pouvez recevoir zéro enregistrement ou plus même s'il y en a dans votre stream. Les enregistrements renvoyés peuvent ne pas représenter tous les enregistrements actuellement présents dans votre flux. C'est normal, et le code de production interrogera le flux à la recherche d'enregistrements à des intervalles appropriés. Cette vitesse d'interrogation varie en fonction des exigences spécifiques de conception de votre application.

Dans votre enregistrement dans cette partie du didacticiel, vous remarquerez que les données semblent être nulles, et ce n'est pas le texte clair que `testdata` nous avons envoyé. Cela provient de la manière dont `put-record` utilise l'encodage en Base64 pour vous permettre d'envoyer des données binaires. Cependant, la prise en charge de Kinesis Data Streams ne permet pas AWS CLI le décodage Base64, car le décodage Base64 en contenu binaire brut imprimé sur `stdout` peut entraîner des comportements indésirables et des problèmes de sécurité potentiels sur certaines plateformes et terminaux. Si vous utilisez un décodeur Base64 (par exemple, <https://www.base64decode.org/>) pour décoder manuellement `dGVzdGRhdGE=`, vous verrez que le texte correspond en fait à `testdata`. Cela est suffisant pour les besoins de ce didacticiel car, dans la pratique, le AWS CLI est rarement utilisé pour consommer des données. Le plus souvent, il est utilisé pour surveiller l'état du flux et obtenir des informations, comme indiqué précédemment (`describe-streametlist-streams`). Pour plus d'informations à ce sujet KCL, consultez la section [Développement de consommateurs personnalisés avec un débit partagé en utilisant KCL](#).

`get-records` ne renvoie pas toujours tous les enregistrements du stream/shard spécifié. Lorsque cela arrive, utilisez le `NextShardIterator` à partir du dernier résultat pour obtenir l'ensemble d'enregistrements suivant. Si davantage de données étaient introduites dans le flux, ce qui est normal dans les applications de production, vous pourriez continuer à interroger les données utilisées à `get-records` chaque fois. Toutefois, si vous n'appellez pas `get-records` en utilisant l'itérateur de partition suivant pendant la durée de vie de 300 secondes de l'itérateur de partition, vous recevrez un message d'erreur et vous devrez utiliser la `get-shard-iterator` commande pour obtenir un nouvel itérateur de partition.

Cette sortie indique également le nombre de millisecondes pendant lesquelles la réponse de l'[GetRecords](#) opération se situe `MillisBehindLatest` à partir de la fin du flux, indiquant à quel point le consommateur est en retard sur l'heure actuelle. Une valeur égale à zéro indique que le traitement des enregistrements est terminé et qu'il ne reste plus d'enregistrements à traiter pour le moment. Dans ce didacticiel, vous pouvez voir un nombre très grand si vous prenez le temps de lire le texte. Par défaut, les enregistrements de données restent dans un flux pendant 24 heures en attendant que vous les récupériez. Ce laps de temps s'appelle la période de conservation et peut être défini sur une valeur maximale de 365 jours.

Un `get-records` résultat positif aura toujours un effet `NextShardIterator` même s'il n'y a plus d'enregistrement actuellement dans le flux. Il s'agit d'un modèle d'interrogation qui suppose qu'un producteur met potentiellement plus d'enregistrements dans le flux à un moment donné. Bien que vous puissiez écrire vos propres routines de sondage, si vous utilisez les méthodes mentionnées précédemment KCL pour développer des applications grand public, ce sondage est pris en charge pour vous.

Si vous appelez `get-records` jusqu'à ce qu'il n'y ait plus d'enregistrements dans le flux et la partition que vous extrayez, vous verrez un résultat contenant des enregistrements vides, comme dans l'exemple suivant :

```
{
  "Records": [],
  "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdh06B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wu1/
EhyNeSs5DYXLSSC5XCapmCAYGFjYER69QsdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

Étape 4 : Nettoyer

Supprimez votre stream pour libérer des ressources et éviter des frais involontaires sur votre compte. Faites-le chaque fois que vous avez créé un flux et que vous n'allez pas l'utiliser, car des frais s'accumulent par flux, que vous y mettiez et que vous obteniez des données avec celui-ci ou non. La commande de nettoyage est la suivante :

```
aws kinesis delete-stream --stream-name Foo
```

Le succès n'entraîne aucune sortie. `describe-stream` À utiliser pour vérifier la progression de la suppression :

```
aws kinesis describe-stream-summary --stream-name Foo
```

Si vous exécutez cette commande immédiatement après la commande de suppression, vous obtiendrez un résultat similaire à l'exemple suivant :

```
{
  "StreamDescriptionSummary": {
    "StreamName": "samplestream",
    "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/samplestream",
    "StreamStatus": "ACTIVE",
```

Une fois que le flux est entièrement supprimé, la commande `describe-stream` génère une erreur de type « introuvable » :

```
A client error (ResourceNotFoundException) occurred when calling the
DescribeStreamSummary operation:
Stream Foo under account 123456789012 not found.
```

Tutoriels de mise en route pour Amazon Kinesis Data Streams

Amazon Kinesis Data Streams propose différentes solutions pour ingérer et consommer des données issues des flux de données Kinesis. Les didacticiels présentés dans cette section sont conçus pour vous aider à mieux comprendre les concepts et fonctionnalités d'Amazon Kinesis Data Streams et à identifier la solution qui répond à vos besoins.

Rubriques

- [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 2.x](#)
- [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#)
- [Tutoriel : Analyser les données boursières en temps réel à l'aide d'Amazon Managed Service pour Apache Flink](#)
- [Tutoriel : Utilisation AWS Lambda avec Amazon Kinesis Data Streams](#)
- [Utilisez la solution de AWS streaming de données pour Amazon Kinesis](#)

Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 2.x

Le scénario de ce didacticiel consiste à intégrer des transactions boursières dans un flux de données et à écrire une application Amazon Kinesis Data Streams de base qui effectue des calculs sur le flux. Vous apprendrez à envoyer un flux d'enregistrements vers Kinesis Data Streams et à implémenter une application qui consomme et traite les enregistrements en temps quasi réel.

Important

Une fois que vous avez créé un stream, votre compte est soumis à des frais minimes pour l'utilisation de Kinesis Data Streams, car Kinesis Data Streams n'est pas éligible au AWS niveau gratuit. Lorsque l'application consommateur a démarré, elle implique également des frais nominaux pour l'utilisation d'Amazon DynamoDB. L'application consommateur utilise DynamoDB pour suivre l'état du traitement. Lorsque vous avez terminé d'utiliser cette application, supprimez vos ressources AWS pour arrêter de générer des frais. Pour de plus amples informations, veuillez consulter [Nettoyage des ressources](#).

Le code n'accède pas aux données boursières réelles, mais simule le flux des opérations boursières. Pour cela, il utilise un générateur d'opérations boursières aléatoires qui part de données de marché réelles pour les 25 actions principales par capitalisation de marché à partir de février 2015. Si vous avez accès à un flux en temps réel d'opérations boursières, tirer des statistiques pratiques et adéquates de ce flux peut vous intéresser. Par exemple, vous pouvez effectuer une analyse par fenêtre glissante pour déterminer l'action la plus couramment achetée au cours des 5 dernières minutes. Vous souhaitez peut-être aussi recevoir une notification chaque fois qu'il y a un ordre de vente qui est trop grand (c'est-à-dire qui contient trop d'actions). Vous pouvez étendre le code illustré dans cette série pour fournir cette fonctionnalité.

Vous pouvez parcourir les étapes de ce didacticiel sur votre ordinateur de bureau ou ordinateur portable et exécuter le code de l'application producteur et de l'application consommateur sur la même machine ou sur une plateforme qui prend en charge les exigences définies.

Les exemples présentés utilisent la région USA Ouest (Oregon), mais ils fonctionnent dans toutes les [régions AWS qui prennent en charge Kinesis Data Streams](#).

Tâches

- [Exécuter les opérations prérequis](#)
- [Création d'un flux de données](#)
- [Création d'une IAM politique et d'un utilisateur](#)
- [Téléchargez et compilez le code](#)
- [Implémenter le producteur](#)
- [Mettre en œuvre le consommateur](#)
- [\(Facultatif\) Étendre le consommateur](#)
- [Nettoyage des ressources](#)

Exécuter les opérations prérequis

Vous devez remplir les conditions suivantes pour suivre ce didacticiel :

Création et utilisation d'un compte Amazon Web Services

Avant de commencer, assurez-vous de bien connaître les concepts abordés dans [Terminologie et concepts relatifs à Amazon Kinesis Data Streams](#), en particulier les streams, les shards, les producteurs et les consommateurs. Il est également utile d'avoir effectué les étapes présentées dans le guide suivant : [Tutoriel : Installation et configuration du AWS CLI pour Kinesis Data Streams](#).

Vous devez disposer d'un AWS compte et d'un navigateur Web pour accéder au AWS Management Console.

Pour accéder à la console, utilisez votre nom IAM d'utilisateur et votre mot de passe pour vous connecter à la page de connexion [AWS Management Console](#) depuis la page de IAM connexion. Pour plus d'informations sur les informations d'identification de AWS sécurité, y compris l'accès programmatique et les alternatives aux informations d'identification à long terme, voir les [informations d'identification AWS de sécurité](#) dans le guide de IAM l'utilisateur. Pour plus d'informations sur la connexion à votre Compte AWS compte, consultez la section [Comment vous connecter AWS dans](#) le guide de Connexion à AWS l'utilisateur.

Pour plus d'informations IAM et pour obtenir des instructions de configuration des clés de sécurité, consultez la section [Création d'un IAM utilisateur](#).

Répondre aux exigences logicielles du système

Java 7 ou version ultérieure installé sur le système que vous utilisez pour exécuter l'application. Pour télécharger et installer le dernier kit de développement Java (JDK), rendez-vous sur le [site d'installation de Java SE d'Oracle](#).

Vous avez besoin de la dernière version [AWS SDK for Java](#).

[L'application grand public nécessite la version 2.2.9 ou supérieure de Kinesis Client Library \(KCL\), que vous pouvez obtenir à l'adresse /tree/master. GitHub <https://github.com/aws-labs/amazon-kinesis-client>](#)

Étapes suivantes

[Création d'un flux de données](#)

Création d'un flux de données

Tout d'abord, vous devez créer le flux de données que vous utiliserez dans les étapes suivantes de ce didacticiel.

Pour créer un flux

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à <https://console.aws.amazon.com/kinesis>](https://console.aws.amazon.com/kinesis)

2. Choisissez Data Streams (Flux de données) dans le volet de navigation.
3. Dans la barre de navigation, développez le sélecteur de région et choisissez une région.
4. Choisissez Create Kinesis stream (Créer un flux Kinesis).
5. Saisissez un nom pour votre flux de données (par exemple, **StockTradeStream**).
6. Entrez **1** le nombre de fragments, mais conservez Estimez le nombre de fragments dont vous aurez besoin réduits.
7. Choisissez Create Kinesis stream (Créer un flux Kinesis).

Sur la page de la liste des flux Kinesis, l'état de votre flux apparaît comme CREATING lorsque le flux est en cours de création. Lorsque le flux est à prêt à être utilisé, le statut passe à ACTIVE.

Si vous choisissez le nom de votre flux, dans la page qui s'affiche, l'onglet Détails affiche un résumé de la configuration de votre flux de données. La section Surveillance affiche des informations de surveillance pour le flux.

Étapes suivantes

[Création d'une IAM politique et d'un utilisateur](#)

Création d'une IAM politique et d'un utilisateur

Les meilleures pratiques de sécurité AWS dictent l'utilisation d'autorisations détaillées pour contrôler l'accès aux différentes ressources. AWS Identity and Access Management (IAM) vous permet de gérer les utilisateurs et les autorisations des utilisateurs dans AWS. Une [IAM politique](#) répertorie explicitement les actions autorisées et les ressources auxquelles ces actions sont applicables.

Les éléments suivants sont les autorisations minimales généralement nécessaires pour une application producteur et une application consommateur Kinesis Data Streams.

Producer

Actions	Ressource	Objectif
DescribeStream , DescribeStreamSummary , DescribeStreamConsumer	Flux de données Kinesis	Avant d'essayer de lire les enregistrements, l'application vérifie si le flux existe, s'il est actif et si les partitions sont contenues dans le flux.

Actions	Ressource	Objectif
SubscribeToShard , RegisterStreamConsumer	Flux de données Kinesis	Abonne et enregistre les applications consommateurs dans
PutRecord , PutRecords	Flux de données Kinesis	Écrit des enregistrements dans Kinesis Data Streams.

Consommateur

Actions	Ressource	Objectif
DescribeStream	Flux de données Kinesis	Avant d'essayer de lire les enregistrements, l'application confirme que le flux existe, s'il est actif et si les partitions sont contenues dans
GetRecords , GetShardIterator	Flux de données Kinesis	Lit les enregistrements d'une partition.
CreateTable , DescribeTable , GetItem, PutItem, Scan, UpdateItem	Table Amazon DynamoDB	Si le client est développé à l'aide de la bibliothèque cliente (1.x ou 2.x), il doit avoir accès à une table DynamoDB pour le traitement de l'application.
DeleteItem	Table Amazon DynamoDB	Pour les moments où le consommateur effectue des opérations de fusion sur les partitions Kinesis Data Streams.
PutMetricData	CloudWatch Journal Amazon	KCLII télécharge également les métriques vers CloudWatch pour surveiller l'application.

Dans le cadre de ce didacticiel, vous allez créer une IAM politique unique qui accorde toutes les autorisations précédentes. En production, vous pouvez créer deux stratégies, l'une pour les applications producteurs et l'autre pour les consommateurs.

Pour créer une stratégie IAM

1. Recherchez le nom de ressource Amazon (ARN) pour le nouveau flux de données que vous avez créé à l'étape précédente. Vous pouvez le trouver ARN répertorié sous le nom Stream ARN en haut de l'onglet Détails. Le ARN format est le suivant :

```
arn:aws:kinesis:region:account:stream/name
```

region

Le code de AWS région ; par exemple, us-west-2. Pour de plus amples informations, veuillez consulter [Concepts de régions et de zones de disponibilité](#).

compte

L'identifiant du AWS compte, comme indiqué dans les [paramètres du compte](#).

name

Le nom du flux de données que vous avez créé à l'étape précédente, qui est `estStockTradeStream`.

2. Déterminez ARN la table DynamoDB à utiliser par le consommateur (et à créer par la première instance de consommateur). Il doit être au format suivant :

```
arn:aws:dynamodb:region:account:table/name
```

La région et l'ID de compte sont identiques aux valeurs du flux ARN de données que vous utilisez pour ce didacticiel, mais le nom est le nom de la table DynamoDB créée et utilisée par l'application client. KCL utilise le nom de l'application comme nom de table. Lors de cette étape, utilisez `StockTradesProcessor` comme nom de table DynamoDB, car il s'agit du nom d'application utilisé dans les étapes ultérieures de ce tutoriel.

3. Dans la IAM console, dans Politiques (<https://console.aws.amazon.com/iam/home#policies>), choisissez Create policy. Si c'est la première fois que vous travaillez avec des IAM politiques, choisissez Get Started, Create Policy.
4. Choisissez Sélectionner à côté de Policy Generator (Générateur de stratégies).
5. Choisissez Amazon Kinesis comme service. AWS
6. Sélectionnez DescribeStream, GetShardIterator, GetRecords, PutRecord et PutRecords comme actions autorisées.

7. Entrez le flux ARN de données que vous utilisez dans ce didacticiel.
8. Utilisez Ajouter une instruction pour chacune des propositions suivantes :

AWS Service	Actions	ARN
Amazon DynamoDB	CreateTable , DeleteItem , DescribeTable , GetItem, PutItem, Scan, UpdateItem	Le ARN de la table DynamoDB que vous avez créée à l'étape 2 de cette procédure.
Amazon CloudWatch	PutMetricData	*

L'astérisque (*) utilisé pour spécifier un n'ARNest pas obligatoire. Dans ce cas, c'est parce qu'il n'existe aucune ressource spécifique CloudWatch sur laquelle l'PutMetricDataaction est invoquée.

9. Choisissez Étape suivante.
10. Remplacez le Nom de la stratégie par StockTradeStreamPolicy, vérifiez le code et choisissez Create Policy (Créer stratégie).

Le document de stratégie qui en résulte doit ressembler à ceci :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ]
    }
  ],
}
```

```
    "Resource": [
      "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
    ]
  },
  {
    "Sid": "Stmt234",
    "Effect": "Allow",
    "Action": [
      "kinesis:SubscribeToShard",
      "kinesis:DescribeStreamConsumer"
    ],
    "Resource": [
      "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
    ]
  },
  {
    "Sid": "Stmt456",
    "Effect": "Allow",
    "Action": [
      "dynamodb:*"
    ],
    "Resource": [
      "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
    ]
  },
  {
    "Sid": "Stmt789",
    "Effect": "Allow",
    "Action": [
      "cloudwatch:PutMetricData"
    ],
    "Resource": [
      "*"
    ]
  }
]
```

Pour créer un utilisateur IAM

1. Ouvrez la IAM console à l'adresse <https://console.aws.amazon.com/iam/>.
2. Sur la page Users (Utilisateurs), choisissez Add user (Ajouter un utilisateur).

3. Pour User name (nom d'utilisateur), saisissez `StockTradeStreamUser`.
4. Pour Access type (Type d'accès), choisissez Programmatic access (Accès programmatique), puis Next: Permissions (Suivant : permissions).
5. Choisissez Attach existing policies directly (Attacher directement les politiques existantes).
6. Recherchez par nom la politique que vous avez créée dans la procédure précédente (`StockTradeStreamPolicy`). Cochez la case à gauche du nom de la stratégie, puis choisissez Next: Review (Suivant : réviser).
7. Vérifiez les détails et le récapitulatif, puis choisissez Create user (Créer utilisateur).
8. Copiez l'ID de clé d'accès et enregistrez-le en privé. Sous Clé d'accès secrète, choisissez Afficher et enregistrez également cette clé en privé.
9. Collez les clés d'accès et secrètes dans un fichier local dans un emplacement sûr auquel vous êtes le seul à pouvoir accéder. Pour cette application, créez un fichier nommé `~/.aws/credentials` (avec des autorisations strictes). Le fichier doit être au format suivant :

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

Pour associer une IAM politique à un utilisateur

1. Dans la IAM console, ouvrez [Politiques](#) et choisissez Policy Actions.
2. Choisissez `StockTradeStreamPolicy` et Attacher.
3. Choisissez `StockTradeStreamUser` et Attacher la stratégie.

Étapes suivantes

[Téléchargez et compilez le code](#)

Téléchargez et compilez le code

Cette rubrique fournit un exemple de code d'implémentation pour l'ingestion d'exemples d'opérations boursières dans le flux de données (application producteur) et pour le traitement de ces données (application consommateur).

Pour télécharger et générer le code

1. Téléchargez le code source du <https://github.com/aws-samples/amazon-kinesis-learning> GitHub dépôt sur votre ordinateur.
2. Créez un projet dans le vôtre IDE avec le code source, en respectant la structure de répertoire fournie.
3. Ajoutez les bibliothèques suivantes au projet :
 - Bibliothèque cliente Amazon Kinesis () KCL
 - AWS SDK
 - Apache HttpCore
 - Apache HttpClient
 - Apache Commons Lang
 - Apache Commons Logging
 - Guava (Google Core Libraries For Java)
 - Jackson Annotations
 - Jackson Core
 - Jackson Databind
 - Format de données Jackson : CBOR
 - Joda Time
4. En fonction de votre IDE, le projet peut être créé automatiquement. Si ce n'est pas le cas, créez le projet en suivant les étapes appropriées pour votre IDE.

Si vous avez exécuté correctement ces étapes, vous êtes maintenant prêt à passer à la section suivante, [the section called "Implémenter le producteur"](#).

Étapes suivantes

Implémenter le producteur

Ce didacticiel utilise le scénario réel de surveillance des opérations boursières. Les principes suivants expliquent brièvement comment ce scénario est mis en correspondance avec l'application producteur et la structure de code associée.

Reportez-vous au [code source](#) et vérifiez les informations suivantes.

StockTrade classe

Une transaction boursière individuelle est représentée par une instance de la `StockTrade` classe. Cette instance contient des attributs tels que le symbole boursier, le prix, le nombre d'actions, le type de l'opération (achat ou vente) et un ID identifiant l'opération de manière unique. Cette classe est implémentée pour vous.

Enregistrement de flux

Un flux est une séquence d'enregistrements. Un enregistrement est une sérialisation d'une `StockTrade` instance au JSON format. Par exemple :

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator classe

`StockTradeGenerator` possède une méthode appelée `getRandomTrade()` qui renvoie une nouvelle transaction boursière générée de manière aléatoire chaque fois qu'elle est invoquée. Cette classe est implémentée pour vous.

StockTradesWriter classe

La `main` méthode du producteur récupère `StockTradesWriter` en permanence une transaction aléatoire, puis l'envoie à Kinesis Data Streams en effectuant les tâches suivantes :

1. Lit le nom du flux de données et le nom de la région en entrée.
2. Utilise le `KinesisAsyncClientBuilder` pour définir la région, les informations d'identification et la configuration du client.
3. Elle vérifie que le flux existe et qu'il est actif (sinon il se ferme et génère une erreur).
4. Dans une boucle continue, elle appelle la méthode `StockTradeGenerator.getRandomTrade()`, puis la méthode `sendStockTrade` pour envoyer l'opération boursière au flux toutes les 100 millisecondes.

La méthode `sendStockTrade` de la classe `StockTradesWriter` comprend le code suivant :

```
private static void sendStockTrade(StockTrade trade, KinesisAsyncClient
kinesisClient,
    String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization
    by the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol
        as the partition key, explained in the Supplemental Information section below.
        .streamName(streamName)
        .data(SdkBytes.fromByteArray(bytes))
        .build();
    try {
        kinesisClient.putRecord(request).get();
    } catch (InterruptedException e) {
        LOG.info("Interrupted, assuming shutdown.");
    } catch (ExecutionException e) {
        LOG.error("Exception while sending data to Kinesis. Will try again next
        cycle.", e);
    }
}
```

Reportez-vous à la ventilation de code suivante :

- Le `PutRecord` API attend un tableau d'octets, et vous devez convertir le trade en JSON format. La seule ligne de code suivante effectue cette opération :

```
byte[] bytes = trade.toJsonAsBytes();
```

- Avant d'envoyer l'opération, vous créez une nouvelle instance `PutRecordRequest` (appelée `request` dans le cas suivant) : Chaque appel de `request` requiert le nom du flux, la clé de partition et un blob de données.

```
PutRecordRequest request = PutRecordRequest.builder()
    .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol as the
    partition key, explained in the Supplemental Information section below.
    .streamName(streamName)
    .data(SdkBytes.fromByteArray(bytes))
    .build();
```

L'exemple utilise un ticker boursier comme clé de partition, qui fait correspondre l'enregistrement à une partition spécifique. En pratique, vous devez avoir des centaines ou des milliers de clés de partition par partition afin de répartir les enregistrements de façon égale dans votre flux. Pour plus d'informations sur la façon d'ajouter des données à un flux, consultez la page [Écrire des données dans Amazon Kinesis Data Streams](#).

Maintenant `request` est prêt à envoyer au client (l'opération `put`) :

```
kinesisClient.putRecord(request).get();
```

- La vérification des erreurs et leur consignation sont toujours très utiles. Le code suivant consigne les conditions d'erreur :

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}
```

Ajoutez le bloc `try/catch` autour de l'opération `put` :

```
try {
    kinesisClient.putRecord(request).get();
} catch (InterruptedException e) {
    LOG.info("Interrupted, assuming shutdown.");
} catch (ExecutionException e) {
    LOG.error("Exception while sending data to Kinesis. Will try again
next cycle.", e);
}
```



```
}
```

Une opération put de Kinesis Data Streams peut échouer en raison d'une erreur réseau ou si le flux de données atteint ses limites de débit. Il est recommandé d'examiner attentivement votre politique de nouvelle tentative pour les put opérations afin d'éviter toute perte de données, par exemple en utilisant une nouvelle tentative.

- La journalisation d'état est très utile, mais elle est facultative :

```
LOG.info("Putting trade: " + trade.toString());
```

Le producteur présenté ici utilise la fonctionnalité PutRecord d'enregistrement unique de Kinesis Data API Streams. En pratique, si une application producteur génère de nombreux enregistrements, il est souvent plus efficace d'utiliser la fonctionnalité Plusieurs enregistrements de PutRecords et d'envoyer les lots d'enregistrements en même temps. Pour de plus amples informations, veuillez consulter [Écrire des données dans Amazon Kinesis Data Streams](#).

Pour exécuter l'application producteur

1. Vérifiez que la clé d'accès et la paire de clés secrètes récupérées dans [Création d'une IAM politique et d'un utilisateur](#) sont enregistrées dans le fichier `~/.aws/credentials`.
2. Exécutez la classe `StockTradeWriter` avec les arguments suivants :

```
StockTradeStream us-west-2
```

Si vous avez créé votre flux dans une région autre que `us-west-2`, vous devez spécifier cette région ici.

Vous devez voir des résultats similaires à ce qui suit :

```
Feb 16, 2015 3:53:00 PM  
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter  
sendStockTrade
```

```
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Vos transactions boursières sont maintenant ingérées par Kinesis Data Streams.

Étapes suivantes

[Mettre en œuvre le consommateur](#)

Mettre en œuvre le consommateur

L'application consommateur de ce didacticiel traite en continu les transactions boursières dans votre flux de données. Elle génère ensuite les actions les plus populaires achetées et vendues toutes les minutes. L'application est construite sur la base de la bibliothèque cliente Kinesis (KCL), qui effectue une grande partie du travail fastidieux propre aux applications grand public. Pour de plus amples informations, veuillez consulter [Utiliser la bibliothèque cliente Kinesis](#).

Reportez-vous au code source et vérifiez les informations suivantes.

StockTradesProcessor classe

La classe principale du consommateur, fournie pour vous, qui exécute les tâches suivantes :

- Lit les noms de l'application, du flux de données et des régions, transmis sous forme d'arguments.
- Crée une `KinesisAsyncClient` instance avec le nom de la région.
- Crée une instance `StockTradeRecordProcessorFactory` qui sert les instances de `ShardRecordProcessor`, implémentée par une instance `StockTradeRecordProcessor`.
- Crée une `ConfigsBuilder` instance avec `IStockTradeRecordProcessorFactory` instance `KinesisAsyncClient` `StreamNameApplicationName`, et. Ceci est utile pour créer toutes les configurations avec des valeurs par défaut.
- Crée un KCL planificateur (auparavant, dans KCL les versions 1.x, il était connu sous le nom de `KCL worker`) avec l'instance `ConfigsBuilder`

- Le planificateur crée un nouveau thread pour chaque partition (affectée à cette instance de consommateur), qui fonctionne en boucle pour lire des enregistrements dans le flux de données. Elle appelle alors l'instance `StockTradeRecordProcessor` pour traiter chaque lot d'enregistrements reçu.

StockTradeRecordProcessor classe

Implémentation de l'instance `StockTradeRecordProcessor`, qui implémente à son tour cinq méthodes requises : `initialize`, `processRecords`, `leaseLost`, `shardEnded` et `shutdownRequested`.

Les `shutdownRequested` méthodes `initialize` et sont utilisées par le processeur d'enregistrements KCL pour indiquer au processeur d'enregistrements quand il doit être prêt à commencer à recevoir des enregistrements et quand il doit s'attendre à arrêter de recevoir des enregistrements, respectivement, afin qu'il puisse effectuer toutes les tâches de configuration et de terminaison spécifiques à l'application. `leaseLost` et `shardEnded` sont utilisés pour implémenter toute logique indiquant ce qu'il faut faire lorsqu'un bail est perdu ou qu'un traitement a atteint la fin d'une partition. Dans cet exemple, nous enregistrons simplement les messages indiquant ces événements.

Le code de ces méthodes est fourni. Le traitement principal se déroule dans la méthode `processRecords`, qui utilise à son tour `processRecord` pour chaque enregistrement. Cette dernière méthode est fournie la plupart du temps sous forme de squelette de code vide à implémenter à l'étape suivante qui fournit des explications détaillées.

Notez également l'implémentation des méthodes d'assistance pour `processRecord` : `reportStats` et `resetStats`, qui sont vides dans le code source d'origine.

La méthode `processRecords` est implémentée pour vous et effectue les opérations suivantes :

- Pour chaque enregistrement passé, il appelle `processRecord` dessus.
- Si au moins 1 minute s'est écoulée depuis le dernier rapport, il appelle `reportStats()`, qui imprime les dernières statistiques, puis de `resetStats()`, qui efface les statistiques afin que l'intervalle suivant n'inclut que les nouveaux enregistrements.
- Règle l'heure du rapport suivant.
- Si au moins 1 minute s'est écoulée depuis le dernier point de contrôle, il appelle `checkpoint()`.
- Règle l'heure du point de contrôle suivante.

Cette méthode utilise des intervalles de 60 secondes pour la fréquence de création de rapports et de point de contrôle. Pour plus d'informations sur les points de contrôle, consultez la section [Utilisation de Kinesis Client Library](#) (français non garanti).

StockStats classe

Cette classe prend en charge la conservation des données et le suivi des statistiques dans le temps pour les actions les plus populaires. Ce code est fourni pour vous et contient les méthodes suivantes :

- `addStockTrade(StockTrade)` : Injecte le `StockTrade` donné dans les statistiques en cours d'exécution.
- `toString()` : renvoie les statistiques dans une chaîne formatée.

Cette classe suit les actions les plus populaires en comptabilisant le nombre total de transactions pour chaque action et le nombre maximum. Elle met à jour ces chiffres chaque fois qu'une opération boursière arrive.

Ajoutez du code aux méthodes de la classe `StockTradeRecordProcessor`, comme l'illustrent les étapes suivantes.

Pour implémenter l'application consommateur

1. Implémentez la méthode `processRecord` en instanciant un objet `StockTrade` correctement dimensionné et en lui ajoutant les données d'enregistrement, puis en consignnant un avertissement en cas de problème.

```
byte[] arr = new byte[record.data().remaining()];
record.data().get(arr);
StockTrade trade = StockTrade.fromJsonAsBytes(arr);
    if (trade == null) {
        log.warn("Skipping record. Unable to parse record into StockTrade.
Partition Key: " + record.partitionKey());
        return;
    }
stockStats.addStockTrade(trade);
```

2. Implémentez une `reportStats` méthode. Modifiez le format de sortie en fonction de vos préférences.

```
System.out.println("***** Shard " + kinesisShardId + " stats for last 1 minute
*****\n" +
stockStats + "\n" +
"*****\n");
```

3. Implémentez la méthode `resetStats`, qui crée une nouvelle instance `stockStats`.

```
stockStats = new StockStats();
```

4. Implémentez les méthodes suivantes requises par `ShardRecordProcessor` l'interface :

```
@Override
public void leaseLost(LeaseLostInput leaseLostInput) {
    log.info("Lost lease, so terminating.");
}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    log.info("Scheduler is shutting down, checkpointing.");
    checkpoint(shutdownRequestedInput.checkpointer());
}

private void checkpoint(RecordProcessorCheckpointer checkpointer) {
    log.info("Checkpointing shard " + kinesisShardId);
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException se) {
```

```

        // Ignore checkpoint if the processor instance has been shutdown (fail
over).
        log.info("Caught shutdown exception, skipping checkpoint.", se);
    } catch (ThrottlingException e) {
        // Skip checkpoint when throttled. In practice, consider a backoff and
retry policy.
        log.error("Caught throttling exception, skipping checkpoint.", e);
    } catch (InvalidStateException e) {
        // This indicates an issue with the DynamoDB table (check for table,
provisioned IOPS).
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
Kinesis Client Library.", e);
    }
}

```

Pour exécuter l'application consommateur

1. Exécutez l'application producteur que vous avez écrite dans `producer` pour injecter des enregistrements d'opérations boursières simulées dans votre flux.
2. Vérifiez que la clé d'accès et la paire de clés secrètes récupérées précédemment (lors de la création de IAM l'utilisateur) sont enregistrées dans le fichier `~/.aws/credentials`.
3. Exécutez la classe `StockTradesProcessor` avec les arguments suivants :

```
StockTradesProcessor StockTradeStream us-west-2
```

Notez que, si vous avez créé votre flux dans une région autre que `us-west-2`, vous devez spécifier cette région ici.

Au bout d'une minute, vous devez voir une sortie similaire à la suivante, actualisée toutes les minutes :

```

***** Shard shardId-000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
*****

```

Étapes suivantes

[\(Facultatif\) Étendre le consommateur](#)

(Facultatif) Étendre le consommateur

Cette section facultative montre comment étendre le code de l'application consommateur pour obtenir un scénario un peu plus complexe.

Si vous voulez connaître les ordres de vente les plus importants toutes les minutes, vous pouvez modifier la classe `StockStats` à trois emplacements pour accueillir cette nouvelle priorité.

Pour étendre l'application consommateur

1. Ajoutez de nouvelles variables d'instance :

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Ajoutez le code suivant à `addStockTrade`:

```
if (type == TradeType.SELL) {
    if (largestSellOrderStock == null || trade.getQuantity() >
        largestSellOrderQuantity) {
        largestSellOrderStock = trade.getTickerSymbol();
        largestSellOrderQuantity = trade.getQuantity();
    }
}
```

3. Modifiez la méthode `toString` pour imprimer les informations supplémentaires :

```
public String toString() {
    return String.format(
        "Most popular stock being bought: %s, %d buys.%n" +
        "Most popular stock being sold: %s, %d sells.%n" +
```

```
        "Largest sell order: %d shares of %s.",
        getMostPopularStock(TradeType.BUY),
        getMostPopularStockCount(TradeType.BUY),
        getMostPopularStock(TradeType.SELL),
        getMostPopularStockCount(TradeType.SELL),
        largestSellOrderQuantity, largestSellOrderStock);
    }
```

Si vous exécutez l'application consommateur maintenant (n'oubliez pas d'exécuter également l'application producteur), vous devez voir une sortie similaire à la suivante :

```
***** Shard shardId-000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
*****
```

Étapes suivantes

[Nettoyage des ressources](#)

Nettoyage des ressources

Étant donné que vous payez pour utiliser le flux de données Kinesis, veillez à le supprimer, ainsi que la table Amazon DynamoDB correspondante, lorsque vous avez terminé. Des frais nominaux s'appliquent sur un flux actif même lorsque vous n'envoyez ni ne recevez d'enregistrements. Cela provient du fait qu'un flux actif utilise des ressources en écoutant en permanence les demandes et enregistrements entrants pour obtenir des enregistrements.

Pour supprimer le flux et la table

1. Fermez tous les producteurs et consommateurs que vous pourriez avoir encore en activité.
2. [Ouvrez la console Kinesis à l'adresse /kinesis. https://console.aws.amazon.com](https://console.aws.amazon.com/kinesis)
3. Choisissez le flux que vous avez créé pour cette application (StockTradeStream).
4. Choisissez Supprimer flux.
5. Ouvrez la console DynamoDB à l'adresse. <https://console.aws.amazon.com/dynamodb/>

6. Supprimez la table `StockTradesProcessor`.

Récapitulatif

Le traitement d'une grande quantité de données en temps quasi réel ne nécessite pas l'écriture de code compliqué ni le développement d'une énorme infrastructure. C'est aussi simple que la logique d'écriture pour traiter une petite quantité de données (comme l'écriture `processRecord(Record)`), tout en utilisant Kinesis Data Streams pour l'adapter à une grande quantité de données en streaming. Vous n'avez pas à vous soucier de la mise à l'échelle de votre traitement, car Kinesis Data Streams s'en charge pour vous. Tout ce que vous avez à faire est d'envoyer vos enregistrements en flux continu à Kinesis Data Streams et d'écrire la logique pour traiter chaque nouvel enregistrement reçu.

Voici quelques améliorations potentielles pour cette application.

Regroupement à travers toutes les partitions

Actuellement, vous obtenez des statistiques résultant de l'agrégation des enregistrements de données reçues par un seul programme exécutant à partir d'une seule partition. (Une partition ne peut pas être traitée par plus d'une application de travail dans une seule application en même temps.) Bien entendu, lorsque vous mettez à l'échelle et que vous avez plusieurs partitions, vous pouvez regrouper toutes les partitions. Vous pouvez y procéder avec une architecture pipeline où la sortie de chaque application de travail alimente un autre flux comportant une seule partition, qui est traité par une application de travail qui regroupe les sorties de la première phase. Etant donné que les données de la première phase sont limitées (un échantillon par minute par partition), elles sont facilement gérées par une seule partition.

Mise à l'échelle du traitement

Lorsque le flux évolue vers plusieurs partitions (suite à l'envoi de données par plusieurs applications producteur), la façon de faire évoluer le traitement consiste à ajouter des applications de travail. Vous pouvez exécuter les travailleurs dans des EC2 instances Amazon et utiliser des groupes Auto Scaling.

Utilisation de connecteurs vers Amazon S3/DynamoDB/Amazon Redshift/Storm

Lorsqu'un flux est traité en continu, sa sortie peut être envoyée vers d'autres destinations. AWS fournit des [connecteurs](#) permettant d'intégrer Kinesis Data Streams à d'AWS autres services et outils tiers.

Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x

Pour ce tutoriel, le scénario implique d'ingérer des opérations boursières dans un flux de données et d'écrire une application Amazon Kinesis Data Streams simple qui effectue des calculs sur le flux. Vous apprendrez à envoyer un flux d'enregistrements vers Kinesis Data Streams et à implémenter une application qui consomme et traite les enregistrements en temps quasi réel.

Important

Une fois que vous avez créé un stream, votre compte est soumis à des frais minimes pour l'utilisation de Kinesis Data Streams, car Kinesis Data Streams n'est pas éligible au AWS niveau gratuit. Lorsque l'application consommateur a démarré, elle implique également des frais nominaux pour l'utilisation d'Amazon DynamoDB. L'application consommateur utilise DynamoDB pour suivre l'état du traitement. Lorsque vous avez terminé d'utiliser cette application, supprimez vos ressources AWS pour arrêter de générer des frais. Pour de plus amples informations, veuillez consulter [Nettoyage des ressources](#).

Le code n'accède pas aux données boursières réelles, mais simule le flux des opérations boursières. Pour cela, il utilise un générateur d'opérations boursières aléatoires qui part de données de marché réelles pour les 25 actions principales par capitalisation de marché à partir de février 2015. Si vous avez accès à un flux en temps réel d'opérations boursières, tirer des statistiques pratiques et adéquates de ce flux peut vous intéresser. Par exemple, vous pouvez effectuer une analyse par fenêtre glissante pour déterminer l'action la plus couramment achetée au cours des 5 dernières minutes. Vous souhaitez peut-être aussi recevoir une notification chaque fois qu'il y a un ordre de vente qui est trop grand (c'est-à-dire qui contient trop d'actions). Vous pouvez étendre le code illustré dans cette série pour fournir cette fonctionnalité.

Vous pouvez suivre les étapes de ce didacticiel sur votre ordinateur de bureau ou portable et exécuter le code producteur et le code consommateur sur la même machine ou sur toute plateforme répondant aux exigences définies, telle qu'Amazon Elastic Compute Cloud (AmazonEC2).

Les exemples présentés utilisent la région USA Ouest (Oregon), mais ils fonctionnent dans toutes les [régions AWS qui prennent en charge Kinesis Data Streams](#).

Tâches

- [Exécuter les opérations prérequis](#)
- [Création d'un flux de données](#)
- [Création d'une IAM politique et d'un utilisateur](#)
- [Téléchargez et compilez le code d'implémentation](#)
- [Implémenter le producteur](#)
- [Mettre en œuvre le consommateur](#)
- [\(Facultatif\) Étendre le consommateur](#)
- [Nettoyage des ressources](#)

Exécuter les opérations prérequis

Les exigences sont les suivantes pour la réalisation de [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#).

Création et utilisation d'un compte Amazon Web Services

Avant de commencer, assurez-vous de vous familiariser avec les concepts abordés dans [Terminologie et concepts relatifs à Amazon Kinesis Data Streams](#), notamment les flux, les partitions, les applications producteur et les applications consommateur. Il est également utile d'avoir consulté [Tutoriel : Installation et configuration du AWS CLI pour Kinesis Data Streams](#).

Vous avez besoin d'un AWS compte et d'un navigateur Web pour accéder au AWS Management Console.

Pour accéder à la console, utilisez votre nom IAM d'utilisateur et votre mot de passe pour vous connecter à la page de connexion [AWS Management Console](#) depuis la page de IAM connexion. Pour plus d'informations sur les informations d'identification de AWS sécurité, y compris l'accès programmatique et les alternatives aux informations d'identification à long terme, voir les [informations d'identification AWS de sécurité](#) dans le guide de IAM l'utilisateur. Pour en savoir plus sur la connexion à votre Compte AWS compte, consultez la section [Comment vous connecter AWS dans](#) le guide de Connexion à AWS l'utilisateur.

Pour plus d'informations IAM et pour obtenir des instructions de configuration des clés de sécurité, consultez la section [Créer un IAM utilisateur](#).

Répondre aux exigences logicielles du système

Java 7 ou une version supérieure doit être installé sur le système utilisé pour exécuter l'application. Pour télécharger et installer le dernier kit de développement Java (JDK), rendez-vous sur le [site d'installation de Java SE d'Oracle](#).

Si vous avez un JavalDE, tel qu'[Eclipse](#), vous pouvez ouvrir le code source, le modifier, le compiler et l'exécuter.

Vous avez besoin de la dernière version [AWS SDK for Java](#). Si vous utilisez EclipseIDE, vous pouvez plutôt installer le [AWS Toolkit for Eclipse](#).

L'application grand public nécessite la version 1.2.1 ou supérieure de Kinesis Client Library (KCL), que vous pouvez obtenir auprès de la GitHub [Kinesis Client Library](#) (Java).

Étapes suivantes

[Création d'un flux de données](#)

Création d'un flux de données

Dans la première étape de [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#), vous créez le flux que vous allez utiliser dans les étapes suivantes.

Pour créer un flux

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à https://console.aws.amazon.com l'adresse /kinesis](https://console.aws.amazon.com/kinesis).
2. Choisissez Data Streams (Flux de données) dans le volet de navigation.
3. Dans la barre de navigation, développez le sélecteur de région et choisissez une région.
4. Choisissez Create Kinesis stream (Créer un flux Kinesis).
5. Entrez un nom pour votre flux (**StockTradeStream**, par exemple).
6. Entrez **1** le nombre de fragments, mais conservez Estimez le nombre de fragments dont vous aurez besoin réduits.
7. Choisissez Create Kinesis stream (Créer un flux Kinesis).

Sur la page de la liste de flux Kinesis, l'état de votre flux est CREATING lorsque le flux est en cours de création. Lorsque le flux est à prêt à être utilisé, le statut passe à ACTIVE. Choisissez le nom de votre

flux. Sur la page qui s'affiche, l'onglet Détails affiche un récapitulatif de votre configuration de flux. La section Surveillance affiche des informations de surveillance pour le flux.

Informations supplémentaires sur les shards

Lorsque vous commencez à utiliser Kinesis Data Streams en dehors de ce tutoriel, vous devrez peut-être planifier plus soigneusement la création du flux. Lorsque vous mettez en service des partitions, vous devez prévoir pour la demande maximale prévue. En utilisant ce scénario comme exemple, le trafic de la bourse américaine est maximal durant la journée à une certaine heure et les estimations de la demande doivent être échantillonnées à partir de cette heure de la journée. Ensuite, vous avez le choix de prévoir pour faire face à la demande attendue maximale ou d'augmenter ou réduire les capacités de votre flux pour répondre aux fluctuations de la demande.

Une partition est une unité de capacité de débit. Sur la page Créer un flux Kinesis, développez Estimez le nombre de partitions dont vous aurez besoin. Indiquez la taille moyenne d'un enregistrement, le nombre maximal d'enregistrements écrits par seconde et le nombre d'applications consommateur en utilisant les instructions suivantes :

Taille moyenne d'enregistrement

Estimation de la taille moyenne calculée de vos enregistrements. Si vous ne connaissez pas cette valeur, indiquez la taille maximale estimée des enregistrements pour cette valeur.

Nombre max. d'enregistrements écrits

Tenez compte du nombre d'entités fournissant des données et du nombre approximatif d'enregistrements par seconde produits par chacune d'elles. Par exemple, si vous extrayez des données d'opérations boursières de 20 serveurs boursiers et que chacun génère 250 opérations par seconde, le nombre total d'opérations boursières par seconde est de 5 000/s.

Nombre d'applications utilisatrices

Un certain nombre d'applications qui lisent indépendamment dans le flux pour traiter le flux de façon différente et générer une sortie différente. Chaque application peut avoir plusieurs instances en cours d'exécution sur différentes machines (par exemple, s'exécuter dans un cluster) afin de faire face à un flux présentant un volume élevé.

Si le nombre estimé de partitions dépasse le nombre limite actuel de partitions, vous devrez peut-être soumettre une demande d'augmentation de cette limite avant de pouvoir créer un flux comprenant ce nombre de partitions. Pour demander une augmentation de la limite de partitions, utilisez le [formulaire](#)

[de limites Kinesis Data Streams](#). Pour plus d'informations sur les flux et les partitions, consultez [Création et gestion de flux de données Kinesis](#).

Étapes suivantes

[Création d'une IAM politique et d'un utilisateur](#)

Création d'une IAM politique et d'un utilisateur

Les meilleures pratiques de sécurité AWS dictent l'utilisation d'autorisations détaillées pour contrôler l'accès aux différentes ressources. AWS Identity and Access Management (IAM) vous permet de gérer les utilisateurs et les autorisations des utilisateurs dans AWS. Une [IAM politique](#) répertorie explicitement les actions autorisées et les ressources auxquelles ces actions sont applicables.

Voici les autorisations minimales généralement requises pour une application producteur et une application consommateur Kinesis Data Streams.

Producer

Actions	Ressource	Objectif
<code>DescribeStream</code> , <code>DescribeStreamSummary</code> , <code>DescribeStreamConsumer</code>	Flux de données Kinesis	Avant d'essayer d'écrire des enregistrements, le producteur flux existe et est actif, les fragments sont contenus dans le consommateur.
<code>SubscribeToShard</code> , <code>RegisterStreamConsumer</code>	Flux de données Kinesis	Abonner et enregistrer un consommateur à une partition de Kinesis.
<code>PutRecord</code> , <code>PutRecords</code>	Flux de données Kinesis	Écrire des enregistrements dans Kinesis Data Streams.

Consommateur

Actions	Ressource	Objectif
<code>DescribeStream</code>	Flux de données Kinesis	Avant d'essayer de lire les enregistrements, l'application doit vérifier que le flux existe et est actif et si les partitions sont contenues dans le flux.
<code>GetRecords</code> , <code>GetShardIterator</code>	Flux de données Kinesis	Lire les enregistrements à partir d'une partition Kinesis Data Streams.
<code>CreateTable</code> , <code>DescribeTable</code> , <code>GetItem</code> , <code>PutItem</code> , <code>Scan</code> , <code>UpdateItem</code>	Table Amazon DynamoDB	Si le client est développé à l'aide de la bibliothèque cliente Kinesis, le client doit être autorisé à accéder à une table DynamoDB pour suivre les données de l'application. La première application producteur démarrée crée une table DynamoDB.
<code>DeleteItem</code>	Table Amazon DynamoDB	Pour les moments où le consommateur effectue des opérations de fusion sur les partitions Kinesis Data Streams.
<code>PutMetricData</code>	CloudWatch Journal Amazon	KCLII télécharge également les métriques vers CloudWatch pour surveiller l'application.

Pour cette application, vous créez une IAM politique unique qui accorde toutes les autorisations précédentes. En pratique, vous pouvez envisager de créer deux stratégies, une pour les applications producteur et une pour les applications consommateur.

Pour créer une stratégie IAM

1. Trouvez le nom de la ressource Amazon (ARN) pour le nouveau flux. Vous pouvez le trouver dans le console IAM. L'ARN répertorié sous le nom Stream ARN en haut de l'onglet Détails. Le ARN format est le suivant :

```
arn:aws:kinesis:region:account:stream/name
```

region

Le code de la région, par exemple, us-west-2. Pour de plus amples informations, veuillez consulter [Concepts de régions et de zones de disponibilité](#).

compte

L'identifiant du AWS compte, comme indiqué dans les [paramètres du compte](#).

name

Le nom du flux indiqué dans [Création d'un flux de données](#), qui est StockTradeStream.

- Déterminez ARN la table DynamoDB à utiliser par le consommateur (et créée par la première instance de consommateur). Il doit être au format suivant :

```
arn:aws:dynamodb:region:account:table/name
```

La région et le compte proviennent du même endroit qu'à l'étape précédente, mais cette fois le nom est le nom de la table créée et utilisée par l'application consommateur. Le nom KCL utilisé par le consommateur utilise le nom de l'application comme nom de table. Utilisez StockTradesProcessor, qui est le nom d'application utilisé plus tard.

- Dans la IAM console, dans Politiques (<https://console.aws.amazon.com/iam/home#policies>), choisissez Create policy. Si c'est la première fois que vous travaillez avec des IAM politiques, choisissez Get Started, Create Policy.
- Choisissez Sélectionner à côté de Policy Generator (Générateur de stratégies).
- Choisissez Amazon Kinesis comme service. AWS
- Sélectionnez DescribeStream, GetShardIterator, GetRecords, PutRecord et PutRecords comme actions autorisées.
- Entrez celui ARN que vous avez créé à l'étape 1.
- Utilisez Ajouter une instruction pour chacune des propositions suivantes :

AWS Service	Actions	ARN
Amazon DynamoDB	CreateTable , DeleteItem , DescribeTable ,	Le ARN que vous avez créé à l'étape 2

AWS Service	Actions	ARN
	GetItem, PutItem, Scan, UpdateItem	
Amazon CloudWatch	PutMetricData	*

L'astérisque (*) utilisé pour spécifier un n'ARNest pas obligatoire. Dans ce cas, c'est parce qu'il n'existe aucune ressource spécifique CloudWatch sur laquelle l'PutMetricDataaction est invoquée.

9. Choisissez l'étape suivante.
10. Remplacez le Nom de la stratégie par `StockTradeStreamPolicy`, vérifiez le code et choisissez `Create Policy` (Créer stratégie).

Le document de stratégie obtenu doit ressembler à ce qui suit :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt123",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:ListShards",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer"
      ],
      "Resource": [
        "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream"
      ]
    },
    {
      "Sid": "Stmt234",
      "Effect": "Allow",
      "Action": [
```

```
    "kinesis:SubscribeToShard",
    "kinesis:DescribeStreamConsumer"
  ],
  "Resource": [
    "arn:aws:kinesis:us-west-2:123:stream/StockTradeStream/*"
  ]
},
{
  "Sid": "Stmt456",
  "Effect": "Allow",
  "Action": [
    "dynamodb:*"
  ],
  "Resource": [
    "arn:aws:dynamodb:us-west-2:123:table/StockTradesProcessor"
  ]
},
{
  "Sid": "Stmt789",
  "Effect": "Allow",
  "Action": [
    "cloudwatch:PutMetricData"
  ],
  "Resource": [
    "*"
  ]
}
]
```

Pour créer un utilisateur IAM

1. Ouvrez la IAM console à l'adresse <https://console.aws.amazon.com/iam/>.
2. Sur la page Users (Utilisateurs), choisissez Add user (Ajouter un utilisateur).
3. Pour User name (nom d'utilisateur), saisissez `StockTradeStreamUser`.
4. Pour Access type (Type d'accès), choisissez Programmatic access (Accès programmatique), puis Next: Permissions (Suivant : permissions).
5. Choisissez Attach existing policies directly (Attacher directement les politiques existantes).
6. Recherchez par nom la stratégie que vous avez créée. Cochez la case à gauche du nom de la stratégie, puis choisissez Next: Review (Suivant : réviser).

7. Vérifiez les détails et le récapitulatif, puis choisissez `Create user` (Créer utilisateur).
8. Copiez l'ID de clé d'accès et enregistrez-le en privé. Sous `Clé d'accès secrète`, choisissez `Afficher` et enregistrez également cette clé en privé.
9. Collez les clés d'accès et secrètes dans un fichier local dans un emplacement sûr auquel vous êtes le seul à pouvoir accéder. Pour cette application, créez un fichier nommé `~/.aws/credentials` (avec des autorisations strictes). Le fichier doit être au format suivant :

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

Pour associer une IAM politique à un utilisateur

1. Dans la IAM console, ouvrez [Politiques](#) et choisissez `Policy Actions`.
2. Choisissez `StockTradeStreamPolicy` et `Attacher`.
3. Choisissez `StockTradeStreamUser` et `Attacher` la stratégie.

Étapes suivantes

[Téléchargez et compilez le code d'implémentation](#)

Téléchargez et compilez le code d'implémentation

Le squelette de code est fourni pour le [the section called “Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x”](#). Il contient une implémentation stub pour l'ingestion du flux d'opérations boursières (application producteur) et le traitement des données (application consommateur). La procédure suivante indique comment terminer l'implémentation.

Pour télécharger et construire le code d'implémentation

1. Téléchargez le [code source](#) sur votre ordinateur.
2. Créez un projet dans vos favoris IDE avec le code source, en respectant la structure de répertoire fournie.
3. Ajoutez les bibliothèques suivantes au projet :
 - Bibliothèque cliente Amazon Kinesis () KCL
 - AWS SDK

- Apache HttpCore
 - Apache HttpClient
 - Apache Commons Lang
 - Apache Commons Logging
 - Guava (Google Core Libraries For Java)
 - Jackson Annotations
 - Jackson Core
 - Jackson Databind
 - Format de données Jackson : CBOR
 - Joda Time
4. En fonction de votre IDE, le projet peut être créé automatiquement. Si ce n'est pas le cas, créez le projet en suivant les étapes appropriées pour votre IDE.

Si vous avez exécuté correctement ces étapes, vous êtes maintenant prêt à passer à la section suivante, [the section called "Implémenter le producteur"](#). Si votre build génère des erreurs à un stade, examinez-les et corrigez-les avant de continuer.

Étapes suivantes

Implémenter le producteur

L'application dans le [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#) utilise le scénario réel de surveillance des opérations boursières. Les principes suivants expliquent brièvement comment ce scénario est mis en correspondance avec la structure de code de l'application producteur et celle qui est associée.

Reportez-vous au code source et vérifiez les informations suivantes.

StockTrade classe

Une opération boursière individuelle est représentée par une instance de la classe `StockTrade`. Cette instance contient des attributs tels que le symbole boursier, le prix, le nombre d'actions, le type de l'opération (achat ou vente) et un ID identifiant l'opération de manière unique. Cette classe est implémentée pour vous.

Enregistrement de flux

Un flux est une séquence d'enregistrements. Un enregistrement est une sérialisation d'une `StockTrade` instance au JSON format. Par exemple :

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator classe

`StockTradeGenerator` a une méthode appelée `getRandomTrade()` qui renvoie une nouvelle opération boursière générée de façon aléatoire chaque fois qu'elle est appelée. Cette classe est implémentée pour vous.

StockTradesWriter classe

La méthode `main` de l'application producteur, `StockTradesWriter` extrait en permanence une opération boursière aléatoire et l'envoie à Kinesis Data Streams en effectuant les tâches suivantes :

1. Elle lit le nom du flux et le nom de la région comme entrée.
2. Elle crée un `AmazonKinesisClientBuilder`.
3. Elle utilise le générateur client pour définir la région, les informations d'identification et la configuration du client.
4. Elle génère un client `AmazonKinesis` à l'aide du générateur client.
5. Elle vérifie que le flux existe et qu'il est actif (sinon il se ferme et génère une erreur).
6. Dans une boucle continue, elle appelle la méthode `StockTradeGenerator.getRandomTrade()`, puis la méthode `sendStockTrade` pour envoyer l'opération boursière au flux toutes les 100 millisecondes.

La méthode `sendStockTrade` de la classe `StockTradesWriter` comprend le code suivant :

```
private static void sendStockTrade(StockTrade trade, AmazonKinesis kinesisClient,
String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
```

```
// The bytes could be null if there is an issue with the JSON serialization by
the Jackson JSON library.
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
}

LOG.info("Putting trade: " + trade.toString());
PutRecordRequest putRecord = new PutRecordRequest();
putRecord.setStreamName(streamName);
// We use the ticker symbol as the partition key, explained in the Supplemental
Information section below.
putRecord.setPartitionKey(trade.getTickerSymbol());
putRecord.setData(ByteBuffer.wrap(bytes));

try {
    kinesisClient.putRecord(putRecord);
} catch (AmazonClientException ex) {
    LOG.warn("Error sending record to Amazon Kinesis.", ex);
}
}
```

Reportez-vous à la ventilation de code suivante :

- Le `PutRecord` API attend un tableau d'octets, et vous devez le convertir `trade` au JSON format. La seule ligne de code suivante effectue cette opération :

```
byte[] bytes = trade.toJsonAsBytes();
```

- Avant de pouvoir envoyer l'opération boursière, vous créez une nouvelle instance `PutRecordRequest` (appelée `putRecord` dans le cas suivant) :

```
PutRecordRequest putRecord = new PutRecordRequest();
```

Chaque appel de `PutRecord` requiert le nom du flux, la clé de partition et le blob de données. Le code suivant remplit ces champs dans l'objet `putRecord` à l'aide de ses méthodes `setXxxx()` :

```
putRecord.setStreamName(streamName);
putRecord.setPartitionKey(trade.getTickerSymbol());
putRecord.setData(ByteBuffer.wrap(bytes));
```

L'exemple utilise un symbole boursier comme clé de partition qui mappe l'enregistrement à une partition spécifique. En pratique, vous devez avoir des centaines ou des milliers de clés de partition par partition afin de répartir les enregistrements de façon égale dans votre flux. Pour plus d'informations sur la façon d'ajouter des données à un flux, consultez la page [Ajouter des données à un flux](#).

Maintenant `putRecord` est prêt à envoyer au client (l'opération `put`) :

```
kinesisClient.putRecord(putRecord);
```

- La vérification des erreurs et leur consignation sont toujours très utiles. Le code suivant consigne les conditions d'erreur :

```
if (bytes == null) {  
    LOG.warn("Could not get JSON bytes for stock trade");  
    return;  
}
```

Ajoutez le bloc `try/catch` autour de l'opération `put` :

```
try {  
    kinesisClient.putRecord(putRecord);  
} catch (AmazonClientException ex) {  
    LOG.warn("Error sending record to Amazon Kinesis.", ex);  
}
```

En effet, une opération `put` de Kinesis Data Streams peut échouer en raison d'une erreur de réseau ou parce que le flux atteint ses limites de débit et est limité. Nous vous recommandons d'examiner attentivement votre politique de nouvelle tentative pour les `put` opérations afin d'éviter toute perte de données, par exemple en utilisant une nouvelle tentative.

- La journalisation d'état est très utile, mais elle est facultative :

```
LOG.info("Putting trade: " + trade.toString());
```

Le producteur présenté ici utilise la fonctionnalité `PutRecord` d'enregistrement unique de Kinesis Data API Streams. En pratique, si une application producteur génère de nombreux enregistrements, il est souvent plus efficace d'utiliser la fonctionnalité `PutRecords`

de `PutRecords` et d'envoyer les lots d'enregistrements en même temps. Pour de plus amples informations, veuillez consulter [Ajouter des données à un flux](#).

Pour exécuter l'application producteur

1. Vérifiez que la clé d'accès et la paire de clés secrètes récupérées précédemment (lors de la création de IAM l'utilisateur) sont enregistrées dans le fichier `~/ .aws/credentials`.
2. Exécutez la classe `StockTradeWriter` avec les arguments suivants :

```
StockTradeStream us-west-2
```

Si vous avez créé votre flux dans une région autre que `us-west-2`, vous devez spécifier cette région ici.

Vous devez voir des résultats similaires à ce qui suit :

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
  sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Votre flux d'opérations boursières est maintenant en cours d'ingestion par Kinesis Data Streams.

Étapes suivantes

[Mettre en œuvre le consommateur](#)

Mettre en œuvre le consommateur

L'application consommateur dans [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#) traite en continue le flux d'opérations boursières que vous avez créé dans . Elle

gène ensuite les actions les plus populaires achetées et vendues toutes les minutes. L'application est construite sur la base de la bibliothèque cliente Kinesis (KCL), qui effectue une grande partie du travail fastidieux propre aux applications grand public. Pour de plus amples informations, veuillez consulter [Développez des KCL consommateurs 1.x](#).

Reportez-vous au code source et vérifiez les informations suivantes.

StockTradesProcessor classe

Classe principale de l'application consommateur, qui est fournie pour vous et effectue les tâches suivantes :

- Lit les noms d'application, de flux et de région transmis comme arguments.
- Lit les informations d'identification à partir de `~/.aws/credentials`.
- Crée une instance `RecordProcessorFactory` qui sert les instances de `RecordProcessor`, implémentée par une instance `StockTradeRecordProcessor`.
- Crée un KCL worker avec l'`RecordProcessorFactory` instance et une configuration standard comprenant le nom du flux, les informations d'identification et le nom de l'application.
- L'application de travail crée un nouveau thread pour chaque partition (affectée à cette instance de consommateur), qui fonctionne en boucle pour lire des enregistrements à partir de Kinesis Data Streams. Elle appelle alors l'instance `RecordProcessor` pour traiter chaque lot d'enregistrements reçu.

StockTradeRecordProcessor classe

Implémentation de l'instance `RecordProcessor`, qui implémente à son tour trois méthodes requises : `initialize`, `processRecords` et `shutdown`.

Comme leur nom l'indique, les méthodes `initialize` et `shutdown` sont utilisées par la Kinesis Client Library pour informer le processeur d'enregistrements du moment où il doit être prêt à recevoir des enregistrements et où il doit s'attendre à arrêter de recevoir des enregistrements afin d'effectuer des tâches d'installation et de mise hors service propres à l'application. Le code correspondant vous est fourni. Le traitement principal se déroule dans la méthode `processRecords`, qui utilise à son tour `processRecord` pour chaque enregistrement. La méthode `processRecord` est fournie la plupart du temps sous forme de squelette de code vide à implémenter à l'étape suivante qui fournit des explications détaillées.

Notez également l'implémentation des méthodes d'assistance pour `processRecord` : `reportStats` et `resetStats`, qui sont vides dans le code source d'origine.

La méthode `processRecords` est implémentée pour vous et effectue les opérations suivantes :

- Pour chaque enregistrement passé, il appelle `processRecord` dessus.
- Si au moins 1 minute s'est écoulée depuis le dernier rapport, il appelle `reportStats()`, qui imprime les dernières statistiques, puis de `resetStats()`, qui efface les statistiques afin que l'intervalle suivant n'inclut que les nouveaux enregistrements.
- Règle l'heure du rapport suivant.
- Si au moins 1 minute s'est écoulée depuis le dernier point de contrôle, il appelle `checkpoint()`.
- Règle l'heure du point de contrôle suivante.

Cette méthode utilise des intervalles de 60 secondes pour la fréquence de création de rapports et de point de contrôle. Pour en savoir plus sur les points de contrôle, consultez [Informations supplémentaires sur le consommateur](#).

StockStats classe

Cette classe prend en charge la conservation des données et le suivi des statistiques dans le temps pour les actions les plus populaires. Ce code est fourni pour vous et contient les méthodes suivantes :

- `addStockTrade(StockTrade)` : Injecte le `StockTrade` donné dans les statistiques en cours d'exécution.
- `toString()` : renvoie les statistiques dans une chaîne formatée.

Cette classe suit les actions les plus populaires en comptabilisant le nombre total de transactions pour chaque action et le nombre maximum. Elle met à jour ces chiffres chaque fois qu'une opération boursière arrive.

Ajoutez du code aux méthodes de la classe `StockTradeRecordProcessor`, comme l'illustrent les étapes suivantes.

Pour implémenter l'application consommateur

1. Implémentez la méthode `processRecord` en instanciant un objet `StockTrade` correctement dimensionné et en lui ajoutant les données d'enregistrement, puis en consignant un avertissement en cas de problème.

```
StockTrade trade = StockTrade.fromJsonAsBytes(record.getData().array());
```

```

if (trade == null) {
    LOG.warn("Skipping record. Unable to parse record into StockTrade. Partition
    Key: " + record.getPartitionKey());
    return;
}
stockStats.addStockTrade(trade);

```

2. Implémentez une méthode `reportStats` simple. N'hésitez pas à modifier le format de sortie selon vos préférences.

```

System.out.println("***** Shard " + kinesisShardId + " stats for last 1 minute
*****\n" +
                 stockStats + "\n" +
                 "*****\n");

```

3. Pour finir, implémentez la méthode `resetStats`, qui crée une nouvelle instance `stockStats`.

```

stockStats = new StockStats();

```

Pour exécuter l'application consommateur

1. Exécutez l'application producteur que vous avez écrite dans `producer` pour injecter des enregistrements d'opérations boursières simulées dans votre flux.
2. Vérifiez que la clé d'accès et la paire de clés secrètes récupérées précédemment (lors de la création de IAM l'utilisateur) sont enregistrées dans le fichier `~/.aws/credentials`.
3. Exécutez la classe `StockTradesProcessor` avec les arguments suivants :

```

StockTradesProcessor StockTradeStream us-west-2

```

Notez que, si vous avez créé votre flux dans une région autre que `us-west-2`, vous devez spécifier cette région ici.

Au bout d'une minute, vous devez voir une sortie similaire à la suivante, actualisée toutes les minutes :

```

***** Shard shardId-000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.

```

Informations supplémentaires sur le consommateur

Si vous connaissez bien les avantages de la Kinesis Client Library, présentés dans [Développez des KCL consommateurs 1.x](#) et ailleurs, vous vous demandez peut-être pourquoi vous devez les utiliser ici. Bien que vous n'utilisiez qu'un seul flux de partition et une seule instance de consommateur pour le traiter, il est toujours plus facile d'implémenter le consommateur à l'aide du KCL. Comparez les étapes d'implémentation du code dans l'application producteur à celles de l'application consommateur pour voir combien il est plus facile d'implémenter une application consommateur. Cela est dû en grande partie aux services qu'ils KCL fournissent.

Dans cette application, vous vous concentrez sur l'implémentation d'une classe de processeur d'enregistrements de processeur qui peut traiter des enregistrements individuels. Vous n'avez pas à vous soucier de la manière dont les enregistrements sont extraits de Kinesis Data Streams ; KCL le système récupère les enregistrements et appelle le processeur d'enregistrements chaque fois que de nouveaux enregistrements sont disponibles. En outre, vous n'avez pas à vous soucier du nombre d'instances de partitions et de consommateurs qui existent. Si le flux est mis à l'échelle, vous n'avez pas à réécrire votre application afin de gérer plus d'une partition ou une instance de consommateurs.

Le terme point de contrôle signifie enregistrer le point dans le flux jusqu'aux enregistrements de données qui ont été consommés et traités jusqu'à présent. Si l'application plante, le flux est lu à partir de ce point et non depuis le début du flux. L'objet de contrôle, les différents modèles de conception et les meilleures pratiques pour à ce sujet sont exclus de ce chapitre. Cependant, il est probable que vous rencontriez ces éléments dans les environnements de production.

Comme vous l'avez appris dans les `put` opérations dans les Kinesis Data API Streams utilisent une clé de partition comme entrée. Kinesis Data Streams utilise une clé de partition comme mécanisme pour fractionner les enregistrements sur plusieurs partitions (lorsqu'il y a plusieurs partitions dans le flux). La même clé de partition achemine toujours vers la même partition. Cela permet à l'application consommateur qui traite une partition spécifique d'être conçue en supposant que les enregistrements ayant la même clé de partition ne sont envoyés qu'à cette application consommateur, et qu'aucun enregistrement ayant cette même clé de partition ne parvient à une autre application consommateur. Par conséquent, une application de travail d'une application consommateur peut regrouper tous les enregistrements ayant la même clé de partition sans se soucier de savoir s'il manque des données nécessaires.

Dans cette application, le traitement des enregistrements par le consommateur n'est pas intensif. Vous pouvez donc utiliser une seule partition et effectuer le traitement dans le même fil que le KCL

fil. Toutefois, il convient en pratique de prendre en considération la mise à l'échelle du nombre de partitions. Dans certains cas, vous pouvez basculer le traitement vers un thread différent ou utiliser un pool de threads si le traitement des enregistrements est prévu être intensif.⁰ De cette façon, ils KCL peuvent récupérer de nouveaux enregistrements plus rapidement tandis que les autres threads peuvent traiter les enregistrements en parallèle. La conception multithread n'est pas banale et doit être abordée à l'aide de techniques avancées. L'augmentation du nombre de partitions est donc généralement le moyen le plus efficace d'augmenter le nombre de partitions.

Étapes suivantes

[\(Facultatif\) Étendre le consommateur](#)

(Facultatif) Étendre le consommateur

L'application présentée dans le [Tutoriel : Traitez les données boursières en temps réel à l'aide KPL de et KCL 1.x](#) suffit peut-être déjà à vos objectifs. Cette section facultative montre comment étendre le code de l'application consommateur pour obtenir un scénario un peu plus complexe.

Si vous voulez connaître les ordres de vente les plus importants toutes les minutes, vous pouvez modifier la classe `StockStats` à trois emplacements pour accueillir cette nouvelle priorité.

Pour étendre l'application consommateur

1. Ajoutez de nouvelles variables d'instance :

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Ajoutez le code suivant à `addStockTrade`:

```
if (type == TradeType.SELL) {
    if (largestSellOrderStock == null || trade.getQuantity() >
        largestSellOrderQuantity) {
        largestSellOrderStock = trade.getTickerSymbol();
        largestSellOrderQuantity = trade.getQuantity();
    }
}
```

3. Modifiez la méthode `toString` pour imprimer les informations supplémentaires :

```
public String toString() {
    return String.format(
        "Most popular stock being bought: %s, %d buys.%n" +
        "Most popular stock being sold: %s, %d sells.%n" +
        "Largest sell order: %d shares of %s.",
        getMostPopularStock(TradeType.BUY),
        getMostPopularStockCount(TradeType.BUY),
        getMostPopularStock(TradeType.SELL),
        getMostPopularStockCount(TradeType.SELL),
        largestSellOrderQuantity, largestSellOrderStock);
}
```

Si vous exécutez l'application consommateur maintenant (n'oubliez pas d'exécuter également l'application producteur), vous devez voir une sortie similaire à la suivante :

```
***** Shard shardId-000000000001 stats for last 1 minute *****
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
*****
```

Étapes suivantes

[Nettoyage des ressources](#)

Nettoyage des ressources

Étant donné que vous payez pour utiliser le flux de données Kinesis, veillez à le supprimer, ainsi que la table Amazon DynamoDB correspondante, lorsque vous avez terminé. Des frais nominaux s'appliquent sur un flux actif même lorsque vous n'envoyez ni ne recevez d'enregistrements. Cela provient du fait qu'un flux actif utilise des ressources en écoutant en permanence les demandes et enregistrements entrants pour obtenir des enregistrements.

Pour supprimer le flux et la table

1. Arrêtez les applications producteur et consommateur qui sont toujours en cours d'exécution.
2. [Ouvrez la console Kinesis à l'adresse /kinesis. https://console.aws.amazon.com](https://console.aws.amazon.com/kinesis)
3. Choisissez le flux que vous avez créé pour cette application (StockTradeStream).

4. Choisissez Supprimer flux.
5. Ouvrez la console DynamoDB à l'adresse. <https://console.aws.amazon.com/dynamodb/>
6. Supprimez la table StockTradesProcessor.

Récapitulatif

Le traitement d'une grande quantité de données en temps quasi réel ne nécessite pas l'écriture de code compliqué ni le développement d'une énorme infrastructure. C'est aussi simple que la logique d'écriture pour traiter une petite quantité de données (comme l'écriture `processRecord(Record)`), tout en utilisant Kinesis Data Streams pour l'adapter à une grande quantité de données en streaming. Vous n'avez pas à vous soucier de la mise à l'échelle de votre traitement, car Kinesis Data Streams s'en charge pour vous. Tout ce que vous avez à faire est d'envoyer vos enregistrements en flux continu à Kinesis Data Streams et d'écrire la logique pour traiter chaque nouvel enregistrement reçu.

Voici quelques améliorations potentielles pour cette application.

Regroupement à travers toutes les partitions

Actuellement, vous obtenez des statistiques résultant de l'agrégation des enregistrements de données reçues par un seul programme exécutant à partir d'une seule partition. (Une partition ne peut pas être traitée par plus d'une application de travail dans une seule application en même temps.) Bien entendu, lorsque vous mettez à l'échelle et que vous avez plusieurs partitions, vous pouvez regrouper toutes les partitions. Vous pouvez y procéder avec une architecture pipeline où la sortie de chaque application de travail alimente un autre flux comportant une seule partition, qui est traité par une application de travail qui regroupe les sorties de la première phase. Etant donné que les données de la première phase sont limitées (un échantillon par minute par partition), elles sont facilement gérées par une seule partition.

Mise à l'échelle du traitement

Lorsque le flux évolue vers plusieurs partitions (suite à l'envoi de données par plusieurs applications producteur), la façon de faire évoluer le traitement consiste à ajouter des applications de travail. Vous pouvez exécuter les travailleurs dans des EC2 instances Amazon et utiliser des groupes Auto Scaling.

Utilisation de connecteurs vers Amazon S3/DynamoDB/Amazon Redshift/Storm

Lorsqu'un flux est traité en continu, sa sortie peut être envoyée vers d'autres destinations. AWS fournit des [connecteurs](#) permettant d'intégrer Kinesis Data Streams à d' AWS autres services et outils tiers.

Étapes suivantes

- Pour plus d'informations sur l'utilisation des opérations Kinesis Data API Streams, [Développez les producteurs en utilisant Amazon Kinesis Data API Streams avec AWS SDK for Java](#) reportez-vous aux sections [Développez des consommateurs personnalisés avec un débit partagé à l'aide du AWS SDK for Java](#), [Création et gestion de flux de données Kinesis](#) et.
- Pour plus d'informations sur la Kinesis Client Library, consultez [Développez des KCL consommateurs 1.x](#).
- Pour plus d'informations sur la façon d'optimiser votre application, consultez la page [Optimisez les utilisateurs d'Amazon Kinesis Data Streams](#).

Tutoriel : Analyser les données boursières en temps réel à l'aide d'Amazon Managed Service pour Apache Flink

Pour ce tutoriel, le scénario implique d'ingérer des opérations boursières dans un flux de données et d'écrire une application [Amazon Managed Service for Apache Flink](#) simple qui effectue des calculs sur le flux. Vous apprendrez à envoyer un flux d'enregistrements vers Kinesis Data Streams et à implémenter une application qui consomme et traite les enregistrements en temps quasi réel.

Avec Amazon Managed Service pour Apache Flink, vous pouvez utiliser Java ou Scala pour traiter et analyser les données de streaming. Le service vous permet de créer et d'exécuter du code Java ou Scala sur des sources de streaming pour effectuer des analyses de séries chronologiques, alimenter des tableaux de bord en temps réel et créer des métriques en temps réel.

[Vous pouvez créer des applications Flink dans Managed Service for Apache Flink en utilisant des bibliothèques open-source basées sur Apache Flink](#). Apache Flink est un framework et un moteur populaires permettant de traiter des flux de données.

Important

Une fois que vous avez créé deux flux de données et une application, votre compte est soumis à des frais minimes pour l'utilisation de Kinesis Data Streams et du Managed Service for Apache Flink, car ils ne sont pas éligibles au niveau gratuit. AWS Lorsque vous aurez terminé d'utiliser cette application, supprimez vos AWS ressources pour ne plus être facturée.

Le code n'accède pas aux données boursières réelles, mais simule le flux des opérations boursières. Pour cela, il utilise un générateur de transactions boursières aléatoires. Si vous avez accès à un flux en temps réel d'opérations boursières, tirer des statistiques pratiques et adéquates de ce flux peut vous intéresser. Par exemple, vous pouvez effectuer une analyse par fenêtre glissante pour déterminer l'action la plus couramment achetée au cours des 5 dernières minutes. Vous souhaitez peut-être aussi recevoir une notification chaque fois qu'il y a un ordre de vente qui est trop grand (c'est-à-dire qui contient trop d'actions). Vous pouvez étendre le code illustré dans cette série pour fournir cette fonctionnalité.

Les exemples présentés utilisent la région USA Ouest (Oregon), mais ils fonctionnent dans toutes les [régions AWS qui prennent en charge Managed Service for Apache Flink](#).

Tâches

- [Conditions préalables pour terminer les exercices](#)
- [Configuration d'un AWS compte et création d'un utilisateur administrateur](#)
- [Configurez le AWS Command Line Interface \(AWS CLI\)](#)
- [Création et exécution d'un service géré pour l'application Apache Flink](#)

Conditions préalables pour terminer les exercices

Pour exécuter la procédure indiquée dans ce guide, vous devez disposer des éléments suivants :

- [Kit de développement Java](#) (JDK) version 8. Définissez la variable d'JAVA_HOME environnement pour qu'elle pointe vers l'emplacement de votre JDK installation.
- Nous vous recommandons d'utiliser un environnement de développement (par exemple [Eclipse Java Neon](#) ou [IntelliJ Idea](#)) pour développer et compiler votre application.
- [Client Git](#). Installez le client Git si vous ne l'avez pas déjà fait.
- [Apache Maven Compiler Plugin](#). Maven doit être installé dans votre chemin de travail. Pour tester votre installation Apache Maven, saisissez les informations suivantes :

```
$ mvn -version
```

Pour démarrer, accédez à [Configuration d'un AWS compte et création d'un utilisateur administrateur](#).

Configuration d'un AWS compte et création d'un utilisateur administrateur

Avant d'utiliser Amazon Managed Service pour Apache Flink pour la première fois, effectuez les tâches suivantes :

1. [Inscrivez-vous pour AWS](#)
2. [Création d'un utilisateur IAM](#)

Inscrivez-vous pour AWS

Lorsque vous vous inscrivez à Amazon Web Services (AWS), votre AWS compte est automatiquement ouvert pour tous les services AWS, y compris Amazon Managed Service pour Apache Flink. Seuls les services que vous utilisez vous sont facturés.

Grâce à Managed Service pour Apache Flink, vous ne payez que pour les ressources que vous utilisez. Si vous êtes un nouveau client AWS , vous pouvez commencer à utiliser Managed Service pour Apache Flink gratuitement. Pour plus d'informations, consultez la page sur l'[offre gratuite AWS](#).

Si vous avez déjà un AWS compte, passez à la tâche suivante. Si vous n'avez pas de compte AWS , suivez les étapes ci-dessous pour en créer un.

Pour créer un AWS compte

1. Ouvrez l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des AWS services et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Notez votre identifiant de AWS compte car vous en aurez besoin pour la prochaine tâche.

Création d'un utilisateur IAM

Les services tels qu'Amazon Managed Service pour Apache Flink nécessitent que vous fournissiez des informations d'identification lorsque vous y accédez. AWS Cela permet au service de déterminer si vous avez les autorisations nécessaires pour accéder aux ressources détenues par ce service. Vous devez saisir votre mot de passe. AWS Management Console

Vous pouvez créer des clés d'accès pour que votre AWS compte accède au AWS Command Line Interface (AWS CLI) ou API. Toutefois, nous vous déconseillons d'accéder à AWS l'aide des informations d'identification de votre AWS compte. Nous vous recommandons plutôt d'utiliser AWS Identity and Access Management (IAM). Créez un IAM utilisateur, ajoutez-le à un IAM groupe doté d'autorisations administratives, puis accordez des autorisations administratives à l'IAM utilisateur que vous avez créé. Vous pouvez ensuite y accéder AWS en utilisant un identifiant spécial URL et les informations d'identification de cet IAM utilisateur.

Si vous vous êtes inscrit AWS, mais que vous ne vous êtes pas encore créé d'IAM utilisateur, vous pouvez en créer un à l'aide de la IAM console.

Les exercices de mise en route de ce guide présument que l'utilisateur (`adminuser`) dispose d'autorisations d'administrateur. Suivez la procédure pour créer `adminuser` dans votre compte.

Pour créer un groupe pour les administrateurs

1. Connectez-vous à la IAM console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/iam/>.
2. Dans le volet de navigation, choisissez Groups, puis Create New Group.
3. Dans Nom du groupe, entrez un nom pour le groupe, par exemple **Administrators**, puis choisissez Étape suivante.
4. Dans la liste des politiques, cochez la case à côté de la AdministratorAccesstratégie. Vous pouvez utiliser le menu Filter et la zone Search pour filtrer la liste de stratégies.
5. Cliquez sur Étape suivante, puis sur Créer un groupe.

Votre nouveau groupe apparaît sous Nom du groupe.

Pour créer un IAM utilisateur pour vous-même, ajoutez-le au groupe des administrateurs et créez un mot de passe

1. Dans le panneau de navigation, choisissez Users (Utilisateurs), puis Add user (Ajouter un utilisateur).
2. Dans le champ Nom utilisateur, saisissez le nom d'un utilisateur.
3. Sélectionnez à la fois l'accès programmatique et l'accès à la console de gestion AWS .
4. Sélectionnez Next: Permissions (Étape suivante : autorisations).
5. Cochez la case en regard du groupe Administrators (Administrateurs). Choisissez ensuite Next: Review.
6. Choisissez Create user (Créer un utilisateur).

Pour vous connecter en tant que nouvel IAM utilisateur

1. Déconnectez-vous du AWS Management Console.
2. Utilisez le URL format suivant pour vous connecter à la console :

`https://aws_account_number.signin.aws.amazon.com/console/`

Le *aws_account_number* est l'identifiant de votre AWS compte sans tiret. Par exemple, si votre numéro de AWS compte est 1234-5678-9012, remplacez *aws_account_number* avec **123456789012**. Pour savoir comment trouver votre numéro de compte, consultez la section [Votre identifiant de AWS compte et son alias](#) dans le guide de IAM l'utilisateur.

3. Entrez le nom IAM d'utilisateur et le mot de passe que vous venez de créer. Lorsque vous êtes connecté, la barre de navigation s'affiche *your_user_name @ your_aws_account_id*.

Note

Si vous ne souhaitez pas que la page URL de connexion contienne votre identifiant de AWS compte, vous pouvez créer un alias de compte.

Pour créer ou supprimer un alias de compte

1. Ouvrez la IAM console à l'adresse <https://console.aws.amazon.com/iam/>.
2. Dans le volet de navigation, sélectionnez Dashboard (Tableau de bord).

3. Trouvez le lien de connexion de l'IAMutilisateur.
4. Pour créer l'alias, cliquez sur Personnaliser. Entrez le nom que vous souhaitez utiliser pour l'alias, puis choisissez Oui, créer.
5. Pour supprimer l'alias, cliquez sur Customize (Personnaliser), puis sur Yes, Delete (Oui, Supprimer). La connexion URL revient à l'aide de votre identifiant de AWS compte.

Pour vous connecter après avoir créé un alias de compte, utilisez ce qui suit URL :

`https://your_account_alias.signin.aws.amazon.com/console/`

Pour vérifier le lien de connexion des IAM utilisateurs de votre compte, ouvrez la IAM console et vérifiez le lien de connexion des IAM utilisateurs sur le tableau de bord.

Pour plus d'informations surIAM, consultez les rubriques suivantes :

- [AWS Identity and Access Management \(IAM\)](#)
- [Prise en main](#)
- [Guide de l'utilisateur IAM](#)

Étape suivante

[Configurez le AWS Command Line Interface \(AWS CLI\)](#)

Configurez le AWS Command Line Interface (AWS CLI)

Au cours de cette étape, vous allez télécharger et configurer le AWS CLI à utiliser avec Amazon Managed Service pour Apache Flink.

Note

Les exercices de mise en route de ce guide supposent que vous utilisez les informations d'identification d'administrateur (`adminuser`) de votre compte pour effectuer les opérations.

Note

Si vous l'avez déjà AWS CLI installé, vous devrez peut-être effectuer une mise à niveau pour bénéficier des dernières fonctionnalités. Pour plus d'informations, consultez la section

[Installation de l'interface de ligne de AWS commande](#) dans le guide de AWS Command Line Interface l'utilisateur. Pour vérifier la version du AWS CLI, exécutez la commande suivante :

```
aws --version
```

Les exercices présentés dans ce didacticiel nécessitent la AWS CLI version suivante ou une version ultérieure :

```
aws-cli/1.16.63
```

Pour configurer le AWS CLI

1. Téléchargez et configurez l'interface AWS CLI. Pour obtenir des instructions, consultez les rubriques suivantes dans le Guide de l'utilisateur de l'interface AWS Command Line Interface :
 - [Installation de AWS Command Line Interface](#)
 - [Configuration de l'interface AWS CLI](#) (français non garanti)
2. Ajoutez un profil nommé pour l'utilisateur administrateur dans le fichier de AWS CLI configuration. Vous utilisez ce profil lors de l'exécution des AWS CLI commandes. Pour plus d'informations sur les profils nommés, consultez la rubrique [Profils nommés](#) dans le Guide de l'utilisateur AWS Command Line Interface .

```
[profile adminuser]
aws_access_key_id = adminuser access key ID
aws_secret_access_key = adminuser secret access key
region = aws-region
```

Pour obtenir la liste des AWS régions disponibles, consultez la section [AWS Régions et points de terminaison](#) dans le Référence générale d'Amazon Web Services.

3. Vérifiez la configuration en saisissant la commande d'aide suivante à l'invite de commande :

```
aws help
```

Après avoir configuré un AWS compte AWS CLI, vous pouvez passer à l'exercice suivant, dans lequel vous configurez un exemple d'application et testez la end-to-end configuration.

Étape suivante

[Création et exécution d'un service géré pour l'application Apache Flink](#)

Création et exécution d'un service géré pour l'application Apache Flink

Dans cet exercice, vous allez créer une application de service géré pour Apache Flink avec des flux de données comme source et comme récepteur.

Cette section contient les étapes suivantes :

- [Créez deux flux de données Amazon Kinesis](#)
- [Écriture d'exemples d'enregistrements dans le flux d'entrée](#)
- [Téléchargez et examinez le code Java de streaming d'Apache Flink](#)
- [Compilez le code de l'application](#)
- [Téléchargez le code Java de streaming Apache Flink](#)
- [Création et exécution du service géré pour l'application Apache Flink](#)

Créez deux flux de données Amazon Kinesis

Avant de créer un Amazon Managed Service pour Apache Flink dans le cadre de cet exercice, créez deux flux de données Kinesis `ExampleInputStream` (`ExampleOutputStream`). Votre application utilise ces flux pour les flux source et de destination de l'application.

Vous pouvez créer ces flux à l'aide de la console Amazon Kinesis ou de la commande AWS CLI suivante. Pour de plus amples informations, veuillez consulter [Création et mise à jour des flux de données](#).

Pour créer les flux de données (AWS CLI)

1. Pour créer le premier flux (`ExampleInputStream`), utilisez la commande Amazon Kinesis `create-stream` AWS CLI suivante.

```
$ aws kinesis create-stream \  
--stream-name ExampleInputStream \  
--shard-count 1 \  
--region us-west-2 \  
--profile adminuser
```

2. Pour créer le second flux utilisé par l'application pour écrire la sortie, exécutez la même commande en remplaçant le nom du flux par `ExampleOutputStream`.

```
$ aws kinesis create-stream \  
--stream-name ExampleOutputStream \  
--shard-count 1 \  
--region us-west-2 \  
--profile adminuser
```

Écriture d'exemples d'enregistrements dans le flux d'entrée

Dans cette section, vous utilisez un script Python pour écrire les exemples d'enregistrements dans le flux pour que l'application les traite.

Note

Cette section nécessite le kit [AWS SDK for Python \(Boto\)](#).

1. Créez un fichier nommé `stock.py` avec le contenu suivant :

```
import datetime  
import json  
import random  
import boto3  
  
STREAM_NAME = "ExampleInputStream"  
  
def get_data():  
    return {  
        "EVENT_TIME": datetime.datetime.now().isoformat(),  
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),  
        "PRICE": round(random.random() * 100, 2),  
    }  
  
def generate(stream_name, kinesis_client):  
    while True:  
        data = get_data()
```



```
print(data)
kinesis_client.put_record(
    StreamName=stream_name, Data=json.dumps(data),
    PartitionKey="partitionkey"
)

if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. Plus loin dans ce didacticiel, vous exécutez le script `stock.py` pour envoyer des données à l'application.

```
$ python stock.py
```

Téléchargez et examinez le code Java de streaming d'Apache Flink

Le code d'application Java pour ces exemples est disponible sur GitHub. Pour télécharger le code d'application, procédez comme suit :

1. Cloner le référentiel distant à l'aide de la commande suivante :

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-examples.git
```

2. Accédez au répertoire `GettingStarted`.

Le code d'application est situé dans les fichiers `CustomSinkStreamingJob.java` et `CloudWatchLogSink.java`. Notez les informations suivantes à propos du code d'application :

- L'application utilise une source Kinesis pour lire à partir du flux source. L'extrait de code suivant crée le récepteur Kinesis :

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
    new SimpleStringSchema(), inputProperties));
```

Compilez le code de l'application

Dans cette section, vous allez utiliser le compilateur Apache Maven pour créer le code Java pour l'application. Pour plus d'informations sur l'installation d'Apache Maven et du kit de développement Java (JDK), consultez [Conditions préalables pour terminer les exercices](#).

Votre application Java nécessite les composants suivants :

- Un fichier de [modèle d'objet du projet \(pom.xml\)](#). Ce fichier contient des informations sur la configuration et les dépendances de l'application, y compris le service Amazon Managed Service pour les bibliothèques Apache Flink.
- Une méthode `main` qui contient la logique de l'application.

Note

Pour utiliser le connecteur Kinesis pour l'application suivante, vous devez télécharger le code source du connecteur et le créer comme décrit dans la documentation [Apache Flink](#).

Pour créer et compiler le code d'application

1. Créez une application Java/Maven dans votre environnement de développement. Pour de plus amples informations sur la création d'une application, veuillez consulter la documentation relative à votre environnement de développement :
 - [Création de votre premier projet Java \(Eclipse Java Neon\)](#) (français non garanti)
 - [Création, exécution et mise en package de votre première application Java \(IntelliJ Idea\)](#) (français non garanti)
2. Utilisez le code suivant dans un fichier nommé `StreamingJob.java`.

```
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
```

```
import
  org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    private static DataStream<String>
createSourceFromStaticConfig(StreamExecutionEnvironment env) {
    Properties inputProperties = new Properties();
    inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

    inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
"LATEST");

    return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(), inputProperties));
}

    private static DataStream<String>
createSourceFromApplicationProperties(StreamExecutionEnvironment env)
throws IOException {
    Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
    return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(),
        applicationProperties.get("ConsumerConfigProperties")));
}

    private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
    Properties outputProperties = new Properties();
    outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
    outputProperties.setProperty("AggregationEnabled", "false");

    FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(), outputProperties);
    sink.setDefaultStream(outputStreamName);
    sink.setDefaultPartition("0");
}
```

```
        return sink;
    }

    private static FlinkKinesisProducer<String>
createSinkFromApplicationProperties() throws IOException {
    Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
    FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
SimpleStringSchema(),
        applicationProperties.get("ProducerConfigProperties"));

    sink.setDefaultStream(outputStreamName);
    sink.setDefaultPartition("0");
    return sink;
}

public static void main(String[] args) throws Exception {
    // set up the streaming execution environment
    final StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();

    /*
     * if you would like to use runtime configuration properties, uncomment the
     * lines below
     * DataStream<String> input = createSourceFromApplicationProperties(env);
     */

    DataStream<String> input = createSourceFromStaticConfig(env);

    /*
     * if you would like to use runtime configuration properties, uncomment the
     * lines below
     * input.addSink(createSinkFromApplicationProperties())
     */

    input.addSink(createSinkFromStaticConfig());

    env.execute("Flink Streaming Java API Skeleton");
}
}
```

Notez les informations suivantes à propos de l'exemple de code précédent :

- Ce fichier contient la méthode `main` qui définit la fonctionnalité de l'application.
 - Votre application crée les connecteurs source et récepteur pour accéder aux ressources externes à l'aide d'un objet `StreamExecutionEnvironment`.
 - L'application crée les connecteurs source et récepteur à l'aide de propriétés statiques. Pour utiliser les propriétés de l'application dynamique, utilisez les méthodes `createSourceFromApplicationProperties` et `createSinkFromApplicationProperties` pour créer les connecteurs. Ces méthodes lisent les propriétés de l'application pour configurer les connecteurs.
3. Pour utiliser le code de votre application, vous devez le compiler et le regrouper dans un JAR fichier. Vous pouvez compiler et intégrer votre code de deux manières :
 - À l'aide de l'outil de ligne de commande Maven. Créez votre JAR fichier en exécutant la commande suivante dans le répertoire qui contient le `pom.xml` fichier :

```
mvn package
```

- À l'aide de votre environnement de développement. Consultez la documentation de votre environnement de développement pour plus de détails.

Vous pouvez soit télécharger votre package sous forme de JAR fichier, soit compresser votre package et le télécharger sous forme de ZIP fichier. Si vous créez votre application à l'aide du AWS CLI, vous spécifiez le type de contenu de votre code (JARouZIP).

4. En cas d'erreur lors de la compilation, vérifiez que votre variable d'environnement `JAVA_HOME` est correctement définie.

Si la compilation de l'application aboutit, le fichier suivant est créé :

```
target/java-getting-started-1.0.jar
```

Téléchargez le code Java de streaming Apache Flink

Dans cette section, vous allez créer un compartiment Amazon Simple Storage Service (Amazon S3) et charger votre code d'application.

Pour charger le code d'application

1. Ouvrez la console Amazon S3 à l'adresse <https://console.aws.amazon.com/s3/>.

2. Choisissez Créer un compartiment.
3. Saisissez **ka-app-code-*<username>*** dans le champ Nom du compartiment. Ajoutez un suffixe au nom du compartiment, par exemple votre nom d'utilisateur, pour qu'il soit unique. Choisissez Suivant.
4. À l'étape Configurer les options, conservez les paramètres, puis choisissez Suivant.
5. À l'étape Définir des autorisations, conservez les paramètres, puis choisissez Suivant.
6. Choisissez Créer un compartiment.
7. Dans la console Amazon S3, choisissez le **ka-app-code-*<username>*bucket**, puis choisissez Upload.
8. À l'étape Sélectionner les fichiers, choisissez Ajouter des fichiers. Accédez au fichier `java-getting-started-1.0.jar` que vous avez créé à l'étape précédente. Choisissez Suivant.
9. À l'étape Définir des autorisations, conservez les paramètres. Choisissez Suivant.
10. À l'étape Définir les propriétés, conservez les paramètres. Sélectionnez Charger.

Votre code d'application est désormais stocké dans un compartiment Amazon S3 auquel votre application peut accéder.

Création et exécution du service géré pour l'application Apache Flink

Vous pouvez créer et exécuter une application de service géré pour Apache Flink à l'aide de la console ou de l'interface AWS CLI.

Note

Lorsque vous créez l'application à l'aide de la console, vos ressources AWS Identity and Access Management (IAM) et Amazon CloudWatch Logs sont créées pour vous. Lorsque vous créez l'application à l'aide du AWS CLI, vous créez ces ressources séparément.

Rubriques

- [Création et exécution de l'application \(console\)](#)
- [Créez et exécutez l'application \(AWS CLI\)](#)

Création et exécution de l'application (console)

Suivez ces étapes pour créer, configurer, mettre à jour et exécuter l'application à l'aide de la console.

Pour créer l'application

1. [Ouvrez la console Kinesis à l'adresse /kinesis. https://console.aws.amazon.com](https://console.aws.amazon.com/kinesis)
2. Dans le tableau de bord Amazon Kinesis, choisissez Création d'une application d'analyse.
3. Sur la page Kinesis Analytics - Créer une application, fournissez les détails de l'application comme suit :
 - Pour Nom de l'application, saisissez **MyApplication**.
 - Pour Description, saisissez **My java test app**.
 - Pour Runtime (Exécution), choisissez Apache Flink 1.6.
4. Pour les autorisations d'accès, choisissez Create/update IAM role **kinesis-analytics-MyApplication-us-west-2**.
5. Choisissez Créer une application.

Note

Lorsque vous créez une application Amazon Managed Service pour Apache Flink à l'aide de la console, vous avez la possibilité de créer un IAM rôle et une politique pour votre application. Votre application utilise ce rôle et cette politique pour accéder à ses ressources dépendantes. Ces IAM ressources sont nommées à l'aide du nom de votre application et de votre région, comme suit :

- Stratégie : `kinesis-analytics-service-MyApplication-us-west-2`
- Rôle : `kinesis-analytics-MyApplication-us-west-2`

Modifier la IAM politique

Modifiez la IAM politique pour ajouter des autorisations d'accès aux flux de données Kinesis.

1. Ouvrez la IAM console à l'adresse <https://console.aws.amazon.com/iam/>.
2. Choisissez Stratégies. Choisissez la politique **kinesis-analytics-service-MyApplication-us-west-2** créée pour vous par la console dans la section précédente.
3. Sur la page Récapitulatif, choisissez Modifier la politique. Choisissez l'JSONonglet.
4. Ajoutez la section mise en surbrillance dans l'exemple de stratégie suivant à la politique. Remplacez le compte d'exemple IDs (`012345678901`) avec votre identifiant de compte.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadCode",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
      ]
    },
    {
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:*"
      ]
    },
    {
      "Sid": "ListCloudwatchLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
      ]
    },
    {
      "Sid": "PutCloudwatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
```



```

        "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
    ]
  },
  {
    "Sid": "ReadInputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
  },
  {
    "Sid": "WriteOutputStream",
    "Effect": "Allow",
    "Action": "kinesis:*",
    "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
  }
]
}

```

Configuration de l'application

1. Sur la MyApplicationpage, choisissez Configurer.
2. Sur la page Configurer l'application, indiquez l'emplacement du code:
 - Pour le compartiment Amazon S3, saisissez **ka-app-code-*<username>***.
 - Pour le chemin de l'objet Amazon S3, saisissez **java-getting-started-1.0.jar**.
3. Sous Accès aux ressources de l'application, pour les autorisations d'accès, choisissez Create/update IAM role **kinesis-analytics-MyApplication-us-west-2**.
4. Sous Propriétés, pour ID de groupe, saisissez **ProducerConfigProperties**.
5. Entrez les valeurs et propriétés d'application suivantes :

Clé	Valeur
flink.inputstream.initpos	LATEST
aws:region	us-west-2

Clé	Valeur
AggregationEnabled	false

6. Sous Surveillance, assurez-vous que Surveillance du niveau des métriques est défini sur Application.
7. Pour la CloudWatch journalisation, cochez la case Activer.
8. Choisissez Mettre à jour.

Note

Lorsque vous choisissez d'activer la CloudWatch journalisation, le service géré pour Apache Flink crée un groupe de journaux et un flux de journaux pour vous. Les noms de ces ressources sont les suivants :

- Groupe de journaux : `/aws/kinesis-analytics/MyApplication`
- Flux de journaux : `kinesis-analytics-log-stream`

Exécutez l'application

1. Sur la MyApplicationpage, choisissez Exécuter. Confirmez l'action.
2. Lorsque l'application est en cours d'exécution, actualisez la page. La console affiche le graphique de l'application.

Arrêtez l'application

Sur la MyApplicationpage, choisissez Stop. Confirmez l'action.

Mise à jour de l'application

À l'aide de la console, vous pouvez mettre à jour les paramètres de l'application, tels que les propriétés de l'application, les paramètres de surveillance, ainsi que l'emplacement ou le nom de fichier de l'applicationJAR. Vous pouvez également recharger l'application JAR depuis le compartiment Amazon S3 si vous devez mettre à jour le code de l'application.

Sur la MyApplicationpage, choisissez Configurer. Mettez à jour les paramètres de l'application, puis choisissez Mettre à jour.

Créez et exécutez l'application (AWS CLI)

Dans cette section, vous allez utiliser le AWS CLI pour créer et exécuter l'application Managed Service for Apache Flink. Le service géré pour Apache Flink utilise la `kinesisanalyticsv2` AWS CLI commande pour créer et interagir avec le service géré pour les applications Apache Flink.

Créer une stratégie d'autorisations

Vous commencez par créer une stratégie d'autorisations avec deux instructions : une qui accorde des autorisations pour l'action `read` sur le flux source et une autre qui accorde des autorisations pour les actions `write` sur le flux récepteur. Vous associez ensuite la politique à un IAM rôle (que vous créez dans la section suivante). Ainsi, lorsque le service géré pour Apache Flink assume le rôle, le service dispose des autorisations nécessaires pour lire à partir du flux source et écrire dans le flux récepteur.

Utilisez le code suivant pour créer la politique d'autorisations

`KAReadSourceStreamWriteSinkStream`. Remplacez *username* par le nom d'utilisateur que vous avez utilisé pour créer le compartiment Amazon S3 pour stocker le code d'application. Remplacez l'ID de compte dans Amazon Resource Names (ARNs) (*012345678901*) par votre identifiant de compte.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "S3",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:::ka-app-code-username",
        "arn:aws:s3:::ka-app-code-username/*"
      ]
    },
    {
      "Sid": "ReadInputStream",
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
    }
  ]
}
```

```
        "Sid": "WriteOutputStream",
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
    }
  ]
}
```

Pour step-by-step obtenir des instructions sur la création d'une politique d'autorisations, voir [Tutoriel : créer et joindre votre première politique gérée par le client](#) dans le guide de IAM l'utilisateur.

Note

Pour accéder à d'autres AWS services, vous pouvez utiliser le AWS SDK for Java. Le service géré pour Apache Flink définit automatiquement les informations d'identification requises par le SDK en fonction du IAM rôle d'exécution du service associé à votre application. Aucune étape supplémentaire n'est nécessaire.

Créez un rôle IAM.

Dans cette section, vous allez créer un IAM rôle que Managed Service for Apache Flink peut assumer pour lire un flux source et écrire dans le flux récepteur.

Le service géré pour Apache Flink ne peut pas accéder à votre flux sans autorisation. Vous accordez ces autorisations via un IAM rôle. Deux politiques sont associées à chaque IAM rôle. La politique d'approbation accorde au service géré pour Apache Flink l'autorisation d'assumer le rôle, et la politique d'autorisation détermine ce que le service géré pour Apache Flink peut faire après avoir assumé le rôle.

Vous attachez la politique d'autorisations que vous avez créée dans la section précédente à ce rôle.

Pour créer un rôle IAM


1. Ouvrez la IAM console à l'adresse <https://console.aws.amazon.com/iam/>.
2. Dans le volet de navigation, choisissez Rôles, puis Créer un rôle.
3. Sous Sélectionner le type d'identité approuvée, choisissez Service AWS . Sous Choisir le service qui utilisera ce rôle, choisissez EC2. Sous Sélectionner votre cas d'utilisation, choisissez Kinesis Analytics.

Sélectionnez Next: Permissions (Étape suivante : autorisations).

4. Dans la page Attacher des stratégies d'autorisations, choisissez Suivant : vérification. Vous attachez des stratégies d'autorisations après avoir créé le rôle.
5. Sur la page Créer un rôle, saisissez **KA-stream-rw-role** pour le Nom du rôle. Sélectionnez Créer un rôle.

Vous avez maintenant créé un nouveau IAM rôle appelé `KA-stream-rw-role`. Ensuite, vous mettez à jour les stratégies d'approbation et d'autorisation pour le rôle.

6. Attachez la politique d'autorisation au rôle.

 Note

Dans le cadre de cet exercice, le service géré pour Apache Flink assume ce rôle à la fois pour la lecture des données à partir d'un flux de données Kinesis (source) et pour l'écriture des résultats dans un autre flux de données Kinesis. Vous attachez donc la politique que vous avez créée à l'étape précédente, [the section called "Créer une stratégie d'autorisations"](#).

- a. Sur la page Récapitulatif, choisissez l'onglet Autorisations.
- b. Choisissez Attacher des stratégies.
- c. Dans la zone de recherche, saisissez **KAReadSourceStreamWriteSinkStream** (la politique que vous avez créée dans la section précédente).
- d. Choisissez la `KAReadInputStreamWriteOutputStream` politique, puis choisissez Attacher la politique.

Vous avez maintenant créé le rôle d'exécution de service que votre application utilise pour accéder aux ressources. Prenez note ARN du nouveau rôle.

Pour step-by-step obtenir des instructions sur la création d'un rôle, consultez [la section Création d'un IAM rôle \(console\)](#) dans le guide de IAM l'utilisateur.

Création de l'application de service géré pour Apache Flink

1. Enregistrez le JSON code suivant dans un fichier nommé `create_request.json`. Remplacez le rôle ARN d'exemple par le rôle que vous avez créé précédemment. ARN Remplacez le ARN

suffixe du bucket (*username*) par le suffixe que vous avez choisi dans la section précédente. Remplacez l'exemple d'ID de compte (*012345678901*) dans le rôle d'exécution de service par votre ID de compte.

```
{
  "ApplicationName": "test",
  "ApplicationDescription": "my java test app",
  "RuntimeEnvironment": "FLINK-1_6",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "java-getting-started-1.0.jar"
        }
      },
      "CodeContentType": "ZIPFILE"
    },
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap": {
            "flink.stream.initpos" : "LATEST",
            "aws.region" : "us-west-2",
            "AggregationEnabled" : "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap": {
            "aws.region" : "us-west-2"
          }
        }
      ]
    }
  }
}
```

2. Exécutez l'action [CreateApplication](#) avec la demande précédente pour créer l'application :

```
aws kinesisanalyticstv2 create-application --cli-input-json file://  
create_request.json
```

L'application est maintenant créée. Vous démarrez l'application dans l'étape suivante.

Démarrage de l'application

Dans cette section, vous utilisez l'action [StartApplication](#) pour démarrer l'application.

Pour démarrer l'application

1. Enregistrez le JSON code suivant dans un fichier nommé `start_request.json`.

```
{  
  "ApplicationName": "test",  
  "RunConfiguration": {  
    "ApplicationRestoreConfiguration": {  
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"  
    }  
  }  
}
```

2. Exécutez l'action [StartApplication](#) avec la demande précédente pour démarrer l'application :

```
aws kinesisanalyticstv2 start-application --cli-input-json file://start_request.json
```

L'application est maintenant en cours d'exécution. Vous pouvez consulter les métriques du service géré pour Apache Flink sur la CloudWatch console Amazon pour vérifier que l'application fonctionne.

Arrêt de l'application

Dans cette section, vous allez utiliser l'action [StopApplication](#) pour arrêter l'application.

Pour arrêter l'application

1. Enregistrez le JSON code suivant dans un fichier nommé `stop_request.json`.

```
{"ApplicationName": "test"}
```

```
}
```

2. Exécutez l'action [StopApplication](#) avec la demande suivante pour arrêter l'application :

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

L'application est maintenant arrêtée.

Tutoriel : Utilisation AWS Lambda avec Amazon Kinesis Data Streams

Dans ce didacticiel, vous créez une fonction Lambda pour consommer des événements à partir d'un flux de données Kinesis. Dans cet exemple de scénario, une application personnalisée écrit des enregistrements dans un flux de données Kinesis. AWS Lambda interroge ensuite ce flux de données et, lorsqu'il détecte de nouveaux enregistrements de données, invoque votre fonction Lambda. AWS Lambda exécute ensuite la fonction Lambda en assumant le rôle d'exécution que vous avez spécifié lors de la création de la fonction Lambda.

Pour obtenir des instructions détaillées étape par étape, consultez [Tutoriel : Utilisation de AWS Lambda avec Amazon Kinesis](#).

Note

Ce didacticiel suppose que vous avez une certaine connaissance des opérations Lambda de base et de la AWS Lambda console. Si ce n'est pas déjà fait, suivez les instructions de la section [Getting Started with AWS Lambda](#) pour créer votre première fonction Lambda.

Utilisez la solution de AWS streaming de données pour Amazon Kinesis

La solution de données de AWS streaming pour Amazon Kinesis configure automatiquement les AWS services nécessaires pour capturer, stocker, traiter et diffuser facilement des données de streaming. La solution propose plusieurs options pour résoudre les cas d'utilisation de données en streaming qui utilisent plusieurs AWS services, notamment Kinesis Data Streams AWS Lambda, Amazon Gateway et API Amazon Managed Service pour Apache Flink.

Chaque solution comprend les composants suivants :

- Un AWS CloudFormation package pour déployer l'exemple complet.
- Un CloudWatch tableau de bord pour afficher les métriques des applications.
- CloudWatch des alarmes sur les métriques les plus pertinentes de l'application.
- Tous les IAM rôles et politiques nécessaires.

La solution se trouve ici : [Solution de streaming de données pour Amazon Kinesis](#)

Création et gestion de flux de données Kinesis

Amazon Kinesis Data Streams ingère une grande quantité de données en temps réel, stocke durablement les données et rend les données disponibles pour leur utilisation. L'unité de données stockée par Kinesis Data Streams est un enregistrement de données. Un flux de données représente un groupe d'enregistrements de données. Les enregistrements de données figurant dans un flux de données sont répartis dans des partitions.

Une partition comporte une séquence d'enregistrements de données dans un flux. Elle sert d'unité de débit de base d'un flux de données Kinesis. Une partition prend en charge 1 Mo/s et 1 000 enregistrements par seconde pour les écritures et 2 Mo/s pour les lectures à la fois en mode à la demande et en mode capacité provisionnée. Les limites de partition garantissent des performances prévisibles, ce qui facilite la conception et l'exploitation d'un flux de données en continu très fiable.

Dans cette section, vous apprendrez à définir le mode de capacité du flux et à créer un flux à l'aide du AWS Management Console ou APIs. Vous pouvez ensuite effectuer des actions supplémentaires sur le stream.

Rubriques

- [Choisissez le mode de capacité du flux de données](#)
- [Créez un flux à l'aide du AWS Management Console](#)
- [Créez un flux à l'aide du APIs](#)
- [Mettre à jour un stream](#)
- [Afficher la liste des flux](#)
- [Répertorier les fragments](#)
- [Supprimer un stream](#)
- [Revisiter un stream](#)
- [Modifier la période de conservation des données](#)
- [Ajoutez des balises à vos flux dans Amazon Kinesis Data Streams](#)

Choisissez le mode de capacité du flux de données

Les rubriques suivantes expliquent comment choisir le mode de capacité adapté à votre application et comment passer d'un mode de capacité à un autre si nécessaire.

Rubriques

- [Qu'est-ce qu'un mode de capacité de flux de données ?](#)
- [Fonctionnalités et cas d'utilisation du mode à la demande](#)
- [Fonctionnalités et cas d'utilisation du mode provisionné](#)
- [Basculer entre les modes de capacité](#)

Qu'est-ce qu'un mode de capacité de flux de données ?

Un mode capacité détermine comment la capacité d'un flux de données est gérée et comment l'utilisation de votre flux de données vous est facturée. Dans Amazon Kinesis Data Streams, vous pouvez choisir entre un mode à la demande et un mode provisionné pour vos flux de données.

- À la demande : les flux de données en mode à la demande ne nécessitent aucune planification de la capacité et se mettent automatiquement à l'échelle pour gérer des gigaoctets de débit d'écriture et de lecture par minute. En mode à la demande, Kinesis Data Streams gère automatiquement les partitions afin de fournir le débit nécessaire.
- Provisionné : les flux de données en mode provisionné nécessitent la spécification du nombre de partitions. La capacité totale d'un flux de données est la somme des capacités de ses partitions. Vous pouvez augmenter ou diminuer le nombre de partitions présentes dans un flux de données en fonction des besoins.

Vous pouvez utiliser Kinesis Data Streams pour écrire des données dans vos PutRecord flux PutRecords APIs de données à la fois en mode à la demande et en mode capacité provisionnée. Pour récupérer des données, les deux modes de capacité prennent en charge les consommateurs par défaut qui utilisent le GetRecords API et les consommateurs Enhanced Fan-Out (EFO) qui utilisent le. SubscribeToShard API

Toutes les fonctionnalités de Kinesis Data Streams, y compris le mode de rétention, le chiffrement, les mesures de surveillance et autres, sont prises en charge à la fois pour le mode à la demande et le mode provisionné. Kinesis Data Streams garantit une durabilité et une disponibilité élevées à la fois en mode de capacité à la demande et en mode de capacité provisionnée.

Fonctionnalités et cas d'utilisation du mode à la demande

Les flux de données en mode à la demande ne nécessitent aucune planification de la capacité et se mettent automatiquement à l'échelle pour gérer des gigaoctets de débit d'écriture et de lecture

par minute. Le mode à la demande simplifie l'ingestion et le stockage de gros volumes de données avec une faible latence, car il élimine le provisionnement et la gestion des serveurs, du stockage ou du débit. Vous pouvez ingérer des milliards d'enregistrements par jour sans aucune surcharge opérationnelle.

Le mode à la demande est idéal pour répondre aux besoins d'un trafic d'application très variable et imprévisible. Il n'est plus nécessaire de provisionner ces charges de travail pour une capacité maximale, ce qui peut entraîner des coûts plus élevés pour une faible utilisation. Le mode à la demande convient aux charges de travail caractérisées par des modèles de trafic imprévisibles et très variables.

En mode de capacité à la demande, vous payez par gigaoctet de données écrites et lues à partir de vos flux de données. Il n'est pas nécessaire de spécifier le débit de lecture et d'écriture que vous attendez de votre application. Kinesis Data Streams s'adapte instantanément à vos charges de travail lorsqu'elles augmentent ou diminuent. Pour plus d'informations, consultez la [Tarification d'Amazon Kinesis Data Streams](#).

Un flux de données en mode à la demande peut atteindre le double du débit d'écriture maximal observé au cours des 30 jours précédents. Lorsque le débit d'écriture de votre flux de données atteint un nouveau maximum, Kinesis Data Streams adapte automatiquement la capacité du flux de données. Par exemple, si le débit d'écriture de votre flux de données varie entre 10 Mo/s et 40 Mo/s, Kinesis Data Streams vous permet de doubler facilement votre débit maximal précédent, soit 80 Mo/s. Si le même flux de données atteint un nouveau débit maximal de 50 Mo/s, Kinesis Data Streams s'assure qu'il y a suffisamment de capacité pour ingérer 100 Mo/s de débit d'écriture. Toutefois, une limitation du nombre d'écritures peut se produire si votre trafic augmente jusqu'à plus du double du maximum précédent dans un délai de 15 minutes. Vous devez réessayer ces demandes limitées.

La capacité de lecture agrégée d'un flux de données en mode à la demande augmente proportionnellement au débit d'écriture. Cela permet de garantir que les applications consommateur disposent toujours d'un débit de lecture suffisant pour traiter les données entrantes en temps réel. Vous obtenez un débit d'écriture au moins deux fois supérieur à celui des données de lecture utilisant le `GetRecordsAPI`. Nous vous recommandons d'utiliser une seule application grand public avec le `GetRecordsAPI`, afin qu'elle dispose de suffisamment d'espace pour rattraper son retard lorsque l'application doit se remettre d'une interruption de service. Il est recommandé d'utiliser la fonctionnalité de débit amélioré `Enhanced Fan-Out` de Kinesis Data Streams pour les scénarios nécessitant l'ajout de plusieurs applications consommateur. `Enhanced Fan-Out` permet d'ajouter jusqu'à 20 applications grand public à un flux de données à l'aide du `SubscribeToShardAPI`, chaque application client disposant d'un débit dédié.

Gérer les exceptions de débit de lecture et d'écriture

En mode de capacité à la demande (tout comme en mode de capacité provisionnée), vous devez spécifier une clé de partition pour chaque enregistrement afin d'écrire des données dans votre flux de données. Kinesis Data Streams utilise vos clés de partition pour répartir les données entre les partitions. Kinesis Data Streams surveille le trafic de chaque partition. Lorsque le trafic entrant dépasse 500 Ko/s par partition, le service fractionne la partition en moins de 15 minutes. Les valeurs de la clé de hachage de la partition parent sont redistribuées uniformément sur les partitions enfants.

Si votre trafic entrant dépasse le double du maximum précédent, vous pouvez rencontrer des exceptions de lecture ou d'écriture pendant environ 15 minutes, même si vos données sont réparties de manière uniforme sur l'ensemble des partitions. Nous vous recommandons de réessayer toutes ces demandes afin que tous les enregistrements soient correctement stockés dans Kinesis Data Streams.

Il est possible que vous rencontriez des exceptions de lecture et d'écriture si vous utilisez une clé de partition qui entraîne une distribution inégale des données et que les enregistrements attribués à une partition particulière dépassent les limites de cette dernière. En mode à la demande, le flux de données s'adapte automatiquement pour gérer les modèles de distribution inégale des données, à moins qu'une seule clé de partition ne dépasse les limites de débit de 1 Mo/s et de 1 000 enregistrements par seconde d'une partition.

En mode à la demande, Kinesis Data Streams fractionne les partitions de manière égale lorsqu'il détecte une augmentation du trafic. Toutefois, il ne détecte ni n'isole les clés de hachage qui acheminent une plus grande partie du trafic entrant vers une partition donnée. Si vous utilisez des clés de partition très inégales, il se peut que vous continuiez à recevoir des exceptions d'écriture. Pour de tels cas d'utilisation, nous vous recommandons d'utiliser le mode de capacité provisionnée qui prend en charge les fractionnements granulaires des partitions.

Fonctionnalités et cas d'utilisation du mode provisionné

En mode provisionné, après avoir créé le flux de données, vous pouvez augmenter ou diminuer dynamiquement la capacité de votre partition à l'aide du AWS Management Console ou du [UpdateShardCount](#) API. Vous pouvez effectuer des mises à jour pendant qu'une application producteur ou consommateur de Kinesis Data Streams écrit ou lit des données dans le flux.

Le mode provisionné est adapté à un trafic prévisible dont les besoins en capacité sont faciles à prévoir. Vous pouvez utiliser le mode provisionné si vous souhaitez un contrôle précis sur la manière dont les données sont distribuées à travers les partitions.

En mode provisionné, vous devez spécifier le nombre de partitions du flux de données. Pour déterminer la taille d'un flux de données en mode provisionné, vous avez besoin des valeurs d'entrée suivantes :

- Taille moyenne des enregistrements de données écrits dans le flux en kilo-octets (Ko), arrondie au Ko le plus proche (`average_data_size_in_KB`).
- Nombre d'enregistrements de données devant être ajoutés au flux et lus depuis celui-ci, par seconde (`records_per_second`).
- Nombre de consommateurs, c'est-à-dire d'applications Kinesis Data Streams qui consomment des données simultanément et indépendamment du flux (`number_of_consumers`).
- Bande passante d'écriture entrante en Ko (`incoming_write_bandwidth_in_KB`), qui est égale à la taille `average_data_size_in_KB` multipliée par la valeur de `records_per_second`.
- Bande passante de lecture sortante en Ko (`outgoing_read_bandwidth_in_KB`), qui est égale à la taille `incoming_write_bandwidth_in_KB` multipliée par la valeur de `number_of_consumers`.

Vous pouvez calculer le nombre initial de partitions (`number_of_shards`) dont votre flux a besoin en utilisant les valeurs d'entrée de la formule ci-dessous.

```
number_of_shards = max(incoming_write_bandwidth_in_KiB/1024,  
outgoing_read_bandwidth_in_KiB/2048)
```

Vous pouvez toujours rencontrer des exceptions de débit de lecture et d'écriture en mode provisionné si vous ne configurez pas votre flux de données pour gérer votre débit maximal. Dans ce cas, vous devez mettre à l'échelle manuellement votre flux de données pour l'adapter à votre trafic de données.

Il est également possible que vous rencontriez des exceptions de lecture et d'écriture si vous utilisez une clé de partition qui entraîne une distribution inégale des données et que les enregistrements attribués à une partition dépassent les limites de cette dernière. Pour résoudre ce problème en mode provisionné, identifiez ces partitions et fractionnez-les manuellement afin de mieux s'adapter à votre trafic. Pour plus d'informations, consultez la rubrique [Repartitionnement d'un flux](#) (français non garanti).

Basculer entre les modes de capacité

Vous pouvez changer le mode de capacité de votre flux de données de « à la demande » à « provisionné », ou de « provisionné » à « à la demande ». Pour chaque flux de données de votre

compte AWS , vous pouvez passer deux fois en 24 heures du mode de capacité à la demande au mode de capacité provisionnée.

Le passage d'un mode de capacité d'un flux de données à un autre ne perturbe pas les applications qui utilisent ce flux de données. Vous pouvez continuer à écrire et à lire dans ce flux de données. Lorsque vous passez d'un mode de capacité à l'autre, soit du mode à la demande au mode provisionné, soit du mode provisionné au mode à la demande, l'état du flux est défini comme étant En cours de mise à jour. Vous devez attendre que l'état du flux de données passe à Actif afin de pouvoir modifier à nouveau ses propriétés.

Lorsque vous passez du mode capacité provisionnée au mode capacité à la demande, votre flux de données conserve initialement le nombre de partitions qu'il possédait avant la transition. À partir de ce moment, Kinesis Data Streams surveille votre trafic de données et adapte le nombre de partitions de ce flux de données à la demande en fonction de votre débit d'écriture.

Lorsque vous passez du mode à la demande au mode provisionné, votre flux de données conserve initialement le nombre de partitions qu'il possédait avant la transition, mais à partir de ce moment, il vous incombe de surveiller et d'ajuster le nombre de partitions de ce flux de données afin de l'adapter correctement à votre débit d'écriture.

Créez un flux à l'aide du AWS Management Console

Vous pouvez créer un flux à l'aide de la console Kinesis Data Streams, de Kinesis Data API Streams ou AWS Command Line Interface du (.AWS CLI

Pour créer un flux de données avec la console

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à https://console.aws.amazon.com/kinesis](https://console.aws.amazon.com/kinesis).
2. Dans la barre de navigation, développez le sélecteur de région et choisissez une région.
3. Choisissez Create data stream (Créer un flux de données).
4. Sur la page Créer un flux Kinesis, saisissez le nom de votre flux de données, puis choisissez le mode de capacité à la demande ou provisionné. Le mode à la demande est sélectionné par défaut. Pour de plus amples informations, veuillez consulter [Choisissez le mode de capacité du flux de données](#).

En mode à la demande, vous pouvez ensuite choisir Créer un flux Kinesis pour créer votre flux de données. En mode provisionné, vous devez ensuite spécifier le nombre de partitions dont vous avez besoin, puis choisir Créer un flux Kinesis.

Dans la page Flux Kinesis, la valeur Statut de votre flux est En création lorsque le flux est en cours de création. Lorsque le flux est à prêt à être utilisé, le statut passe à Actif.

5. Choisissez le nom de votre flux. La page Détails du flux affiche un récapitulatif de la configuration de flux, ainsi que des informations de surveillance.

Pour créer un flux à l'aide des Kinesis Data Streams API

- Pour plus d'informations sur la création d'un flux à l'aide des Kinesis Data API Streams, [Créez un flux à l'aide du APIs](#) consultez.

Pour créer un flux à l'aide du AWS CLI

- Pour plus d'informations sur la création d'un flux à l'aide de AWS CLI, consultez la commande [create-stream](#).

Créez un flux à l'aide du APIs

Suivez les étapes ci-dessous pour créer votre flux de données Kinesis.

Créez le client Kinesis Data Streams

Afin de pouvoir travailler avec les flux de données Kinesis, vous devez créer un objet client. Le code Java suivant instancie un générateur client et l'utilise pour définir la région, les informations d'identification et la configuration du client. Il génère ensuite un objet client.

```
AmazonKinesisClientBuilder clientBuilder = AmazonKinesisClientBuilder.standard();

clientBuilder.setRegion(regionName);
clientBuilder.setCredentials(credentialsProvider);
clientBuilder.setClientConfiguration(config);

AmazonKinesis client = clientBuilder.build();
```


Pour plus d'informations, consultez la rubrique [Régions et points d'extrémité des flux de données Kinesis](#) dans Références générales AWS.

Créez le stream

Maintenant que vous avez créé votre client Kinesis Data Streams, vous pouvez créer un flux à l'aide de la console ou par programmation. Pour créer un flux par programmation, instanciez un `CreateStreamRequest` objet et attribuez un nom au flux. Si vous souhaitez utiliser le mode provisionné, spécifiez le nombre de partitions que le flux doit utiliser.

- À la demande :

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
```

- Provisionné :

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
createStreamRequest.setShardCount( myStreamSize );
```

Le nom du flux identifie le flux. Le nom est limité au AWS compte utilisé par l'application. Il est également défini pour chaque région. En d'autres termes, deux flux provenant de deux AWS comptes différents peuvent porter le même nom, et deux flux appartenant au même AWS compte mais situés dans deux régions différentes peuvent porter le même nom, mais pas deux flux sur le même compte et dans la même région.

Le débit du flux est fonction du nombre de partitions. Pour augmenter le débit provisionné, vous avez besoin d'un plus grand nombre de partitions. L'augmentation du nombre de partitions augmente également le AWS coût du stream. Pour plus d'informations sur le calcul d'un nombre approprié de partitions pour votre application, consultez [Choisissez le mode de capacité du flux de données](#).

Après avoir configuré l'`createStreamRequest` objet, créez un flux en appelant la `createStream` méthode sur le client. Après avoir appelé `createStream`, attendez que le flux passe à l'état `ACTIVE` avant d'effectuer des opérations sur le flux. Pour vérifier l'état du flux, appelez la méthode `describeStream`. Cependant, `describeStream` lève une exception si le flux n'existe pas. Par conséquent, intégrez l'appel `describeStream` dans un bloc `try/catch`.

```
client.createStream( createStreamRequest );
```

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime ) {
    try {
        Thread.sleep(20 * 1000);
    }
    catch ( Exception e ) {}

    try {
        DescribeStreamResult describeStreamResponse =
client.describeStream( describeStreamRequest );
        String streamStatus =
describeStreamResponse.getStreamDescription().getStreamStatus();
        if ( streamStatus.equals( "ACTIVE" ) ) {
            break;
        }
        //
        // sleep for one second
        //
        try {
            Thread.sleep( 1000 );
        }
        catch ( Exception e ) {}
    }
    catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime ) {
    throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Mettre à jour un stream

Vous pouvez mettre à jour les détails d'un flux à l'aide de la console Kinesis Data Streams, de Kinesis Data API Streams ou du AWS CLI

 Note

Vous pouvez activer un chiffrement côté serveur pour des flux de données existants ou pour des flux que vous avez créé récemment.

Utilisation de la console

Pour mettre à jour un flux de données avec la console

1. Ouvrez la console Amazon Kinesis à l'adresse. <https://console.aws.amazon.com/kinesis/>
2. Dans la barre de navigation, développez le sélecteur de région et choisissez une région.
3. Choisissez le nom de votre flux dans la liste. La page Détails du flux affiche un récapitulatif de la configuration de flux, ainsi que des informations de surveillance.
4. Pour passer du mode de capacité à la demande au mode de capacité provisionnée pour un flux de données, choisissez le mode Modifier la capacité dans l'onglet Configuration. Pour de plus amples informations, veuillez consulter [Choisissez le mode de capacité du flux de données](#).

 Important

Pour chaque flux de données de votre AWS compte, vous pouvez passer du mode à la demande au mode provisionné deux fois en 24 heures.

5. Pour un flux de données en mode provisionné, pour modifier le nombre de partitions, choisissez Modifier les partitions provisionnées dans l'onglet Configuration, puis saisissez un nouveau nombre de partitions.
6. Pour activer le chiffrement côté serveur d'enregistrements de données, choisissez Modifier dans la section Server-side encryption (Chiffrement côté serveur). Choisissez une KMS clé à utiliser comme clé principale pour le chiffrement, ou utilisez la clé principale par défaut, aws/kinesis, gérée par Kinesis. Si vous activez le chiffrement d'un flux et que vous utilisez votre propre clé AWS KMS principale, assurez-vous que vos applications de production et de consommation ont accès à la clé AWS KMS principale que vous avez utilisée. Pour attribuer des autorisations à une application afin qu'elle puisse accéder à une clé AWS KMS générée par un utilisateur, consultez [the section called "Autorisations d'utilisation des clés générées par l'utilisateur KMS"](#).
7. Pour modifier la période de conservation des données, choisissez Modifier dans la section Durée de conservation des données, puis saisissez une nouvelle durée de conservation des données.

8. Si vous avez activé les métriques personnalisées sur votre compte, choisissez Modifier dans la section Mesures au niveau des partitions, puis spécifiez les métriques pour votre flux. Pour de plus amples informations, veuillez consulter [the section called “Surveillez le service Kinesis Data Streams avec CloudWatch”](#).

Utilisez le API

Pour mettre à jour les détails du flux à l'aide de l'API, consultez les méthodes suivantes :

- [AddTagsToStream](#)
- [DecreaseStreamRetentionPeriod](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [IncreaseStreamRetentionPeriod](#)
- [RemoveTagsFromStream](#)
- [StartStreamEncryption](#)
- [StopStreamEncryption](#)
- [UpdateShardCount](#)

Utilisez le AWS CLI

Pour plus d'informations sur la mise à jour d'un flux à l'aide du AWS CLI, consultez la [référence Kinesis CLI](#).

Afficher la liste des flux

Les flux sont limités au AWS compte associé aux AWS informations d'identification utilisées pour instancier le client Kinesis Data Streams ainsi qu'à la région spécifiée pour le client. Un compte AWS peut avoir plusieurs flux actifs à la fois. Vous pouvez répertorier vos flux dans la console Kinesis Data Streams ou par programmation. Le code de cette section indique comment répertorier tous les streams de votre AWS compte.

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(20);
ListStreamsResult listStreamsResult = client.listStreams(listStreamsRequest);
```

```
List<String> streamNames = listStreamsResult.getStreamNames();
```

Cet exemple de code crée d'abord une nouvelle instance de `ListStreamsRequest` et appelle sa méthode `setLimit` pour spécifier que 20 flux au maximum doivent être renvoyés pour chaque appel de `listStreams`. Si vous ne spécifiez pas de valeur pour `setLimit`, Kinesis Data Streams renvoie un nombre de flux inférieur ou égal au nombre de flux présents dans le compte. Le code passe ensuite `listStreamsRequest` à la méthode `listStreams` du client. La valeur de retour `listStreams` est stockée dans un objet `ListStreamsResult`. Le code appelle la méthode `getStreamNames` sur cet objet et stocke les noms de flux renvoyés dans la liste `streamNames`. Notez que Kinesis Data Streams peut renvoyer un nombre de flux inférieur à celui qui est spécifié par la limite définie même si le compte et la région contiennent un nombre de flux supérieur. Pour être sûr d'extraire tous les flux, utilisez la méthode `getHasMoreStreams` comme il est décrit dans l'exemple de code suivant.

```
while (listStreamsResult.getHasMoreStreams())
{
    if (streamNames.size() > 0) {
        listStreamsRequest.setExclusiveStartStreamName(streamNames.get(streamNames.size()
- 1));
    }
    listStreamsResult = client.listStreams(listStreamsRequest);
    streamNames.addAll(listStreamsResult.getStreamNames());
}
```

Ce code appelle la méthode `getHasMoreStreams` sur `listStreamsRequest` pour vérifier s'il y a des flux supplémentaires disponibles en plus de ceux qui ont été renvoyés dans l'appel initial de `listStreams`. Si c'est le cas, le code appelle la méthode `setExclusiveStartStreamName` en indiquant le nom du dernier flux qui a été renvoyé dans l'appel précédent de `listStreams`. La méthode `setExclusiveStartStreamName` fait que l'appel suivant de `listStreams` démarre après ce flux. Le groupe de noms de flux renvoyé par cet appel est ensuite ajouté à la liste `streamNames`. Ce processus continue jusqu'à ce que tous les noms de flux aient été recueillis dans la liste.

Les flux renvoyés par `listStreams` peuvent avoir l'un des états suivants :

- CREATING
- ACTIVE
- UPDATING

- DELETING

Vous pouvez vérifier l'état d'un flux à l'aide de la méthode `describeStream`, comme illustré dans la section précédente [Créez un flux à l'aide du APIs](#).

Répertoirer les fragments

Un flux de données peut comporter une ou plusieurs partitions. La méthode recommandée pour répertoirer ou récupérer les fragments d'un flux de données consiste à utiliser le [ListShardsAPI](#). L'exemple suivant montre comment vous pouvez obtenir une liste des partitions d'un flux de données. Pour une description complète de l'opération principale utilisée dans cet exemple et de tous les paramètres que vous pouvez définir pour l'opération, consultez [ListShards](#).

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ListShardsRequest;
import software.amazon.awssdk.services.kinesis.model.ListShardsResponse;

import java.util.concurrent.TimeUnit;

public class ShardSample {

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.builder().build();

        ListShardsRequest request = ListShardsRequest
            .builder().streamName("myFirstStream")
            .build();

        try {
            ListShardsResponse response = client.listShards(request).get(5000,
                TimeUnit.MILLISECONDS);
            System.out.println(response.toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Pour exécuter l'exemple de code précédent, vous pouvez utiliser un POM fichier comme le suivant.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>kinesis.data.streams.samples</groupId>
  <artifactId>shards</artifactId>
  <version>1.0-SNAPSHOT</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>8</source>
          <target>8</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>kinesis</artifactId>
      <version>2.0.0</version>
    </dependency>
  </dependencies>
</project>
```

Avec le `ListShardsAPI`, vous pouvez utiliser le [ShardFilter](#) paramètre pour filtrer la réponse du API. Vous ne pouvez spécifier qu'un seul filtre à la fois.

Si vous utilisez le `ShardFilter` paramètre lors de l'appel de `ListShardsAPI`, la propriété `Type` est requise et doit être spécifiée. Si vous spécifiez les types `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON` ou `AT_LATEST`, vous n'avez pas besoin d'indiquer le `ShardId` ou les propriétés optionnelles `Timestamp`.

Si vous spécifiez le type `AFTER_SHARD_ID`, vous devez également indiquer la valeur de la propriété optionnelle `ShardId`. La fonctionnalité de la `ShardId` propriété est identique à celle du `ExclusiveStartShardId` paramètre de `ListShards API`. Lorsque la propriété `ShardId`

est indiquée, la réponse inclut les partitions en commençant par la partition dont l'identifiant suit immédiatement le `ShardId` que vous avez indiqué.

Si vous spécifiez le type `AT_TIMESTAMP` ou `FROM_TIMESTAMP_ID`, vous devez également indiquer la valeur de la propriété optionnelle `Timestamp`. Si vous spécifiez le type `AT_TIMESTAMP`, toutes les partitions ouvertes à l'horodatage indiqué sont renvoyées. Si vous spécifiez le `FROM_TIMESTAMP` type, toutes les partitions commençant par l'horodatage fourni sont TIP renvoyées.

Important

`DescribeStreamSummary` et `ListShard` APIs offrent un moyen plus évolutif de récupérer des informations sur vos flux de données. Plus précisément, les quotas pour le `DescribeStream` API peuvent entraîner un ralentissement. Pour de plus amples informations, veuillez consulter [Quotas et limites](#). Notez également que les `DescribeStream` quotas sont partagés entre toutes les applications qui interagissent avec tous les flux de données de votre AWS compte. Les quotas pour `ListShards` API, en revanche, sont spécifiques à un seul flux de données. Ainsi, non seulement vous progressez TPS avec le `ListShards` API, mais l'action s'adapte mieux à mesure que vous créez davantage de flux de données.

Nous vous recommandons de migrer tous vos producteurs et consommateurs qui appellent le `DescribeStream` API to au lieu d'invoquer le `DescribeStreamSummary` et le `ListShard` APIs. Pour identifier ces producteurs et consommateurs, nous recommandons d'utiliser Athena pour analyser les CloudTrail journaux en tant qu'agents utilisateurs KPL et KCL capturés dans les appels. API

```
SELECT useridentity.sessioncontext.sessionissuer.username,
useridentity.arn,eventname,useragent, count(*) FROM
cloudtrail_logs WHERE Eventname IN ('DescribeStream') AND
eventtime
    BETWEEN ''
    AND ''
GROUP BY
    useridentity.sessioncontext.sessionissuer.username,useridentity.arn,eventname,useragent
ORDER BY count(*) DESC LIMIT 100
```

Nous recommandons également de reconfigurer les intégrations AWS Lambda et Amazon Firehose avec Kinesis Data Streams qui invoquent `DescribeStream` API le. `DescribeStreamSummary` `ListShards` Plus précisément, pour AWS Lambda, vous

devez mettre à jour le mappage de votre source d'événements. Pour Amazon Firehose, les IAM autorisations correspondantes doivent être mises à jour afin qu'elles incluent l'`ListShardsIAM` autorisation.

Supprimer un stream

Vous pouvez supprimer un flux à l'aide de la console Kinesis Data Streams ou par programmation. Pour supprimer un flux par programmation, utilisez `DeleteStreamRequest`, comme illustré dans le code suivant.

```
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(myStreamName);
client.deleteStream(deleteStreamRequest);
```

Fermez toutes les applications qui opèrent sur le flux avant de supprimer ce dernier. Si une application tente d'opérer sur un flux supprimé, elle reçoit les exceptions `ResourceNotFound`. En outre, si vous créez ensuite un flux ayant le même nom que votre flux précédent et que les applications qui opéraient sur le flux précédent sont toujours en cours d'exécution, ces applications peuvent essayer d'interagir avec le nouveau flux comme si c'était le précédent, ce qui entraîne des résultats imprévisibles.

Revisionner un stream

Important

Vous pouvez redéfinir votre stream à l'aide du [UpdateShardCount](#) API. Sinon, vous pouvez continuer à diviser et à fusionner comme expliqué ici.

Amazon Kinesis Data Streams prend en charge le repartitionnement, ce qui vous permet d'ajuster le nombre de partitions dans votre flux afin de répondre aux modifications du débit de données dans le flux. Le repartitionnement est considéré comme une opération avancée. Si vous utilisez Kinesis Data Streams pour la première fois, revenez à ce sujet après vous être familiarisé avec tous les autres aspects de Kinesis Data Streams.

Il existe deux types d'opérations de repartitionnement : les fusions et les divisions de partitions. Lorsque vous fractionnez une partition, vous divisez une seule partition en deux partitions.

Lorsque vous fusionnez une partition, vous combinez deux partitions en une seule partition. Le repartitionnement s'effectue toujours par paire, car vous ne pouvez ni fractionner en plus de deux partitions dans une seule opération, ni fusionner plus de deux partitions en une seule opération. La partition ou la paire de partitions qui fait l'objet du repartitionnement s'appelle les partitions parent. La partition ou la paire de partitions générée par le repartitionnement s'appelle les partitions enfant.

Le fractionnement augmente le nombre de partitions dans votre flux et donc la capacité de données du flux. Etant donné que vous êtes facturé par partition, le fractionnement augmente le coût de votre flux. De même, la fusion réduit le nombre de partitions dans votre flux et diminue ainsi la capacité de données, et donc le coût du flux.

En général, le repartitionnement est effectué par une application administrative qui est distincte des applications producteur (put) et des applications consommateur (get). Une telle application administrative surveille les performances globales du flux sur la base des métriques fournies par Amazon CloudWatch ou sur la base des métriques collectées auprès des producteurs et des consommateurs. L'application administrative a également besoin d'un ensemble d'IAM autorisations plus large que celui des consommateurs ou des producteurs, car les consommateurs et les producteurs ne devraient généralement pas avoir besoin d'accéder au contenu APIs utilisé pour le repartage. Pour plus d'informations sur IAM les autorisations pour Kinesis Data Streams, [Contrôler l'accès aux ressources Amazon Kinesis Data Streams à l'aide de IAM](#) consultez.

Pour plus d'informations sur le repartitionnement, consultez la rubrique [Comment modifier le nombre de partitions ouvertes dans Kinesis Data Streams ?](#)

Rubriques

- [Décidez d'une stratégie de repartage](#)
- [Séparer un fragment](#)
- [Fusionner deux partitions](#)
- [Terminez l'action de repartage](#)

Décidez d'une stratégie de repartage

L'objectif du repartitionnement dans Amazon Kinesis Data Streams est de permettre à votre flux de s'adapter aux changements de débit des données. Vous fractionnez les partitions pour accroître la capacité (et le coût) de votre flux. Vous fusionnez les partitions pour réduire le coût (et la capacité) de votre flux.

Une approche du partitionnement peut consister à fractionner chaque partition du flux et à doubler ainsi la capacité du flux. Cependant, cela peut fournir une capacité supplémentaire supérieure à celle dont vous avez réellement besoin et donc engendrer des coûts inutiles.

Vous pouvez également utiliser des métriques pour déterminer quelles sont vos partitions chaudes et vos partitions froides, c'est-à-dire les partitions qui reçoivent plus de données que prévu et les partitions qui reçoivent moins de données que prévu. Vous pouvez ensuite fractionner les partitions chaudes de manière sélective afin d'augmenter la capacité des clés de hachage qui ciblent ces partitions. De même, vous pouvez fusionner les partitions froides afin de mieux utiliser leurs capacités inutilisées.

Vous pouvez obtenir certaines données de performance pour votre flux à partir des CloudWatch métriques Amazon publiées par Kinesis Data Streams. Cependant, vous pouvez également collecter certaines de vos propres métriques pour vos flux. Pour cela, vous pouvez consigner les valeurs de clé de hachage générées par les clés de partition pour vos enregistrements de données. N'oubliez pas que vous spécifiez la clé de partition lorsque vous ajoutez l'enregistrement au flux.

```
putRecordRequest.setPartitionKey( String.format( "myPartitionKey" ) );
```

Kinesis Data Streams calcule la [MD5](#) clé de hachage à partir de la clé de partition. Comme vous spécifiez la clé de partition pour l'enregistrement, vous pouvez l'utiliser MD5 pour calculer la valeur de la clé de hachage pour cet enregistrement et l'enregistrer.

Vous pouvez également enregistrer les IDs des partitions auxquelles vos enregistrements de données sont affectés. L'ID de partition s'obtient en utilisant la méthode `getShardId` de l'objet `putRecordResults` renvoyé par la méthode `putRecords` et l'objet `putRecordResult` renvoyé par la méthode `putRecord`.

```
String shardId = putRecordResult.getShardId();
```

Avec les valeurs de la partition IDs et de la clé de hachage, vous pouvez déterminer quelles partitions et quelles clés de hachage reçoivent le plus ou le moins de trafic. Vous pouvez ensuite utiliser le repartitionnement pour augmenter ou diminuer les capacités, de manière appropriée pour ces clés.

Séparer un fragment

Pour fractionner une partition dans Amazon Kinesis Data Streams, vous devez spécifier comment les valeurs de clé de hachage de la partition parent doivent être redistribuées aux partitions

enfants. Lorsque vous ajoutez un enregistrement de données à un flux, il est affecté à une partition suivant une valeur de clé de hachage. La valeur de la clé de hachage est le [MD5](#)hachage de la clé de partition que vous spécifiez pour l'enregistrement de données au moment où vous ajoutez l'enregistrement de données au flux. Les données des enregistrements ayant la même clé de partition disposent également de la même valeur de clé de hachage.

Les valeurs de clé de hachage admises pour une partition donnée constituent un ensemble d'entiers non négatifs contigus ordonnés. Cette gamme de valeurs de clé de hachage admises est donnée par ce qui suit :

```
shard.getHashKeyRange().getStartingHashKey();
shard.getHashKeyRange().getEndingHashKey();
```

Lorsque vous fractionnez la partition, vous spécifiez une valeur comprise dans cette plage. Cette valeur de clé de hachage et toutes les valeurs de clé de hachage supérieures sont distribuées à l'une des partitions enfant. Toutes les valeurs de clé de hachage inférieures sont distribuées à l'autre partition enfant.

Le code suivant illustre une opération de fractionnement de partition qui répartit de nouveau les clés de hachage uniformément entre chaque partition enfant, en fractionnant principalement en deux la partition parent. Il présente simplement une des manières possibles de diviser la partition parent. Vous pouvez, par exemple, diviser la partition de façon à ce que le tiers inférieur des clés issues de la partition parent passe dans une partition enfant et que les deux tiers supérieurs des clés passent dans l'autre partition enfant. Cependant, pour de nombreuses applications, le fractionnement des partitions en deux est une approche efficace.

Le code suppose que `myStreamName` contient le nom de votre flux et que la variable d'objet `shard` contient la partition à fractionner. Commencez par instancier un nouvel objet `splitShardRequest` et définir le nom de flux et l'ID de partition.

```
SplitShardRequest splitShardRequest = new SplitShardRequest();
splitShardRequest.setStreamName(myStreamName);
splitShardRequest.setShardToSplit(shard.getShardId());
```

Déterminez la valeur de clé de hachage qui se trouve à mi-chemin entre les valeurs inférieures et supérieures de la partition. Il s'agit de la valeur de clé de hachage de départ de la partition enfant qui contient la partie supérieure des clés de hachage de la partition parent. Spécifiez cette valeur dans la méthode `setNewStartingHashKey`. Vous ne devez spécifier que cette valeur. Kinesis Data

Streams distribue automatiquement les clés de hachage inférieures à cette valeur à l'autre partition enfant créée par le fractionnement. La dernière étape consiste à appeler la méthode `splitShard` sur le client Kinesis Data Streams.

```
BigInteger startingHashKey = new
    BigInteger(shard.getHashKeyRange().getStartingHashKey());
BigInteger endingHashKey = new
    BigInteger(shard.getHashKeyRange().getEndingHashKey());
String newStartingHashKey = startingHashKey.add(endingHashKey).divide(new
    BigInteger("2")).toString();

splitShardRequest.setNewStartingHashKey(newStartingHashKey);
client.splitShard(splitShardRequest);
```

La première étape qui suit cette procédure est présentée dans [Attendre qu'un stream redevienne actif](#).

Fusionner deux partitions

Dans une fusion de partitions, deux partitions spécifiées sont combinées en une seule partition. Après la fusion, la partition enfant unique reçoit des données pour toutes les valeurs de clé de hachage couvertes par les deux partitions parent.

Partitions adjacentes

Pour fusionner deux partitions, celles-ci doivent être adjacentes. Deux partitions sont considérées adjacentes si l'union des plages de clés de hachage des deux partitions forme un ensemble contigu sans interruption. Par exemple, supposons que vous avez deux partitions, l'un avec une plage de clé de hachage de 276...381 et l'autre avec une plage de clé de hachage de 382...454. Vous pouvez fusionner ces deux partitions dans une seule partition qui aurait une plage de clé de hachage de 276...454.

Pour un exemple supplémentaire, supposons que vous avez deux partitions, dont une avec une plage de clé de hachage de 276...381 et l'autre avec une plage de clé de hachage de 455...560. Vous ne pouvez pas fusionner ces deux partitions, car il y a une ou plusieurs partitions entre celles-ci qui couvrent la plage 382...454.

L'ensemble de toutes les OPEN partitions d'un flux, en tant que groupe, couvre toujours l'ensemble des valeurs des clés de hachage. MD5 Pour plus d'informations sur les états des partitions, par

exemple CLOSED, consultez [Tenez compte du routage des données, de la persistance des données et de l'état de la partition après une refonte](#) (français non garanti).

Pour identifier les partitions qui sont candidates à la fusion, vous devez filtrer toutes les partitions qui ont l'état CLOSED. Les partitions qui ont l'état OPEN, c'est-à-dire qui n'ont pas l'état CLOSED, ont un numéro de séquence de fin de null. Vous pouvez tester le numéro de séquence de fin d'une partition en utilisant :

```
if( null == shard.getSequenceNumberRange().getEndingSequenceNumber() )
{
    // Shard is OPEN, so it is a possible candidate to be merged.
}
```

Après avoir filtré les partitions fermées, triez les partitions restantes sur la valeur de clé de hachage maximale prise en charge par chaque partition. Vous pouvez extraire cette valeur en utilisant :

```
shard.getHashKeyRange().getEndingHashKey();
```

Si deux partitions sont adjacentes dans cette liste filtrée et triée, elles peuvent être fusionnées.

Code pour l'opération de fusion

Le code ci-après fusionne deux partitions. Ce code suppose que `myStreamName` contient le nom de votre flux et que les variables d'objet `shard1` et `shard2` contiennent les deux partitions adjacentes à fusionner.

Pour l'opération de fusion, commencez par instancier un nouvel objet `mergeShardsRequest`. Spécifiez le nom du flux avec la méthode `setStreamName`. Spécifiez ensuite les deux partitions à fusionner à l'aide des méthodes `setShardToMerge` et `setAdjacentShardToMerge`. Pour finir, appelez la méthode `mergeShards` sur le client Kinesis Data Streams pour effectuer l'opération.

```
MergeShardsRequest mergeShardsRequest = new MergeShardsRequest();
mergeShardsRequest.setStreamName(myStreamName);
mergeShardsRequest.setShardToMerge(shard1.getShardId());
mergeShardsRequest.setAdjacentShardToMerge(shard2.getShardId());
client.mergeShards(mergeShardsRequest);
```

La première étape qui suit cette procédure est présentée dans [Attendre qu'un stream redevienne actif](#).

Terminez l'action de repartage

Après tout type de procédure de repartitionnement dans Amazon Kinesis Data Streams et avant la reprise du traitement normal des enregistrements, d'autres procédures doivent être effectuées et d'autres éléments doivent être pris en compte. Les sections suivantes décrivent ces procédures et considérations.

Rubriques

- [Attendre qu'un stream redevienne actif](#)
- [Tenez compte du routage des données, de la persistance des données et de l'état de la partition après une refonte](#)

Attendre qu'un stream redevienne actif

Après avoir appelé une opération de repartage `mergeShards`, `splitShard` vous devez attendre que le flux redevienne actif. Le code à utiliser est le même que lorsque vous attendez qu'un flux devienne actif après la [création d'un flux](#). Ce code se présente comme suit :

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime )
{
    try {
        Thread.sleep(20 * 1000);
    }
    catch ( Exception e ) {}

    try {
        DescribeStreamResult describeStreamResponse =
client.describeStream( describeStreamRequest );
        String streamStatus =
describeStreamResponse.getStreamDescription().getStreamStatus();
        if ( streamStatus.equals( "ACTIVE" ) ) {
            break;
        }
    }
    //
    // sleep for one second
}
```

```
//
try {
    Thread.sleep( 1000 );
}
catch ( Exception e ) {}
}
catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime )
{
    throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Tenez compte du routage des données, de la persistance des données et de l'état de la partition après une refonte

Kinesis Data Streams est un service de streaming de données en temps réel. Vos applications doivent partir du principe que les données circulent en continu à travers les fragments de votre flux. Lorsque vous repartitionnez, les enregistrements de données qui étaient acheminés vers les partitions parent sont réacheminés vers les partitions enfant en se basant sur les valeurs de clé de hachage auxquelles sont mappées les clés de partition des enregistrements de données. Toutefois, les enregistrements de données qui se trouvaient dans les partitions parent avant le repartitionnement demeurent dans ces partitions. Les partitions parentes ne disparaissent pas lors du remaniement. Elles demeurent avec les données qu'elles contiennent avant le repartitionnement. Les enregistrements de données des partitions parents sont accessibles à l'aide des `getRecords` opérations [getShardIterateur](#) dans les Kinesis Data API Streams ou via la bibliothèque cliente Kinesis.

Note

Les enregistrements de données sont accessibles à partir du moment où ils sont ajoutés au flux jusqu'à la période de conservation actuelle. Cette constatation demeure vraie, quelles que soient les modifications apportées aux partitions du flux au cours de cette période. Pour plus d'informations sur la période de conservation d'un flux, consultez la page [Modifier la période de conservation des données](#).

Au cours du processus de repartitionnement, une partition parent passe de l'état OPEN à l'état CLOSED, puis à l'état EXPIRED.

- **OPEN:** Avant une opération de refonte, une partition parent est dans l'OPEN état, ce qui signifie que les enregistrements de données peuvent être à la fois ajoutés à la partition et extraits de la partition.
- **CLOSED:** Après une opération de refonte, la partition parent passe à un CLOSED état. Cela signifie qu'aucun enregistrement de données n'est plus ajouté à la partition. Les enregistrements de données qui auraient été ajoutés à cette partition sont maintenant ajoutés à une partition enfant. Toutefois, des enregistrements de données peuvent encore être extraits de la partition pendant une durée limitée.
- **EXPIRED:** une fois la période de conservation du flux expirée, tous les enregistrements de données de la partition parent ont expiré et ne sont plus accessibles. À ce stade, la partition elle-même passe à l'état EXPIRED. Les appels de `getStreamDescription().getShards` pour énumérer les partitions du flux n'incluent pas les partitions à l'état EXPIRED dans la liste des partitions renvoyées. Pour plus d'informations sur la période de conservation d'un flux, consultez la page [Modifier la période de conservation des données](#).

Lorsque le repartitionnement est terminé et que le flux a de nouveau l'état ACTIVE, vous pouvez commencer immédiatement à lire les données dans les partitions enfant. Cependant, les partitions parentes qui restent après la refonte peuvent toujours contenir des données que vous n'avez pas encore lues et qui ont été ajoutées au flux avant la refonte. Si vous lisez les données des partitions enfant avant d'avoir lu toutes les données des partitions parent, vous pouvez lire les données pour une clé de hachage spécifique sans respecter l'ordre donné par les numéros de séquence des enregistrements de données. Par conséquent, en supposant que l'ordre des données soit important, après un repartitionnement, vous devez toujours continuer à lire les données des partitions parentes jusqu'à leur épuisement. Ensuite seulement, vous devez commencer à lire les données des partitions enfants. Lorsque `getRecordsResult.getNextShardIterator` renvoie `null`, cela indique que vous avez lu toutes les données de la partition parent. Si vous lisez des données à l'aide de la bibliothèque cliente Kinesis, il est possible que vous ne les receviez pas dans l'ordre après une refonte.

Modifier la période de conservation des données

Amazon Kinesis Data Streams prend en charge les modifications de la période de conservation des enregistrements de données de votre flux de données. Le flux de données Kinesis est une séquence ordonnée d'enregistrements de données destinés à être écrits et lus en temps réel. Les enregistrements de données sont donc stockés provisoirement dans des partitions de votre flux. La période entre le moment où un enregistrement est ajouté et celui où il n'est plus accessible est

appelée la période de conservation. Un flux de données Kinesis stocke des enregistrements pendant 24 heures par défaut, jusqu'à 8 760 heures (365 jours).

Vous pouvez mettre à jour la période de rétention via la console Kinesis Data Streams ou en utilisant [IncreaseStreamRetentionPeriod](#) les opérations et [DecreaseStreamRetentionPeriod](#). La console Kinesis Data Streams vous permet de modifier en bloc la durée de conservation de plusieurs flux de données à la fois. Vous pouvez augmenter la période de rétention jusqu'à un maximum de 8 760 heures (365 jours) à l'aide de l'[IncreaseStreamRetentionPeriod](#) opération ou de la console Kinesis Data Streams. Vous pouvez réduire la période de rétention à un minimum de 24 heures à l'aide de l'[DecreaseStreamRetentionPeriod](#) opération ou de la console Kinesis Data Streams. La syntaxe de demande pour les deux opérations inclut le nom du flux et la période de conservation en heures. Enfin, vous pouvez vérifier la période de rétention actuelle d'un flux en appelant l'[DescribeStream](#) opération.

Voici un exemple de modification de la période de conservation à l'aide de l' AWS CLI :

```
aws kinesis increase-stream-retention-period --stream-name retentionPeriodDemo --  
retention-period-hours 72
```

Lorsque vous augmentez la période de conservation de Kinesis Data Streams, les enregistrements qui étaient auparavant inaccessibles en raison de l'ancienne période de conservation redeviennent accessibles en quelques minutes. Par exemple, si la période de conservation est portée de 24 heures à 48 heures, les enregistrements ajoutés au flux 23 heures 55 minutes auparavant continuent d'être disponibles au bout de 24 heures.

Lorsque vous diminuez la période de conservation de Kinesis Data Streams, les enregistrements plus anciens que la nouvelle période de conservation deviennent presque immédiatement inaccessibles. Par conséquent, soyez très prudent lorsque vous appelez l'[DecreaseStreamRetentionPeriod](#) opération.

Définissez votre période de conservation de données de manière à garantir que vos consommateurs pourront lire les données avant leur expiration, si des problèmes se produisent. Vous devez prendre soin de tenir compte de toutes les situations possibles, par exemple d'un problème avec votre logique de traitement des enregistrements ou d'une dépendance en aval hors service pendant une longue période. Considérez la période de conservation comme un filet de sécurité pour accorder davantage de temps à vos consommateurs de données pour extraire des données. Les API opérations relatives à la période de rétention vous permettent de le configurer de manière proactive ou de répondre aux événements opérationnels de manière réactive.

Des frais supplémentaires s'appliquent pour les flux dont la période de conservation définie est supérieure à 24 heures. Pour en savoir plus, consultez la [Tarification Amazon Kinesis Data Streams](#).

Ajoutez des balises à vos flux dans Amazon Kinesis Data Streams

Vous pouvez attribuer vos propres métadonnées aux flux que vous créez dans Amazon Kinesis Data Streams sous forme de balises. Une balise est une paire clé-valeur que vous définissez pour un flux. L'utilisation de balises est un moyen simple mais puissant de gérer les AWS ressources et d'organiser les données, y compris les données de facturation.

Table des matières

- [Passez en revue les bases des tags](#)
- [Suivez les coûts à l'aide du balisage](#)
- [Comprendre les restrictions relatives aux balises](#)
- [Étiquetez les flux à l'aide de la console Kinesis Data Streams](#)
- [Étiquetez les flux à l'aide du AWS CLI](#)
- [Étiquetez les flux à l'aide des Kinesis Data Streams API](#)

Passez en revue les bases des tags

Vous utilisez la console AWS CLI Kinesis Data Streams ou Kinesis Data API Streams pour effectuer les tâches suivantes :

- Ajouter des balises à un flux
- Afficher les balises de vos flux
- Supprimer les balises d'un flux

Les balises vous permettent de classer vos flux. Par exemple, vous pouvez classer les flux par objectif, propriétaire ou environnement. Dans la mesure où vous avez défini la clé et la valeur de chaque balise, vous pouvez créer un ensemble personnalisé de catégories répondant à vos besoins spécifiques. Par exemple, vous pouvez définir un ensemble de balises vous permettant de suivre les flux par propriétaire et application associée. Voici quelques exemples de balises :

- Projet : nom du projet

- Propriétaire : nom
- Objectif : test de la charge
- Application : nom de l'application
- Environnement : production

Suivez les coûts à l'aide du balisage

Vous pouvez utiliser des balises pour classer et suivre vos AWS coûts. Lorsque vous appliquez des balises à vos AWS ressources, y compris aux flux, votre rapport de répartition des AWS coûts inclut l'utilisation et les coûts agrégés par balises. Vous pouvez appliquer des balises associées à des catégories métier (telles que les centres de coûts, les noms d'applications ou les propriétaires) pour organiser les coûts relatifs à divers services. Pour plus d'informations, consultez [Utilisation des identifications de répartition des coûts pour les rapports de facturation personnalisés](#) dans le Guide de l'utilisateur AWS Billing .

Comprendre les restrictions relatives aux balises

Les restrictions suivantes s'appliquent aux balises :

Restrictions de base

- Le nombre maximum de balises par ressource (flux) est de 50.
- Les clés et valeurs d'étiquette sont sensibles à la casse.
- Vous ne pouvez pas changer ou modifier les balises d'un flux supprimé.

Restrictions relatives aux clés de balise

- Chaque clé de balise doit être unique. Si vous ajoutez une balise avec une clé qui est déjà en cours d'utilisation, la nouvelle balise remplacera la paire clé-valeur existante.
- Vous ne pouvez pas commencer une clé de balise par, aws : car ce préfixe est réservé à l'usage de AWS. AWS crée des balises qui commencent par ce préfixe en votre nom, mais vous ne pouvez ni les modifier ni les supprimer.
- Les clés de balise doivent comporter entre 1 et 128 caractères Unicode.
- Les clés de balise doivent comporter les caractères suivants : lettres Unicode, chiffres, espaces et les caractères spéciaux suivants : _ . / = + - @.

Restrictions relatives à la valeur de balise

- Les valeurs de balise doivent comporter entre 0 et 255 caractères Unicode.
- Les valeurs de balise peuvent être vides. Si tel n'est pas le cas, elles doivent être composées des caractères suivants : lettres Unicode, chiffres, espaces et les caractères spéciaux suivants : _ . / = + - @.

Étiquetez les flux à l'aide de la console Kinesis Data Streams

Vous pouvez ajouter, répertorier et supprimer des balises à l'aide de la console Kinesis Data Streams.

Pour afficher les balises d'un flux

1. Ouvrez la console Kinesis Data Streams. Dans la barre de navigation, développez le sélecteur de région et sélectionnez une région.
2. Dans la page Liste de flux, sélectionnez un flux.
3. Sur la page Détails du stream, choisissez l'onglet Tags.

Pour ajouter une balise à un flux

1. Ouvrez la console Kinesis Data Streams. Dans la barre de navigation, développez le sélecteur de région et sélectionnez une région.
2. Dans la page Liste de flux, sélectionnez un flux.
3. Sur la page Détails du stream, choisissez l'onglet Tags.
4. Spécifiez la clé de balise dans le champ Clé, spécifiez éventuellement une valeur de balise dans le champ Valeur, puis choisissez Ajouter une étiquette.

Si le bouton Ajouter balise n'est pas activé, la clé ou la valeur de la balise que vous avez spécifiée ne répond pas aux restrictions de balise. Pour plus d'informations, consultez [Comprendre les restrictions relatives aux balises](#).

5. Pour afficher votre nouveau tag dans la liste de l'onglet Tags, cliquez sur l'icône d'actualisation.

Pour supprimer une balise d'un flux

1. Ouvrez la console Kinesis Data Streams. Dans la barre de navigation, développez le sélecteur de région et sélectionnez une région.
2. Dans la page Liste de flux, sélectionnez un flux.
3. Sur la page Détails du stream, cliquez sur l'onglet Tags, puis sur l'icône Supprimer pour le tag.
4. Dans la boîte de dialogue Supprimer le tag, choisissez Oui, Supprimer.

Étiquetez les flux à l'aide du AWS CLI

Vous pouvez ajouter, répertorier et supprimer des balises à l'aide de l' AWS CLI. Pour obtenir des exemples, consultez la documentation suivante :

[add-tags-to-stream](#)

Ajoute ou met à jour les balises du flux spécifié.

[list-tags-for-stream](#)

Répertorie les balises du flux spécifié.

[remove-tags-from-stream](#)

Supprime les balises du flux spécifié.

Étiquetez les flux à l'aide des Kinesis Data Streams API

Vous pouvez ajouter, répertorier et supprimer des balises à l'aide des Kinesis Data API Streams. Pour obtenir des exemples, consultez la documentation suivante :

[AddTagsToStream](#)

Ajoute ou met à jour les balises du flux spécifié.

[ListTagsForStream](#)

Répertorie les balises du flux spécifié.

[RemoveTagsFromStream](#)

Supprime les balises du flux spécifié.

Écrire des données dans Amazon Kinesis Data Streams

Un producteur est une application qui écrit des données dans Amazon Kinesis Data Streams. Vous pouvez créer des producteurs pour Kinesis Data Streams à l'aide de la Kinesis Producer Library () et de AWS SDK for Java la Kinesis Producer Library (). KPL

Si vous ne connaissez pas Kinesis Data Streams, commencez par vous familiariser avec les concepts et la terminologie présentés dans [Qu'est-ce qu'Amazon Kinesis Data Streams ?](#) et [Utilisez le AWS CLI pour effectuer des opérations Amazon Kinesis Data Streams](#).

Important

Kinesis Data Streams prend en charge les modifications de la période de conservation des enregistrements de données de votre flux de données. Pour plus d'informations, consultez [Modifier la période de conservation des données](#).

Pour placer des données dans le flux, vous devez spécifier le nom du flux, une clé de partition et le blob de données à ajouter au flux. La clé de partition sert à déterminer la partition à laquelle l'enregistrement de données est ajouté dans le flux.

Toutes les données de la partition sont envoyées à la même application de travail qui traite la partition. La clé de partition que vous utilisez dépend de votre logique d'application. En général, le nombre de clés de partition doit être bien supérieur au nombre de partitions. Cela provient du fait que la clé de partition sert à déterminer comment mapper un enregistrement de données à une partition spécifique. Si vous avez suffisamment de clés de partition, les données peuvent être réparties uniformément entre les partitions d'un flux.

Rubriques

- [Développez des producteurs à l'aide de la bibliothèque Amazon Kinesis Producer \(\) KPL](#)
- [Développez les producteurs en utilisant Amazon Kinesis Data API Streams avec AWS SDK for Java](#)
- [Écrire sur Amazon Kinesis Data Streams à l'aide de Kinesis Agent](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'autres services AWS](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'intégrations tierces](#)
- [Résoudre les problèmes liés aux producteurs d'Amazon Kinesis Data Streams](#)

- [Optimisez les producteurs de Kinesis Data Streams](#)

Développez des producteurs à l'aide de la bibliothèque Amazon Kinesis Producer () KPL

Un producteur Amazon Kinesis Data Streams est une application qui place des enregistrements de données utilisateur dans un flux de données Kinesis (également appelé ingestion de données). La Kinesis Producer Library (KPL) simplifie le développement des applications de production, permettant aux développeurs d'atteindre un débit d'écriture élevé dans un flux de données Kinesis.

Vous pouvez les surveiller KPL avec Amazon CloudWatch. Pour plus d'informations, consultez [Surveillez la bibliothèque Kinesis Producer avec Amazon CloudWatch](#).

Rubriques

- [Passez en revue le rôle du KPL](#)
- [Découvrez les avantages de l'utilisation du KPL](#)
- [Sachez quand vous ne devez pas utiliser le KPL](#)
- [Installer le KPL](#)
- [Transition vers les certificats Amazon Trust Services \(ATS\) pour KPL](#)
- [Plateformes prises en charge par KPL](#)
- [KPL concepts clés](#)
- [Intégrez le KPL avec le code du producteur](#)
- [Écrivez dans votre flux de données Kinesis à l'aide du KPL](#)
- [Configuration de la bibliothèque Kinesis Producer](#)
- [Mettre en œuvre la désagrégation des consommateurs](#)
- [Utilisez le KPL avec Amazon Data Firehose](#)
- [Utiliser le KPL avec le registre des AWS Glue schémas](#)
- [Configuration de la configuration du KPL proxy](#)

Note

Il est recommandé de passer à la dernière KPL version. KPL est régulièrement mis à jour avec les nouvelles versions qui incluent les derniers correctifs de dépendance et de sécurité,

les corrections de bogues et les nouvelles fonctionnalités rétrocompatibles. Pour plus d'informations, consultez <https://github.com/awslabs/amazon-kinesis-producer/releases/>.

Passez en revue le rôle du KPL

KPLII s'agit d'une easy-to-use bibliothèque hautement configurable qui vous permet d'écrire dans un flux de données Kinesis. Il sert d'intermédiaire entre le code de votre application de production et les actions Kinesis Data API Streams. KPL exécute les tâches principales suivantes :

- Écrit dans un ou plusieurs flux de données Kinesis avec un mécanisme de nouvelle tentative configurable
- Collecte les enregistrements et utilise PutRecords pour écrire plusieurs enregistrements dans plusieurs partitions par demande
- Regroupe les enregistrements d'utilisateur pour accroître la taille de la charge utile et améliorer le débit
- S'intègre parfaitement à la [bibliothèque cliente Kinesis](#) (KCL) pour désagréger les enregistrements par lots concernant le consommateur
- Soumet CloudWatch les statistiques Amazon en votre nom afin de fournir une visibilité sur les performances des producteurs

Notez que le KPL est différent des Kinesis Data API Streams disponibles dans le [AWS SDKs](#). Les Kinesis Data API Streams vous aident à gérer de nombreux aspects de Kinesis Data Streams (notamment la création de flux, le repartage, le transfert et l'obtention d'enregistrements), tout en fournissant une couche d'abstraction spécialement conçue pour KPL l'ingestion de données. Pour plus d'informations sur les Kinesis Data API Streams, consultez [le API Amazon Kinesis Reference](#).

Découvrez les avantages de l'utilisation du KPL

La liste suivante présente certains des principaux avantages liés à l'utilisation des producteurs de Kinesis Data Streams KPL pour le développement.

Ils KPL peuvent être utilisés dans des cas d'utilisation synchrones ou asynchrones. Nous recommandons d'utiliser les meilleures performances de l'interface asynchrone, à moins d'avoir une raison précise d'utiliser le comportement synchrone. Pour plus d'informations sur ces deux cas d'utilisation et l'exemple de code, consultez la page [Écrivez dans votre flux de données Kinesis à l'aide du KPL](#).

Avantages en termes de performances

Ils KPL peuvent aider à créer des producteurs performants. Imaginons que vos EC2 instances Amazon servent de proxy pour collecter des événements de 100 octets provenant de centaines ou de milliers d'appareils à faible consommation d'énergie et pour écrire des enregistrements dans un flux de données Kinesis. Ces EC2 instances doivent chacune écrire des milliers d'événements par seconde dans votre flux de données. Pour atteindre le débit nécessaire, les applications producteur doivent mettre en œuvre une logique complexe, telle que le traitement par lots ou multithreading, en plus d'une logique de nouvelle tentative et du dégroupement des enregistrements du côté consommateur. Il KPL exécute toutes ces tâches pour vous.

Facilité d'utilisation côté consommateur

Pour les développeurs côté consommateur utilisant Java, KCL l'intégration se fait sans effort supplémentaire. Lorsque le KCL extrait un enregistrement Kinesis Data Streams agrégé composé de KPL plusieurs enregistrements utilisateur, il invoque automatiquement KPL le pour extraire les enregistrements utilisateur individuels avant de les renvoyer à l'utilisateur.

Pour les développeurs côté consommateur qui n'utilisent pas l'API opération KCL mais l'utilisent `GetRecords` directement, une bibliothèque KPL Java est disponible pour extraire les enregistrements utilisateur individuels avant de les renvoyer à l'utilisateur.

Surveillance d'une application producteur

Vous pouvez collecter, surveiller et analyser vos producteurs de Kinesis Data Streams à l'aide d' `CloudWatch` Amazon et KPL du. Le KPL système émet des indicateurs de débit, d'erreur et d'autres indicateurs en votre `CloudWatch` nom, et est configurable pour être surveillé au niveau du flux, de la partition ou du producteur.

Architecture asynchrone

Comme les enregistrements KPL peuvent être mis en mémoire tampon avant de les envoyer à Kinesis Data Streams, cela n'oblige pas l'application appelant à bloquer et à attendre la confirmation que l'enregistrement est arrivé sur le serveur avant de poursuivre l'exécution. Un appel pour insérer un enregistrement dans le renvoie KPL toujours immédiatement et n'attend pas que l'enregistrement soit envoyé ou qu'une réponse soit reçue du serveur. En revanche, un objet `Future` est créé, qui reçoit le résultat de l'envoi ultérieur de l'enregistrement à Kinesis Data Streams. Ce comportement est identique à celui des clients asynchrones dans le `AWS SDK`

Sachez quand vous ne devez pas utiliser le KPL

Les KPL peuvent entraîner un délai de traitement supplémentaire pouvant aller jusqu'à l'`RecordMaxBufferedTime` de la bibliothèque (configurable par l'utilisateur). Les valeurs élevées de `RecordMaxBufferedTime` se traduisent par une efficacité de compression supérieure et de meilleures performances. Les applications qui ne tolèrent pas ce délai supplémentaire devront peut-être utiliser l'AWS SDK directement. Pour plus d'informations sur l'utilisation de l'AWS SDK avec Kinesis Data Streams, consultez [Développez les producteurs en utilisant Amazon Kinesis Data API Streams avec l'AWS SDK for Java](#). Pour plus d'informations sur les propriétés configurables par l'utilisateur `RecordMaxBufferedTime` et sur les autres propriétés configurables par l'utilisateur KPL, consultez [Configuration de la bibliothèque Kinesis Producer](#).

Installer le KPL

Amazon fournit des fichiers binaires prédéfinis de la bibliothèque C++ Kinesis Producer (KPL) pour macOS, Windows et les distributions Linux récentes (pour plus d'informations sur les plateformes prises en charge, consultez la section suivante). Ces fichiers binaires sont emballés sous forme de fichiers `.jar` Java et sont appelés automatiquement et utilisés si vous utilisez Maven pour installer le package. Pour trouver les dernières versions de KPL et KCL, utilisez les liens de recherche Maven suivants :

- [KPL](#)
- [KCL](#)

Les fichiers binaires Linux ont été compilés avec GNU Compiler Collection (GCC) et liés statiquement à `libstdc++` sous Linux. Ils sont censés fonctionner sur toutes les distributions Linux 64 bits qui incluent une `glibc` version 2.5 ou ultérieure.

Les utilisateurs d'anciennes distributions Linux peuvent créer le KPL en utilisant les instructions de compilation fournies avec le code source activé GitHub. Pour télécharger le formulaire KPL GitHub, consultez la bibliothèque [Kinesis Producer](#).

Transition vers les certificats Amazon Trust Services (ATS) pour KPL

Le 9 février 2018, à 9h00PST, Amazon Kinesis Data Streams ATS a installé des certificats. Pour continuer à écrire des enregistrements dans Kinesis Data Streams à l'aide de la Kinesis Producer Library KPL, vous devez mettre à niveau votre installation [vers KPL la version](#) 0.12.6 ou ultérieure. Ce changement concerne toutes les AWS régions.

Pour plus d'informations sur le passage àATS, voir [Comment préparer AWS le passage à sa propre autorité de certification](#).

Si vous rencontrez des problèmes et que vous avez besoin d'une assistance technique, [créez une demande](#) auprès du centre de support AWS .

Plateformes prises en charge par KPL

La Kinesis Producer Library (KPL) est écrite en C++ et s'exécute en tant que processus enfant du processus utilisateur principal. Des fichiers binaires natifs 64 bits précompilés sont regroupés avec la version Java et sont gérés par le wrapper Java.

Le package Java s'exécute sans avoir besoin d'installer les bibliothèques supplémentaires sur les systèmes d'exploitation suivants :

- Distributions Linux avec le noyau 2.6.18 (septembre 2006) et ultérieur
- Apple OS X version 10.9 et ultérieure
- Windows Server 2008 et version ultérieure

Important

Windows Server 2008 et versions ultérieures sont pris en charge pour toutes les KPL versions jusqu'à la version 0.14.0.

La plate-forme Windows est NOT prise en charge à partir de KPL la version 0.14.0 ou supérieure.

Notez qu'il s'agit uniquement de 64 bits.

Code source

Si les fichiers binaires fournis dans l'installation ne sont pas suffisants pour votre environnement, le cœur du KPL est écrit sous forme de module C++. Le code source du module C++ et de l'interface Java est publié sous Amazon Public License et est disponible sur GitHub [Kinesis Producer Library](#). Bien qu'il KPL puisse être utilisé sur toutes les plateformes pour lesquelles un compilateur C++ récent conforme aux normes est disponible, Amazon ne prend officiellement en charge aucune plate-forme ne figurant pas sur la liste des plateformes prises en charge. JRE

KPL concepts clés

Les sections suivantes contiennent les concepts et la terminologie nécessaires pour comprendre et tirer parti de la Kinesis Producer Library (KPL).

Rubriques

- [Enregistrements](#)
- [Traitement par lot](#)
- [Agrégation](#)
- [Collection](#)

Enregistrements

Dans ce guide, nous distinguons les enregistrements KPL utilisateur des enregistrements Kinesis Data Streams. Lorsque nous utilisons le terme enregistrement sans qualificatif, nous faisons référence à un enregistrement KPL utilisateur. Lorsque nous faisons référence à un enregistrement Kinesis Data Streams, il est explicitement question d'enregistrement Kinesis Data Streams.

Un enregistrement KPL utilisateur est un bloc de données qui a une signification particulière pour l'utilisateur. Les exemples incluent un JSON blob représentant un événement d'interface utilisateur sur un site Web ou une entrée de journal provenant d'un serveur Web.

Un enregistrement Kinesis Data Streams est une instance de Record la structure de données définie par le service Kinesis Data Streams. API Il contient une clé de partition, un numéro de séquence et un blob de données.

Traitement par lot

Le traitement par lot consiste à exécuter une action sur plusieurs éléments au lieu d'exécuter cette action à plusieurs reprises sur chaque élément individuel.

Dans ce contexte, l'« élément » est un enregistrement et l'action est l'envoi de cet enregistrement à Kinesis Data Streams. En l'absence de traitement par lots, vous devez placer chaque enregistrement dans un enregistrement Kinesis Data Streams distinct et faire une HTTP demande pour l'envoyer à Kinesis Data Streams. Avec le traitement par lots, chaque HTTP demande peut contenir plusieurs enregistrements au lieu d'un seul.

Il KPL prend en charge deux types de traitement par lots :

- Regroupement : stockage de plusieurs enregistrements dans un seul enregistrement Kinesis Data Streams.
- Collecte : utilisation de l'API opération `PutRecords` pour envoyer plusieurs enregistrements Kinesis Data Streams vers une ou plusieurs partitions de votre flux de données Kinesis.

Les deux types de traitement par KPL lots sont conçus pour coexister et peuvent être activés ou désactivés indépendamment l'un de l'autre. Par défaut, les deux types sont activés.

Agrégation

Le regroupement consiste à stocker plusieurs enregistrements dans un enregistrement Kinesis Data Streams. L'agrégation permet aux clients d'augmenter le nombre d'enregistrements envoyés par API appel, ce qui augmente efficacement le débit des producteurs.

Les partitions Kinesis Data Streams acceptent jusqu'à 1 000 enregistrements Kinesis Data Streams par seconde, soit un débit de 1 Mo. Le nombre limite d'enregistrements Kinesis Data Streams par seconde force les clients à avoir des enregistrements inférieurs à 1 Ko. Le regroupement des enregistrements permet aux clients de combiner plusieurs enregistrements en un seul enregistrement Kinesis Data Streams. Cela permet aux clients d'améliorer leur débit par partition.

Prenons le cas d'une partition dans la région us-east-1 qui fonctionne actuellement à un rythme constant de 1 000 enregistrements par seconde, avec des enregistrements de 512 octets chacun. Grâce à KPL l'agrégation, vous pouvez regrouper 1 000 enregistrements dans seulement 10 enregistrements Kinesis Data Streams, ce qui réduit RPS le nombre à 10 (à 50 Ko chacun).

Collection

La collecte consiste à regrouper plusieurs enregistrements Kinesis Data Streams et à les envoyer en une HTTP seule demande avec un appel à API l'opération `PutRecords`, au lieu d'envoyer chaque enregistrement Kinesis Data Streams dans sa propre demande. HTTP

Cela augmente le débit par rapport à l'absence de collecte, car cela réduit les frais liés à l'envoi de nombreuses HTTP demandes distinctes. En fait, l'API `PutRecords` elle-même a été spécifiquement conçue à cet effet.

La collecte est différente du regroupement en cela qu'elle utilise des groupes d'enregistrements Kinesis Data Streams. Les enregistrements Kinesis Data Streams collectés peuvent encore contenir plusieurs enregistrements provenant de l'utilisateur. La relation peut être visualisée comme suit :

```
record 0 -- |
```

```

record 1   |           [ Aggregation ]
...       |--> Amazon Kinesis record 0 --|
...       |
record A --|
...       |
...       |
record K --|
record L   |           [ Collection ]
...       |--> Amazon Kinesis record C --|--> PutRecords Request
...       |
record S --|
...       |
...       |
record AA--|
record BB  |
...       |--> Amazon Kinesis record M --|
...       |
record ZZ--|

```

Intégrez le KPL avec le code du producteur

La Kinesis Producer Library (KPL) s'exécute dans le cadre d'un processus distinct et communique avec le processus utilisateur parent à l'aide de IPC. Cette architecture est parfois appelée un [microservice](#) et est choisie pour deux raisons principales :

1) Votre processus utilisateur ne se bloquera pas même s'il se KPL bloque

Votre processus peut comporter des tâches non liées à Kinesis Data Streams et peut continuer à fonctionner même en cas KPL de panne. Il est également possible que le processus utilisateur de votre parent redémarre KPL et revienne à un état de fonctionnement complet (cette fonctionnalité se trouve dans les wrappers officiels).

Prenons l'exemple d'un serveur Web qui envoie des métriques à Kinesis Data Streams. Le serveur peut continuer à servir des pages même si la partie Kinesis Data Streams a cessé de fonctionner. Le fait de planter l'ensemble du serveur à cause d'un bogue KPL provoquerait donc une panne inutile.

2) Les clients arbitraires peuvent être pris en charge

Il y a toujours des clients qui utilisent des langages autres que ceux qui sont officiellement pris en charge. Ces clients devraient également pouvoir utiliser le KPL facilement.

Matrice d'utilisation recommandée

La matrice d'utilisation suivante énumère les paramètres recommandés pour les différents utilisateurs et vous indique si et comment vous devez utiliser le KPL. N'oubliez pas que si le regroupement est activé, le dégroupement doit aussi être utilisé pour extraire vos enregistrements du côté consommateur.

Langage côté producteur	Langage côté consommateur	KCLVersion	Logique de contrôle	Pouvez-vous utiliser la KPL ?	Mises en garde
Tout sauf Java	*	*	*	Non	N/A
Java	Java	Utilise Java SDK directement	N/A	Oui	Si le regroupement est utilisé, vous devez utiliser la bibliothèque de regroupement fournie après les appels de <code>GetRecords</code> .
Java	Tout sauf Java	Utilisations SDK directes	N/A	Oui	Désactiver obligatoirement le regroupement.
Java	Java	1.3.x	N/A	Oui	Désactiver obligatoirement le

Langage côté producteur	Langage côté consommateur	KCLVersion	Logique de contrôle	Pouvez-vous utiliser la KPL ?	Mises en garde
					regroupement.
Java	Java	1.4.x	Appelle le contrôle sans argument	Oui	Aucun
Java	Java	1.4.x	Appelle le contrôle avec un numéro de séquence explicite	Oui	Désactiver le regroupement ou modifier le code pour utiliser des numéros de séquence étendus pour le contrôle.
Java	Tout sauf Java	1.3.x + démon multilingue + wrapper propre au langage	N/A	Oui	Désactiver obligatoirement le regroupement.

Écrivez dans votre flux de données Kinesis à l'aide du KPL

Les sections suivantes présentent des exemples de code dans une progression allant du code de production le plus élémentaire au code entièrement asynchrone.

Code du producteur Barebones

Le code suivant contient tous les éléments nécessaires pour écrire une application producteur convenable minimale. Les enregistrements utilisateur de la Kinesis Producer Library (KPL) sont traités en arrière-plan.

```
// KinesisProducer gets credentials automatically like
// DefaultAWSCredentialsProviderChain.
// It also gets region automatically from the EC2 metadata service.
KinesisProducer kinesis = new KinesisProducer();
// Put some records
for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    kinesis.addUserRecord("myStream", "myPartitionKey", data);
}
// Do other stuff ...
```

Réagir aux résultats de manière synchrone

Dans l'exemple précédent, le code ne vérifiait pas si les enregistrements KPL utilisateur étaient corrects. Il KPL effectue toutes les tentatives nécessaires pour tenir compte des échecs. Si vous voulez vérifier les résultats, vous pouvez les examiner à l'aide des objets Future renvoyés par `addUserRecord`, comme dans l'exemple suivant (exemple précédent indiqué pour le contexte) :

```
KinesisProducer kinesis = new KinesisProducer();

// Put some records and save the Futures
List<Future<UserRecordResult>> putFutures = new
    LinkedList<Future<UserRecordResult>>();
for (int i = 0; i < 100; i++) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    putFutures.add(
        kinesis.addUserRecord("myStream", "myPartitionKey", data));
}

// Wait for puts to finish and check the results
for (Future<UserRecordResult> f : putFutures) {
    UserRecordResult result = f.get(); // this does block
    if (result.isSuccessful()) {
        System.out.println("Put record into shard " +
```

```
        result.getShardId());
    } else {
        for (Attempt attempt : result.getAttempts()) {
            // Analyze and respond to the failure
        }
    }
}
```

Répondre aux résultats de manière asynchrone

L'exemple précédent appelle `get()` sur un objet `Future`, ce qui bloque l'exécution. Si vous ne voulez pas bloquer l'exécution, vous pouvez utiliser un rappel asynchrone, comme illustré dans l'exemple suivant :

```
KinesisProducer kinesis = new KinesisProducer();

FutureCallback<UserRecordResult> myCallback = new FutureCallback<UserRecordResult>() {

    @Override public void onFailure(Throwable t) {
        /* Analyze and respond to the failure */
    };
    @Override public void onSuccess(UserRecordResult result) {
        /* Respond to the success */
    };
};

for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    ListenableFuture<UserRecordResult> f = kinesis.addUserRecord("myStream",
"myPartitionKey", data);
    // If the Future is complete by the time we call addCallback, the callback will be
invoked immediately.
    Futures.addCallback(f, myCallback);
}
```

Configuration de la bibliothèque Kinesis Producer

Bien que les paramètres par défaut fonctionnent en principe bien pour la plupart des cas d'utilisation, vous pouvez en modifier certains paramètres par défaut pour adapter le comportement de `KinesisProducer` à vos besoins. Pour cela, une instance de la classe

`KinesisProducerConfiguration` peut être passée au constructeur `KinesisProducer`, par exemple :

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
    .setRecordMaxBufferedTime(3000)
    .setMaxConnections(1)
    .setRequestTimeout(60000)
    .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

Vous pouvez également charger une configuration à partir d'un fichier de propriétés :

```
KinesisProducerConfiguration config =
    KinesisProducerConfiguration.fromPropertiesFile("default_config.properties");
```

Vous pouvez remplacer un chemin d'accès et un nom de fichier auxquels le processus de l'utilisateur peut accéder. Vous pouvez aussi appeler les méthodes `set` sur l'instance `KinesisProducerConfiguration` créée de cette façon afin de personnaliser la configuration.

Le fichier de propriétés doit spécifier les paramètres en utilisant leur nom dans `PascalCase`. Les noms correspondent à ceux qui sont utilisés dans les méthodes `set` de la classe `KinesisProducerConfiguration`. Par exemple :

```
RecordMaxBufferedTime = 100
MaxConnections = 4
RequestTimeout = 6000
Region = us-west-1
```

Pour plus d'informations sur les règles d'utilisation des paramètres de configuration et les limites de valeurs, consultez [l'exemple de fichier de propriétés de configuration sur GitHub](#).

Notez que, une fois que `KinesisProducer` est initialisé, la modification de l'instance `KinesisProducerConfiguration` qui a été utilisée n'a aucun autre effet. `KinesisProducer` ne soutient pas actuellement la reconfiguration dynamique.

Mettre en œuvre la désagrégation des consommateurs

À partir de la version 1.4.0, la désagrégation automatique des enregistrements KPL utilisateur est prise en charge. Le code d'application grand public écrit avec les versions précédentes du KCL

sera compilé sans aucune modification après la mise à jour du KCL. Toutefois, si KPL l'agrégation est utilisée du côté du producteur, il existe une subtilité liée au point de contrôle : tous les sous-enregistrements d'un enregistrement agrégé ont le même numéro de séquence. Des données supplémentaires doivent donc être stockées avec le point de contrôle si vous devez faire la distinction entre les sous-enregistrements. Ces données supplémentaires s'appellent le numéro de sous-séquence.

Options

- [Migrer à partir des versions précédentes du KCL](#)
- [Utiliser des KCL extensions pour la KPL désagrégation](#)
- [Utiliser GetRecords directement](#)

Migrer à partir des versions précédentes du KCL

Vous n'avez pas besoin de modifier vos appels existants pour effectuer le contrôle associé au regroupement. L'extraction de tous les enregistrements stockés avec succès dans Kinesis Data Streams demeure garantie. KCLII propose désormais deux nouvelles opérations de point de contrôle pour prendre en charge des cas d'utilisation particuliers, décrits ci-dessous.

Si votre code existant a été écrit pour le KPL support KCL antérieur et que votre opération de point de contrôle est appelée sans arguments, cela revient à vérifier le numéro de séquence du dernier enregistrement KPL utilisateur du lot. Si votre opération de contrôle est appelée avec une chaîne de numéro de séquence, cela équivaut à contrôler le numéro de séquence donné du lot ainsi que le numéro de sous-séquence implicite 0 (zéro).

L'appel de la nouvelle opération de KCL point de contrôle `checkpoint()` sans aucun argument équivaut sémantiquement à vérifier le numéro de séquence du dernier `Record` appelé du lot, ainsi que le numéro de sous-séquence implicite 0 (zéro).

L'appel de la nouvelle opération de KCL point de contrôle `checkpoint(Record record)` équivaut sémantiquement à vérifier le numéro de séquence `Record` du point donné avec le numéro de sous-séquence implicite 0 (zéro). Si l'appel `Record` est en fait un `UserRecord`, le numéro de séquence et le numéro de sous-séquence `UserRecord` sont contrôlés.

L'appel de la nouvelle opération de KCL point de contrôle `vérifie_checkpoint(String sequenceNumber, long subSequenceNumber)` explicitement le numéro de séquence donné ainsi que le numéro de sous-séquence donné.

Dans tous les cas, une fois le point de contrôle enregistré dans la table des points de contrôle Amazon DynamoDB, la récupération des enregistrements peut reprendre correctement KCL, même lorsque l'application se bloque et redémarre. Si la séquence contient plusieurs enregistrements, l'extraction commence par l'enregistrement ayant le numéro de sous-séquence suivant dans l'enregistrement ayant le numéro de séquence contrôlé le plus récemment. Si le dernier contrôle incluait le tout dernier numéro de sous-séquence de l'enregistrement ayant le numéro de séquence précédent, l'extraction commence par l'enregistrement avec le numéro de séquence suivant.

La section suivante décrit en détail le contrôle de séquence et de sous-séquence pour les consommateurs qui doivent éviter de sauter ou de dupliquer des enregistrements. S'il importe peu que des enregistrements soient sautés (ou dupliqués) lors de l'arrêt ou du redémarrage du traitement des enregistrements du consommateur, vous pouvez exécuter votre code existant sans modification.

Utiliser des KCL extensions pour la KPL désagrégation

KPL la désagrégation peut impliquer un point de contrôle des sous-séquences. Pour faciliter l'utilisation du point de contrôle de sous-séquence, une `UserRecord` classe a été ajoutée au : KCL

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
        /* ... */
    }
    @Override
    public int hashCode() {
        /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
        /* contract-satisfying implementation */
    }
}
```

Cette classe est maintenant utilisée à la place de `Record`. Elle ne casse pas le code existant, car c'est une sous-classe de `Record`. La classe `UserRecord` représente à la fois les sous-enregistrements réels et les enregistrements non regroupés standard. Les enregistrements non regroupés peuvent être considérés comme des enregistrements regroupés contenant un seul sous-enregistrement.

En outre, deux nouvelles opérations ont été ajoutées à `IRecordProcessorCheckpoint` :

```
public void checkpoint(Record record);
```

```
public void checkpoint(String sequenceNumber, long subSequenceNumber);
```

Pour commencer à utiliser le contrôle du numéro de sous-séquence, vous pouvez effectuer la conversion ci-dessous. Modifiez le code de formulaire suivant :

```
checkpointer.checkpoint(record.getSequenceNumber());
```

Nouveau code de formulaire :

```
checkpointer.checkpoint(record);
```

Nous vous recommandons d'utiliser le formulaire `checkpoint(Record record)` pour le contrôle de sous-séquence. Toutefois, si vous stockez déjà `sequenceNumbers` dans les chaînes à utiliser pour les points de contrôle, vous devez désormais stocker aussi `subSequenceNumber`, comme l'illustre l'exemple suivant :

```
String sequenceNumber = record.getSequenceNumber();
long subSequenceNumber = ((UserRecord) record).getSubSequenceNumber(); // ... do other
processing
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

Le cast from `Record` to réussit `UserRecord` toujours car l'implémentation utilise `UserRecord` toujours. À moins qu'il ne soit nécessaire d'effectuer des opérations arithmétiques sur les numéros de séquence, cette approche n'est pas recommandée.

Lors du traitement des enregistrements KPL utilisateur, le numéro de sous-séquence KCL écrit le numéro de sous-séquence dans Amazon DynamoDB sous forme de champ supplémentaire pour chaque ligne. Les versions précédentes du étaient KCL utilisées `AFTER_SEQUENCE_NUMBER` pour récupérer des enregistrements lors de la reprise des points de contrôle. Le courant KCL avec KPL support utilise à la `AT_SEQUENCE_NUMBER` place. Lorsque l'enregistrement situé au numéro de séquence contrôlé est extrait, le numéro de sous-séquence contrôlé est vérifié, et les sous-enregistrements sont abandonnés le cas échéant (ce peut être tous si le dernier sous-enregistrement est celui qui est contrôlé). Encore, les enregistrements non regroupés peuvent être considérés comme des enregistrements regroupés contenant un seul sous-enregistrement, si bien que le même algorithme fonctionne aussi bien pour les enregistrements regroupés que pour les enregistrements non regroupés.

Utiliser GetRecords directement

Vous pouvez également choisir de ne pas utiliser l'API opération KCL mais de l'invoquer `GetRecords` directement pour récupérer les enregistrements Kinesis Data Streams. Pour décompresser ces enregistrements récupérés dans vos dossiers KPL utilisateur d'origine, effectuez l'une des opérations statiques suivantes dans `UserRecord.java` :

```
public static List<Record> deaggregate(List<Record> records)

public static List<UserRecord> deaggregate(List<UserRecord> records, BigInteger
    startingHashKey, BigInteger endingHashKey)
```

La première opération utilise la valeur par défaut 0 (zéro) pour `startingHashKey` et la valeur par défaut $2^{128} - 1$ pour `endingHashKey`.

Chacune de ces opérations désagrège la liste donnée d'enregistrements Kinesis Data Streams en une liste d'enregistrements utilisateur. KPL Tous les enregistrements KPL utilisateur dont la clé de hachage ou de partition explicite se situe en dehors de la plage comprise entre `startingHashKey` (inclus) et `endingHashKey` (inclus) sont supprimés de la liste d'enregistrements renvoyée.

Utilisez le KPL avec Amazon Data Firehose

Si vous utilisez la bibliothèque Kinesis Producer (KPL) pour écrire des données dans un flux de données Kinesis, vous pouvez utiliser l'agrégation pour combiner les enregistrements que vous écrivez dans ce flux de données Kinesis. Si vous utilisez ensuite ce flux de données comme source pour votre flux de diffusion Firehose, Firehose désagrège les enregistrements avant de les livrer à destination. Si vous configurez votre flux de diffusion pour transformer les données, Firehose désagrège les enregistrements avant de les transmettre. AWS Lambda Pour plus d'informations, consultez [Écriture de données à Amazon Kinesis Firehose à l'aide de flux de données Kinesis](#) (français non garanti).

Utiliser le KPL avec le registre des AWS Glue schémas

Vous pouvez intégrer vos flux de données Kinesis au registre des AWS Glue schémas. Le registre des AWS Glue schémas vous permet de découvrir, de contrôler et de faire évoluer les schémas de manière centralisée, tout en garantissant que les données produites sont validées en permanence par un schéma enregistré. Un schéma définit la structure et le format d'un enregistrement de données. Un schéma est une spécification versionnée pour la publication, la consommation ou le stockage des données fiables. Le registre des AWS Glue schémas vous permet d'améliorer la qualité

end-to-end des données et la gouvernance des données au sein de vos applications de streaming. Pour plus d'informations, consultez le registre [AWS Glue Schema](#) (français non garanti). L'un des moyens de configurer cette intégration consiste à utiliser les bibliothèques KPL et Kinesis Client Library (KCL) en Java.

Important

Actuellement, l'intégration de Kinesis Data Streams AWS Glue et de registres de schémas n'est prise en charge que pour les flux de données Kinesis qui KPL utilisent des producteurs implémentés en Java. La prise en charge multilingue n'est pas fournie.

Pour obtenir des instructions détaillées sur la façon de configurer l'intégration de Kinesis Data Streams à Schema Registry à l'aide de, consultez KPL la section « Interaction avec les données en utilisant KPL lesKCL/Libraries » [dans Use Case : Integrating Amazon Kinesis Data Streams with the AWS Glue Schema Registry](#).

Configuration de la configuration du KPL proxy

Pour les applications qui ne peuvent pas se connecter directement à Internet, tous les AWS SDK clients prennent en charge l'utilisation de HTTP HTTPS proxys. Au sein d'un environnement d'entreprise classique, tout le trafic réseau sortant doit passer par des serveurs proxy. Si votre application utilise Kinesis Producer Library (KPL) pour collecter et envoyer des données AWS dans un environnement qui utilise des serveurs proxy, elle doit être configurée KPL par proxy. KPL est une bibliothèque de haut niveau basée sur le AWS KinesisSDK. Elle est divisée en un processus natif et un encapsuleur. Le processus natif exécute toutes les tâches de traitement et d'envoi des enregistrements, tandis que l'encapsuleur gère le processus natif et communique avec lui. Pour obtenir plus d'informations, consultez [Implémentation de producteurs efficaces et fiables avec la bibliothèque producteur Amazon Kinesis](#) (français non garanti).

Le wrapper est écrit en Java et le processus natif est écrit en C++ à l'aide de KinesisSDK. KPL les versions 0.14.7 et supérieures prennent désormais en charge la configuration du proxy dans le wrapper Java qui peut transmettre toutes les configurations de proxy au processus natif. Pour plus d'informations, consultez <https://github.com/aws-labs/amazon-kinesis-producer/releases/tag/v0.14.7>.

Vous pouvez utiliser le code suivant pour ajouter des configurations de proxy à vos KPL applications.

```
KinesisProducerConfiguration configuration = new KinesisProducerConfiguration();
```

```
// Next 4 lines used to configure proxy
configuration.setProxyHost("10.0.0.0"); // required
configuration.setProxyPort(3128); // default port is set to 443
configuration.setProxyUserName("username"); // no default
configuration.setProxyPassword("password"); // no default

KinesisProducer kinesisProducer = new KinesisProducer(configuration);
```

Développez les producteurs en utilisant Amazon Kinesis Data API Streams avec AWS SDK for Java

Vous pouvez développer des producteurs à l'aide d'Amazon Kinesis Data API Streams avec AWS SDK le logiciel pour Java. Si vous ne connaissez pas Kinesis Data Streams, commencez par vous familiariser avec les concepts et la terminologie présentés dans [Qu'est-ce qu'Amazon Kinesis Data Streams ?](#) et [Utilisez le AWS CLI pour effectuer des opérations Amazon Kinesis Data Streams.](#)

Ces exemples présentent les [Kinesis Data API Streams](#) et utilisent [AWS SDKle for Java pour](#) ajouter (mettre) des données à un flux. Toutefois, dans la plupart des cas d'utilisation, vous devriez préférer la bibliothèque Kinesis Data KPL Streams. Pour de plus amples informations, veuillez consulter [Développez des producteurs à l'aide de la bibliothèque Amazon Kinesis Producer \(\) KPL.](#)

L'exemple de code Java présenté dans ce chapitre montre comment effectuer des opérations Kinesis Data API Streams de base. Il est divisé logiquement par type d'opération. Ces exemples ne représentent pas du code prêt à la production, car ils ne recherchent pas toutes les exceptions possibles ou ne tiennent pas compte de toutes les considérations possibles en matière de sécurité ou de performances. Vous pouvez également appeler les [Kinesis Data API Streams](#) à l'aide d'autres langages de programmation. Pour plus d'informations sur toutes les options disponibles AWS SDKs, consultez [Commencer à développer avec Amazon Web Services.](#)

Chaque tâche a des conditions préalables ; par exemple, vous ne pouvez pas ajouter de données à un flux tant que vous n'avez pas créé de flux, ce qui demande de créer un client. Pour de plus amples informations, veuillez consulter [Création et gestion de flux de données Kinesis.](#)

Rubriques

- [Ajouter des données à un flux](#)
- [Interagissez avec les données à l'aide du registre des AWS Glue schémas](#)

Ajouter des données à un flux

Une fois qu'un flux est créé, vous pouvez y ajouter des données sous forme d'enregistrements. Un enregistrement est une structure de données qui contient les données à traiter sous la forme d'un blob de données. Une fois que vous avez stocké les données dans l'enregistrement, Kinesis Data Streams n'inspecte pas, n'interprète pas ou ne modifie absolument pas les données. Chaque enregistrement a également un numéro de séquence et une clé de partition qui lui sont associés.

Les Kinesis Data API Streams proposent deux opérations différentes qui ajoutent des données à un flux [PutRecords](#), et [PutRecord](#). L'[PutRecords](#) opération envoie plusieurs enregistrements à votre flux par HTTP demande, et l'[PutRecord](#) opération unique envoie des enregistrements à votre flux un par un (une HTTP demande distincte est requise pour chaque enregistrement). Vous préférerez sans doute utiliser [PutRecords](#) pour la plupart des applications, car cette opération permet d'atteindre un débit supérieur par application producteur. Pour plus d'informations sur chacune de ces opérations, consultez les sous-sections distinctes ci-dessous.

Rubriques

- [Ajoutez plusieurs enregistrements avec PutRecords](#)
- [Ajoutez un seul enregistrement avec PutRecord](#)

Gardez toujours à l'esprit que, lorsque votre application source ajoute des données au flux à l'aide des Kinesis Data API Streams, il est fort probable qu'une ou plusieurs applications grand public traitent simultanément des données en dehors du flux. Pour plus d'informations sur la manière dont les consommateurs obtiennent des données à l'aide des Kinesis Data API Streams, [Obtenir des données à partir d'un flux](#) consultez.

Important

[Modifier la période de conservation des données](#)

Ajoutez plusieurs enregistrements avec PutRecords

L'opération [PutRecords](#) envoie plusieurs enregistrements à Kinesis Data Streams dans une seule demande. En utilisant [PutRecords](#), les applications producteur peuvent atteindre un débit supérieur lors de l'envoi de données à leur flux de données Kinesis. Chaque demande [PutRecords](#) peut prendre en charge jusqu'à 500 enregistrements. Chaque enregistrement de la demande

peut atteindre 1 Mo, jusqu'à une limite de 5 Mo pour l'ensemble de la demande, y compris les clés de partition. Comme avec la seule opération `PutRecord` décrite ci-dessous, `PutRecords` utilise des numéros de séquence et des clés de partition. Toutefois, le paramètre `PutRecordSequenceNumberForOrdering` n'est pas inclus dans un appel `PutRecords`. L'opération `PutRecords` tente de traiter tous les enregistrements dans l'ordre naturel de la demande.

Chaque enregistrement de données a un numéro de séquence unique. Le numéro de séquence est attribué par Kinesis Data Streams une fois que vous avez appelé `client.putRecords` pour ajouter les enregistrements de données au flux. Les numéros de séquence correspondant à une même clé de partition deviennent généralement de plus en plus longs au fil du temps ; plus l'intervalle de temps entre chaque demande `PutRecords` est élevé, plus les numéros de séquence sont longs.

Note

Les numéros de séquence ne peuvent pas servir d'index aux ensembles de données d'un même flux. Pour séparer logiquement les ensembles de données, utilisez des clés de partition ou créez un flux distinct pour chaque ensemble de données.

Une demande `PutRecords` peut inclure des enregistrements ayant différentes clés de partition. La portée de la demande est un flux ; chaque demande peut inclure une combinaison de clés de partition et d'enregistrements allant jusqu'aux limites définies pour la demande. Les demandes effectuées avec de nombreuses clés de partition différentes pour des flux comportant de nombreuses partitions différentes sont généralement plus rapides que les demandes comportant un petit nombre de clés de partition pour un petit nombre de partitions. Le nombre de clés de partition doit être beaucoup plus grand que le nombre de partitions pour réduire la latence et optimiser le débit.

PutRecordsexemple

Le code suivant crée 100 enregistrements de données avec des clés de partition séquentielles et les place dans un flux appelé `DataStream`.

```
AmazonKinesisClientBuilder clientBuilder =
AmazonKinesisClientBuilder.standard();

clientBuilder.setRegion(regionName);
clientBuilder.setCredentials(credentialsProvider);
clientBuilder.setClientConfiguration(config);
```

```
AmazonKinesis kinesisClient = clientBuilder.build();

PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(streamName);
List <PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new
PutRecordsRequestEntry();

putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult =
kinesisClient.putRecords(putRecordsRequest);
System.out.println("Put Result" + putRecordsResult);
```

La réponse `PutRecords` comprend un tableau de réponse `Records`. Chaque enregistrement compris dans ce tableau de réponse correspond directement à un enregistrement dans le tableau de demande. Ces entrées sont classées dans l'ordre naturel, soit de haut en bas de la demande et de la réponse. Le tableau de réponse `Records` comprend toujours le même nombre d'enregistrements que le tableau de demande.

Gérez les défaillances lors de l'utilisation `PutRecords`

Par défaut, la défaillance d'enregistrements individuels dans une demande n'arrête pas le traitement des enregistrements suivants dans une demande `PutRecords`. Cela signifie qu'un tableau `Records` de réponse comprend à la fois des enregistrements traités avec succès et sans succès. Vous devez détecter les enregistrements traités sans succès et les inclure dans un appel ultérieur.

Les enregistrements qui ont réussi incluent les valeurs `SequenceNumber` et `ShardID`. Ceux qui ont échoué incluent les valeurs `ErrorCode` et `ErrorMessage`. Le paramètre `ErrorCode` reflète le type d'erreur et peut avoir une des valeurs suivantes : `ProvisionedThroughputExceededException` ou `InternalFailure`. `ErrorMessage` fournit des informations plus détaillées sur l'exception `ProvisionedThroughputExceededException`, y compris l'ID de compte, le nom du flux et les ID de partition de l'enregistrement qui a été limité. L'exemple ci-dessous contient trois enregistrements dans une demande `PutRecords`. Le second enregistrement échoue et est reflété dans la réponse.

Exemple PutRecords Syntaxe de la demande

```
{
  "Records": [
    {
      "Data": "XzxkYXRhPl8w",
      "PartitionKey": "partitionKey1"
    },
    {
      "Data": "AbceddeRFfg12asd",
      "PartitionKey": "partitionKey1"
    },
    {
      "Data": "KFpcd98*7nd1",
      "PartitionKey": "partitionKey3"
    }
  ],
  "StreamName": "myStream"
}
```

Exemple PutRecords Syntaxe de réponse

```
{
  "FailedRecordCount": 1,
  "Records": [
    {
      "SequenceNumber": "21269319989900637946712965403778482371",
      "ShardId": "shardId-000000000001"
    },
    {
      "ErrorCode": "ProvisionedThroughputExceededException",
      "ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream exampleStreamName under account 111111111111."
    },
    {
      "SequenceNumber": "21269319989999637946712965403778482985",
      "ShardId": "shardId-000000000002"
    }
  ]
}
```

Les enregistrements qui ont été traités sans succès peuvent être inclus dans des demandes `PutRecords` ultérieures. Tout d'abord, vérifiez le paramètre `FailedRecordCount` de `putRecordsResult` afin de savoir si la demande comporte des enregistrements d'échecs. Dans ce cas, chaque `putRecordsEntry` comportant un `ErrorCode` qui n'est pas `null` doit être ajouté à une demande ultérieure. Pour un exemple de ce type de gestionnaire, reportez-vous au code suivant.

Exemple `PutRecords` gestionnaire de défaillances

```
PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(myStreamName);
List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int j = 0; j < 100; j++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
    putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(j).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d", j));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);

while (putRecordsResult.getFailedRecordCount() > 0) {
    final List<PutRecordsRequestEntry> failedRecordsList = new ArrayList<>();
    final List<PutRecordsResultEntry> putRecordsResultEntryList =
putRecordsResult.getRecords();
    for (int i = 0; i < putRecordsResultEntryList.size(); i++) {
        final PutRecordsRequestEntry putRecordRequestEntry =
putRecordsRequestEntryList.get(i);
        final PutRecordsResultEntry putRecordsResultEntry =
putRecordsResultEntryList.get(i);
        if (putRecordsResultEntry.getErrorCode() != null) {
            failedRecordsList.add(putRecordRequestEntry);
        }
    }
    putRecordsRequestEntryList = failedRecordsList;
    putRecordsRequest.setRecords(putRecordsRequestEntryList);
    putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);
}
```

Ajoutez un seul enregistrement avec PutRecord

Chaque appel de [PutRecord](#) est exécuté sur un seul enregistrement. Préférez l'opération PutRecords décrite dans [Ajoutez plusieurs enregistrements avec PutRecords](#), sauf si votre application a particulièrement besoin de toujours envoyer des enregistrements uniques par demande ou qu'une autre raison empêche l'utilisation de PutRecords.

Chaque enregistrement de données a un numéro de séquence unique. Le numéro de séquence est attribué par Kinesis Data Streams une fois que vous avez appelé `client.putRecord` pour ajouter l'enregistrement de données au flux. Les numéros de séquence correspondant à une même clé de partition deviennent généralement de plus en plus longs au fil du temps ; plus l'intervalle de temps entre chaque demande PutRecord est élevé, plus les numéros de séquence sont longs.

Lorsque des opérations se succèdent rapidement, il n'est pas sûr que les numéros de séquence renvoyés augmentent, car ces opérations semblent surtout simultanées pour Kinesis Data Streams. Pour garantir une stricte augmentation des numéros de séquence pour la même clé de partition, utilisez le paramètre `SequenceNumberForOrdering`, comme il est illustré dans l'exemple de code [PutRecordexemple](#).

Que vous utilisiez `SequenceNumberForOrdering` ou non, les enregistrements que Kinesis Data Streams reçoit via un appel `GetRecords` sont strictement classés par numéro de séquence.

Note

Les numéros de séquence ne peuvent pas servir d'index aux ensembles de données d'un même flux. Pour séparer logiquement les ensembles de données, utilisez des clés de partition ou créez un flux distinct pour chaque ensemble de données.

La clé de partition sert à regrouper les données dans le flux. Un enregistrement de données est attribué à une partition du flux suivant sa clé de partition. Plus précisément, Kinesis Data Streams utilise la clé de partition comme entrée d'une fonction de hachage qui mappe la clé de partition (et les données associées) à une partition spécifique.

Ce mécanisme de hachage a pour effet que tous les enregistrements de données ayant la même clé de partition sont mappés à la même partition du flux. Toutefois, si le nombre de clés de partition dépasse le nombre de partitions, certaines partitions contiennent nécessairement des enregistrements ayant des clés de partition différentes. Du point de vue de la conception, afin de garantir que toutes vos partitions sont bien utilisées, le nombre de partitions (spécifié par la méthode

`setShardCount` de `CreateStreamRequest`) doit être nettement inférieur à celui des partitions uniques, et la quantité de données qui passe dans une clé de partition unique doit être nettement inférieure à la capacité de la partition.

PutRecordexemple

Le code suivant crée dix enregistrements de données répartis entre deux clés de partition et les place dans un flux appelé `myStreamName`.

```
for (int j = 0; j < 10; j++)
{
    PutRecordRequest putRecordRequest = new PutRecordRequest();
    putRecordRequest.setStreamName( myStreamName );
    putRecordRequest.setData(ByteBuffer.wrap( String.format( "testData-%d",
j ).getBytes() ));
    putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
    putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
    PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
    sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

L'exemple de code précédent utilise le paramètre `setSequenceNumberForOrdering` pour garantir un ordre croissant dans chaque clé de partition. Pour utiliser ce paramètre efficacement, définissez le `SequenceNumberForOrdering` de l'enregistrement en cours (enregistrement `n`) sur le numéro de séquence de l'enregistrement précédent (enregistrement `n-1`). Pour obtenir le numéro de séquence d'un enregistrement qui a été ajouté au flux, appelez `getSequenceNumber` sur le résultat de `putRecord`.

Le paramètre `SequenceNumberForOrdering` garantit des numéros de séquence strictement croissants pour la même clé de partition. `SequenceNumberForOrdering` ne propose pas l'ordre des enregistrements sur plusieurs clés de partition.

Interagissez avec les données à l'aide du registre des AWS Glue schémas

Vous pouvez intégrer vos flux de données Kinesis au registre des AWS Glue schémas. Le registre des AWS Glue schémas vous permet de découvrir, de contrôler et de faire évoluer les schémas de manière centralisée, tout en garantissant que les données produites sont validées en permanence par un schéma enregistré. Un schéma définit la structure et le format d'un enregistrement de données. Un schéma est une spécification versionnée pour la publication, la consommation ou le stockage des données fiables. Le registre des AWS Glue schémas vous permet d'améliorer la qualité

end-to-end des données et la gouvernance des données au sein de vos applications de streaming. Pour plus d'informations, consultez le registre [AWS Glue Schema](#) (français non garanti). L'un des moyens de configurer cette intégration consiste à utiliser PutRecords les PutRecord Kinesis Data APIs Streams disponibles dans Java AWS . SDK

Pour obtenir des instructions détaillées sur la façon de configurer l'intégration de Kinesis Data Streams au registre des schémas à l'aide PutRecords de Kinesis Data Streams PutRecord et de Kinesis Data APIs Streams, consultez la [section « Interaction avec les données à l'aide des Kinesis APIs Data Streams »](#) dans [Use Case : Integrating Amazon Kinesis Data Streams with the Glue Schema Registry](#). AWS

Écrire sur Amazon Kinesis Data Streams à l'aide de Kinesis Agent

L'agent Kinesis est une application logicielle Java autonome qui permet de collecter et d'envoyer facilement des données à Kinesis Data Streams. L'agent surveille en permanence un ensemble de fichiers et envoie de nouvelles données à votre flux. L'agent gère les rotations de fichier, les points de contrôle et les nouvelles tentatives après échec. Il diffuse toutes vos données de manière fiable, rapide et simple. Il émet également des CloudWatch métriques Amazon pour vous aider à mieux surveiller et résoudre les problèmes liés au processus de streaming.

Par défaut, les enregistrements sont analysés à partir de chaque fichier sur la base du caractère de saut de ligne (' \n '). Toutefois, l'agent peut également être configuré pour analyser les enregistrements de plusieurs lignes (consultez [Spécifiez les paramètres de configuration de l'agent](#)).

Cet agent peut être installé dans des environnements basés sur des serveurs Linux tels que des serveurs Web, serveurs de journaux ou encore serveurs de base de données. Après avoir installé l'agent, configurez-le en spécifiant les fichiers à surveiller et le flux pour les données. Une fois que l'agent est configuré, il collecte de façon durable les données depuis les fichiers et les envoie en toute fiabilité dans le flux.

Rubriques

- [Remplissez les conditions requises pour Kinesis Agent](#)
- [Téléchargez et installez l'agent](#)
- [Configuration et démarrage de l'agent](#)
- [Spécifiez les paramètres de configuration de l'agent](#)
- [Surveillez plusieurs répertoires de fichiers et écrivez dans plusieurs flux](#)
- [Utiliser l'agent pour prétraiter les données](#)

- [Utiliser les CLI commandes de l'agent](#)
- [FAQ](#)

Remplissez les conditions requises pour Kinesis Agent

- Votre système d'exploitation doit être soit Amazon Linux AMI avec la version 2015.09 ou ultérieure, soit Red Hat Enterprise Linux version 7 ou ultérieure.
- Si vous utilisez Amazon EC2 pour exécuter votre agent, lancez votre EC2 instance.
- Gérez vos AWS informations d'identification à l'aide de l'une des méthodes suivantes :
 - Spécifiez un IAM rôle lorsque vous lancez votre EC2 instance.
 - Spécifiez AWS les informations d'identification lorsque vous configurez l'agent (voir [awsAccessKeyId](#) et [awsSecretAccessKey](#)).
 - Modifiez `/etc/sysconfig/aws-kinesis-agent` pour spécifier votre région et vos clés AWS d'accès.
 - Si votre EC2 instance se trouve sur un autre AWS compte, créez un IAM rôle permettant d'accéder au service Kinesis Data Streams et spécifiez ce rôle lorsque vous configurez l'agent ([assumeRoleARN](#) voir [assumeRoleExternalID](#)). Utilisez l'une des méthodes précédentes pour spécifier les AWS informations d'identification d'un utilisateur de l'autre compte autorisé à assumer ce rôle.
- Le IAM rôle ou les AWS informations d'identification que vous spécifiez doivent être autorisés à effectuer l'opération Kinesis Data [PutRecords](#)Streams pour que l'agent envoie des données à votre flux. Si vous activez la CloudWatch surveillance pour l'agent, l'autorisation d'effectuer l'opération [PutMetricData](#) est également nécessaire. Pour plus d'informations, consultez [Contrôler l'accès aux ressources Amazon Kinesis Data Streams à l'aide de IAM](#) [Surveillez l'état de santé des agents Kinesis Data Streams avec Amazon CloudWatch](#), et [Contrôle CloudWatch d'accès](#).

Téléchargez et installez l'agent

Commencez par vous connecter à votre instance. Pour plus d'informations, consultez [Connect to Your Instance](#) dans le guide de EC2 l'utilisateur Amazon. Si vous rencontrez des difficultés pour vous connecter, consultez la section [Résolution des problèmes liés à la connexion à votre instance](#) dans le guide de EC2 l'utilisateur Amazon.

Pour configurer l'agent à l'aide d'Amazon Linux AMI

Utilisez la commande suivante pour télécharger et installer l'agent :

```
sudo yum install -y aws-kinesis-agent
```

Pour configurer l'agent à l'aide de Red Hat Enterprise Linux

Utilisez la commande suivante pour télécharger et installer l'agent :

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-latest.amzn2.noarch.rpm
```

Pour configurer l'agent à l'aide de GitHub

1. Téléchargez l'agent depuis [awlabs/ amazon-kinesis-agent](#).
2. Installez l'agent en accédant au répertoire de téléchargement et en exécutant la commande suivante :

```
sudo ./setup --install
```

Configuration de l'agent dans un conteneur Docker

L'agent Kinesis peut également être exécuté dans un conteneur via la base de conteneurs [amazonlinux](#). Utilisez le Dockerfile suivant, puis exécutez `docker build`.

```
FROM amazonlinux

RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

Configuration et démarrage de l'agent

Pour configurer et démarrer l'agent

1. Ouvrez le fichier de configuration et modifiez-le (en tant que super-utilisateur si vous utilisez les autorisations d'accès fichier par défaut) `:/etc/aws-kinesis/agent.json`

Dans ce fichier de configuration, spécifiez les fichiers ("filePattern") à partir desquels l'agent collecte les données, et le nom du flux ("kinesisStream") dans lequel l'agent envoie les données. Notez que le nom de fichier est un modèle et que l'agent reconnaît les rotations de fichier. Vous pouvez effectuer une rotation de fichier ou créer de nouveaux fichiers pas plus d'une fois par seconde. L'agent utilise l'horodatage de création de fichier pour déterminer quels fichiers doivent être suivis et tracés dans votre flux. Créer de nouveaux fichiers ou effectuer une rotation de fichier plus souvent qu'une fois par seconde ne permet pas à l'agent de différencier correctement les fichiers.

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "yourkinesisstream"
    }
  ]
}
```

2. Démarrez l'agent manuellement :

```
sudo service aws-kinesis-agent start
```

3. (Facultatif) Configurez l'agent pour qu'il soit lancé au démarrage du système :

```
sudo chkconfig aws-kinesis-agent on
```

L'agent fonctionne maintenant en arrière-plan en tant que service système. Il surveille en permanence les fichiers spécifiés et envoie des données dans le flux spécifié. L'activité de l'agent est enregistrée dans `/var/log/aws-kinesis-agent/aws-kinesis-agent.log`.

Spécifiez les paramètres de configuration de l'agent

L'agent prend en charge les deux paramètres de configuration obligatoires `filePattern` et `kinesisStream`, plus les paramètres de configuration facultatifs des fonctionnalités supplémentaires. Vous pouvez spécifier aussi bien une configuration obligatoire que facultative dans `/etc/aws-kinesis/agent.json`.

Chaque fois que vous modifiez le fichier de configuration, vous devez arrêter et démarrer l'agent en utilisant les commandes suivantes :

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Vous pouvez également utiliser la commande suivante :

```
sudo service aws-kinesis-agent restart
```

Les paramètres de configuration générale sont indiqués ci-après.

Paramètre de configuration	Description
<code>assumeRoleARN</code>	Le ARN rôle à assumer par l'utilisateur. Pour plus d'informations, consultez la section Accès délégué entre AWS comptes à l'aide de IAM rôles dans le guide de IAM l'utilisateur.
<code>assumeRoleExternalId</code>	Identifiant facultatif qui détermine qui peut assumer le rôle. Pour plus d'informations, consultez la section Comment utiliser un identifiant externe dans le guide de IAM l'utilisateur.
<code>awsAccessKeyId</code>	AWS ID de clé d'accès qui remplace les informations d'identification par défaut. Ce paramètre est prioritaire sur tous les autres fournisseurs d'informations d'identification.
<code>awsSecretAccessKey</code>	AWS clé secrète qui remplace les informations d'identification par défaut. Ce paramètre est prioritaire sur tous les autres fournisseurs d'informations d'identification.
<code>cloudwatch.emitMetrics</code>	Permet à l'agent d'émettre des métriques CloudWatch s'il est défini (true). Valeur par défaut : true
<code>cloudwatch.endpoint</code>	Le point de terminaison régional pour CloudWatch. Par défaut : <code>monitoring.us-east-1.amazonaws.com</code>

Paramètre de configuration	Description
<code>kinesis.endpoint</code>	Point de terminaison régional pour Kinesis Data Streams. Par défaut : <code>kinesis.us-east-1.amazonaws.com</code>

Les paramètres de configuration de flux sont indiqués ci-après.

Paramètre de configuration	Description
<code>dataProcessingOptions</code>	Liste des options de traitement appliquées à chaque enregistrement analysé avant qu'il ne soit envoyé au flux. Les options de traitement sont exécutées dans l'ordre spécifié. Pour plus d'informations, consultez Utiliser l'agent pour prétraiter les données .
<code>kinesisStream</code>	[Obligatoire] Nom du flux.
<code>filePattern</code>	[Obligatoire] Le répertoire et le modèle de fichier qui doivent correspondre pour être récupérés par l'agent. Pour tous les fichiers correspondant à ce modèle, l'autorisation en lecture doit être accordée à <code>aws-kinesis-agent-user</code> . Pour le répertoire contenant les fichiers, les autorisations en lecture et exécution doivent être accordées à <code>aws-kinesis-agent-user</code> .
<code>initialPosition</code>	Position initiale à partir de laquelle le fichier a commencé à être analysé. Les valeurs valides sont <code>START_OF_FILE</code> et <code>END_OF_FILE</code> . Par défaut : <code>END_OF_FILE</code>
<code>maxBufferAgeMillis</code>	Durée maximale, en millisecondes, pendant laquelle l'agent met les données en tampon avant de les envoyer dans le flux. Plage de valeurs : 1 000 à 900 000 (1 seconde à 15 minutes) Par défaut : 60 000 (1 minute)

Paramètre de configuration	Description
<code>maxBuffer SizeBytes</code>	<p>Taille maximale, en octets, pour laquelle l'agent met les données en tampon avant de les envoyer dans le flux.</p> <p>Plage de valeurs : 1 à 4 194 304 (4 Mo)</p> <p>Par défaut : 4 194 304 (4 Mo)</p>
<code>maxBuffer SizeRecords</code>	<p>Nombre maximal d'enregistrements pour lequel l'agent met les données en tampon avant de les envoyer dans le flux.</p> <p>Plage de valeurs : 1 à 500</p> <p>Par défaut : 500</p>
<code>minTimeBe tweenFile PollsMillis</code>	<p>Fréquence, en millisecondes, à laquelle l'agent interroge et analyse les fichiers surveillés pour rechercher les nouvelles données.</p> <p>Plage de valeurs : 1 ou plus</p> <p>Par défaut : 100</p>
<code>multiLine StartPattern</code>	<p>Modèle d'identification du début d'un enregistrement. Un enregistrement se compose d'une ligne qui correspond au modèle et de lignes suivantes qui ne correspondent pas au modèle. Les valeurs valides sont les expressions régulières. Par défaut, chaque nouvelle ligne comprise dans les fichiers journaux est analysée comme étant un enregistrement.</p>
<code>partition KeyOption</code>	<p>Méthode de génération de la clé de partition. Les valeurs valides sont <code>RANDOM</code> (entier généré uniquement aléatoirement) et <code>DETERMINISTIC</code> (une valeur de hachage calculée à partir des données).</p> <p>Par défaut : <code>RANDOM</code></p>

Paramètre de configuration	Description
<code>skipHeaderLines</code>	<p>Nombre de lignes que l'agent doit ignorer lors de l'analyse au début des fichiers surveillés.</p> <p>Plage de valeurs : 0 ou plus</p> <p>Par défaut : 0 (zéro)</p>
<code>truncatedRecord Terminator</code>	<p>Chaîne que l'agent utilise pour tronquer un enregistrement analysé lorsque la taille de ce dernier dépasse la taille limite d'un enregistrement Kinesis Data Streams. (1,000 Ko)</p> <p>Par défaut : <code>'\n'</code> (nouvelle ligne)</p>

Surveillez plusieurs répertoires de fichiers et écrivez dans plusieurs flux

En spécifiant plusieurs paramètres de configuration de flux, vous pouvez configurer l'agent pour surveiller plusieurs répertoires de fichiers et envoyer des données dans plusieurs flux. Dans l'exemple de configuration suivant, l'agent surveille deux répertoires de fichiers et envoie des données respectivement à un flux Kinesis et à un flux de diffusion Firehose. Notez que vous pouvez spécifier différents points de terminaison pour Kinesis Data Streams et Firehose afin que votre flux Kinesis et votre flux de diffusion Firehose ne soient pas nécessairement situés dans la même région.

```
{
  "cloudwatch.emitMetrics": true,
  "kinesis.endpoint": "https://your/kinesis/endpoint",
  "firehose.endpoint": "https://your/firehose/endpoint",
  "flows": [
    {
      "filePattern": "/tmp/app1.log*",
      "kinesisStream": "yourkinesisstream"
    },
    {
      "filePattern": "/tmp/app2.log*",
      "deliveryStream": "yourfirehosedeliverystream"
    }
  ]
}
```

```
}
```

Pour des informations plus détaillées sur l'utilisation de l'agent avec Firehose, consultez [Writing to Amazon Data Firehose with Kinesis Agent](#).

Utiliser l'agent pour prétraiter les données

L'agent peut prétraiter les enregistrements analysés à partir des fichiers surveillés avant de les envoyer dans votre flux. Vous pouvez activer cette fonctionnalité en ajoutant le paramètre de configuration `dataProcessingOptions` à votre flux de fichiers. Une ou plusieurs options de traitement peuvent être ajoutées. Elles sont exécutées dans l'ordre spécifié.

L'agent prend en charge les options de traitement répertoriées ci-après. Etant donné que l'agent est open source, vous pouvez continuer à développer et étendre ses options de traitement. Vous pouvez télécharger l'agent à partir de [Kinesis Agent](#).

Options de traitement

SINGLELINE

Convertit un enregistrement de plusieurs lignes en un enregistrement d'une seule ligne en supprimant les caractères de saut de ligne, les espaces de début et les espaces de fin.

```
{
  "optionName": "SINGLELINE"
}
```

CSVTOJSON

Convertit un enregistrement d'un format séparé par un délimiteur en JSON format.

```
{
  "optionName": "CSVTOJSON",
  "customFieldNames": [ "field1", "field2", ... ],
  "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Obligatoire] Les noms de champs utilisés comme clés dans chaque JSON paire clé-valeur. Par exemple, si vous spécifiez `["f1", "f2"]`, l'enregistrement « `v1, v2` » est converti en `{"f1": "v1", "f2": "v2"}`.

delimiter

Chaîne utilisée comme délimiteur dans l'enregistrement. La valeur par défaut est une virgule (,).

LOGTOJSON

Convertit un enregistrement d'un format journal en JSON format. Les formats de journal pris en charge sont Apache Common Log, Apache Combined Log, Apache Error Log et RFC3164 Syslog.

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "logformat",
  "matchPattern": "yourregexpattern",
  "customFieldNames": [ "field1", "field2", ... ]
}
```

logFormat

[Obligatoire] Format d'entrée de journal. Les valeurs admises sont les suivantes :

- COMMONAPACHELOG : le format de journal courant d'Apache. Chaque entrée de journal a le schéma suivant par défaut : « `%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}` ».
- COMBINEDAPACHELOG : le format de journal combiné d'Apache. Chaque entrée de journal a le schéma suivant par défaut : « `%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}` ».
- APACHEERRORLOG : le format de journal d'erreurs d'Apache. Chaque entrée de journal a le schéma suivant par défaut : « `[%{timestamp}] [%{module}:%{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}` ».
- SYSLOG— Le format RFC3164 Syslog. Chaque entrée de journal a le schéma suivant par défaut : « `%{timestamp} %{hostname} %{program}[%{processid}]: %{message}` ».

matchPattern

Modèle d'expression régulière utilisé pour extraire les valeurs des entrées de journal. Ce paramètre est utilisé si votre entrée de journal n'a pas l'un des formats de journalisation prédéfinis. Si ce paramètre est utilisé, vous devez également spécifier `customFieldNames`.

customFieldNames

Les noms de champs personnalisés utilisés comme clés dans chaque paire JSON clé-valeur. Vous pouvez utiliser ce paramètre pour définir les noms de champ pour les valeurs extraites de `matchPattern`, ou remplacer les noms de champ par défaut des formats de journalisation prédéfinis.

Exemple : LOGTOJSON Configuration

Voici un exemple de LOGTOJSON configuration pour une entrée Apache Common Log convertie au JSON format :

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG"
}
```

Avant la conversion :

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
HTTP/1.1" 200 6291
```

Après la conversion :

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
HTTP/1.1","response":"200","bytes":"6291"}
```

Exemple : LOGTOJSON Configuration avec champs personnalisés

Voici un autre exemple de configuration LOGTOJSON :

```
{
  "optionName": "LOGTOJSON",
  "logFormat": "COMMONAPACHELOG",
  "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Avec ce paramètre de configuration, la même entrée Apache Common Log que dans l'exemple précédent est convertie au JSON format suivant :

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

Exemple : Conversion d'une entrée au format Journal courant Apache

La configuration de flux suivante convertit une entrée Apache Common Log en un enregistrement d'une seule ligne au JSON format :

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "my-stream",
      "dataProcessingOptions": [
        {
          "optionName": "LOGTOJSON",
          "logFormat": "COMMONAPACHELOG"
        }
      ]
    }
  ]
}
```

Exemple : Conversion des enregistrements de plusieurs lignes

La configuration de flux suivante analyse les enregistrements de plusieurs lignes dont la première ligne commence par « [SEQUENCE= ». Chaque enregistrement est d'abord converti en un enregistrement d'une seule ligne. Les valeurs sont ensuite extraites de l'enregistrement sur la base d'un séparateur tabulation. Les valeurs extraites sont mappées aux customFieldNames valeurs spécifiées pour former un enregistrement d'une seule ligne au JSON format.

```
{
  "flows": [
    {
      "filePattern": "/tmp/app.log*",
      "kinesisStream": "my-stream",
      "multiLineStartPattern": "\\[SEQUENCE=",
      "dataProcessingOptions": [
        {
          "optionName": "SINGLELINE"
        }
      ],
    }
  ]
}
```

```

        {
            "optionName": "CSVTOJSON",
            "customFieldNames": [ "field1", "field2", "field3" ],
            "delimiter": "\\t"
        }
    ]
}

```

Exemple : LOGTOJSON Configuration avec Match Pattern

Voici un exemple de LOGTOJSON configuration pour une entrée Apache Common Log convertie au JSON format, le dernier champ (octets) étant omis :

```

{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^(([\\d.]+) (\\S+) (\\S+) \\[[([\\w:/]+\\s[+\\-]\\d{4})\\] \\\"(.+?)\\\" (\\d{3})\"",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request", "response"]
}

```

Avant la conversion :

```

123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
200

```

Après la conversion :

```

{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
-0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}

```

Utiliser les CLI commandes de l'agent

Lancer automatiquement l'agent au démarrage du système :

```

sudo chkconfig aws-kinesis-agent on

```

Vérifier le statut de l'agent :

```
sudo service aws-kinesis-agent status
```

Arrêter l'agent :

```
sudo service aws-kinesis-agent stop
```

Lire le fichier journal de l'agent à partir de cet emplacement :

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Désinstaller l'agent :

```
sudo yum remove aws-kinesis-agent
```

FAQ

Existe-t-il un Kinesis Agent pour Windows ?

[Kinesis Agent pour Windows](#) est un logiciel différent de Kinesis Agent pour les plateformes Linux.

Pourquoi Kinesis Agent ralentit-il ou **RecordSendErrors** augmente-t-il ?

Cela est généralement dû à une limitation de Kinesis. Vérifiez la `WriteProvisionedThroughputExceeded` métrique pour Kinesis Data Streams ou `ThrottledRecords` la métrique pour Firehose Delivery Streams. Toute augmentation de ces métriques par rapport à 0 indique que les limites de flux doivent être augmentées. Pour plus d'informations, consultez les rubriques [Limites des flux de données Kinesis](#) (français non garanti) et [Flux de diffusion Amazon Firehose](#) (français non garanti).

Une fois que vous avez exclu la limitation, vérifiez si Kinesis Agent est configuré pour suivre un grand nombre de petits fichiers. Il y a un délai lorsque Kinesis Agent suit un nouveau fichier, c'est pourquoi Kinesis Agent doit suivre un petit nombre de fichiers plus volumineux. Essayez de regrouper vos fichiers journaux dans des fichiers plus volumineux.

Pourquoi est-ce que je reçois des exceptions **java.lang.OutOfMemoryError** ?

Kinesis Agent ne dispose pas de suffisamment de mémoire pour gérer sa charge de travail actuelle. Essayez d'augmenter `JAVA_START_HEAP` et `JAVA_MAX_HEAP` dans `/usr/bin/start-aws-kinesis-agent` et de redémarrer l'agent.

Pourquoi est-ce que je reçois des exceptions **IllegalStateException** : **connection pool shut down** ?

Kinesis Agent ne dispose pas de suffisamment de connexions pour gérer sa charge de travail actuelle. Essayez d'augmenter `maxConnections` et `maxSendingThreads` dans les paramètres de configuration de votre agent général à `/etc/aws-kinesis/agent.json`. La valeur par défaut de ces champs est 12 fois supérieure au nombre de processeurs d'exécution disponibles. Consultez [AgentConfiguration.java](#) pour en savoir plus sur les paramètres de configuration avancés des agents.

Comment puis-je déboguer un autre problème avec Kinesis Agent ?

Les journaux de niveau DEBUG peuvent être activés dans `/etc/aws-kinesis/log4j.xml`.

Comment dois-je configurer Kinesis Agent ?

Plus la taille de `maxBufferSizeBytes` est petite, plus Kinesis Agent envoie fréquemment des données. Cela peut être une bonne chose, car cela réduit le délai de livraison des enregistrements, mais cela augmente également le nombre de requêtes par seconde adressées à Kinesis.

Pourquoi Kinesis Agent envoie-t-il des enregistrements dupliqués ?

Cela se produit en raison d'une mauvaise configuration dans le suivi des fichiers. Assurez-vous que chaque `fileFlow's filePattern` ne correspond qu'à un seul fichier. Cela peut également se produire si le mode `logrotate` utilisé est en mode `copytruncate`. Essayez de remplacer le mode par défaut ou de créer le mode pour éviter les doublons. Pour plus d'informations sur la gestion des enregistrements dupliqués, consultez la rubrique [Gestion des enregistrements dupliqués](#) (français non garanti).

Écrivez sur Kinesis Data Streams à l'aide d'autres services AWS

Les AWS services suivants peuvent s'intégrer directement à Amazon Kinesis Data Streams pour écrire des données dans des flux de données Kinesis. Passez en revue les informations relatives à chaque service qui vous intéresse et reportez-vous aux références fournies.

Rubriques

- [Écrivez dans Kinesis Data Streams à l'aide de AWS Amplify](#)
- [Écrire sur Kinesis Data Streams à l'aide d'Amazon Aurora](#)

- [Écrivez sur Kinesis Data Streams à l'aide d'Amazon CloudFront](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'Amazon Logs CloudWatch](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'Amazon Connect](#)
- [Écrivez dans Kinesis Data Streams à l'aide de AWS Database Migration Service](#)
- [Écrire sur Kinesis Data Streams à l'aide d'Amazon DynamoDB](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'Amazon EventBridge](#)
- [Écrivez dans Kinesis Data Streams à l'aide de AWS IoT Core](#)
- [Écrivez sur Kinesis Data Streams à l'aide d'Amazon Relational Database Service](#)
- [Écrire sur Kinesis Data Streams Pinpoint using Amazon](#)
- [Écrivez dans Kinesis Data Streams à l'aide de la base de données Amazon Quantum Ledger \(Amazon\) QLDB](#)

Écrivez dans Kinesis Data Streams à l'aide de AWS Amplify

Vous pouvez utiliser Amazon Kinesis Data Streams pour diffuser des données à partir de vos applications mobiles conçues avec AWS Amplify pour un traitement en temps réel. Vous pouvez ensuite créer des tableaux de bord en temps réel, capturer les exceptions et générer des alertes, formuler des recommandations et prendre d'autres décisions commerciales ou opérationnelles en temps réel. Vous pouvez également envoyer des données à d'autres services tels qu'Amazon Simple Storage Service, Amazon DynamoDB et Amazon Redshift.

Pour plus d'informations, consultez la rubrique [Utilisation d'Amazon Kinesis](#) (français non garanti) dans le Guide du développeur AWS Amplify.

Écrire sur Kinesis Data Streams à l'aide d'Amazon Aurora

Vous pouvez utiliser Amazon Kinesis Data Streams pour surveiller les activités de vos clusters de bases de données Amazon Aurora. Grâce aux flux d'activités de base de données, votre cluster de bases de données Aurora pousse les activités vers un flux de données Amazon Kinesis Data Streams en temps réel. Vous pouvez ensuite créer des applications de gestion de la conformité qui utilisent ces activités, les contrôlent et génèrent des alertes. Vous pouvez également utiliser Amazon Firehose pour stocker les données.

Pour plus d'informations, consultez la rubrique [Flux d'activités de base de données](#) dans le Guide du développeur Amazon Aurora (français non garanti).

Écrivez sur Kinesis Data Streams à l'aide d'Amazon CloudFront

Vous pouvez utiliser Amazon Kinesis Data Streams CloudFront avec des journaux en temps réel et obtenir des informations sur les demandes adressées à une distribution en temps réel. Vous pouvez ensuite créer votre propre [consommateur de flux de données Kinesis](#) ou utiliser Amazon Data Firehose pour envoyer les données de journal à Amazon S3, Amazon Redshift, Amazon OpenSearch Service ou à un service de traitement des journaux tiers.

Pour plus d'informations, consultez la section [Journaux en temps réel](#) du manuel Amazon CloudFront Developer Guide.

Écrivez sur Kinesis Data Streams à l'aide d'Amazon Logs CloudWatch

Vous pouvez utiliser des CloudWatch abonnements pour accéder à un flux en temps réel des événements du journal depuis Amazon CloudWatch Logs et le transmettre à un flux de données Kinesis à des fins de traitement, d'analyse et de chargement vers d'autres systèmes.

Pour plus d'informations, consultez la section [Traitement en temps réel des données de journal avec les abonnements](#) dans le guide de l'utilisateur Amazon CloudWatch Logs.

Écrivez sur Kinesis Data Streams à l'aide d'Amazon Connect

Vous pouvez utiliser les flux de données Kinesis Data Streams pour exporter les enregistrements de contacts et les événements des agents en temps réel à partir de votre instance Amazon Connect. Vous pouvez également activer le streaming de données à partir des profils clients Amazon Connect pour recevoir automatiquement des mises à jour d'un flux de données Kinesis concernant la création de nouveaux profils ou la modification de profils existants.

Vous pouvez ensuite créer des applications grand public pour traiter et analyser les données en temps réel. Par exemple, en utilisant les enregistrements de contacts et les données de profil client, vous pouvez conserver les données de vos systèmes sources, telles que CRMs les outils d'automatisation du marketing, up-to-date avec les informations les plus récentes. En utilisant les données relatives aux événements des agents, vous pouvez créer des tableaux de bord qui affichent les informations et les événements relatifs aux agents, et déclencher des notifications personnalisées en cas d'activités spécifiques des agents.

Pour plus d'informations, consultez les rubriques [Flux de données pour votre instance](#), [Configuration de l'exportation en temps réel](#) et [Flux d'événements de l'agent](#) dans le Guide de l'administrateur Amazon Connect (français non garanti).

Écrivez dans Kinesis Data Streams à l'aide de AWS Database Migration Service

Vous pouvez l'utiliser AWS Database Migration Service pour migrer des données vers un flux de données Kinesis. Vous pouvez ensuite créer des applications consommateur qui traitent les enregistrements de données en temps réel. Vous pouvez également facilement envoyer des données en aval vers d'autres services tels qu'Amazon Simple Storage Service, Amazon DynamoDB et Amazon Redshift

Pour plus d'informations, consultez la rubrique [Utilisation de Kinesis Data Streams](#) dans le Guide de l'utilisateur AWS Database Migration Service .

Écrire sur Kinesis Data Streams à l'aide d'Amazon DynamoDB

Vous pouvez utiliser Amazon Kinesis Data Streams pour capturer les modifications apportées à Amazon DynamoDB. Kinesis Data Streams récupère les modifications au niveau élément dans n'importe quelle table DynamoDB et les réplique dans un Kinesis Data Stream de votre choix. Vos applications consommateur peuvent accéder à ce flux pour visualiser les modifications au niveau de l'élément en temps réel et transmettre ces modifications en aval ou prendre des mesures en fonction du contenu.

Pour plus d'informations, consultez la rubrique [Fonctionnement des flux de données Kinesis Data Streams avec DynamoDB](#) dans le Guide du développeur Amazon DynamoDB.

Écrivez sur Kinesis Data Streams à l'aide d'Amazon EventBridge

Kinesis Data Streams vous permet d' AWS APIenvoyer des événements EventBridge d'[appel](#) vers un flux, de créer des applications grand public et de traiter de grandes quantités de données. Vous pouvez également utiliser Kinesis Data Streams comme cible EventBridge dans Pipes et diffuser des enregistrements sous forme de flux à partir de l'une des sources disponibles après un filtrage et un enrichissement facultatifs.

Pour plus d'informations, consultez [Envoyer des événements vers un flux Amazon Kinesis et EventBridge Pipes](#) dans le guide de EventBridge l'utilisateur Amazon.

Écrivez dans Kinesis Data Streams à l'aide de AWS IoT Core

Vous pouvez écrire des données en temps réel à partir de MQTT messages dans AWS IoT Core à l'aide des actions AWS IoT Rule. Vous pouvez ensuite créer des applications qui traitent les données,

analysent leur contenu, génèrent des alertes et les transmettent à des applications d'analyse ou à d'autres services AWS .

Pour plus d'informations, consultez la rubrique [Kinesis Data Streams](#) (français non garanti) dans le Guide du développeur AWS (français non garanti).

Écrivez sur Kinesis Data Streams à l'aide d'Amazon Relational Database Service

Vous pouvez utiliser Amazon Kinesis Data Streams pour surveiller les activités sur vos instances RDS Amazon. À l'aide des flux d'activité de base de données RDS, Amazon transmet les activités vers un flux de données Kinesis en temps réel. Vous pouvez ensuite créer des applications de gestion de la conformité qui utilisent ces activités, les contrôlent et génèrent des alertes. Vous pouvez également utiliser Amazon Data Firehose pour stocker les données.

Pour plus d'informations, consultez [Database Activity Streams](#) dans le manuel Amazon RDS Developer Guide.

Écrire sur Kinesis Data Streams Pinpoint using Amazon

Vous pouvez configurer Amazon Pinpoint pour qu'il envoie des données d'événements à Amazon Kinesis Data Streams. Amazon Pinpoint peut envoyer des données d'événements pour les campagnes, les parcours, les e-mails et les messages transactionnels. SMS Vous avez ensuite la possibilité d'ingérer ces données dans des applications d'analyse ou de créer vos propres applications consommateur qui agissent en fonction du contenu des événements.

Pour plus d'informations, consultez la rubrique [Événements en flux continu](#) dans le Guide du développeur Amazon Pinpoint.

Écrivez dans Kinesis Data Streams à l'aide de la base de données Amazon Quantum Ledger (Amazon) QLDB

Vous pouvez créer un flux dans Amazon QLDB qui capture chaque révision de document enregistrée dans votre journal et transmet ces données à Amazon Kinesis Data Streams en temps réel. Un QLDB flux est un flux continu de données entre le journal de votre registre et une ressource de flux de données Kinesis. Vous pouvez ensuite utiliser la plateforme de flux continu Kinesis ou la bibliothèque client Kinesis Client Library pour consommer votre flux, traiter les enregistrements de données et analyser le contenu des données. Un QLDB flux écrit vos données dans Kinesis Data Streams sous trois types d'enregistrements `control` : `block summary`, `revision details` et.

Pour plus d'informations, consultez [Streams](#) dans le guide du QLDB développeur Amazon.

Écrivez sur Kinesis Data Streams à l'aide d'intégrations tierces

Vous pouvez écrire des données dans Kinesis Data Streams à l'aide de l'une des options tierces suivantes qui s'intègrent à Kinesis Data Streams. Sélectionnez l'option sur laquelle vous souhaitez en savoir plus et recherchez des ressources et des liens vers la documentation pertinente.

Rubriques

- [Apache Flink](#)
- [Fluentd](#)
- [Debezium](#)
- [Oracle GoldenGate](#)
- [Kafka Connect](#)
- [Adobe Experience](#)
- [Striim](#)

Apache Flink

Apache Flink est un environnement et un moteur de traitement distribué pour les calculs avec état sur des flux de données illimités et limités. Pour plus d'informations sur l'écriture dans Kinesis Data Streams à partir d'Apache Flink, consultez la rubrique [Connecteur Amazon Kinesis Data Streams](#).

Fluentd

Fluentd est un collecteur de données open source pour une couche de journalisation unifiée. Pour plus d'informations sur l'écriture dans Kinesis Data Streams à partir de Fluentd. Pour plus d'informations, consultez la rubrique [PTraitement des flux à l'aide de Kinesis](#).

Debezium

Debezium est une plateforme distribuée open source dédiée à la capture des modifications de données. Pour plus d'informations sur l'écriture vers Kinesis Data Streams depuis Debezium, consultez [Streaming SQL My Data Changes to Amazon Kinesis](#).

Oracle GoldenGate

Oracle GoldenGate est un produit logiciel qui vous permet de répliquer, de filtrer et de transformer des données d'une base de données à une autre. Pour plus d'informations sur l'écriture vers Kinesis Data Streams depuis GoldenGate Oracle, [voir Réplication de données vers Kinesis Data Stream](#) à l'aide d'Oracle. GoldenGate

Kafka Connect

Kafka Connect est un outil permettant de diffuser des données de manière évolutive et fiable entre Apache Kafka et d'autres systèmes. Pour plus d'informations sur l'écriture de données à partir d'Apache Kafka vers Kinesis Data Streams, consultez la rubrique [Connecteur Kinesis Kafka](#) (français non garanti).

Adobe Experience

Adobe Experience Platform permet aux entreprises de centraliser et de standardiser les données des consommateurs depuis n'importe quel système. Elle applique ensuite la science des données et le machine learning pour améliorer considérablement la conception et la fourniture d'expériences riches et personnalisées. Pour en savoir plus sur l'écriture de données à partir d'Adobe Experience Platform vers Kinesis Data Streams, consultez la procédure de création d'une [connexion Amazon Kinesis](#).

Striim

Striim est une plateforme complète en mémoire pour la collecte end-to-end, le filtrage, la transformation, l'enrichissement, l'agrégation, l'analyse et la diffusion de données en temps réel. Pour plus d'informations sur la façon d'écrire des données dans Kinesis Data Streams à partir de Striim, consultez la rubrique [Composant d'écriture Kinesis](#).

Résoudre les problèmes liés aux producteurs d'Amazon Kinesis Data Streams

Les rubriques suivantes proposent des solutions aux problèmes courants rencontrés par les producteurs d'Amazon Kinesis Data Streams :

- [Ma candidature de producteur s'écrit plus lentement que prévu](#)
- [Je reçois une erreur d'autorisation de clé KMS principale non autorisée](#)

- [Résoudre d'autres problèmes courants rencontrés par les producteurs](#)

Ma candidature de producteur s'écrit plus lentement que prévu

Les raisons les plus courantes pour lesquelles le débit d'écriture est plus lent que prévu sont les suivantes :

- [Limites de service dépassées](#)
- [Je souhaite optimiser mon producteur](#)

Limites de service dépassées

Pour savoir si les limites de service sont dépassées, vérifiez si votre producteur émet des exceptions de débit de la part du service et validez quelles API opérations sont limitées. N'oubliez pas qu'il existe différentes limites basées sur l'appel, consultez [Quotas et limites](#). Par exemple, en plus des nombres limite d'écritures et de lectures au niveau d'une partition, qui sont connus la plupart du temps, les nombres limite suivants sont applicables au niveau du flux :

- [CreateStream](#)
- [DeleteStream](#)
- [ListStreams](#)
- [GetShardIterator](#)
- [MergeShards](#)
- [DescribeStream](#)
- [DescribeStreamSummary](#)

Les opérations `CreateStream`, `DeleteStream`, `ListStreams`, `GetShardIterator` et `MergeShards` sont limitées à 5 appels par seconde. L'opération `DescribeStream` est limitée à 10 appels par seconde. L'opération `DescribeStreamSummary` est limitée à 20 appels par seconde.

Si ces appels ne constituent pas le problème, assurez-vous que vous avez sélectionné une clé de partition qui permet de répartir également les opérations put entre toutes les partitions, et que vous n'avez pas de clé de partition spécifique qui se heurte aux limites de service alors que le reste ne le fait pas. Pour cela, vous devez mesurer le débit maximal et tenir compte du nombre de partitions du flux. Pour plus d'informations sur la gestion des flux, consultez la page [Création et gestion de flux de données Kinesis](#).

i Tip

N'oubliez pas d'arrondir au kilo-octet le plus proche pour les calculs de limitation du débit lorsque vous utilisez l'opération à enregistrement unique [PutRecord](#), tandis que l'opération à enregistrements multiples [PutRecords](#) arrondit la somme cumulée des enregistrements de chaque appel. Par exemple, une demande PutRecords contenant 600 enregistrements de 1,1 Ko n'est pas limitée.

Je souhaite optimiser mon producteur

Avant de commencer à optimiser votre producteur, effectuez les tâches clés suivantes. Vous devez commencer par identifier votre débit maximal voulu en termes de taille d'enregistrement et de nombre d'enregistrements par seconde. Éliminez ensuite la capacité de flux comme facteur limitant ([Limites de service dépassées](#)). Si vous avez éliminé la capacité du flux, utilisez les conseils de dépannage et instructions d'optimisation suivants pour les deux types courants d'applications producteur.

Application producteur de grande capacité

Un grand producteur fonctionne généralement à partir d'un serveur sur site ou d'une EC2 instance Amazon. Les clients qui ont besoin d'un débit supérieur pour une application producteur de grande capacité s'occupent de la latence par enregistrement. Les stratégies pour gérer la latence sont les suivantes : si le client peut microloter/mettre en mémoire tampon des enregistrements, utilisez la [bibliothèque Kinesis Producer](#) (qui possède une logique d'agrégation avancée), l'opération multi-enregistrements [PutRecords](#), ou agréger les enregistrements dans un fichier plus volumineux avant d'utiliser l'opération à enregistrement unique. [PutRecord](#) Si vous ne pouvez pas regrouper/mettre en mémoire tampon, utilisez plusieurs threads pour écrire simultanément dans le service Kinesis Data Streams. Les AWS SDK for Java autres SDKs incluent des clients asynchrones qui peuvent le faire avec très peu de code.

Application producteur de petite capacité

Une application producteur de petite capacité est généralement une application mobile, un appareil IoT ou un client Web. S'il s'agit d'une application mobile, nous vous recommandons d'utiliser l'[PutRecords](#) appareil ou l'enregistreur Kinesis dans le AWS mobile. SDKs Pour plus d'informations, consultez le [AWS Mobile SDK for Android Guide de démarrage](#) et [AWS Mobile SDK for iOS le Guide de démarrage](#). Les applications mobiles doivent gérer les connexions intermittentes de façon intrinsèque et ont besoin d'une sorte de placement par lots, par exemple [PutRecords](#). Si vous ne pouvez pas regrouper pour une raison quelconque, consultez les informations ci-dessus relatives

aux applications producteur de grande capacité. Si votre application producteur est un navigateur, le volume de données généré est généralement très faible. Cependant, vous placez les opérations sur le chemin critique de l'application, ce que nous ne recommandons pas.

Je reçois une erreur d'autorisation de clé KMS principale non autorisée

Cette erreur se produit lorsqu'une application productrice écrit dans un flux chiffré sans autorisation sur la clé KMS principale. Pour attribuer à une application des autorisations d'accès à une KMS clé, consultez les sections [Utilisation de politiques clés dans AWS KMS](#) et [Utilisation de IAM politiques avec AWS KMS](#).

Résoudre d'autres problèmes courants rencontrés par les producteurs

- [Pourquoi mon flux de données Kinesis renvoie-t-il une erreur interne de serveur 500 ?](#)
- [Comment résoudre les erreurs de dépassement de délai lors de l'écriture à partir de Flink vers Kinesis Data Streams ?](#)
- [Comment résoudre les erreurs de limitation dans Kinesis Data Streams ?](#)
- [Pourquoi mon flux de données Kinesis est-il limité ?](#)
- [Comment puis-je insérer des enregistrements de données dans un flux de données Kinesis à l'aide du ? KPL](#)

Optimisez les producteurs de Kinesis Data Streams

Vous pouvez optimiser davantage vos producteurs Amazon Kinesis Data Streams en fonction du comportement spécifique que vous observez. Consultez les rubriques suivantes pour identifier des solutions.

Rubriques

- [Personnaliser le KPL comportement des nouvelles tentatives et des limites de débit](#)
- [Appliquer les meilleures pratiques à l'KPLagrégation](#)

Personnaliser le KPL comportement des nouvelles tentatives et des limites de débit

Lorsque vous ajoutez des enregistrements utilisateur Kinesis Producer Library (KPL) à l'aide de cette KPL `addUserRecord()` opération, un enregistrement reçoit un horodatage et est ajouté à

une mémoire tampon avec une date limite définie par le paramètre de `RecordMaxBufferedTime` configuration. Cette combinaison de l'horodatage et du délai définit la priorité tampon. Les enregistrements sont vidés de la mémoire tampon selon les critères suivants :

- Priorité tampon
- Configuration du regroupement
- Configuration de la collecte

Les paramètres de configuration du regroupement et de la collecte qui influencent le comportement du tampon sont les suivants :

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

Les enregistrements vidés sont ensuite envoyés à votre flux de données Kinesis sous forme d'enregistrements Amazon Kinesis Data Streams à l'aide d'un appel à l'opération Kinesis Data Streams. `API PutRecords` L'opération `PutRecords` envoie des demandes à votre flux qui échouent parfois entièrement ou partiellement. Les enregistrements qui échouent sont automatiquement ajoutés à la KPL mémoire tampon. Le nouveau délai est défini sur la base du minimum des deux valeurs suivantes :

- La moitié de la configuration `RecordMaxBufferedTime` actuelle
- La `time-to-live` valeur de l'enregistrement

Cette stratégie permet d'inclure les enregistrements KPL d'utilisateurs réessayés dans les appels Kinesis Data API Streams suivants, afin d'améliorer le débit et de réduire la complexité tout en renforçant la valeur de l'enregistrement Kinesis Data Streams. `time-to-live` Il n'y a aucun algorithme d'interruption, ce qui rend cette stratégie de nouvelle tentative relativement brutale. Le spam provoqué par des tentatives excessives est empêché par la limitation de vitesse, qui est présentée à la section suivante.

Limitation du débit

KPLII inclut une fonctionnalité de limitation de débit, qui limite le débit par partition envoyée par un seul producteur. La limitation de vitesse est implémentée à l'aide d'un algorithme de compartiment à jeton, avec des compartiments distincts pour les enregistrements Kinesis Data Streams et les octets. Chaque écriture réussie dans un flux de données Kinesis ajoute un jeton (ou plusieurs jetons) à chaque compartiment, jusqu'à un certain seuil. Ce seuil est configurable, mais est défini par défaut sur une valeur 50 % supérieure au nombre limite de partitions réel afin d'autoriser la saturation de partition par une seule application producteur.

Vous pouvez réduire ce nombre limite pour diminuer le spam dû aux tentatives excessives. Toutefois, la bonne pratique consiste, pour chaque application producteur, à effectuer des tentatives brutales pour atteindre un débit maximal et à traiter toute limitation obtenue comme excessive en étendant la capacité du flux et en mettant en œuvre une stratégie de clé de partition appropriée.

Appliquer les meilleures pratiques à l'KPLagrégation

Bien que le schéma de numérotation des enregistrements Amazon Kinesis Data Streams obtenus reste le même, l'agrégation entraîne l'indexation des enregistrements utilisateur Kinesis Producer Library KPL () contenus dans un enregistrement Kinesis Data Streams agrégé à partir de 0 (zéro) ; toutefois, tant que vous ne vous fiez pas aux numéros de séquence pour KPL identifier de manière unique vos enregistrements utilisateur, votre code peut l'ignorer, car l'agrégation (de vos enregistrements utilisateur dans un Kinesis Data Streams (enregistrement) et désagrégation ultérieure (KPLd'un enregistrement Kinesis Data Streams) dans votre KPLuser records) s'en charge automatiquement pour vous. Cela s'applique que votre consommateur utilise le KCL ou le AWS SDK. Pour utiliser cette fonctionnalité d'agrégation, vous devez intégrer la partie Java du KPL dans votre build si votre client est écrit en utilisant le code API fourni dans le AWS SDK.

Si vous avez l'intention d'utiliser des numéros de séquence comme identifiants uniques pour vos dossiers KPL d'utilisateur, nous vous recommandons d'utiliser les `public boolean equals(Object obj)` opérations `public int hashCode()` et de respecter les contrats prévus dans `Record` et `UserRecord` pour permettre la comparaison de vos dossiers d'utilisateur. KPL En outre, si vous souhaitez examiner le numéro de sous-séquence de votre enregistrement KPL utilisateur, vous pouvez le convertir en `UserRecord` instance et récupérer son numéro de sous-séquence.

Pour de plus amples informations, veuillez consulter [Mettre en œuvre la désagrégation des consommateurs](#).

Lire les données d'Amazon Kinesis Data Streams

Un consommateur est une application qui traite toutes les données d'un flux de données Kinesis. Lorsqu'une application consommateur utilise la diffusion améliorée, elle obtient sa propre allocation de 2 Mo/sec de débit de lecture, ce qui permet à plusieurs applications consommateur de lire les données à partir du même flux en parallèle, sans devoir partager le débit de lecture avec d'autres applications consommateur. Pour utiliser la capacité de diffusion améliorée des partitions, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Par défaut, les partitions présentes dans un flux fournissent un débit de lecture de 2 Mo/sec par partition. Ce débit est réparti entre toutes les applications consommateur qui lisent une partition donnée. En d'autres termes, la valeur par défaut de 2 Mo/sec de débit par partition est fixe, même si plusieurs applications consommateur lisent la partition. Pour utiliser ce débit par défaut des partitions, consultez [Développez des consommateurs personnalisés avec un débit partagé](#).

Le tableau suivant compare le débit par défaut à la diffusion améliorée. Le délai de propagation des messages est défini comme le temps nécessaire, en millisecondes, pour qu'une charge utile envoyée à l'aide de la méthode d'expédition de charge utile APIs (comme PutRecord et PutRecords) atteigne l'application client via l'application consommatrice de charge utile (comme et). APIs GetRecords SubscribeToShard

Caractéristiques	Consommateurs non enregistrés sans la diffusion améliorée	Consommateurs enregistrés avec la diffusion améliorée
Débit de lecture des partitions	Fixé à un total de 2 Mo/sec par partition. S'il y a plusieurs applications consommateur qui lisent la même partition, elles partagent toutes ce débit. La somme des débits qu'elles reçoivent de la partition ne dépasse pas 2 Mo/sec.	S'adapte au fur et à mesure de l'enregistrement des applications consommateur pour utiliser la diffusion améliorée. Chaque application consommateur enregistrée pour utiliser la diffusion améliorée reçoit son propre débit de lecture par partition, jusqu'à 2 Mo/sec, indépendamment des autres applications consommateur.

Caractéristiques	Consommateurs non enregistrés sans la diffusion améliorée	Consommateurs enregistrés avec la diffusion améliorée
Retard de propagation des messages	En moyenne, environ 200 ms si un consommateur lit depuis le flux. Cette moyenne monte à environ 1000 ms si vous avez cinq consommateurs.	En général, une moyenne de 70 ms que vous ayez un consommateur ou cinq consommateurs.
Coût	N/A	Il y a un coût d'extraction des données et un coût horaire application consommateur-partition. Pour en savoir plus, consultez la Tarification Amazon Kinesis Data Streams .
Modèle de livraison de l'enregistrement	Tirez le modèle sur le dessus HTTP en utilisant GetRecords .	Kinesis Data Streams vous envoie les enregistrements HTTP sur /2 en utilisant. SubscribeToShard

Rubriques

- [Utiliser le visualiseur de données dans la console Kinesis](#)
- [Interrogez vos flux de données dans la console Kinesis](#)
- [Développez les consommateurs en utilisant AWS Lambda](#)
- [Développez vos clients à l'aide d'Amazon Managed Service pour Apache Flink](#)
- [Développez vos clients grâce à Amazon Data Firehose](#)
- [Utiliser la bibliothèque cliente Kinesis](#)
- [Développez des consommateurs personnalisés avec un débit partagé](#)
- [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#)
- [Migrer les consommateurs de la version KCL 1.x vers la version KCL 2.x](#)
- [Lisez les données de Kinesis Data Streams à l'aide d'autres services AWS](#)
- [Lisez depuis Kinesis Data Streams à l'aide d'intégrations tierces](#)
- [Résoudre les problèmes des utilisateurs de Kinesis Data Streams](#)
- [Optimisez les utilisateurs d'Amazon Kinesis Data Streams](#)

Utiliser le visualiseur de données dans la console Kinesis

Le visualiseur de données de la Kinesis Management Console vous permet de visualiser les enregistrements de données contenus dans la partition spécifiée de votre flux de données sans avoir à développer d'application grand public. Pour utiliser le visualiseur de données, procédez comme suit :

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/kinesis](https://console.aws.amazon.com/kinesis).
2. Choisissez le flux de données actif dont vous souhaitez afficher les enregistrements avec le visualiseur de données, puis cliquez sur l'onglet Visualiseur de données.
3. Dans l'onglet Visualiseur de données du flux de données actif sélectionné, choisissez la partition dont vous souhaitez afficher les enregistrements, choisissez la position de départ, puis cliquez sur Obtenir des enregistrements. Vous pouvez définir la position de départ sur l'une des valeurs suivantes :
 - Au numéro de séquence : affiche les enregistrements à partir de la position indiquée par le numéro de séquence spécifié dans le champ du numéro de séquence.
 - Après le numéro de séquence : affiche les enregistrements juste après la position indiquée par le numéro de séquence spécifié dans le champ du numéro de séquence.
 - À l'horodatage : affiche les enregistrements à partir de la position indiquée par l'horodatage spécifié dans le champ d'horodatage.
 - Horizon de découpe : affiche les enregistrements au dernier enregistrement non découpé de la partition, qui correspond au plus ancien enregistrement de données de la partition.
 - Plus récent : affiche les enregistrements juste après l'enregistrement le plus récent de la partition, afin de toujours lire les données les plus récentes de la partition.

Les enregistrements de données générés correspondant à l'ID de partition et à la position de départ spécifiés sont ensuite affichés dans un tableau d'enregistrements de la console. Un maximum de 50 enregistrements est affiché à la fois. Pour afficher les enregistrements suivants, cliquez sur le bouton Suivant.

4. Cliquez sur un enregistrement individuel pour afficher la charge utile de cet enregistrement sous forme de données brutes ou de JSON format dans une fenêtre séparée.

Notez que lorsque vous cliquez sur les boutons Obtenir des enregistrements ou Suivant dans le visualiseur de données, cela invoque le `GetRecordsAPI` et cela s'applique à la `GetRecordsAPI` limitée de 5 transactions par seconde.

Interrogez vos flux de données dans la console Kinesis

L'onglet Data Analytics de la console Kinesis Data Streams vous permet d'interroger vos flux de données à l'aide de SQL. Pour utiliser cette fonctionnalité, procédez comme suit :

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à l'adresse `https://console.aws.amazon.com/kinesis`.](https://console.aws.amazon.com/kinesis)
2. Choisissez le flux de données actif que vous souhaitez interroger, SQL puis cliquez sur l'onglet Analyse des données.
3. Dans l'onglet Analyse des données, vous pouvez inspecter et visualiser les flux à l'aide d'un bloc-notes Apache Flink Studio géré. Vous pouvez effectuer des SQL requêtes ad hoc pour inspecter votre flux de données et afficher les résultats en quelques secondes à l'aide d'Apache Zeppelin. Dans l'onglet Analyse des données, choisissez J'accepte, puis choisissez Créer un bloc-notes pour créer un bloc-notes.
4. Une fois le bloc-notes créé, choisissez Ouvrir dans Apache Zeppelin. Cela ouvrira votre bloc-notes dans un nouvel onglet. Un bloc-notes est une interface interactive où vous pouvez soumettre vos SQL requêtes. Choisissez la note contenant le nom de votre stream.
5. Vous verrez une note avec un exemple de SELECT requête pour afficher les données du flux déjà en cours d'exécution. Cela vous permet de visualiser le schéma de votre flux de données.
6. Pour essayer d'autres requêtes, telles que des fenêtres coulissantes ou coulissantes, choisissez Afficher des exemples de requêtes dans l'onglet Analyse des données. Copiez la requête, modifiez-la en fonction de votre schéma de flux de données, puis exécutez-la dans un nouveau paragraphe de votre note Zeppelin.

Développez les consommateurs en utilisant AWS Lambda

Vous pouvez utiliser une AWS Lambda fonction pour traiter les enregistrements d'un flux de données. AWS Lambda est un service de calcul qui vous permet d'exécuter du code sans provisionner ni gérer de serveurs. Il exécute votre code uniquement lorsque c'est nécessaire et l'adapte automatiquement, de quelques demandes par jour à plusieurs milliers par seconde. Vous payez uniquement pour le temps de calcul consommé. Aucun frais n'est facturé si votre code n'est pas en cours d'exécution.

Avec AWS Lambda, vous pouvez exécuter du code pour pratiquement n'importe quel type d'application ou de service principal, le tout sans aucune administration. Il exécute votre code sur une infrastructure de calcul à haute disponibilité et effectue toute l'administration des ressources de calcul, y compris la maintenance des serveurs et du système d'exploitation, la mise en service des capacités et la scalabilité automatique, ainsi que la surveillance et la journalisation du code. Pour plus d'informations, consultez la section [Utilisation AWS Lambda avec Amazon Kinesis](#).

Pour obtenir plus d'informations sur la résolution des problèmes, consultez [Pourquoi le déclencheur Kinesis Data Streams ne parvient-il pas à invoquer ma fonction Lambda ?](#)

Développez vos clients à l'aide d'Amazon Managed Service pour Apache Flink

Vous pouvez utiliser une application Amazon Managed Service for Apache Flink pour traiter et analyser les données d'un flux Kinesis à SQL l'aide de Java ou de Scala. Le service géré pour les applications Apache Flink peut enrichir les données à l'aide de sources de référence, agréger les données au fil du temps ou utiliser le machine learning pour détecter les anomalies des données. Vous pouvez ensuite écrire les résultats de l'analyse dans un autre flux Kinesis, un flux de diffusion Firehose ou une fonction Lambda. Pour plus d'informations, consultez le guide du développeur du [service géré pour Apache Flink pour les SQL applications](#) ou le [guide du développeur du service géré pour Apache Flink pour les applications Flink](#).

Développez vos clients grâce à Amazon Data Firehose

Vous pouvez utiliser un Firehose pour lire et traiter les enregistrements d'un flux Kinesis. Firehose est un service entièrement géré permettant de diffuser des données de streaming en temps réel vers des destinations telles qu'Amazon S3, Amazon Redshift, OpenSearch Amazon Service et Splunk. Firehose prend également en charge tout HTTP point de HTTP terminaison personnalisé ou appartenant à des fournisseurs de services tiers pris en charge, notamment Datadog, MongoDB et New Relic. Vous pouvez également configurer Firehose pour transformer vos enregistrements de données et convertir le format d'enregistrement avant de livrer vos données à destination. Pour plus d'informations, consultez [Writing to Firehose Using Kinesis](#) Data Streams.

Utiliser la bibliothèque cliente Kinesis

L'une des méthodes de développement d'applications grand public personnalisées capables de traiter des données issues de flux de KDS données consiste à utiliser la bibliothèque cliente Kinesis (JKCL).

Rubriques

- [Qu'est-ce que Kinesis Client Library ?](#)
- [KCLversions disponibles](#)
- [Concepts KCL](#)
- [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#)
- [Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public Java](#)
- [Utiliser le KCL avec le registre des AWS Glue schémas](#)

Note

Pour les versions KCL 1.x et KCL 2.x, il est recommandé de passer à la dernière version KCL 1.x ou KCL 2.x, selon votre scénario d'utilisation. Les versions KCL 1.x et KCL 2.x sont régulièrement mises à jour avec de nouvelles versions qui incluent les derniers correctifs de dépendance et de sécurité, des corrections de bogues et de nouvelles fonctionnalités rétrocompatibles. Pour plus d'informations, consultez <https://github.com/aws-labs/amazon-kinesis-client/releases>.

Qu'est-ce que Kinesis Client Library ?

KCL vous aide à consommer et à traiter les données issues d'un flux de données Kinesis en prenant en charge de nombreuses tâches complexes associées à l'informatique distribuée. Il s'agit notamment de l'équilibrage de charge entre plusieurs instances d'applications consommateur, de la réponse aux défaillances des instances d'applications consommateur, du contrôle des enregistrements traités et de la réaction au repartitionnement. KCL s'occupe de toutes ces sous-tâches afin que vous puissiez concentrer vos efforts sur l'écriture de votre logique de traitement des enregistrements personnalisée.

KCL est différent des Kinesis Data APIs Streams disponibles dans le AWS SDKs. Les Kinesis Data APIs Streams vous aident à gérer de nombreux aspects de Kinesis Data Streams, notamment la création de flux, le repartage, ainsi que le transfert et l'obtention d'enregistrements. KCL fournit une couche d'abstraction autour de toutes ces sous-tâches, notamment pour que vous puissiez vous concentrer sur la logique de traitement des données personnalisée de votre application client. Pour plus d'informations sur les Kinesis Data API Streams, consultez [le API Amazon Kinesis Reference](#).

⚠ Important

KCLII s'agit d'une bibliothèque Java. Support pour les langages autres que Java est fourni à l'aide d'une interface multilingue appelée. MultiLangDaemon Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par exemple, si vous installez le KCL pour Python et que vous écrivez votre application grand public entièrement en Python, vous devez toujours installer Java sur votre système en raison du MultiLangDaemon. En outre, MultiLangDaemon comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations sur l' MultiLangDaemon on GitHub, voir le [KCL MultiLangDaemon projet](#).

Il KCL agit comme un intermédiaire entre votre logique de traitement des enregistrements et Kinesis Data Streams. La KCL effectue les tâches suivantes :

- Se connecte au flux de données
- Énumère les partitions du flux de données
- Utilise les baux pour coordonner les associations de partitions avec ses employés
- Instancie un processeur d'enregistrements pour chaque partition qu'il gère
- Extrait les enregistrements de données du flux de données
- Pousse les enregistrements sur le processeur d'enregistrements correspondant
- Contrôle les enregistrements traités
- Équilibre les associations partition-application de travail (baux) lorsque le nombre d'instances de travailleurs change ou lorsque le flux de données est repartitionné (les partitions sont fractionnées ou fusionnées)

KCLversions disponibles

À l'heure actuelle, vous pouvez utiliser l'une des versions prises en charge suivantes KCL pour créer vos applications grand public personnalisées :

- KCL1. x

Pour plus d'informations, consultez [Développez des KCL consommateurs 1.x](#).

- KCL2. x

Pour plus d'informations, consultez [Développez des KCL consommateurs 2.x](#).

Vous pouvez utiliser la version KCL 1.x ou la version KCL 2.x pour créer des applications grand public utilisant un débit partagé. Pour plus d'informations, consultez [Développez des consommateurs personnalisés avec un débit partagé en utilisant KCL](#).

Pour créer des applications grand public utilisant un débit dédié (consommateurs ventilés améliorés), vous ne pouvez utiliser que la version 2.x. KCL Pour plus d'informations, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Pour plus d'informations sur les différences entre la version KCL 1.x et la version KCL 2.x, et pour obtenir des instructions sur la façon de migrer de la version KCL 1.x vers la version KCL 2.x, consultez [Migrer les consommateurs de la version KCL 1.x vers la version KCL 2.x](#)

Concepts KCL

- KCLapplication grand public : application conçue sur mesure pour lire KCL et traiter des enregistrements à partir de flux de données.
- Instance d'application KCL grand public : les applications grand public sont généralement distribuées, une ou plusieurs instances d'application s'exécutant simultanément afin de coordonner les défaillances et d'équilibrer dynamiquement la charge du traitement des enregistrements de données.
- Travailleur : classe de haut niveau utilisée par une instance d'application KCL grand public pour commencer à traiter des données.

Important

Chaque instance d'application KCL grand public dispose d'un travailleur.

L'application de travail initialise et supervise diverses tâches, y compris la synchronisation des informations de partition et de bail, le suivi des affectations de partitions et le traitement des données provenant des partitions. Un travailleur fournit KCL les informations de configuration de l'application client, telles que le nom du flux de données dont cette application KCL client va traiter les enregistrements de données et les AWS informations d'identification nécessaires pour accéder à ce flux de données. Le travailleur lance également cette instance d'application KCL

client spécifique pour fournir des enregistrements de données du flux de données aux processeurs d'enregistrements.

⚠ Important

Dans la KCL version 1.x, cette classe s'appelle Worker. Pour plus d'informations (il s'agit des KCL référentiels Java), consultez <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/worker.java>. Dans la KCL version 2.x, cette classe s'appelle Scheduler. L'objectif du planificateur dans la version KCL 2.x est identique à celui du Worker dans la version 1.x. KCL [Pour plus d'informations sur la classe Scheduler dans la version KCL 2.x, consultez https://github.com/aws-labs/amazon-kinesis-client/blob/master/src/main/java/software.amazon.kinesis.coordinator/Scheduler.java](https://github.com/aws-labs/amazon-kinesis-client/blob/master/src/main/java/software.amazon.kinesis.coordinator/Scheduler.java). [amazon-kinesis-client](https://github.com/aws-labs/amazon-kinesis-client)

- Bail : données qui définissent le lien entre une application de travail et une partition. Les applications KCL grand public distribuées utilisent des contrats de location pour répartir le traitement des enregistrements de données entre un parc de travailleurs. À tout moment, chaque fragment d'enregistrements de données est lié à un travailleur en particulier par un bail identifié par la leaseKeyvariable.

Par défaut, un travailleur peut détenir un ou plusieurs baux (sous réserve de la valeur de la variable maxLeasesForWorker) en même temps.

⚠ Important

Chaque application de travail devra posséder tous les baux disponibles pour toutes les partitions disponibles dans un flux de données. Cependant, à tout moment, une seule application de travail peut posséder avec succès un bail spécifique.

Par exemple, si vous avez une instance d'application consommateur A avec l'application de travail A qui traite un flux de données contenant 4 partitions, l'application de travail A peut posséder simultanément des baux pour les partitions 1, 2, 3 et 4. Cependant, si vous avez deux instances d'applications consommateur, nommées A et B, chacune avec sa propre application de travail (application de travail A et application de travail B), traitant un flux de données composé de 4 partitions, alors l'application de travail A et l'application de travail B ne peuvent pas détenir simultanément le bail de la partition 1. Une application de travail possède le bail d'une partition spécifique jusqu'à ce qu'elle soit prête à arrêter de traiter les enregistrements de données de cette

partition ou jusqu'à ce qu'elle rencontre une défaillance. Lorsqu'une application de travail cesse de posséder le bail, une autre application de travail prend et possède ce bail.

[Pour plus d'informations \(il s'agit des KCL référentiels Java\), consultez <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/lease.java> pour 1.x et \[/blob/master/ /src/main/java/software/amazon/kinesis/leases/Lease.java\]\(https://github.com/aws-labs/amazon-kinesis-client/blob/master/src/main/java/software/amazon/kinesis/leases/Lease.java\) pour 2.x. KCL \[https://github.com/aws-labs/ amazon-kinesis-client\]\(https://github.com/aws-labs/amazon-kinesis-client\) \[amazon-kinesis-client\]\(https://github.com/aws-labs/amazon-kinesis-client\) KCL](https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/lease.java)

- **Table des baux** : table Amazon DynamoDB unique utilisée pour suivre les partitions d'KDSun flux de données louées et traitées par les employés de l'application client. KCL La table des baux doit rester synchronisée (au sein d'un travailleur et entre tous les travailleurs) avec les dernières informations relatives aux partitions provenant du flux de données pendant l'exécution de l'application KCL client. Pour plus d'informations, consultez [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client.](#)
- **Processeur d'enregistrement** : logique qui définit la manière dont votre application KCL client traite les données issues des flux de données. Au moment de l'exécution, une instance d'application KCL grand public instancie un travailleur, qui instancie un processeur d'enregistrement pour chaque partition qu'il loue.

Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client

Rubriques

- [Qu'est-ce qu'une table de location](#)
- [Débit](#)
- [Comment une table de location est synchronisée avec les partitions d'un flux de données Kinesis](#)

Qu'est-ce qu'une table de location

Pour chaque application Amazon Kinesis Data StreamsKCL, utilise une table de location unique (stockée dans une table Amazon DynamoDB) afin de suivre les fragments KDS d'un flux de données loués et traités par les employés de l'application client. KCL

⚠ Important

KCL utilise le nom de l'application client pour créer le nom de la table de location utilisée par cette application client. Par conséquent, chaque nom d'application client doit être unique.

Vous pouvez consulter cette table à l'aide de la [console Amazon DynamoDB](#) lors que l'application est en cours d'exécution.

Si la table des baux de votre application KCL client n'existe pas au démarrage de l'application, l'un des travailleurs crée la table des baux pour cette application.

⚠ Important


Votre compte est facturé pour les coûts associés à la table DynamoDB, en plus des coûts associés au service Kinesis Data Streams lui-même.

Chaque ligne de la table des baux représente une partition qui est en cours de traitement par les applications de travail de votre application consommateur. Si votre application KCL client ne traite qu'un seul flux de données, leaseKey l'ID de partition est la clé de hachage de la table de location. Si c'est le cas [Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public Java](#), alors la structure du leaseKey ressemble à ceci :account-id:StreamName:streamCreationTimestamp:ShardId. Par exemple, 111111111:multiStreamTest-1:12345:shardId-000000000336.

En plus de l'ID de partition, chaque ligne inclut également les données suivantes :


- checkpoint : le plus récent numéro de séquence de point de contrôle de la partition. Cette valeur est unique dans toutes les partitions du flux de données.
- checkpointSubSequenceNuméro : lorsque vous utilisez la fonctionnalité d'agrégation de la bibliothèque Kinesis Producer, il s'agit d'une extension du point de contrôle qui permet de suivre les enregistrements utilisateur individuels au sein de l'enregistrement Kinesis.
- leaseCounter: utilisé pour le contrôle des versions de bail afin que les travailleurs puissent détecter que leur bail a été pris par un autre travailleur.
- leaseKey: identifiant unique pour un bail. Chaque bail est propre à une partition du flux de données et est détenu par une seule application de travail à la fois.

- `leaseOwner`: Le travailleur titulaire de ce bail.
- `ownerSwitchesSincePoint` de contrôle : combien de fois ce bail a changé de travailleur depuis la dernière fois qu'un point de contrôle a été écrit.
- `parentShardId`: Utilisé pour garantir que la partition parent est entièrement traitée avant le début du traitement sur les partitions enfants. Cela garantit que les enregistrements sont traités dans l'ordre dans lequel ils ont été placés dans le flux.
- `hashrange` : utilisé par le `PeriodicShardSyncManager` pour exécuter des synchronisations périodiques afin de trouver les partitions manquantes dans la table des baux et de créer des baux pour celles-ci si nécessaire.

 Note


Ces données sont présentes dans le tableau des baux pour chaque partition à partir des versions KCL 1.14 et KCL 2.3. Pour plus d'informations sur `PeriodicShardSyncManager` et la synchronisation périodique entre les baux et les partitions, consultez [Comment une table de location est synchronisée avec les partitions d'un flux de données Kinesis](#).

- `childshards` : utilisé par le `LeaseCleanupManager` pour vérifier l'état de traitement de la partition enfant et décider si la partition parent peut être supprimée de la table des baux.

 Note

Ces données sont présentes dans le tableau des baux pour chaque partition à partir des versions KCL 1.14 et KCL 2.3.

- `shardID` : ID de la partition.

 Note

Ces données ne sont présentes dans le tableau des baux que si vous êtes [Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public Java](#). Ceci n'est pris en charge que dans la version KCL 2.x pour Java, à partir de la version KCL 2.3 pour Java et versions ultérieures.

- `nom du flux` : identifiant du flux de données au format suivant : `account-id:StreamName:streamCreationTimestamp`.

Note

Ces données ne sont présentes dans le tableau des baux que si vous êtes [Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public Java](#). Ceci n'est pris en charge que dans la version KCL 2.x pour Java, à partir de la version KCL 2.3 pour Java et versions ultérieures.

Débit

Si votre application Amazon Kinesis Data Streams reçoit des exceptions de débit provisionné, vous devez augmenter le débit provisionné pour la table DynamoDB. La table est KCL créée avec un débit provisionné de 10 lectures par seconde et 10 écritures par seconde, mais cela risque de ne pas être suffisant pour votre application. Par exemple, si votre application Amazon Kinesis Data Streams effectue des contrôles fréquents ou fonctionne sur un flux composé de nombreuses partitions, il se peut que vous ayez besoin d'un débit plus élevé.

Pour plus d'informations sur le débit provisionné dans DynamoDB consultez les rubriques [Mode de capacité en lecture/écriture](#) et [Utilisation des tables et des données](#) dans le Guide du développeur Amazon DynamoDB.

Comment une table de location est synchronisée avec les partitions d'un flux de données Kinesis

Les employés des applications KCL grand public utilisent des contrats de location pour traiter les fragments provenant d'un flux de données donné. Les informations relatives à l'application de travail qui loue une partition à un moment donné sont stockées dans une table des baux. La table des baux doit rester synchronisée avec les dernières informations relatives à la partition provenant du flux de données pendant l'exécution de l'application KCL client. KCL synchronise la table des baux avec les informations relatives aux partitions obtenues auprès du service Kinesis Data Streams lors du démarrage de l'application client (soit lorsque l'application client est initialisée, soit redémarrée) et également chaque fois qu'une partition en cours de traitement arrive à son terme (repartition). En d'autres termes, les applications de travail ou de KCL consommation sont synchronisées avec le flux de données qu'elles traitent lors du démarrage initial de l'application client et chaque fois que l'application client rencontre un événement de reconfiguration du flux de données.

Rubriques

- [Synchronisation en KCL 1.0 - 1.13 et KCL 2.0 - 2.2](#)
- [Synchronisation dans la KCL version 2.x, à partir de la version KCL 2.3 et des versions ultérieures](#)
- [Synchronisation dans la KCL version 1.x, à partir de la version KCL 1.14 et ultérieure](#)

Synchronisation en KCL 1.0 - 1.13 et KCL 2.0 - 2.2

Dans les KCL versions 1.0 à 1.13 et KCL 2.0 à 2.2, lors du démarrage de l'application client et également lors de chaque événement de reconfiguration du flux de données, KCL synchronise la table des baux avec les informations relatives aux partitions obtenues auprès du service Kinesis Data Streams en invoquant le ou la découverte. `ListShards` `DescribeStream` APIs Dans toutes les KCL versions répertoriées ci-dessus, chaque utilisateur d'une application KCL client effectue les étapes suivantes pour effectuer le processus de synchronisation entre location et partition lors du démarrage de l'application client et à chaque événement de redéfinition du flux :

- Récupère toutes les partitions contenant les données du flux en cours de traitement
- Récupère tous les baux de partitions à partir de la table des baux
- Filtre chaque partition ouverte qui n'a pas de bail dans la table des baux
- Effectue une itération sur toutes les partitions ouvertes trouvées et pour chaque partition ouverte sans parent ouvert :
 - Parcourt l'arbre hiérarchique en suivant le chemin de ses ancêtres pour déterminer si la partition est une descendante. Une partition est définie comme descendante si une partition ancestrale est actuellement en cours de traitement (c'est-à-dire, l'entrée de bail pour cette partition ancestrale est présente dans la table des baux) ou si une partition ancestrale est prévue pour être traitée (par exemple, si la position initiale est `TRIM_HORIZON` ou `AT_TIMESTAMP`)
 - Si le fragment ouvert dans le contexte est un descendant, le KCL point de contrôle du fragment en fonction de sa position initiale et crée des baux pour ses parents, si nécessaire

Synchronisation dans la KCL version 2.x, à partir de la version KCL 2.3 et des versions ultérieures

À partir des dernières versions prises en charge de KCL 2.x (KCL2.3) et versions ultérieures, la bibliothèque prend désormais en charge les modifications suivantes apportées au processus de synchronisation. Ces modifications de synchronisation entre bail et partition réduisent considérablement le nombre d'API appels passés par les applications KCL grand public vers le service Kinesis Data Streams et optimisent la gestion des baux dans votre application client. KCL

- Pendant le démarrage de l'application, si la table des baux est vide, KCL utilise l'option `ListShard` API de filtrage's (le paramètre de demande `ShardFilter` facultatif) pour récupérer et créer des baux uniquement pour un instantané des partitions ouvertes à l'heure spécifiée par le paramètre. `ShardFilter` Le `ShardFilter` paramètre vous permet de filtrer la réponse du `ListShards` API. La seule propriété obligatoire du paramètre `ShardFilter` est `Type`. KCL utilise la propriété `Type filter` et ses valeurs valides suivantes pour identifier et renvoyer un instantané des partitions ouvertes susceptibles de nécessiter de nouveaux baux :
 - `AT_TRIM_HORIZON` : la réponse inclut toutes les partitions ouvertes sur `TRIM_HORIZON`.
 - `AT_LATEST` : la réponse inclut uniquement les partitions actuellement ouvertes du flux de données.
 - `AT_TIMESTAMP` : la réponse inclut toutes les partitions dont l'horodatage de début est inférieur ou égal à l'horodatage donné et l'horodatage de fin est supérieur ou égal à l'horodatage donné ou encore ouvertes.

`ShardFilter` est utilisé lors de la création de baux pour une table des baux vide afin d'initialiser les baux pour un instantané des partitions spécifiées à `RetrievalConfig#initialPositionInStreamExtended`.

Pour plus d'informations sur `ShardFilter`, consultez https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.

- Au lieu que toutes les applications de travail effectuent la synchronisation entre bail et partition afin de maintenir la table des baux à jour avec les dernières partitions du flux de données, une seule application de travail leader effectue la synchronisation entre bail et partition.
- KCL 2.3 utilise le paramètre de `ChildShards` retour de `GetRecords` et `SubscribeToShard` APIs pour effectuer la synchronisation entre bail et partition qui se produit `SHARD_END` pour les partitions fermées, permettant à un KCL travailleur de créer des baux uniquement pour les fragments enfants de la partition qu'il a terminé de traiter. Pour les applications grand public partagées, cette optimisation de la synchronisation entre bail et partition utilise le `ChildShards` paramètre de `GetRecords` API. Pour les applications grand public à débit dédié (fan-out amélioré), cette optimisation de la synchronisation entre location et partition utilise le paramètre de `ChildShards` `SubscribeToShard` API. Pour plus d'informations, reportez-vous aux [SubscribeToShards](#) sections [GetRecords](#), et [ChildShard](#).
- Avec les modifications ci-dessus, le comportement de KCL passe du modèle selon lequel tous les travailleurs découvrent toutes les partitions existantes à un modèle selon lequel les travailleurs n'apprennent que les fragments enfants des fragments que chaque travailleur possède. Par conséquent, outre la synchronisation qui se produit lors des événements de démarrage et de

refonte des applications grand public, elle effectue KCL désormais des analyses périodiques supplémentaires des parties/locations afin d'identifier les éventuelles lacunes dans le tableau des baux (en d'autres termes, pour en savoir plus sur toutes les nouvelles partitions) afin de garantir le traitement de la plage de hachage complète du flux de données et de créer des baux pour celles-ci si nécessaire. `PeriodicShardSyncManager` est le composant chargé d'exécuter des analyses périodiques des baux et des partitions.

Pour plus d'informations sur `PeriodicShardSyncManager` la version KCL 2.3, consultez <https://github.com/aws-labs/amazon-kinesis-client/blob/master/src/main/java/software.amazon.kinesis/leases/.java#L201> `amazon-kinesis-client -L213 LeaseManagementConfig`.

Dans la KCL version 2.3, de nouvelles options de configuration sont disponibles pour la configuration `PeriodicShardSyncManager` dans `LeaseManagementConfig` :

Nom	Valeur par défaut	Description
leasesRecoveryAuditorExecutionFrequencyMillis	120 000 (2 minutes)	Fréquence (en millisecondes) à laquelle la tâche d'audit vérifie la présence de baux partiels dans la table des baux. Si l'auditeur détecte une lacune dans les baux d'un flux, il déclenchera la synchronisation des partitions en fonction de leasesRecoveryAuditorInconsistencyConfidenceThreshold .

Nom	Valeur par défaut	Description
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Seuil de confiance pour la tâche périodique d'audit afin de déterminer si les baux d'un flux de données dans la table des baux sont incohérents. Si l'auditeur identifie le même ensemble d'incohérences consécutivement pour un flux de données ce nombre de fois, il déclenche alors une synchronisation des partitions.

De nouvelles CloudWatch mesures sont également désormais émises pour surveiller l'état de santé du `PeriodicShardSyncManager`. Pour plus d'informations, consultez [PeriodicShardSyncManager](#).

- Incluant une optimisation de `HierarchicalShardSyncer` pour créer des baux uniquement pour une couche de partition.

Synchronisation dans la KCL version 1.x, à partir de la version KCL 1.14 et ultérieure

À partir des dernières versions prises en charge de la version KCL 1.x (KCL1.14) et des versions ultérieures, la bibliothèque prend désormais en charge les modifications suivantes apportées au

processus de synchronisation. Ces modifications de synchronisation entre bail et partition réduisent considérablement le nombre d'APIcalls passés par les applications KCL grand public vers le service Kinesis Data Streams et optimisent la gestion des baux dans votre application client. KCL

- Pendant le démarrage de l'application, si la table des baux est vide, KCL utilise l'option `ListShard` API de filtrage's (le paramètre de demande `ShardFilter` facultatif) pour récupérer et créer des baux uniquement pour un instantané des partitions ouvertes à l'heure spécifiée par le paramètre. `ShardFilter` Le `ShardFilter` paramètre vous permet de filtrer la réponse du `ListShardsAPI`. La seule propriété obligatoire du paramètre `ShardFilter` est `Type`. KCL utilise la propriété `Type filter` et ses valeurs valides suivantes pour identifier et renvoyer un instantané des partitions ouvertes susceptibles de nécessiter de nouveaux baux :
 - `AT_TRIM_HORIZON` : la réponse inclut toutes les partitions ouvertes sur `TRIM_HORIZON`.
 - `AT_LATEST` : la réponse inclut uniquement les partitions actuellement ouvertes du flux de données.
 - `AT_TIMESTAMP` : la réponse inclut toutes les partitions dont l'horodatage de début est inférieur ou égal à l'horodatage donné et l'horodatage de fin est supérieur ou égal à l'horodatage donné ou encore ouvertes.

`ShardFilter` est utilisé lors de la création de baux pour une table des baux vide afin d'initialiser les baux pour un instantané des partitions spécifiées à `KinesisClientLibConfiguration#initialPositionInStreamExtended`.

Pour plus d'informations sur `ShardFilter`, consultez https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.

- Au lieu que toutes les applications de travail effectuent la synchronisation entre bail et partition afin de maintenir la table des baux à jour avec les dernières partitions du flux de données, une seule application de travail leader effectue la synchronisation entre bail et partition.
- KCL1.14 utilise le paramètre de `ChildShards` retour de `GetRecords` et `SubscribeToShard` APIs pour effectuer la synchronisation entre bail et partition qui se produit `SHARD_END` pour les partitions fermées, permettant à un KCL travailleur de créer des baux uniquement pour les fragments enfants de la partition qu'il a terminé de traiter. Pour plus d'informations, reportez-vous [GetRecords](#) aux sections et [ChildShard](#).
- Avec les modifications ci-dessus, le comportement de KCL passe du modèle selon lequel tous les travailleurs découvrent toutes les partitions existantes à un modèle selon lequel les travailleurs n'apprennent que les fragments enfants des fragments que chaque travailleur possède. Par conséquent, outre la synchronisation qui se produit lors des événements de démarrage et de

refonte des applications grand public, elle effectue KCL désormais des analyses périodiques supplémentaires des parties/locations afin d'identifier les éventuelles lacunes dans le tableau des baux (en d'autres termes, pour en savoir plus sur toutes les nouvelles partitions) afin de garantir le traitement de la plage de hachage complète du flux de données et de créer des baux pour celles-ci si nécessaire. `PeriodicShardSyncManager` est le composant chargé d'exécuter des analyses périodiques des baux et des partitions.

Lorsque `KinesisClientLibConfiguration#shardSyncStrategyType` est défini sur `ShardSyncStrategyType.SHARD_END`, `PeriodicShardSyncLeasesRecoveryAuditorInconsistencyConfidenceThreshold` est utilisé pour déterminer le seuil du nombre d'analyses consécutives révélant des lacunes dans la table des baux, après lequel une synchronisation des partitions est imposée. Lorsque `KinesisClientLibConfiguration#shardSyncStrategyType` est défini sur `ShardSyncStrategyType.PERIODIC`, `LeasesRecoveryAuditorInconsistencyConfidenceThreshold` est ignoré.

Pour plus d'informations sur la version KCL 1.14, consultez `PeriodicShardSyncManager` <https://github.com/aws-labs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/.java#L987-L999> `KinesisClientLibConfiguration`.

Dans la KCL version 1.14, une nouvelle option de configuration est disponible pour configurer `PeriodicShardSyncManager` dans `LeaseManagementConfig` :

Nom	Valeur par défaut	Description
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Seuil de confiance pour la tâche périodique d'audit afin de déterminer si les baux d'un flux de données dans la table des baux sont incohérents. Si l'auditeur identifie le même ensemble d'incohérences consécutivement pour un flux de données ce nombre de fois, il déclenche alors une synchronisation des partitions.

De nouvelles CloudWatch mesures sont également désormais émises pour surveiller l'état de santé du `PeriodicShardSyncManager`. Pour plus d'informations, consultez [PeriodicShardSyncManager](#).

- KCLLa version 1.14 prend désormais également en charge le nettoyage différé des baux. Les baux sont supprimés de manière asynchrone par `LeaseCleanupManager` lorsqu'ils atteignent `SHARD_END`, c'est-à-dire lorsqu'une partition a dépassé la période de conservation du flux de données ou a été fermée à la suite d'une opération de repartitionnement.

De nouvelles options de configuration sont disponibles pour la configuration de `LeaseCleanupManager`.

Nom	Valeur par défaut	Description
<code>leaseCleanupIntervalMillis</code>	1 minute	Intervalle d'exécution du thread de nettoyage des baux.
<code>completedLeaseCleanupIntervalMillis</code>	5 minutes	Intervalle au bout duquel il faut vérifier si un bail est terminé ou non.
<code>garbageLeaseCleanupIntervalMillis</code>	30 minutes	Intervalle à partir duquel il faut vérifier si un bail est nul (c'est-à-dire s'il a dépassé la période de conservation du flux de données) ou non.

- Incluant une optimisation de `KinesisShardSyncer` pour créer des baux uniquement pour une couche de partition.

Traitez plusieurs flux de données avec le même KCL 2.x pour une application grand public Java

Cette section décrit les modifications suivantes apportées à la KCL version 2.x pour Java, qui vous permettent de créer KCL des applications grand public capables de traiter plusieurs flux de données à la fois.

Important

Le traitement multiflux n'est pris en charge que dans la KCL version 2.x pour Java, à partir de la version KCL 2.3 pour Java et des versions ultérieures.

Le traitement multiflux est NOT pris en charge pour tous les autres langages dans lesquels la version KCL 2.x peut être implémentée.

Le traitement multiflux est NOT pris en charge dans toutes les versions de KCL 1.x.

- MultistreamTracker interface

Pour créer une application grand public capable de traiter plusieurs flux en même temps, vous devez implémenter une nouvelle interface appelée [MultistreamTracker](#). Cette interface inclut la `streamConfigList` méthode qui renvoie la liste des flux de données et leurs configurations à traiter par l'application KCL client. Notez que les flux de données en cours de traitement peuvent être modifiés pendant l'exécution de l'application grand public. `streamConfigList` est appelé périodiquement par le KCL pour prendre connaissance de l'évolution des flux de données à traiter.

La `streamConfigList` méthode remplit la [StreamConfigliste](#).

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

Notez que les champs `StreamIdentifiant` et `InitialPositionInStreamExtended` sont obligatoires, alors que `consumerArn` est facultatif. Vous ne devez fournir le `consumerArn` que si vous utilisez la version KCL 2.x pour implémenter une application grand public améliorée.

Pour plus d'informations `StreamIdentifiant`, consultez <https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/src/main/java/software/amazon/kinesis/common/.java#L129>. [amazon-kinesis-client StreamIdentifiant](#) Pour créer une `StreamIdentifiant`, nous vous recommandons de créer une instance `multistream` à partir de `streamArn` et `streamCreationEpoch` qui est disponible dans les versions 2.5.0 et ultérieures. Dans les KCL versions 2.3 et 2.4, qui ne sont pas prises en charge `streamArn`, créez une instance `multistream` en utilisant le format `account-id:StreamName:streamCreationTimestamp` Ce format sera obsolète et ne sera plus pris en charge à compter de la prochaine version majeure.

`MultiStreamTracker` inclut également une stratégie pour supprimer les baux des anciens flux dans la table des baux (`formerStreamsLeasesDeletionStrategy`).

Notez que la stratégie CANNOT sera modifiée pendant l'exécution de l'application grand public. Pour plus d'informations, consultez <https://github.com/awslabs/amazon-kinesis-client/blob/0c5042dadf794fe988438436252a5a8fe70b6b0b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/.java> `FormerStreamsLeasesDeletionStrategy`

- [ConfigsBuilder](#) est une classe à l'échelle de l'application que vous pouvez utiliser pour spécifier tous les paramètres de configuration KCL 2.x à utiliser lors de la création de votre KCL application grand public. `ConfigsBuilder` la classe prend désormais en charge l'`MultiStreamTracker` interface. Vous pouvez initialiser l'un `ConfigsBuilder` ou l'autre avec le nom du flux de données à partir duquel les enregistrements seront consommés :

```
/**
 * Constructor to initialize ConfigsBuilder with StreamName
 * @param streamName
 * @param applicationName
 * @param kinesisClient
 * @param dynamoDBClient
 * @param cloudWatchClient
 * @param workerIdentifier
 * @param shardRecordProcessorFactory
 */
```

```

    public ConfigsBuilder(@NonNull String streamName, @NonNull String
applicationName,
        @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
        @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
        @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
    this.appStreamTracker = Either.right(streamName);
    this.applicationName = applicationName;
    this.kinesisClient = kinesisClient;
    this.dynamoDBClient = dynamoDBClient;
    this.cloudWatchClient = cloudWatchClient;
    this.workerIdentifier = workerIdentifier;
    this.shardRecordProcessorFactory = shardRecordProcessorFactory;
}

```

Vous pouvez également l'initialiser ConfigsBuilder avec MultiStreamTracker si vous souhaitez implémenter une application KCL grand public qui traite plusieurs flux en même temps.

```

* Constructor to initialize ConfigsBuilder with MultiStreamTracker
* @param multiStreamTracker
* @param applicationName
* @param kinesisClient
* @param dynamoDBClient
* @param cloudWatchClient
* @param workerIdentifier
* @param shardRecordProcessorFactory
*/
    public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
        @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
        @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
        @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
    this.appStreamTracker = Either.left(multiStreamTracker);
    this.applicationName = applicationName;
    this.kinesisClient = kinesisClient;
    this.dynamoDBClient = dynamoDBClient;
    this.cloudWatchClient = cloudWatchClient;
    this.workerIdentifier = workerIdentifier;
    this.shardRecordProcessorFactory = shardRecordProcessorFactory;
}

```

```
}
```

- Grâce à la prise en charge multiflux mise en œuvre pour votre application KCL client, chaque ligne de la table des baux de l'application contient désormais l'ID de partition et le nom de flux des multiples flux de données traités par cette application.
- Lorsque le support multiflux pour votre application KCL grand public est implémenté, la leaseKey structure est la suivante :`account-id:StreamName:streamCreationTimestamp:ShardId`. Par exemple, `11111111:multiStreamTest-1:12345:shardId-000000000336`.

Important

Lorsque votre application KCL client existante est configurée pour traiter un seul flux de données, le leaseKey (qui est la clé de hachage de la table de location) est l'ID de partition. Si vous reconfigurez cette application KCL client existante pour traiter plusieurs flux de données, cela interrompt votre table de location, car avec le support multiflux, la leaseKey structure doit être la suivante : `account-id:StreamName:StreamCreationTimestamp:ShardId`

Utiliser le KCL avec le registre des AWS Glue schémas

Vous pouvez intégrer vos flux de données Kinesis au registre des AWS Glue schémas. Le registre des AWS Glue schémas vous permet de découvrir, de contrôler et de faire évoluer les schémas de manière centralisée, tout en garantissant que les données produites sont validées en permanence par un schéma enregistré. Un schéma définit la structure et le format d'un enregistrement de données. Un schéma est une spécification versionnée pour la publication, la consommation ou le stockage des données fiables. Le registre des AWS Glue schémas vous permet d'améliorer la qualité end-to-end des données et la gouvernance des données au sein de vos applications de streaming. Pour plus d'informations, consultez le registre [AWS Glue Schema](#) (français non garanti). L'un des moyens de configurer cette intégration consiste à utiliser le KCL en Java.

Important

Actuellement, l'intégration de Kinesis Data Streams AWS Glue et de Schema Registry n'est prise en charge que pour les flux de données Kinesis qui KCL utilisent des consommateurs 2.3 implémentés en Java. La prise en charge multilingue n'est pas fournie. KCLLes

consommateurs 1.0 ne sont pas pris en charge. KCL Les utilisateurs 2.x antérieurs à la version KCL 2.3 ne sont pas pris en charge.

Pour obtenir des instructions détaillées sur la façon de configurer l'intégration de Kinesis Data Streams à Schema Registry à l'aide de, consultez KCL la section « Interaction avec les données en utilisant KPL lesKCL/Libraries » [dans Use Case : Integrating Amazon Kinesis Data Streams with the AWS Glue Schema Registry](#).

Développez des consommateurs personnalisés avec un débit partagé

Si vous n'avez pas besoin d'un débit dédié lors de la réception des données à partir de Kinesis Data Streams, et si vous n'avez pas besoin de lire les retards de propagation sous 200 ms, vous pouvez créer des applications consommateur comme décrit dans les rubriques suivantes. Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) ou le AWS SDK for Java

Rubriques

- [Développez des consommateurs personnalisés avec un débit partagé en utilisant KCL](#)
- [Développez des consommateurs personnalisés avec un débit partagé à l'aide du AWS SDK for Java](#)

Pour plus d'informations sur la création d'applications consommateur pouvant recevoir des enregistrements provenant des flux de données Kinesis avec un débit dédié, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Développez des consommateurs personnalisés avec un débit partagé en utilisant KCL

L'une des méthodes de développement d'une application client personnalisée avec un débit partagé consiste à utiliser la bibliothèque KCL cliente Kinesis ().

Choisissez l'une des rubriques suivantes pour la KCL version que vous utilisez.

Rubriques

- [Développez des KCL consommateurs 1.x](#)

- [Développez des KCL consommateurs 2.x](#)

Développez des KCL consommateurs 1.x

Vous pouvez développer une application grand public pour Amazon Kinesis Data Streams à l'aide de la bibliothèque client Kinesis (). KCL

Pour plus d'informations sur KCL, voir [Qu'est-ce que Kinesis Client Library ?](#) .

Choisissez l'une des rubriques suivantes en fonction de l'option que vous souhaitez utiliser.

Table des matières

- [Développez un client de bibliothèque cliente Kinesis en Java](#)
- [Développez un client de bibliothèque cliente Kinesis dans Node.js](#)
- [Développez un client de la bibliothèque cliente Kinesis dans .NET](#)
- [Développez un client de bibliothèque cliente Kinesis en Python](#)
- [Développez un client de bibliothèque cliente Kinesis dans Ruby](#)

Développez un client de bibliothèque cliente Kinesis en Java

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique présente Java. Pour consulter la référence Javadoc, consultez la rubrique [AWS Javadoc](#) consacrée à Class. AmazonKinesisClient

Pour télécharger le Java KCL depuis GitHub, accédez à la [bibliothèque cliente Kinesis \(Java\)](#). Pour localiser le Java KCL sur Apache Maven, rendez-vous sur la page des [résultats KCL de recherche](#). Pour télécharger un exemple de code pour une application Java KCL grand public depuis GitHub, rendez-vous sur la page [d'exemple de projet KCL pour Java](#) sur GitHub.

L'exemple d'application utilise [Apache Commons Logging](#). Vous pouvez modifier la configuration de la journalisation dans la méthode statique `configure` définie dans le fichier `AmazonKinesisApplicationSample.java`. Pour plus d'informations sur l'utilisation d'Apache Commons Logging avec Log4j et les applications AWS Java, consultez la section [Logging with Log4j](#) dans le manuel du développeur.AWS SDK for Java

Vous devez effectuer les tâches suivantes lors de l'implémentation d'une application KCL grand public en Java :

Tâches

- [Implémenter les IRecordProcessor méthodes](#)
- [Implémenter une fabrique de classes pour l'IRecordProcessorinterface](#)
- [Création d'un travailleur](#)
- [Modifier les propriétés de configuration](#)
- [Migrer vers la version 2 de l'interface du processeur d'enregistrements](#)

Implémenter les IRecordProcessor méthodes

KCL prend actuellement en charge deux versions de l'IRecordProcessorinterface : l'interface d'origine est disponible avec la première version du KCL, et la version 2 est disponible à partir de la version 1.5.0. Les deux interfaces sont entièrement prises en charge. Votre choix dépend des exigences spécifiques à votre scénario. Reportez-vous à vos Javadocs locales ou au code source pour voir toutes les différences. Les sections suivantes décrivent l'implémentation minimale pour la mise en route.

IRecordProcessorVersions

- [Interface d'origine \(Version 1\)](#)
- [Interface mise à jour \(version 2\)](#)

Interface d'origine (Version 1)

L'interface d'origine IRecordProcessor (package `com.amazonaws.services.kinesis.clientlibrary.interfaces`) expose les méthodes de processeur d'enregistrements suivantes que votre application producteur doit implémenter. L'exemple fournit des implémentations que vous pouvez utiliser comme point de départ (voir `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpoint
    checkpointer)
public void shutdown(IRecordProcessorCheckpoint
    checkpointer, ShutdownReason reason)
```

initialisation

Il KCL appelle la `initialize` méthode lorsque le processeur d'enregistrement est instancié, en transmettant un ID de partition spécifique en tant que paramètre. Ce processeur d'enregistrements

traite uniquement cette partition et, en règle générale, l'inverse est également vrai (cette partition est traitée uniquement par ce processeur d'enregistrements). Cependant, votre application consommateur doit prendre en compte la possibilité qu'un enregistrement de données peut être traité plusieurs fois. Kinesis Data Streams a la sémantique au moins une fois, ce qui signifie que chaque enregistrement de données issu d'une partition est traité au moins une fois par une application de travail dans votre application consommateur. Pour plus d'informations sur les cas dans lesquels une partition spécifique peut être traitée par plusieurs applications de travail, consultez la page [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).

```
public void initialize(String shardId)
```

processRecords

Il KCL appelle la `processRecords` méthode en transmettant une liste d'enregistrements de données provenant de la partition spécifiée par la `initialize(shardId)` méthode. Le processeur d'enregistrements traite les données contenues dans ces enregistrements selon la sémantique de l'application consommateur. Par exemple, l'application de travail peut exécuter une transformation sur les données et stocker ensuite le résultat dans un compartiment Amazon Simple Storage Service (Amazon S3).

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointter  
    checkpointter)
```

En plus des données elles-même, l'enregistrement contient également un numéro de séquence et une clé de partition. L'application de travail utilise ces valeurs lors du traitement des données. Par exemple, l'application de travail peut choisir le compartiment S3 dans lequel stocker les données en fonction de la valeur de la clé de partition. La classe `Record` expose les méthodes suivantes qui permettent d'accéder aux données, numéro de séquence et clé de partition de l'enregistrement.

```
record.getData()  
record.getSequenceNumber()  
record.getPartitionKey()
```

Dans l'exemple, la méthode privée `processRecordsWithRetries` contient du code qui indique comment une application de travail peut accéder aux données, numéro de séquence et clé de partition de l'enregistrement.

Kinesis Data Streams exige que le processeur d'enregistrements effectue le suivi des enregistrements qui ont déjà été traités dans une partition. Le KCL se charge de ce suivi pour

vous en passant un `checkpoint` (`IRecordProcessorCheckpoint`) à `processRecords`. Le processeur d'enregistrements appelle la `checkpoint` méthode sur cette interface pour indiquer dans quelle mesure il a progressé dans le traitement des enregistrements de la partition. KCL Si le worker échoue, il KCL utilise ces informations pour relancer le traitement de la partition au dernier enregistrement traité connu.

Dans le cas d'une opération de division ou de fusion, le traitement des nouvelles partitions KCL ne commencera pas tant que les processeurs des partitions d'origine n'auront pas appelé `checkpoint` pour signaler que le traitement des partitions d'origine est terminé.

Si vous ne transmettez aucun paramètre, cela KCL suppose que l'appel à `checkpoint` signifie que tous les enregistrements ont été traités, jusqu'au dernier enregistrement transmis au processeur d'enregistrements. Par conséquent, le processeur d'enregistrements doit appeler `checkpoint` seulement après avoir traité tous les enregistrements de la liste qui lui a été passée. Les processeurs d'enregistrements n'ont pas besoin d'appeler `checkpoint` à chaque appel de `processRecords`. Un processeur pourrait, par exemple, appeler `checkpoint` à chaque troisième appel de `processRecords`. Vous pouvez éventuellement spécifier le numéro de séquence précis d'un enregistrement comme paramètre à `checkpoint`. Dans ce cas, le KCL suppose que tous les enregistrements ont été traités jusqu'à cet enregistrement uniquement.

Dans l'exemple, la méthode privée `checkpoint` montre comment appeler `IRecordProcessorCheckpoint`. `checkpoint` en utilisant la logique appropriée de traitement des exceptions et de nouvelle tentative.

Il KCL s'appuie sur le `processRecords` traitement des exceptions liées au traitement des enregistrements de données. Si une exception est émise `processRecords`, les enregistrements KCL de données transmis avant l'exception sont ignorés. En d'autres termes, ces enregistrements ne sont pas renvoyés au processeur d'enregistrements qui a lancé l'exception ou à tout autre processeur d'enregistrement dans l'application consommateur.

shutdown

Il KCL appelle la `shutdown` méthode soit lorsque le traitement est terminé (la raison de l'arrêt est `TERMINATE`), soit lorsque le travailleur ne répond plus (la raison de l'arrêt est `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpoint checkpoint, ShutdownReason reason)
```

Le traitement se termine lorsque le processeur d'enregistrements ne reçoit plus d'enregistrements de la partition, car la partition a été fractionnée ou fusionnée, ou le flux a été supprimé.

KCLII transmet également une `IRecordProcessorCheckpoint` interface à `shutdown`. Si le motif de fermeture est `TERMINATE`, le processeur d'enregistrements doit terminer le traitement des enregistrements de données et appeler ensuite la méthode `checkpoint` sur cette interface.

Interface mise à jour (version 2)

L'interface `IRecordProcessor` mise à jour (package `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) expose les méthodes de processeur d'enregistrements suivantes que votre application producteur doit implémenter :

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Tous les arguments provenant de la version initiale de l'interface sont accessibles via les méthodes `get` sur les objets de conteneur. Par exemple, pour extraire la liste des enregistrements dans `processRecords()`, vous pouvez utiliser `processRecordsInput.getRecords()`.

À partir de la version 2 de cette interface (KCL1.5.0 et versions ultérieures), les nouvelles entrées suivantes sont disponibles en plus des entrées fournies par l'interface d'origine :

Numéro de séquence de début

Dans l'objet `InitializationInput` passé à l'opération `initialize()`, le numéro de séquence de début à partir duquel les enregistrements sont fournis à l'instance de processeur d'enregistrements. C'est le numéro de séquence qui a été contrôlé en dernier par l'instance de processeur d'enregistrements qui a traité précédemment la même partition. Il est fourni si votre application a besoin de cette information.

Numéro de séquence de point de contrôle en attente

Dans l'objet `InitializationInput` passé à l'opération `initialize()`, le numéro de séquence de point de contrôle en attente (le cas échéant) qui n'a pas pu être validé avant l'arrêt de l'instance de processeur d'enregistrements précédente.

Implémenter une fabrique de classes pour l'`IRecordProcessor` interface

Vous avez aussi besoin d'implémenter une fabrique pour la classe qui implémente les méthodes de processeur d'enregistrements. Lorsque votre application consommateur instancie l'application de travail, elle passe une référence à cette fabrique.

L'exemple implémente la classe `Factory` dans le fichier `AmazonKinesisApplicationSampleRecordProcessorFactory.java` à l'aide de l'interface de processeur d'enregistrements d'origine. Si vous voulez que la fabrique de classes crée des processeurs d'enregistrements version 2, utilisez le nom de package `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
     * Constructor.
     */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
     * {@inheritDoc}
     */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Création d'un travailleur

Comme indiqué dans [Implémenter les IRecordProcessor méthodes](#), il existe deux versions de l'interface du processeur d'KCLenregistrements parmi lesquelles choisir, ce qui influe sur la façon dont vous créeriez un travailleur. L'interface de processeur d'enregistrements d'origine utilise la structure de code suivante pour créer un travail :

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Avec la version 2 de l'interface de processeur d'enregistrements, vous pouvez utiliser `Worker.Builder` pour créer un travail sans avoir à vous soucier du constructeur à utiliser et de l'ordre des arguments. L'interface de processeur d'enregistrements mise à jour utilise la structure de code suivante pour créer un travail :

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
```

```
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Modifier les propriétés de configuration

L'exemple fournit les valeurs par défaut des propriétés de configuration. Ces données de configuration du travail sont ensuite consolidées dans un objet `KinesisClientLibConfiguration`. Cet objet et une référence à la fabrique de classes pour `IRecordProcessor` sont passés dans l'appel qui instancie l'application de travail. Vous pouvez remplacer ces propriétés par vos propres valeurs en utilisant un fichier de propriétés Java (voir `AmazonKinesisApplicationSample.java`).

Nom de l'application

KCLCela nécessite un nom d'application unique pour toutes vos applications et pour toutes les tables Amazon DynamoDB d'une même région. Elle utilise la valeur de configuration du nom d'application des manières suivantes :

- Tous les programmes d'exécution associés à ce nom d'application sont considérés comme rattachés au même flux. Ces programmes d'exécution peuvent être répartis sur plusieurs instances. Si vous exécutez une instance supplémentaire du même code d'application, mais avec un nom d'application différent, KCL la deuxième instance est traitée comme une application entièrement distincte qui fonctionne également sur le même flux.
- KCL crée une table DynamoDB avec le nom de l'application et utilise la table pour gérer les informations d'état (telles que les points de contrôle et le mappage worker-shard) de l'application. Chaque application a son propre tableau DynamoDB. Pour de plus amples informations, veuillez consulter [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#).

Configurer les informations d'identification

Vous devez mettre vos AWS informations d'identification à la disposition de l'un des fournisseurs d'informations d'identification de la chaîne de fournisseurs d'informations d'identification par défaut. Par exemple, si vous exécutez votre client sur une EC2 instance, nous vous recommandons de lancer l'instance avec un IAM rôle. AWS les informations d'identification qui reflètent les autorisations associées à ce IAM rôle sont mises à la disposition des applications de l'instance

via ses métadonnées d'instance. Il s'agit de la méthode la plus sûre pour gérer les informations d'identification d'un client exécutant une EC2 instance.

L'exemple d'application tente d'abord de récupérer les IAM informations d'identification à partir des métadonnées de l'instance :

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Si l'exemple d'application ne peut pas obtenir les informations d'identification à partir des métadonnées d'instance, il tente d'extraire les informations d'identification d'un fichier de propriétés :

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Pour plus d'informations sur les métadonnées d'instance, consultez la section [Métadonnées d'instance](#) dans le guide de EC2 l'utilisateur Amazon.

Utiliser l'ID du travailleur pour plusieurs instances

L'exemple de code d'initialisation crée un ID `workerId` pour l'application de travail, en utilisant le nom de l'ordinateur local et en y ajoutant un identifiant unique dans le monde entier, comme illustré dans l'extrait de code ci-après. Cette approche prend en charge le scénario où plusieurs instances de l'application consommateur sont exécutées sur le même ordinateur.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +  
    UUID.randomUUID();
```

Migrer vers la version 2 de l'interface du processeur d'enregistrements

Si vous souhaitez migrer le code qui utilise l'interface d'origine, les étapes suivantes sont nécessaires en plus de celles décrites précédemment :

1. Changez la classe de processeur d'enregistrements pour importer la version 2 de l'interface de processeur d'enregistrements :

```
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
```

2. Modifiez les références aux entrées pour utiliser des méthodes `get` sur les objets de conteneur. Par exemple, dans l'opération `shutdown()`, remplacez `checkpointer` par `shutdownInput.getCheckpointter()`.

3. Changez la classe de la fabrique de processeurs d'enregistrements pour importer la version 2 de l'interface de fabrique de processeurs d'enregistrements :

```
import
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
```

4. Modifiez la construction de l'application de travail pour utiliser `Worker.Builder`. Par exemple :

```
final Worker worker = new Worker.Builder()
  .recordProcessorFactory(recordProcessorFactory)
  .config(config)
  .build();
```

Développez un client de bibliothèque cliente Kinesis dans Node.js

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique présente Node.js.

KCLII s'agit d'une bibliothèque Java ; la prise en charge de langages autres que Java est fournie à l'aide d'une interface multilingue appelée. MultiLangDaemon Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par conséquent, si vous installez le fichier KCL for Node.js et que vous écrivez votre application client entièrement dans Node.js, vous devez toujours installer Java sur votre système en raison du MultiLangDaemon. En outre, MultiLangDaemon il comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations MultiLangDaemon sur l'activation GitHub, rendez-vous sur la page [KCL MultiLangDaemon du projet](#).

Pour télécharger le fichier Node.js KCL depuis GitHub, accédez à la [bibliothèque cliente Kinesis \(Node.js\)](#).

Téléchargements des exemples de code

Deux exemples de code sont disponibles KCL dans Node.js :

- [basic-sample](#)

Utilisé dans les sections suivantes pour illustrer les principes fondamentaux de la création d'une application KCL grand public dans Node.js.

- [click-stream-sample](#)

Il est un peu plus avancé et se sert d'un scénario réel. A utiliser après vous être familiarisé avec l'exemple de code de base. Cet exemple n'est pas abordé ici mais contient un README fichier contenant plus d'informations.

Vous devez effectuer les tâches suivantes lors de l'implémentation d'une application KCL grand public dans Node.js :

Tâches

- [Implémenter le processeur d'enregistrement](#)
- [Modifier les propriétés de configuration](#)

Implémenter le processeur d'enregistrement

Le consommateur le plus simple possible utilisant KCL for Node.js doit implémenter une `recordProcessor` fonction, qui contient à son tour les fonctions `initializeProcessRecords`, `etshutdown`. L'exemple fournit une implémentation que vous pouvez utiliser comme point de départ (voir `sample_kcl_app.js`).

```
function recordProcessor() {  
  // return an object that implements initialize, processRecords and shutdown  
  functions.}
```

initialisation

La KCL `initialize` fonction est appelée lorsque le processeur d'enregistrement démarre. Ce processeur d'enregistrements traite uniquement l'ID de partition passé à `initializeInput.shardId` et, en règle générale, l'inverse est également vrai (cette partition est traitée uniquement par ce processeur d'enregistrements). Cependant, votre application consommateur doit prendre en compte la possibilité qu'un enregistrement de données peut être traité plusieurs fois. Cela provient du fait que Kinesis Data Streams a la sémantique au moins une fois, qui signifie que chaque enregistrement de données issu d'une partition est traité au moins une fois par une application de travail dans votre application consommateur. Pour plus d'informations sur les cas dans lesquels une partition spécifique peut éventuellement être traitée par plusieurs applications de travail, consultez la page [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).


```
initialize: function(initializeInput, completeCallback)
```

processRecords

KCL appelle cette fonction avec une entrée contenant une liste d'enregistrements de données provenant de la partition spécifiée pour la `initialize` fonction. Le processeur d'enregistrements que vous implémentez traite les données figurant dans ces enregistrements suivant la sémantique de votre application consommateur. Par exemple, l'application de travail peut exécuter une transformation sur les données et stocker ensuite le résultat dans un compartiment Amazon Simple Storage Service (Amazon S3).

```
processRecords: function(processRecordsInput, completeCallback)
```

En plus des données elles-même, l'enregistrement contient également un numéro de séquence et une clé de partition, que l'application de travail peut utiliser pour traiter les données. Par exemple, l'application de travail peut choisir le compartiment S3 dans lequel stocker les données en fonction de la valeur de la clé de partition. Le dictionnaire `record` expose les paires clé-valeur suivantes pour accéder aux données, numéro de séquence et clé de partition de l'enregistrement :

```
record.data  
record.sequenceNumber  
record.partitionKey
```

Notez que les données sont encodées en Base64.

Dans l'exemple de base, la fonction `processRecords` contient du code qui indique comment une application de travail peut accéder aux données, numéro de séquence et clé de partition de l'enregistrement.

Kinesis Data Streams exige que le processeur d'enregistrements effectue le suivi des enregistrements qui ont déjà été traités dans une partition. KCL s'occupe de ce suivi avec un `checkpointer` objet transmis sous le nom de `processRecordsInput.checkpointer`. Votre processeur de fichiers appelle la `checkpointer.checkpoint` fonction pour indiquer dans KCL quelle mesure il a progressé dans le traitement des enregistrements de la partition. En cas d'échec du worker, il KCL utilise ces informations lorsque vous redémarrez le traitement de la partition afin qu'il continue à partir du dernier enregistrement traité connu.

Dans le cas d'une opération de division ou de fusion, le traitement des nouvelles partitions KCL ne commence pas tant que les processeurs des partitions d'origine n'ont pas appelé `checkpoint` pour signaler que le traitement des partitions d'origine est terminé.

Si vous ne transmettez pas le numéro de séquence à la `checkpoint` fonction, cela KCL suppose que l'appel à `checkpoint` signifie que tous les enregistrements ont été traités, jusqu'au dernier enregistrement transmis au processeur d'enregistrements. Par conséquent, le processeur d'enregistrements doit appeler `checkpoint` seulement après avoir traité tous les enregistrements de la liste qui lui a été passée. Les processeurs d'enregistrements n'ont pas besoin d'appeler `checkpoint` à chaque appel de `processRecords`. Un processeur peut, par exemple, appeler `checkpoint` tous les trois appels ou lors d'un événement externe au processeur d'enregistrements, tel qu'un service de vérification/validation personnalisé que vous avez implémenté.

Vous pouvez éventuellement spécifier le numéro de séquence précis d'un enregistrement comme paramètre à `checkpoint`. Dans ce cas, le KCL suppose que tous les enregistrements ont été traités jusqu'à cet enregistrement uniquement.

L'exemple d'application de base montre l'appel le plus simple possible de la fonction `shutdownInput.checkpointer`. Vous pouvez ajouter à la fonction une autre logique de points de contrôle nécessaire pour votre application consommateur à ce stade.

shutdown

Il KCL appelle la `shutdown` fonction soit lorsque le traitement se termine (`shutdownInput.reasonestTERMINATE`), soit lorsque le travailleur ne répond plus (`shutdownInput.reasonestZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

Le traitement se termine lorsque le processeur d'enregistrements ne reçoit plus d'enregistrements de la partition, car la partition a été fractionnée ou fusionnée, ou le flux a été supprimé.

KCL transmet également un `shutdownInput.checkpointer` objet à `shutdown`. Si le motif de fermeture est `TERMINATE`, vous devez vous assurer que le processeur d'enregistrements a fini de traiter les enregistrements de données et appeler ensuite la fonction `checkpoint` sur cet objet.

Modifier les propriétés de configuration

L'exemple fournit les valeurs par défaut des propriétés de configuration. Vous pouvez remplacer ces propriétés par vos propres valeurs (voir `sample.properties` dans l'exemple de base).

Nom de l'application

Cela KCL nécessite une application unique parmi vos applications et parmi les tables Amazon DynamoDB d'une même région. Elle utilise la valeur de configuration du nom d'application des manières suivantes :

- Tous les programmes d'exécution associés à ce nom d'application sont considérés comme rattachés au même flux. Ces programmes d'exécution peuvent être répartis sur plusieurs instances. Si vous exécutez une instance supplémentaire du même code d'application, mais avec un nom d'application différent, KCL la deuxième instance est traitée comme une application entièrement distincte qui fonctionne également sur le même flux.
- KCL crée une table DynamoDB avec le nom de l'application et utilise la table pour gérer les informations d'état (telles que les points de contrôle et le mappage worker-shard) de l'application. Chaque application a son propre tableau DynamoDB. Pour de plus amples informations, veuillez consulter [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client.](#)

Configurer les informations d'identification

Vous devez mettre vos AWS informations d'identification à la disposition de l'un des fournisseurs d'informations d'identification de la chaîne de fournisseurs d'informations d'identification par défaut. Vous pouvez utiliser la propriété `AWSCredentialsProvider` pour définir un fournisseur d'informations d'identification. Le fichier `sample.properties` doit mettre vos informations d'identification à disposition de l'un des fournisseurs d'informations d'identification appartenant à la [chaîne des fournisseurs d'informations d'identification par défaut](#). Si vous exécutez votre client sur une EC2 instance Amazon, nous vous recommandons de configurer l'instance avec un IAM rôle. AWS les informations d'identification qui reflètent les autorisations associées à ce IAM rôle sont mises à la disposition des applications de l'instance via ses métadonnées d'instance. Il s'agit de la méthode la plus sûre pour gérer les informations d'identification d'une application grand public exécutée sur une EC2 instance.

L'exemple suivant est configuré KCL pour traiter un flux de données Kinesis `kclnodejssample` nommé à l'aide du processeur d'enregistrement fourni dans : `sample_kcl_app.js`

```
# The Node.js executable script
executableName = node sample_kcl_app.js
# The name of an Amazon Kinesis stream to process
streamName = kclnodejssample
# Unique KCL application name
applicationName = kclnodejssample
```

```
# Use default AWS credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

Développez un client de la bibliothèque cliente Kinesis dans .NET

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique traite de .NET.

KCLII s'agit d'une bibliothèque Java ; la prise en charge de langages autres que Java est fournie à l'aide d'une interface multilingue appelée MultiLangDaemon. Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par conséquent, si vous installez le KCL pour .NET et inscrivez entièrement votre application destinée aux consommateurs .NET, vous devez toujours installer Java sur votre système en raison du MultiLangDaemon. En outre, MultiLangDaemon comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations MultiLangDaemon sur l'activation GitHub, rendez-vous sur la page [KCL MultiLangDaemon du projet](#).

Pour télécharger le .NETKCL à partir de GitHub, accédez à la [bibliothèque cliente Kinesis \(.NET\)](#). Pour télécharger un exemple de code pour un .NETKCL application destinée aux consommateurs, accédez au formulaire de [KCL demande .NET exemple de page de projet destiné aux consommateurs](#) sur GitHub.

Vous devez effectuer les tâches suivantes lors de la mise en œuvre d'une application KCL grand public dans .NET:

Tâches

- [Implémenter les méthodes IRecordProcessor de classe](#)
- [Modifier les propriétés de configuration](#)

Implémenter les méthodes IRecordProcessor de classe

L'application consommateur doit implémenter les méthodes suivantes pour IRecordProcessor. L'exemple d'application consommateur fournit des implémentations que vous pouvez utiliser comme point de départ (voir la classe SampleRecordProcessor dans SampleConsumer/AmazonKinesisSampleConsumer.cs).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Initialiser

Il KCL appelle cette méthode lorsque le processeur d'enregistrements est instancié, en transmettant un identifiant de partition spécifique dans le `input` paramètre (`input.ShardId`). Ce processeur d'enregistrements traite uniquement cette partition et, en règle générale, l'inverse est également vrai (cette partition est traitée uniquement par ce processeur d'enregistrements). Cependant, votre application consommateur doit prendre en compte la possibilité qu'un enregistrement de données peut être traité plusieurs fois. Cela provient du fait que Kinesis Data Streams a la sémantique au moins une fois, qui signifie que chaque enregistrement de données issu d'une partition est traité au moins une fois par une application de travail dans votre application consommateur. Pour plus d'informations sur les cas dans lesquels une partition spécifique peut éventuellement être traitée par plusieurs applications de travail, consultez la page [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).

```
public void Initialize(InitializationInput input)
```

ProcessRecords

Il KCL appelle cette méthode en transmettant une liste d'enregistrements de données dans le `input` paramètre (`input.Records`) à partir de la partition spécifiée par la `Initialize` méthode. Le processeur d'enregistrements que vous implémentez traite les données figurant dans ces enregistrements suivant la sémantique de votre application consommateur. Par exemple, l'application de travail peut exécuter une transformation sur les données et stocker ensuite le résultat dans un compartiment Amazon Simple Storage Service (Amazon S3).

```
public void ProcessRecords(ProcessRecordsInput input)
```

En plus des données elles-même, l'enregistrement contient également un numéro de séquence et une clé de partition. L'application de travail utilise ces valeurs lors du traitement des données. Par exemple, l'application de travail peut choisir le compartiment S3 dans lequel stocker les données en fonction de la valeur de la clé de partition. La classe `Record` expose le code suivant pour accéder aux données, numéro de séquence et clé de partition de l'enregistrement :

```
byte[] Record.Data
```

```
string Record.SequenceNumber
string Record.PartitionKey
```

Dans l'exemple, la méthode `ProcessRecordsWithRetries` contient du code qui montre comment une application de travail peut accéder aux données, numéro de séquence et clé de partition de l'enregistrement.

Kinesis Data Streams exige que le processeur d'enregistrements effectue le suivi des enregistrements qui ont déjà été traités dans une partition. Le KCL se charge de ce suivi pour vous en transmettant un `Checkpoint` objet à `ProcessRecords` (`input.Checkpointer`). Le processeur d'enregistrements appelle la `Checkpoint` méthode pour indiquer dans quelle mesure il a progressé dans le traitement des enregistrements de la partition. KCL Si le worker échoue, il KCL utilise ces informations pour relancer le traitement de la partition au dernier enregistrement traité connu.

Dans le cas d'une opération de division ou de fusion, le traitement des nouvelles partitions KCL ne commence pas tant que les processeurs des partitions d'origine n'ont pas appelé `Checkpoint` pour signaler que le traitement des partitions d'origine est terminé.

Si vous ne transmettez aucun paramètre, KCL l'appel à `Checkpoint` signifie que tous les enregistrements ont été traités, jusqu'au dernier enregistrement transmis au processeur d'enregistrements. Par conséquent, le processeur d'enregistrements doit appeler `Checkpoint` seulement après avoir traité tous les enregistrements de la liste qui lui a été passée. Les processeurs d'enregistrements n'ont pas besoin d'appeler `Checkpoint` à chaque appel de `ProcessRecords`. Un processeur peut, par exemple, appeler `Checkpoint` tous les trois ou quatre appels. Vous pouvez éventuellement spécifier le numéro de séquence précis d'un enregistrement comme paramètre à `Checkpoint`. Dans ce cas, le KCL suppose que les enregistrements n'ont été traités que jusqu'à cet enregistrement.

Dans l'exemple, la méthode privée `Checkpoint(Checkpointer checkpointer)` montre comment appeler la méthode `Checkpoint` en utilisant la logique appropriée de traitement des exceptions et de nouvelle tentative.

Le KCL pour .NET gère les exceptions différemment des autres bibliothèques KCL linguistiques en ce sens qu'elle ne gère aucune exception résultant du traitement des enregistrements de données. Toutes les exceptions non interceptées dans le code utilisateur bloquent le programme.

Fermeture

Il KCL appelle la Shutdown méthode soit lorsque le traitement est terminé (la raison de l'arrêt est `TERMINATE`), soit lorsque le travailleur ne répond plus (la `input.Reason` valeur d'arrêt est `ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

Le traitement se termine lorsque le processeur d'enregistrements ne reçoit plus d'enregistrements de la partition, car la partition a été fractionnée ou fusionnée, ou le flux a été supprimé.

KCL Transmet également un `Checkpoint` objet à `shutdown`. Si le motif de fermeture est `TERMINATE`, le processeur d'enregistrements doit terminer le traitement des enregistrements de données et appeler ensuite la méthode `checkpoint` sur cette interface.

Modifier les propriétés de configuration

L'exemple d'application consommateur fournit les valeurs par défaut des propriétés de configuration. Vous pouvez remplacer ces propriétés par vos propres valeurs (voir `SampleConsumer/kcl.properties`).

Nom de l'application

Cela KCL nécessite une application unique parmi vos applications et parmi les tables Amazon DynamoDB d'une même région. Elle utilise la valeur de configuration du nom d'application des manières suivantes :

- Tous les programmes d'exécution associés à ce nom d'application sont considérés comme rattachés au même flux. Ces programmes d'exécution peuvent être répartis sur plusieurs instances. Si vous exécutez une instance supplémentaire du même code d'application, mais avec un nom d'application différent, KCL la deuxième instance est traitée comme une application entièrement distincte qui fonctionne également sur le même flux.
- KCL crée une table DynamoDB avec le nom de l'application et utilise la table pour gérer les informations d'état (telles que les points de contrôle et le mappage `worker-shard`) de l'application. Chaque application a son propre tableau DynamoDB. Pour de plus amples informations, veuillez consulter [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#).

Configurer les informations d'identification

Vous devez mettre vos AWS informations d'identification à la disposition de l'un des fournisseurs d'informations d'identification de la chaîne de fournisseurs d'informations d'identification par

défaut. Vous pouvez utiliser la propriété `AWSCredentialsProvider` pour définir un fournisseur d'informations d'identification. Le fichier [sample.properties](#) doit mettre vos informations d'identification à disposition de l'un des fournisseurs d'informations d'identification appartenant à la [chaîne des fournisseurs d'informations d'identification par défaut](#). Si vous exécutez votre application grand public sur une EC2 instance, nous vous recommandons de configurer l'instance avec un IAM rôle. AWS les informations d'identification qui reflètent les autorisations associées à ce IAM rôle sont mises à la disposition des applications de l'instance via ses métadonnées d'instance. Il s'agit de la méthode la plus sûre pour gérer les informations d'identification d'un client exécutant une EC2 instance.

Le fichier de propriétés de l'exemple est configuré KCL pour traiter un flux de données Kinesis appelé « mots » à l'aide du processeur d'enregistrement fourni dans `AmazonKinesisSampleConsumer.cs`

Développez un client de bibliothèque cliente Kinesis en Python

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique présente Python.

KCLII s'agit d'une bibliothèque Java ; la prise en charge de langages autres que Java est fournie à l'aide d'une interface multilingue appelée `MultiLangDaemon`. Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par conséquent, si vous installez le KCL pour Python et que vous écrivez votre application grand public entièrement en Python, vous devez toujours installer Java sur votre système en raison du `MultiLangDaemon`. En outre, `MultiLangDaemon` il comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations `MultiLangDaemon` sur l'activation GitHub, rendez-vous sur la page [KCL MultiLangDaemon du projet](#).

Pour télécharger le Python KCL depuis GitHub, accédez à la [bibliothèque cliente Kinesis \(Python\)](#). Pour télécharger un exemple de code pour une application KCL grand public Python, rendez-vous sur la page [d'exemple de projet KCL pour Python](#) sur GitHub.

Vous devez effectuer les tâches suivantes lors de l'implémentation d'une application KCL grand public en Python :

Tâches

- [Implémenter les méthodes `RecordProcessor` de classe](#)
- [Modifier les propriétés de configuration](#)

Implémenter les méthodes RecordProcessor de classe

La classe `RecordProcess` doit étendre la classe `RecordProcessorBase` pour implémenter les méthodes ci-après. L'exemple fournit des implémentations que vous pouvez utiliser comme point de départ (voir `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialisation

Il KCL appelle la `initialize` méthode lorsque le processeur d'enregistrement est instancié, en transmettant un ID de partition spécifique en tant que paramètre. Ce processeur d'enregistrements traite uniquement cette partition et, en règle générale, l'inverse est également vrai (cette partition est traitée uniquement par ce processeur d'enregistrements). Cependant, votre application consommateur doit prendre en compte la possibilité qu'un enregistrement de données peut être traité plusieurs fois. Cela provient du fait que Kinesis Data Streams a la sémantique au moins une fois, qui signifie que chaque enregistrement de données issu d'une partition est traité au moins une fois par une application de travail dans votre application consommateur. Pour plus d'informations sur les cas dans lesquels une partition spécifique peut être traitée par plusieurs applications de travail, consultez la page [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).

```
def initialize(self, shard_id)
```

process_records

Il KCL appelle cette méthode en transmettant une liste d'enregistrements de données provenant de la partition spécifiée par la `initialize` méthode. Le processeur d'enregistrements que vous implémentez traite les données figurant dans ces enregistrements suivant la sémantique de votre application consommateur. Par exemple, l'application de travail peut exécuter une transformation sur les données et stocker ensuite le résultat dans un compartiment Amazon Simple Storage Service (Amazon S3).

```
def process_records(self, records, checkpointer)
```

En plus des données elles-même, l'enregistrement contient également un numéro de séquence et une clé de partition. L'application de travail utilise ces valeurs lors du traitement des données. Par

exemple, l'application de travail peut choisir le compartiment S3 dans lequel stocker les données en fonction de la valeur de la clé de partition. Le dictionnaire `record` expose les paires clé-valeur suivantes pour accéder aux données, numéro de séquence et clé de partition de l'enregistrement :

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Notez que les données sont encodées en Base64.

Dans l'exemple, la méthode `process_records` contient du code qui montre comment une application de travail peut accéder aux données, numéro de séquence et clé de partition de l'enregistrement.

Kinesis Data Streams exige que le processeur d'enregistrements effectue le suivi des enregistrements qui ont déjà été traités dans une partition. Le KCL se charge de ce suivi pour vous en transmettant un `Checkpointer` objet à `process_records`. Le processeur d'enregistrements appelle la `checkpoint` méthode sur cet objet pour indiquer dans quelle mesure il a progressé dans le traitement des enregistrements de la partition. KCL Si le worker échoue, il KCL utilise ces informations pour relancer le traitement de la partition au dernier enregistrement traité connu.

Dans le cas d'une opération de division ou de fusion, le traitement des nouvelles partitions KCL ne commence pas tant que les processeurs des partitions d'origine n'ont pas appelé `checkpoint` pour signaler que le traitement des partitions d'origine est terminé.

Si vous ne transmettez aucun paramètre, cela KCL suppose que l'appel à `checkpoint` signifie que tous les enregistrements ont été traités, jusqu'au dernier enregistrement transmis au processeur d'enregistrements. Par conséquent, le processeur d'enregistrements doit appeler `checkpoint` seulement après avoir traité tous les enregistrements de la liste qui lui a été passée. Les processeurs d'enregistrements n'ont pas besoin d'appeler `checkpoint` à chaque appel de `process_records`. Un processeur peut, par exemple, appeler `checkpoint` tous les trois appels. Vous pouvez éventuellement spécifier le numéro de séquence précis d'un enregistrement comme paramètre à `checkpoint`. Dans ce cas, le KCL suppose que tous les enregistrements ont été traités jusqu'à cet enregistrement uniquement.

Dans l'exemple, la méthode privée `checkpoint` montre comment appeler la méthode `Checkpointer.checkpoint` en utilisant la logique appropriée de traitement des exceptions et de nouvelle tentative.

Il KCL s'appuie sur le `process_records` traitement des exceptions liées au traitement des enregistrements de données. Si une exception est émise `process_records`, les enregistrements de données transmis `process_records` avant l'exception sont KCL ignorés. En d'autres termes, ces enregistrements ne sont pas renvoyés au processeur d'enregistrements qui a lancé l'exception ou à tout autre processeur d'enregistrement dans l'application consommateur.

shutdown

Il KCL appelle la `shutdown` méthode soit lorsque le traitement est terminé (la raison de l'arrêt est `TERMINATE`), soit lorsque le travailleur ne répond plus (la raison est l'arrêt `ZOMBIE`).

```
def shutdown(self, checkpoint, reason)
```

Le traitement se termine lorsque le processeur d'enregistrements ne reçoit plus d'enregistrements de la partition, car la partition a été fractionnée ou fusionnée, ou le flux a été supprimé.

KCL transmet également un `Checkpoint` objet à `shutdown`. Si le motif de fermeture `reason` est `TERMINATE`, le processeur d'enregistrements doit terminer le traitement des enregistrements de données et appeler ensuite la méthode `checkpoint` sur cette interface.

Modifier les propriétés de configuration

L'exemple fournit les valeurs par défaut des propriétés de configuration. Vous pouvez remplacer ces propriétés par vos propres valeurs (voir `sample.properties`).

Nom de l'application

Le KCL nécessite un nom d'application unique parmi vos applications et parmi les tables Amazon DynamoDB de la même région. Elle utilise la valeur de configuration du nom d'application des manières suivantes :

- Tous les programmes d'exécution qui sont associés à ce nom d'application sont considérés comme rattachés au même flux. Ces programmes d'exécution peuvent être répartis sur plusieurs instances. Si vous exécutez une instance supplémentaire du même code d'application, mais avec un nom d'application différent, KCL la deuxième instance est traitée comme une application entièrement distincte qui fonctionne également sur le même flux.
- KCL crée une table DynamoDB avec le nom de l'application et utilise la table pour gérer les informations d'état (telles que les points de contrôle et le mappage `worker-shard`) de l'application. Chaque application a son propre tableau DynamoDB. Pour de plus amples informations, veuillez consulter [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#).

Configurer les informations d'identification

Vous devez mettre vos AWS informations d'identification à la disposition de l'un des fournisseurs d'informations d'identification de la chaîne de fournisseurs d'informations d'identification par défaut. Vous pouvez utiliser la propriété `AWSCredentialsProvider` pour définir un fournisseur d'informations d'identification. Le fichier [sample.properties](#) doit mettre vos informations d'identification à disposition de l'un des fournisseurs d'informations d'identification appartenant à la [chaîne des fournisseurs d'informations d'identification par défaut](#). Si vous exécutez votre application grand public sur une EC2 instance Amazon, nous vous recommandons de configurer l'instance avec un IAM rôle. AWS les informations d'identification qui reflètent les autorisations associées à ce IAM rôle sont mises à la disposition des applications de l'instance via ses métadonnées d'instance. Il s'agit de la méthode la plus sûre pour gérer les informations d'identification d'une application grand public exécutée sur une EC2 instance.

Le fichier de propriétés de l'exemple est configuré KCL pour traiter un flux de données Kinesis appelé « mots » à l'aide du processeur d'enregistrement fourni dans `sample_kc1py_app.py`

Développez un client de bibliothèque cliente Kinesis dans Ruby

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique présente Ruby.

KCLII s'agit d'une bibliothèque Java ; la prise en charge de langages autres que Java est fournie à l'aide d'une interface multilingue appelée. `MultiLangDaemon` Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par conséquent, si vous installez le KCL for Ruby et que vous écrivez votre application grand public entièrement en Ruby, vous devez toujours installer Java sur votre système en raison du `MultiLangDaemon`. En outre, `MultiLangDaemon` il comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations `MultiLangDaemon` sur l'activation GitHub, rendez-vous sur la page [KCL MultiLangDaemon du projet](#).

Pour télécharger le Ruby KCL depuis GitHub, rendez-vous dans la [bibliothèque cliente Kinesis \(Ruby\)](#). Pour télécharger un exemple de code pour une application KCL grand public Ruby, rendez-vous sur la KCL page [d'exemple du projet Ruby](#) sur GitHub.

Pour plus d'informations sur la bibliothèque de support KCL Ruby, consultez la [documentation KCL Ruby Gems](#).

Développez des KCL consommateurs 2.x

Cette rubrique explique comment utiliser la version 2.0 de la bibliothèque cliente Kinesis (KCL).

Pour plus d'informations à ce sujet KCL, consultez la présentation fournie dans [Developing Consumers Using the Kinesis Client Library 1.x](#).

Choisissez l'une des rubriques suivantes en fonction de l'option que vous souhaitez utiliser.

Rubriques

- [Développez un client de bibliothèque cliente Kinesis en Java](#)
- [Développez un client de bibliothèque cliente Kinesis en Python](#)

Développez un client de bibliothèque cliente Kinesis en Java

Le code suivant présente un exemple d'implémentation dans Java de `ProcessorFactory` et `RecordProcessor`. Si vous souhaitez tirer parti de la fonctionnalité de déploiement amélioré, consultez [Utilisation d'applications consommateur avec le déploiement amélioré](#).

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Amazon Software License (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/asl/
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
```

```
*
*   http://www.apache.org/licenses/LICENSE-2.0
*
* or in the "license" file accompanying this file. This file is distributed
* on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
* express or implied. See the License for the specific language governing
* permissions and limitations under the License.
*/

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
```

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

/**
 * This class will run a simple app that uses the KCL to read data and uses the AWS SDK
 * to publish data.
 * Before running this program you must first create a Kinesis stream through the AWS
 * console or AWS SDK.
 */
public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    /**
     * Invoke the main method with 2 args: the stream name and (optionally) the region.
     * Verifies valid inputs and then starts running the app.
     */
    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    /**
     * Constructor sets streamName and region. It also creates a KinesisClient object
     to send data to Kinesis.
     * This KinesisClient is used to send dummy data so that the consumer has something
     to read; it is also used
```

```
    * indirectly by the KCL to handle the consumption of the data.
    */
private SampleSingle(String streamName, String region) {
    this.streamName = streamName;
    this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
    this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
}

private void run() {

    /**
     * Sends dummy data to Kinesis. Not relevant to consuming the data with the KCL
     */
    ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
    ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

    /**
     * Sets up configuration for the KCL, including DynamoDB and CloudWatch
dependencies. The final argument, a
     * ShardRecordProcessorFactory, is where the logic for record processing lives,
and is located in a private
     * class below.
     */
    DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
    CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
    ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

    /**
     * The Scheduler (also called Worker in earlier versions of the KCL) is the
entry point to the KCL. This
     * instance is configured with defaults provided by the ConfigsBuilder.
     */
    Scheduler scheduler = new Scheduler(
        configsBuilder.checkpointConfig(),
        configsBuilder.coordinatorConfig(),
        configsBuilder.leaseManagementConfig(),
        configsBuilder.lifecycleConfig(),
```



```
        configsBuilder.metricsConfig(),
        configsBuilder.processorConfig(),
        configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
    );

    /**
     * Kickoff the Scheduler. Record processing of the stream of dummy data will
continue indefinitely
     * until an exit is triggered.
     */
    Thread schedulerThread = new Thread(scheduler);
    schedulerThread.setDaemon(true);
    schedulerThread.start();

    /**
     * Allows termination of app by pressing Enter.
     */
    System.out.println("Press enter to shutdown");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        reader.readLine();
    } catch (IOException ioex) {
        log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
    }

    /**
     * Stops sending dummy data.
     */
    log.info("Cancelling producer and shutting down executor.");
    producerFuture.cancel(true);
    producerExecutor.shutdownNow();

    /**
     * Stops consuming data. Finishes processing the current batch of data already
received from Kinesis
     * before shutting down.
     */
    Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
    log.info("Waiting up to 20 seconds for shutdown to complete.");
    try {
        gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
```

```
        log.info("Interrupted while waiting for graceful shutdown. Continuing.");
    } catch (ExecutionException e) {
        log.error("Exception while executing graceful shutdown.", e);
    } catch (TimeoutException e) {
        log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
    }
    log.info("Completed, shutting down now.");
}

/**
 * Sends a single record of dummy data to Kinesis.
 */
private void publishRecord() {
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
        .streamName(streamName)
        .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
        .build();

    try {
        kinesisClient.putRecord(request).get();
    } catch (InterruptedException e) {
        log.info("Interrupted, assuming shutdown.");
    } catch (ExecutionException e) {
        log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
    }
}

private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}

/**
 * The implementation of the ShardRecordProcessor interface is where the heart of
the record processing logic lives.
 * In this example all we do to 'process' is log info about the records.
 */
private static class SampleRecordProcessor implements ShardRecordProcessor {

    private static final String SHARD_ID_MDC_KEY = "ShardId";
```

```
private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

private String shardId;

/**
 * Invoked by the KCL before data records are delivered to the
ShardRecordProcessor instance (via
 * processRecords). In this example we do nothing except some logging.
 *
 * @param initializationInput Provides information related to initialization.
 */
public void initialize(InitializationInput initializationInput) {
    shardId = initializationInput.shardId();
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/**
 * Handles record processing logic. The Amazon Kinesis Client Library will
invoke this method to deliver
 * data records to the application. In this example we simply log our records.
 *
 * @param processRecordsInput Provides the records to be processed as well as
information and capabilities
 *                               related to them (e.g. checkpointing).
 */
public void processRecords(ProcessRecordsInput processRecordsInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Processing {} record(s)",
processRecordsInput.records().size());
        processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
    } catch (Throwable t) {
        log.error("Caught throwable while processing records. Aborting.");
        Runtime.getRuntime().halt(1);
    } finally {
```

```
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/** Called when the lease tied to this record processor has been lost. Once the
lease has been lost,
 * the record processor can no longer checkpoint.
 *
 * @param leaseLostInput Provides access to functions and data related to the
loss of the lease.
 */
public void leaseLost(LeaseLostInput leaseLostInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Lost lease, so terminating.");
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/**
 * Called when all data on this shard has been processed. Checkpointing must
occur in the method for record
 * processing to be considered complete; an exception will be thrown otherwise.
 *
 * @param shardEndedInput Provides access to a checkpointer method for
completing processing of the shard.
 */
public void shardEnded(ShardEndedInput shardEndedInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

/**
 * Invoked when Scheduler has been requested to shut down (i.e. we decide to
stop running the app by pressing
 * Enter). Checkpoints and logs the data a final time.
```

```
    *
    * @param shutdownRequestedInput Provides access to a checkpoint, allowing a
record processor to checkpoint
    *
    * before the shutdown is completed.
    */
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Scheduler is shutting down, checkpointing.");
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at requested shutdown. Giving
up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
}
```

Développez un client de bibliothèque cliente Kinesis en Python

Vous pouvez utiliser la bibliothèque cliente Kinesis (KCL) pour créer des applications qui traitent les données issues de vos flux de données Kinesis. La KCL est disponible en plusieurs langues. Cette rubrique présente Python.

KCLII s'agit d'une bibliothèque Java ; la prise en charge de langages autres que Java est fournie à l'aide d'une interface multilingue appelée MultiLangDaemon. Ce démon est basé sur Java et s'exécute en arrière-plan lorsque vous utilisez un KCL langage autre que Java. Par conséquent, si vous installez le KCL pour Python et que vous écrivez votre application grand public entièrement en Python, vous devez toujours installer Java sur votre système en raison du MultiLangDaemon. En outre, MultiLangDaemon il comporte certains paramètres par défaut que vous devrez peut-être personnaliser en fonction de votre cas d'utilisation, par exemple, la AWS région à laquelle il se connecte. Pour plus d'informations MultiLangDaemon sur l'activation GitHub, rendez-vous sur la page [KCL MultiLangDaemon du projet](#).

Pour télécharger le Python KCL depuis GitHub, accédez à la [bibliothèque cliente Kinesis \(Python\)](#). Pour télécharger un exemple de code pour une application KCL grand public Python, rendez-vous sur la page [d'exemple de projet KCL pour Python](#) sur GitHub.

Vous devez effectuer les tâches suivantes lors de l'implémentation d'une application KCL grand public en Python :

Tâches

- [Implémenter les méthodes RecordProcessor de classe](#)
- [Modifier les propriétés de configuration](#)

Implémenter les méthodes RecordProcessor de classe

La classe RecordProcess doit étendre la classe RecordProcessorBase pour implémenter les méthodes ci-après :

```
initialize
process_records
shutdown_requested
```

Cet exemple fournit des implémentations que vous pouvez utiliser comme point de départ.

```
#!/usr/bin/env python

# Copyright 2014-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License.
# A copy of the License is located at
#
# http://aws.amazon.com/asl/
#
# or in the "license" file accompanying this file. This file is distributed
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

from __future__ import print_function

import sys
import time

from amazon_kclpy import kcl
from amazon_kclpy.v3 import processor
```

```

class RecordProcessor(processor.RecordProcessorBase):
    """
    A RecordProcessor processes data from a shard in a stream. Its methods will be
    called with this pattern:

    * initialize will be called once
    * process_records will be called zero or more times
    * shutdown will be called if this MultiLangDaemon instance loses the lease to this
    shard, or the shard ends due
      a scaling change.
    """
    def __init__(self):
        self._SLEEP_SECONDS = 5
        self._CHECKPOINT_RETRIES = 5
        self._CHECKPOINT_FREQ_SECONDS = 60
        self._largest_seq = (None, None)
        self._largest_sub_seq = None
        self._last_checkpoint_time = None

    def log(self, message):
        sys.stderr.write(message)

    def initialize(self, initialize_input):
        """
        Called once by a KCLProcess before any calls to process_records

        :param amazon_kclpy.messages.InitializeInput initialize_input: Information
        about the lease that this record
        processor has been assigned.
        """
        self._largest_seq = (None, None)
        self._last_checkpoint_time = time.time()

    def checkpoint(self, checkpointer, sequence_number=None, sub_sequence_number=None):
        """
        Checkpoints with retries on retryable exceptions.

        :param amazon_kclpy.kcl.Checkpointer checkpointer: the checkpointer provided to
        either process_records
        or shutdown
        :param str or None sequence_number: the sequence number to checkpoint at.
        :param int or None sub_sequence_number: the sub sequence number to checkpoint
        at.

```

```

"""
for n in range(0, self._CHECKPOINT_RETRIES):
    try:
        checkpointer.checkpoint(sequence_number, sub_sequence_number)
        return
    except kcl.CheckpointError as e:
        if 'ShutdownException' == e.value:
            #
            # A ShutdownException indicates that this record processor should
            # be shutdown. This is due to
            # some failover event, e.g. another MultiLangDaemon has taken the
            # lease for this shard.
            #
            print('Encountered shutdown exception, skipping checkpoint')
            return
        elif 'ThrottlingException' == e.value:
            #
            # A ThrottlingException indicates that one of our dependencies is
            # is over burdened, e.g. too many
            # dynamo writes. We will sleep temporarily to let it recover.
            #
            if self._CHECKPOINT_RETRIES - 1 == n:
                sys.stderr.write('Failed to checkpoint after {n} attempts,
giving up.\n'.format(n=n))
                return
            else:
                print('Was throttled while checkpointing, will attempt again in
{s} seconds'
                    .format(s=self._SLEEP_SECONDS))
        elif 'InvalidStateException' == e.value:
            sys.stderr.write('MultiLangDaemon reported an invalid state while
checkpointing.\n')
        else: # Some other error
            sys.stderr.write('Encountered an error while checkpointing, error
was {e}.\n'.format(e=e))
            time.sleep(self._SLEEP_SECONDS)

def process_record(self, data, partition_key, sequence_number,
sub_sequence_number):
    """
    Called for each record that is passed to process_records.

    :param str data: The blob of data that was contained in the record.
    :param str partition_key: The key associated with this record.

```



```

        :param int sequence_number: The sequence number associated with this record.
        :param int sub_sequence_number: the sub sequence number associated with this
record.
        """
        #####
        # Insert your processing logic here
        #####
        self.log("Record (Partition Key: {pk}, Sequence Number: {seq}, Subsequence
Number: {sseq}, Data Size: {ds}"
                .format(pk=partition_key, seq=sequence_number,
sseq=sub_sequence_number, ds=len(data)))

    def should_update_sequence(self, sequence_number, sub_sequence_number):
        """
        Determines whether a new larger sequence number is available

        :param int sequence_number: the sequence number from the current record
        :param int sub_sequence_number: the sub sequence number from the current record
        :return boolean: true if the largest sequence should be updated, false
otherwise
        """
        return self._largest_seq == (None, None) or sequence_number >
self._largest_seq[0] or \
            (sequence_number == self._largest_seq[0] and sub_sequence_number >
self._largest_seq[1])

    def process_records(self, process_records_input):
        """
        Called by a KCLProcess with a list of records to be processed and a
checkpointer which accepts sequence numbers
        from the records to indicate where in the stream to checkpoint.

        :param amazon_kclpy.messages.ProcessRecordsInput process_records_input: the
records, and metadata about the
            records.
        """
        try:
            for record in process_records_input.records:
                data = record.binary_data
                seq = int(record.sequence_number)
                sub_seq = record.sub_sequence_number
                key = record.partition_key
                self.process_record(data, key, seq, sub_seq)
                if self.should_update_sequence(seq, sub_seq):

```

```

        self._largest_seq = (seq, sub_seq)

        #
        # Checkpoints every self._CHECKPOINT_FREQ_SECONDS seconds
        #
        if time.time() - self._last_checkpoint_time >
self._CHECKPOINT_FREQ_SECONDS:
            self.checkpoint(process_records_input.checkpointer,
str(self._largest_seq[0]), self._largest_seq[1])
            self._last_checkpoint_time = time.time()

    except Exception as e:
        self.log("Encountered an exception while processing records. Exception was
{e}\n".format(e=e))

    def lease_lost(self, lease_lost_input):
        self.log("Lease has been lost")

    def shard_ended(self, shard_ended_input):
        self.log("Shard has ended checkpointing")
        shard_ended_input.checkpointer.checkpoint()

    def shutdown_requested(self, shutdown_requested_input):
        self.log("Shutdown has been requested, checkpointing.")
        shutdown_requested_input.checkpointer.checkpoint()

if __name__ == "__main__":
    kcl_process = kcl.KCLProcess(RecordProcessor())
    kcl_process.run()

```

Modifier les propriétés de configuration

L'exemple fournit les valeurs par défaut des propriétés de configuration, comme illustré dans le script suivant. Vous pouvez remplacer ces propriétés par vos propres valeurs.

```

# The script that abides by the multi-language protocol. This script will
# be executed by the MultiLangDaemon, which will communicate with this script
# over STDIN and STDOUT according to the multi-language protocol.
executableName = sample_kclpy_app.py

# The name of an Amazon Kinesis stream to process.
streamName = words

```

```
# Used by the KCL as the name of this application. Will be used as the name
# of an Amazon DynamoDB table which will store the lease and checkpoint
# information for workers with this application name
applicationName = PythonKCLSample

# Users can change the credentials provider the KCL will use to retrieve credentials.
# The DefaultAWSCredentialsProviderChain checks several other providers, which is
# described here:
# http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/
DefaultAWSCredentialsProviderChain.html
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain

# Appended to the user agent of the KCL. Does not impact the functionality of the
# KCL in any other way.
processingLanguage = python/2.7

# Valid options at TRIM_HORIZON or LATEST.
# See http://docs.aws.amazon.com/kinesis/latest/APIReference/
API_GetShardIterator.html#API_GetShardIterator_RequestSyntax
initialPositionInStream = TRIM_HORIZON

# The following properties are also available for configuring the KCL Worker that is
# created
# by the MultiLangDaemon.

# The KCL defaults to us-east-1
#regionName = us-east-1

# Fail over time in milliseconds. A worker which does not renew it's lease within this
# time interval
# will be regarded as having problems and it's shards will be assigned to other
# workers.
# For applications that have a large number of shards, this msy be set to a higher
# number to reduce
# the number of DynamoDB IOPS required for tracking leases
#failoverTimeMillis = 10000

# A worker id that uniquely identifies this worker among all workers using the same
# applicationName
# If this isn't provided a MultiLangDaemon instance will assign a unique workerId to
# itself.
#workerId =
```

```
# Shard sync interval in milliseconds - e.g. wait for this long between shard sync
tasks.
#shardSyncIntervalMillis = 60000

# Max records to fetch from Kinesis in a single GetRecords call.
#maxRecords = 10000

# Idle time between record reads in milliseconds.
#idleTimeBetweenReadsInMillis = 1000

# Enables applications flush/checkpoint (if they have some data "in progress", but
don't get new data for while)
#callProcessRecordsEvenForEmptyRecordList = false

# Interval in milliseconds between polling to check for parent shard completion.
# Polling frequently will take up more DynamoDB IOPS (when there are leases for shards
waiting on
# completion of parent shards).
#parentShardPollIntervalMillis = 10000

# Cleanup leases upon shards completion (don't wait until they expire in Kinesis).
# Keeping leases takes some tracking/resources (e.g. they need to be renewed,
assigned), so by default we try
# to delete the ones we don't need any longer.
#cleanupLeasesUponShardCompletion = true

# Backoff time in milliseconds for Amazon Kinesis Client Library tasks (in the event of
failures).
#taskBackoffTimeMillis = 500

# Buffer metrics for at most this long before publishing to CloudWatch.
#metricsBufferTimeMillis = 10000

# Buffer at most this many metrics before publishing to CloudWatch.
#metricsMaxQueueSize = 10000

# KCL will validate client provided sequence numbers with a call to Amazon Kinesis
before checkpointing for calls
# to RecordProcessorCheckpoint#checkpoint(String) by default.
#validateSequenceNumberBeforeCheckpointing = true

# The maximum number of active threads for the MultiLangDaemon to permit.
# If a value is provided then a FixedThreadPool is used with the maximum
# active threads set to the provided value. If a non-positive integer or no
```

```
# value is provided a CachedThreadPool is used.  
#maxActiveThreads = 0
```

Nom de l'application

Le KCL nécessite un nom d'application unique parmi vos applications et parmi les tables Amazon DynamoDB d'une même région. Elle utilise la valeur de configuration du nom d'application des manières suivantes :

- Tous les programmes d'exécution qui sont associés à ce nom d'application sont considérés comme rattachés au même flux. Ces programmes d'exécution peuvent être répartis sur plusieurs instances. Si vous exécutez une instance supplémentaire du même code d'application, mais avec un nom d'application différent, KCL la deuxième instance est traitée comme une application entièrement distincte qui fonctionne également sur le même flux.
- KCL crée une table DynamoDB avec le nom de l'application et utilise la table pour gérer les informations d'état (telles que les points de contrôle et le mappage worker-shard) de l'application. Chaque application a son propre tableau DynamoDB. Pour de plus amples informations, veuillez consulter [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client.](#)

Informations d'identification

Vous devez mettre vos AWS informations d'identification à la disposition de l'un des fournisseurs d'informations d'identification de la chaîne de fournisseurs [d'informations d'identification par défaut](#). Vous pouvez utiliser la propriété `AWSCredentialsProvider` pour définir un fournisseur d'informations d'identification. Si vous exécutez votre application client sur une EC2 instance Amazon, nous vous recommandons de configurer l'instance avec un IAM rôle. AWS les informations d'identification qui reflètent les autorisations associées à ce IAM rôle sont mises à la disposition des applications de l'instance via ses métadonnées d'instance. Il s'agit de la méthode la plus sûre pour gérer les informations d'identification d'une application grand public exécutée sur une EC2 instance.

Développez des consommateurs personnalisés avec un débit partagé à l'aide du AWS SDK for Java

L'une des méthodes pour développer des clients Kinesis Data Streams personnalisés avec un accès partagé consiste à utiliser Amazon Kinesis Data Streams. APIs Cette section décrit l'utilisation des Kinesis Data APIs Streams avec AWS SDK le pour Java. L'exemple de code Java présenté dans cette section montre comment effectuer des KDS API opérations de base. Il est divisé logiquement par type d'opération.

Ces exemples ne représentent pas du code destiné à la production. Ils ne recherchent pas toutes les exceptions possibles et ils ne tiennent pas compte de toutes les considérations possibles en matière de sécurité ou de performances.

Vous pouvez appeler les Kinesis Data APIs Streams à l'aide d'autres langages de programmation. Pour plus d'informations sur toutes les options disponibles AWS SDKs, consultez [Commencer à développer avec Amazon Web Services](#).

Important

La méthode recommandée pour développer des clients Kinesis Data Streams personnalisés avec Shared Through consiste à utiliser la Kinesis Client Library (). KCL KCLvous aide à consommer et à traiter les données issues d'un flux de données Kinesis en prenant en charge de nombreuses tâches complexes associées à l'informatique distribuée. Pour plus d'informations, consultez la section [Développement de consommateurs personnalisés à l'aide KCL d'un débit partagé](#).

Rubriques

- [Obtenir des données à partir d'un flux](#)
- [Utiliser des itérateurs de partitions](#)
- [Utiliser GetRecords](#)
- [S'adapter à une refonte](#)
- [Interagissez avec les données à l'aide du registre des AWS Glue schémas](#)

Obtenir des données à partir d'un flux

Les Kinesis Data APIs Streams incluent `getShardIterator` les méthodes `getRecords` et que vous pouvez invoquer pour récupérer des enregistrements d'un flux de données. Il s'agit du modèle d'extraction où votre code extrait les enregistrements de données directement depuis les partitions du flux de données.

Important

Nous vous recommandons d'utiliser le support du processeur d'enregistrements fourni par KCL pour récupérer les enregistrements de vos flux de données. Il s'agit du modèle push,

où vous implémentez le code qui traite les données. Il KCL récupère les enregistrements de données du flux de données et les transmet au code de votre application. En outre, il KCL fournit des fonctionnalités de basculement, de restauration et d'équilibrage de charge. Pour plus d'informations, consultez la section [Développement de consommateurs personnalisés à l'aide KCL d'un débit partagé](#).

Toutefois, dans certains cas, vous préférerez peut-être utiliser les Kinesis Data APIs Streams. Par exemple, pour implémenter les outils personnalisés pour surveiller ou déboguer vos flux de données.

Important

Kinesis Data Streams prend en charge les modifications de la période de conservation des enregistrements de données de votre flux de données. Pour de plus amples informations, veuillez consulter [Modifier la période de conservation des données](#).

Utiliser des itérateurs de partitions

Vous extrayez des enregistrements du flux à partir du flux pour chaque partition. Pour chaque partition et chaque lot d'enregistrements que vous extrayez de cette partition, vous devez obtenir un itérateur de partition. L'itérateur de partition est utilisé dans l'objet `getRecordsRequest` pour spécifier la partition à partir de laquelle les enregistrements sont extraits. Le type associé à l'itérateur de partition détermine le point de la partition à partir duquel les enregistrements doivent être extraits (voir plus loin dans cette section pour plus de détails). Avant de pouvoir utiliser l'itérateur de partition, vous devez récupérer la partition. Pour de plus amples informations, veuillez consulter [Répertoire les fragments](#).

Obtenez l'itérateur de partition initial à l'aide de la méthode `getShardIterator`. Obtenez des itérateurs de partition pour les lots supplémentaires d'enregistrements à l'aide de la méthode `getNextShardIterator` de l'objet `getRecordsResult` renvoyé par la méthode `getRecords`. Un itérateur de partition est valide pendant 5 minutes. Si vous utilisez un itérateur de partition pendant qu'il est valide, vous en obtenez un nouveau. Chaque itérateur de partition reste valide 5 minutes, même après avoir été utilisé.

Pour obtenir l'itérateur de partition initial, instanciez `GetShardIteratorRequest` et passez-le à la méthode `getShardIterator`. Pour configurer la demande, spécifiez le flux et l'ID de partition. Pour plus d'informations sur la façon d'obtenir les streams de votre AWS compte, consultez [Afficher la liste](#)

[des flux](#). Pour plus d'informations sur la façon d'obtenir les partitions d'un flux, consultez [Répertoire les fragments](#).

```
String shardIterator;
GetShardIteratorRequest getShardIteratorRequest = new GetShardIteratorRequest();
getShardIteratorRequest.setStreamName(myStreamName);
getShardIteratorRequest.setShardId(shard.getShardId());
getShardIteratorRequest.setShardIteratorType("TRIM_HORIZON");

GetShardIteratorResult getShardIteratorResult =
    client.getShardIterator(getShardIteratorRequest);
shardIterator = getShardIteratorResult.getShardIterator();
```

Cet exemple de code spécifie TRIM_HORIZON comme type d'itérateur lors de l'obtention de l'itérateur de partition initial. Ce type d'itérateur signifie que les enregistrements doivent être renvoyés en commençant par le premier enregistrement ajouté à la partition, et pas en commençant par l'enregistrement ajouté le plus récemment, également appelé l'extrémité. Les types d'itérateur possibles sont les suivants :

- AT_SEQUENCE_NUMBER
- AFTER_SEQUENCE_NUMBER
- AT_TIMESTAMP
- TRIM_HORIZON
- LATEST

Pour plus d'informations, consultez [ShardIteratorType](#).

Certains types d'itérateur nécessitent que vous spécifiez un numéro de séquence en plus du type ; par exemple :

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Après avoir obtenu un enregistrement à l'aide de `getRecords`, vous pouvez obtenir le numéro de séquence de l'enregistrement en appelant la méthode `getSequenceNumber` de l'enregistrement.

```
record.getSequenceNumber()
```


En outre, le code qui ajoute les enregistrements au flux de données peut obtenir le numéro de séquence d'un enregistrement ajouté en appelant `getSequenceNumber` sur le résultat de `putRecord`.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Vous pouvez utiliser des numéros de séquence pour garantir un ordre croissant strict des enregistrements. Pour plus d'informations, consultez l'exemple de code dans [PutRecordexemple](#).

Utiliser GetRecords

Après avoir obtenu l'itérateur de partition, instanciez un objet `GetRecordsRequest`. Spécifiez l'itérateur de la demande à l'aide de la méthode `setShardIterator`.

Le cas échéant, vous pouvez également définir le nombre d'enregistrements à extraire à l'aide de la méthode `setLimit`. Le nombre d'enregistrements renvoyés par `getRecords` est toujours égal ou inférieur à cette limite. Si vous ne spécifiez pas cette limite, `getRecords` renvoie 10 Mo d'enregistrements extraits. L'exemple de code ci-dessous définit cette limite à 25 enregistrements.

Si aucun enregistrement n'est renvoyé, cela signifie qu'aucun enregistrement de données n'est actuellement disponible à partir de cette partition au numéro de séquence référencé par l'itérateur de partition. Dans cette situation, votre application doit attendre pendant le laps de temps approprié pour les sources de données du flux. Essayez ensuite d'extraire les données de la partition à l'aide de l'itérateur de partition renvoyé par l'appel précédent de `getRecords`.

Passez l'objet `getRecordsRequest` à la méthode `getRecords` et capturez la valeur renvoyée en tant qu'objet `getRecordsResult`. Pour extraire les enregistrements de données, appelez la méthode `getRecords` sur l'objet `getRecordsResult`.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Pour vous préparer à un autre appel de `getRecords`, obtenez l'itérateur de partition suivant de `getRecordsResult`.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Pour obtenir les meilleurs résultats, attendez pendant au moins 1 seconde (1 000 millisecondes) entre les appels de `getRecords`, afin d'éviter de dépasser la limite de la fréquence `getRecords`.

```
try {
    Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

En général, vous devez appeler `getRecords` en boucle, même si vous êtes en train d'extraire un seul enregistrement dans un scénario de test. Un seul appel de `getRecords` peut renvoyer une liste d'enregistrements vide, même si la partition contient plusieurs enregistrements à des numéros de séquence ultérieurs. Dans ce cas, l'objet `NextShardIterator` renvoyé avec la liste d'enregistrements vide fait référence à un numéro de séquence ultérieur de la partition, et les appels successifs de `getRecords` finissent par renvoyer les enregistrements. L'exemple suivant illustre l'utilisation d'une boucle.

Exemple : `getRecords`

L'exemple de code suivant reflète les conseils pour `getRecords` de cette section, notamment la réalisation d'appels en boucle.

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

    // Create a new getRecordsRequest with an existing shardIterator
    // Set the maximum records to return to 25

    GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
    getRecordsRequest.setShardIterator(shardIterator);
    getRecordsRequest.setLimit(25);
```

```
GetRecordsResult result = client.getRecords(getRecordsRequest);

// Put the result into record list. The result can be empty.
records = result.getRecords();

try {
    Thread.sleep(1000);
}
catch (InterruptedException exception) {
    throw new RuntimeException(exception);
}

shardIterator = result.getNextShardIterator();
}
```

Si vous utilisez la bibliothèque client Kinesis (KCL), elle peut effectuer plusieurs appels avant de renvoyer des données. Ce comportement est intentionnel et n'indique aucun problème lié aux données KCL ou à vos données.

S'adapter à une refonte

Si `getRecordsResult.getNextShardIterator` renvoie `null`, cela indique qu'une division ou une fusion de partition a eu lieu impliquant cette partition. Cette partition est désormais dans un état `CLOSED`, et vous avez lu tous les enregistrements de données disponibles à partir de cette partition.

Dans ce scénario, vous pouvez utiliser `getRecordsResult.childShards` pour en savoir plus sur les nouvelles partitions secondaires de la partition en cours de traitement qui ont été créées par le fractionnement ou la fusion. Pour plus d'informations, consultez [ChildShard](#).

Dans le cas d'un fractionnement, les deux nouvelles partitions ont `parentShardId` égal à l'ID de la partition que vous avez traitée précédemment. La valeur de `adjacentParentShardId` pour ces deux partitions est `null`.

Dans le cas d'une fusion, la nouvelle partition créée par la fusion a `parentShardId` égal à l'ID de partition de l'une des partitions parent et `adjacentParentShardId` égal à l'ID de partition de l'autre partition parent. Votre application a déjà lu toutes les données à partir de l'une de ces partitions. Il s'agit de la partition pour laquelle `getRecordsResult.getNextShardIterator` a renvoyé `null`. Si l'ordre des données est important pour votre application, assurez-vous qu'elle lit aussi toutes les données de l'autre partition parente avant de lire les nouvelles données de la partition enfant créée par la fusion.

Si vous utilisez plusieurs processeurs pour extraire des données du flux (disons un processeur par partition) et qu'un fractionnement ou une fusion a lieu, augmentez ou réduisez le nombre de processeurs pour prendre en compte le nouveau nombre de partitions.

Pour plus d'informations sur le repartitionnement, plus une présentation des états des partitions, par exemple CLOSED, consultez [Revisiter un stream](#).

Interagissez avec les données à l'aide du registre des AWS Glue schémas

Vous pouvez intégrer vos flux de données Kinesis au registre des AWS Glue schémas. Le registre des AWS Glue schémas vous permet de découvrir, de contrôler et de faire évoluer les schémas de manière centralisée, tout en garantissant que les données produites sont validées en permanence par un schéma enregistré. Un schéma définit la structure et le format d'un enregistrement de données. Un schéma est une spécification versionnée pour la publication, la consommation ou le stockage des données fiables. Le registre des AWS Glue schémas vous permet d'améliorer la qualité end-to-end des données et la gouvernance des données au sein de vos applications de streaming. Pour plus d'informations, consultez le registre [AWS Glue Schema](#) (français non garanti). L'un des moyens de configurer cette intégration consiste à utiliser les `GetRecords` Kinesis Data API Streams disponibles dans Java AWS . SDK

Pour obtenir des instructions détaillées sur la façon de configurer l'intégration de Kinesis Data Streams au registre des schémas à l'aide de Kinesis Data APIs Streams, consultez `GetRecords` la [section « Interaction avec les données à l'aide des Kinesis Data Streams » dans Use Case : Integrating Amazon Kinesis APIs Data Streams](#) with the Glue Schema Registry. AWS

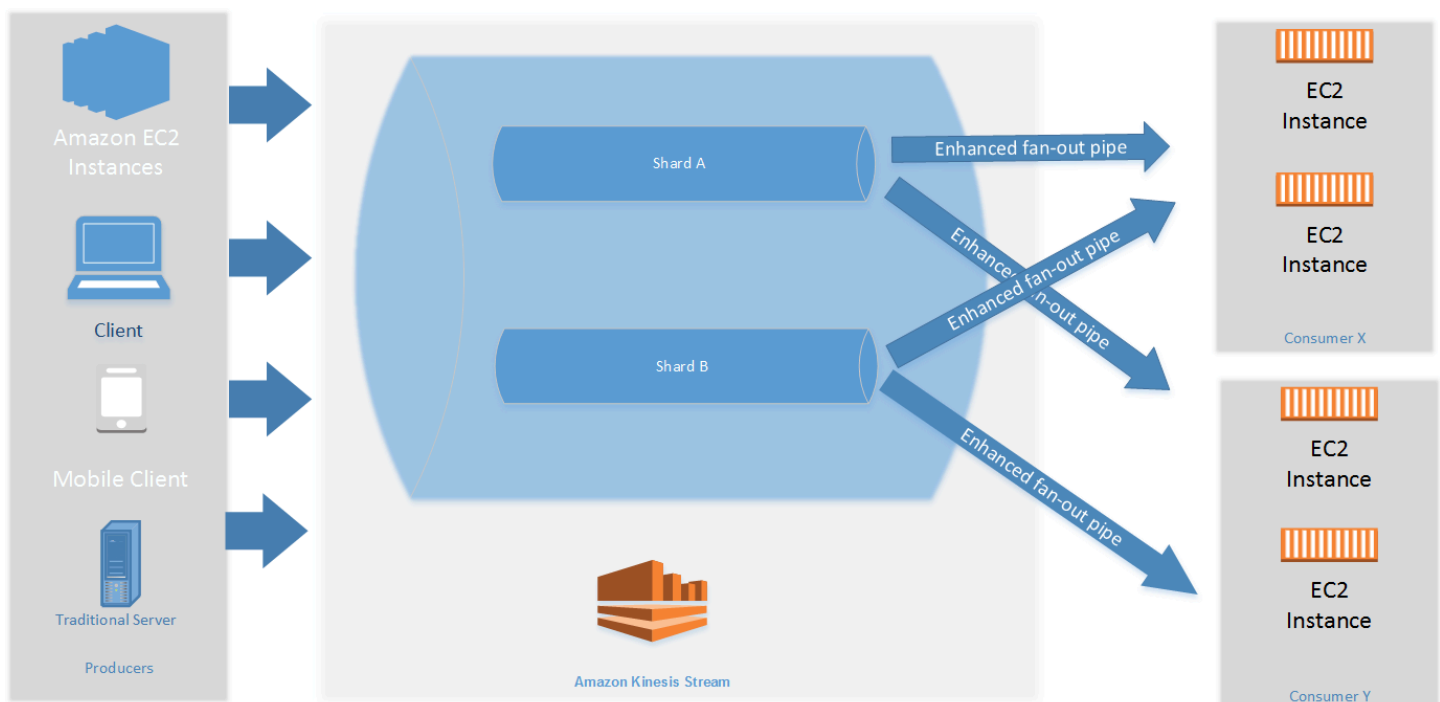
Développez des consommateurs personnalisés avec un débit dédié (fan-out amélioré)

Dans Amazon Kinesis Data Streams, vous pouvez créer des applications consommateur qui utilisent une fonctionnalité appelée diffusion améliorée. Cette fonction permet aux applications consommateur de recevoir des enregistrements provenant d'un flux avec un débit de jusqu'à 2 Mo de données par seconde par partition. Ce débit est dédié. En d'autres termes, les consommateurs qui utilisent la diffusion améliorée n'ont pas besoin de se heurter à d'autres consommateurs qui reçoivent des données à partir du flux. Kinesis Data Streams envoie les enregistrements de données depuis le flux vers les applications consommateur qui utilisent la diffusion améliorée. Par conséquent, ces applications consommateur n'ont pas besoin de sonder des données.

⚠ Important

Vous pouvez enregistrer jusqu'à vingt applications consommateur par flux pour utiliser la diffusion améliorée.

Le schéma suivant illustre l'architecture de diffusion améliorée proposée. Si vous utilisez la version 2.0 ou ultérieure de la bibliothèque client Amazon Kinesis (KCL) pour créer un consommateur, le consommateur est KCL configuré pour qu'il utilise un ventilateur amélioré pour recevoir les données de toutes les partitions du flux. Si vous utilisez le API pour créer un consommateur utilisant un ventilateur amélioré, vous pouvez vous abonner à des partitions individuelles.



Le diagramme décrit les éléments suivants :

- Un flux comportant deux partitions.
- Deux applications consommateur qui utilisent la diffusion améliorée pour recevoir des données à partir du flux : Consommateur X et Consommateur Y. Chacune des deux applications consommateur est abonnée à toutes les partitions et tous les enregistrements du flux. Si vous utilisez la version 2.0 ou ultérieure de KCL pour créer un consommateur, celui-ci s'abonne automatiquement à toutes les partitions du stream. D'autre part, si vous utilisez le API pour créer un consommateur, vous pouvez vous abonner à des partitions individuelles.

- Les flèches représentant les canaux de diffusion améliorée des tuyaux que les applications consommateur utilisent pour recevoir des données à partir du flux. Un canal de diffusion améliorée fournit jusqu'à 2 Mo/sec de données par partition, indépendamment des autres canaux ou du nombre total d'applications consommateur.

Rubriques

- [Développez de meilleurs fans grâce à la version 2.x KCL](#)
- [Développez un plus grand nombre de fans grâce aux Kinesis Data Streams API](#)
- [Gérez davantage de clients fans grâce au AWS Management Console](#)

Développez de meilleurs fans grâce à la version 2.x KCL

Les applications consommateur utilisant la diffusion améliorée dans Amazon Kinesis Data Streams peuvent recevoir des enregistrements provenant d'un flux de données avec un débit dédié de jusqu'à 2 Mo de données par seconde par partition. Ce type d'application consommateur n'a pas besoin de se heurter à d'autres applications consommateur qui reçoivent des données à partir du flux. Pour plus d'informations, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Vous pouvez utiliser la version 2.0 ou ultérieure de la bibliothèque cliente Kinesis (KCL) pour développer des applications qui utilisent un ventilateur amélioré pour recevoir des données provenant de flux. abonne KCL automatiquement votre application à toutes les partitions d'un flux et garantit que votre application client peut lire avec une valeur de débit de 2 Mo/sec par partition. Si vous souhaitez utiliser le KCL sans activer le ventilateur amélioré, consultez [Developing Consumers Using the Kinesis Client Library 2.0](#).

Rubriques

- [Développez des consommateurs de fans améliorés à l'aide de la version KCL 2.x en Java](#)

Développez des consommateurs de fans améliorés à l'aide de la version KCL 2.x en Java

Vous pouvez utiliser la version 2.0 ou ultérieure de la bibliothèque client Kinesis (KCL) pour développer des applications dans Amazon Kinesis Data Streams afin de recevoir des données provenant de flux à l'aide d'un ventilateur amélioré. Le code suivant présente un exemple d'implémentation dans Java de `ProcessorFactory` et `RecordProcessor`.

Il est recommandé d'utiliser `KinesisClientUtil` pour créer `KinesisAsyncClient` et configurer `maxConcurrency` dans `KinesisAsyncClient`.

Important

Le client Amazon Kinesis peut voir une augmentation significative de la latence, sauf si vous configurez `KinesisAsyncClient` pour avoir un `maxConcurrency` suffisamment élevée pour autoriser tous les baux et les utilisations supplémentaires de `KinesisAsyncClient`.

```
/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Amazon Software License (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://aws.amazon.com/asl/
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
import java.io.IOException;
```

```
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
The Region may be specified as the second argument.");
        }
    }
}
```



```
        System.exit(1);
    }

    String streamName = args[0];
    String region = null;
    if (args.length > 1) {
        region = args[1];
    }

    new SampleSingle(streamName, region).run();
}

private final String streamName;
private final Region region;
private final KinesisAsyncClient kinesisClient;

private SampleSingle(String streamName, String region) {
    this.streamName = streamName;
    this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
    this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
}

private void run() {
    ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
    ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

    DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
    CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
    ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

    Scheduler scheduler = new Scheduler(
        configsBuilder.checkpointConfig(),
        configsBuilder.coordinatorConfig(),
        configsBuilder.leaseManagementConfig(),
        configsBuilder.lifecycleConfig(),
        configsBuilder.metricsConfig(),
        configsBuilder.processorConfig(),
```

```
        configsBuilder.retrievalConfig()
    );

    Thread schedulerThread = new Thread(scheduler);
    schedulerThread.setDaemon(true);
    schedulerThread.start();

    System.out.println("Press enter to shutdown");
    BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
    try {
        reader.readLine();
    } catch (IOException ioex) {
        log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
    }

    log.info("Cancelling producer, and shutting down executor.");
    producerFuture.cancel(true);
    producerExecutor.shutdownNow();

    Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
    log.info("Waiting up to 20 seconds for shutdown to complete.");
    try {
        gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
    } catch (InterruptedException e) {
        log.info("Interrupted while waiting for graceful shutdown. Continuing.");
    } catch (ExecutionException e) {
        log.error("Exception while executing graceful shutdown.", e);
    } catch (TimeoutException e) {
        log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
    }
    log.info("Completed, shutting down now.");
}

private void publishRecord() {
    PutRecordRequest request = PutRecordRequest.builder()
        .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
        .streamName(streamName)
        .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
        .build();
    try {
        kinesisClient.putRecord(request).get();
    } catch (InterruptedException e) {
```

```
        log.info("Interrupted, assuming shutdown.");
    } catch (ExecutionException e) {
        log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
    }
}

private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
}

private static class SampleRecordProcessor implements ShardRecordProcessor {

    private static final String SHARD_ID_MDC_KEY = "ShardId";

    private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

    private String shardId;

    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)",
processRecordsInput.records().size());
            processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.");
        }
    }
}
```

```
        Runtime.getRuntime().halt(1);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

public void leaseLost(LeaseLostInput leaseLostInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Lost lease, so terminating.");
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

public void shardEnded(ShardEndedInput shardEndedInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}

public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    MDC.put(SHARD_ID_MDC_KEY, shardId);
    try {
        log.info("Scheduler is shutting down, checkpointing.");
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at requested shutdown. Giving
up.", e);
    } finally {
        MDC.remove(SHARD_ID_MDC_KEY);
    }
}
}
```

Développez un plus grand nombre de fans grâce aux Kinesis Data Streams API

La diffusion améliorée est une fonction Amazon Kinesis Data Streams permettant aux applications consommateur de recevoir des enregistrements provenant d'un flux de données avec un débit dédié de jusqu'à 2 Mo de données par seconde par partition. Une application consommateur utilisant la diffusion améliorée n'a pas besoin de se heurter à d'autres applications consommateur qui reçoivent des données à partir du flux. Pour plus d'informations, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Vous pouvez utiliser API les opérations pour créer un consommateur qui utilise le ventilateur amélioré dans Kinesis Data Streams.

Pour inscrire un consommateur à un système de ventilation amélioré à l'aide des Kinesis Data Streams API

1. Appelez [RegisterStreamConsumer](#) pour enregistrer votre application en tant que client utilisant un ventilateur amélioré. Kinesis Data Streams génère un Amazon Resource Name ARN () pour le consommateur et le renvoie dans la réponse.
2. Pour commencer à écouter un fragment spécifique, transmettez le consommateur ARN dans un appel à [SubscribeToShard](#). Kinesis Data Streams commence alors à vous transmettre les enregistrements de cette partition, sous la forme d'événements de [SubscribeToShardEvent](#) type 2 via HTTP une connexion /2. La connexion demeure ouvert pour une durée maximum de 5 minutes. Appelez [SubscribeToShard](#) à nouveau si vous souhaitez continuer à recevoir des enregistrements de la partition une fois future que l'appel les a renvoyés [SubscribeToShard](#) normalement ou exceptionnellement.

Note

[SubscribeToShardAPI](#) renvoie également la liste des fragments enfants de la partition actuelle lorsque la fin de la partition actuelle est atteinte.

3. Pour annuler l'enregistrement d'un consommateur qui utilise le ventilateur amélioré, appelez [DeregisterStreamConsumer](#)

Le code suivant est un exemple de la façon dont vous pouvez abonner votre application consommateur à une partition, renouveler l'abonnement périodiquement et gérer les événements.

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
import
software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;

import java.util.concurrent.CompletableFuture;

/**
 * See https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/example\_code/kinesis/src/main/java/com/example/kinesis/KinesisStreamEx.java
 * for complete code and more examples.
 */
public class SubscribeToShardSimpleImpl {

    private static final String CONSUMER_ARN = "arn:aws:kinesis:us-
east-1:123456789123:stream/foobar/consumer/test-consumer:1525898737";
    private static final String SHARD_ID = "shardId-000000000000";

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.create();

        SubscribeToShardRequest request = SubscribeToShardRequest.builder()
            .consumerARN(CONSUMER_ARN)
            .shardId(SHARD_ID)
            .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();

        // Call SubscribeToShard iteratively to renew the subscription
        periodically.
        while(true) {
            // Wait for the CompletableFuture to complete normally or
            exceptionally.
            callSubscribeToShardWithVisitor(client, request).join();
        }

        // Close the connection before exiting.
        // client.close();
    }

    /**
```

```
    * Subscribes to the stream of events by implementing the
    SubscribeToShardResponseHandler.Visitor interface.
    */
    private static CompletableFuture<Void>
    callSubscribeToShardWithVisitor(KinesisAsyncClient client, SubscribeToShardRequest
    request) {
        SubscribeToShardResponseHandler.Visitor visitor = new
    SubscribeToShardResponseHandler.Visitor() {
            @Override
            public void visit(SubscribeToShardEvent event) {
                System.out.println("Received subscribe to shard event " + event);
            }
        };
        SubscribeToShardResponseHandler responseHandler =
    SubscribeToShardResponseHandler
            .builder()
            .onError(t -> System.err.println("Error during stream - " +
    t.getMessage()))
            .subscriber(visitor)
            .build();
        return client.subscribeToShard(request, responseHandler);
    }
}
```

Si `event.ContinuationSequenceNumber` renvoie `null`, cela indique qu'une division ou une fusion de partition a eu lieu impliquant cette partition. Cette partition est maintenant dans un état `CLOSED`, et vous avez lu tous les enregistrements de données disponibles à partir de cette partition. Dans ce scénario, comme indiqué ci-dessus, vous pouvez utiliser `event.childShards` pour en savoir plus sur les nouvelles partitions secondaires de la partition en cours de traitement qui ont été créées par la scission ou la fusion. Pour plus d'informations, consultez [ChildShard](#).

Interagissez avec les données à l'aide du registre des AWS Glue schémas

Vous pouvez intégrer vos flux de données Kinesis au registre des AWS Glue schémas. Le registre des AWS Glue schémas vous permet de découvrir, de contrôler et de faire évoluer les schémas de manière centralisée, tout en garantissant que les données produites sont validées en permanence par un schéma enregistré. Un schéma définit la structure et le format d'un enregistrement de données. Un schéma est une spécification versionnée pour la publication, la consommation ou le stockage des données fiables. Le registre des AWS Glue schémas vous permet d'améliorer la qualité end-to-end des données et la gouvernance des données au sein de vos applications de streaming. Pour plus d'informations, consultez le registre [AWS Glue Schema](#) (français non garanti). L'un des

moyens de configurer cette intégration consiste à utiliser les GetRecords Kinesis Data API Streams disponibles dans Java AWS . SDK

Pour obtenir des instructions détaillées sur la façon de configurer l'intégration de Kinesis Data Streams au registre des schémas à l'aide de Kinesis Data APIs Streams, consultez GetRecords la [section « Interaction avec les données à l'aide des Kinesis Data Streams » dans Use Case : Integrating Amazon Kinesis APIs Data Streams with the Glue Schema Registry. AWS](#)

Gérez davantage de clients fans grâce au AWS Management Console

Les applications consommateur utilisant la diffusion améliorée dans Amazon Kinesis Data Streams peuvent recevoir des enregistrements provenant d'un flux de données avec un débit dédié de jusqu'à 2 Mo de données par seconde par partition. Pour plus d'informations, consultez [Développez des consommateurs personnalisés avec un débit dédié \(fan-out amélioré\)](#).

Vous pouvez utiliser le AWS Management Console pour voir la liste de tous les consommateurs enregistrés pour utiliser le fan-out amélioré avec un stream spécifique. Pour chacun de ces consommateurs, vous pouvez voir des détails tels que le statutARN, la date de création et les mesures de surveillance.

Pour afficher les applications consommateur qui sont enregistrés pour utiliser la diffusion améliorée, leur statut, leur date de création et les métriques sur la console

1. [Connectez-vous à la console Kinesis AWS Management Console et ouvrez-la à l'adresse https://console.aws.amazon.com/kinesis.](https://console.aws.amazon.com/kinesis)
2. Choisissez Data Streams (Flux de données) dans le volet de navigation.
3. Choisissez un flux de données Kinesis pour en afficher les détails.
4. Sur la page des détails du flux, choisissez l'onglet Diffusion améliorée.
5. Choisissez une application consommateur pour afficher son nom, son 'état et la date de son enregistrement.

Pour annuler l'enregistrement d'une application consommateur

1. Ouvrez la console Kinesis à l'adresse <https://console.aws.amazon.com/kinesis>.
2. Choisissez Data Streams (Flux de données) dans le volet de navigation.
3. Choisissez un flux de données Kinesis pour en afficher les détails.
4. Sur la page des détails du flux, choisissez l'onglet Diffusion améliorée.

5. Cochez située à gauche du nom de chaque application consommateur que vous souhaitez désinscrire.
6. Choisissez Annuler l'enregistrement d'un consommateur.

Migrer les consommateurs de la version KCL 1.x vers la version KCL 2.x

Cette rubrique explique les différences entre les versions 1.x et 2.x de la bibliothèque cliente Kinesis (). KCL II vous montre également comment faire migrer votre client de la version 1.x vers la version 2.x du. KCL Après avoir migré votre client, il commencera à traiter les enregistrements à partir du dernier emplacement contrôlé.

La version 2.0 KCL introduit les modifications d'interface suivantes :

KCL Changements d'interface

KCLInterface 1.x	KCLInterface 2.0
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor</code>	<code>software.amazon.kinesis.processor.ShardRecordProcessor</code>
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory</code>	<code>software.amazon.kinesis.processor.ShardRecordProcessorFactory</code>
<code>com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware</code>	Intégré à <code>software.amazon.kinesis.processor.ShardRecordProcessor</code>

Rubriques

- [Migrer le processeur d'enregistrements](#)
- [Migrer l'usine de traitement des enregistrements](#)
- [Migrer le travailleur](#)
- [Configuration du client Amazon Kinesis](#)

- [Suppression des temps d'inactivité](#)
- [Suppressions de configurations clientes](#)

Migrer le processeur d'enregistrements

L'exemple suivant montre un processeur d'enregistrement implémenté pour la version KCL 1.x :

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpoint;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class TestRecordProcessor implements IRecordProcessor,
    IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        //
        // Process records, and possibly checkpoint
        //
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpoint().checkpoint();
            }
        }
    }
}
```

```

        } catch (ShutdownException | InvalidStateException e) {
            throw new RuntimeException(e);
        }
    }
}

@Override
public void shutdownRequested(IRecordProcessorCheckpointter checkpointer) {
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow exception
        //
        e.printStackTrace();
    }
}
}
}

```

Pour migrer la classe de processeur d'enregistrements

1. Modifiez les interfaces de

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`
et
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`
de `software.amazon.kinesis.processor.ShardRecordProcessor`, comme suit :

```

// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// public class TestRecordProcessor implements IRecordProcessor,
// IShutdownNotificationAware {
public class TestRecordProcessor implements ShardRecordProcessor {

```

2. Mettez à jour les instructions `import` pour les méthodes `initialize` et `processRecords`.

```

// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

```

```
//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. Remplacez la méthode `shutdown` par les nouvelles méthodes suivantes : `leaseLost`, `shardEnded` et `shutdownRequested`.

```
// @Override
// public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
//     //
//     // This is moved to shardEnded(...)
//     //
//     try {
//         checkpoint.checkpoint();
//     } catch (ShutdownException | InvalidStateException e) {
//         //
//         // Swallow exception
//         //
//         e.printStackTrace();
//     }
// }

@Override
public void leaseLost(LeaseLostInput leaseLostInput) {

}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}

// @Override
// public void shutdownRequested(IRecordProcessorCheckpoint checkpoint) {
//     //
//     // This is moved to shutdownRequested(ShutdownRequestedInput)
//     //
```

```
//      try {
//          checkpointer.checkpoint();
//      } catch (ShutdownException | InvalidStateException e) {
//          //
//          // Swallow exception
//          //
//          e.printStackTrace();
//      }
//  }

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}
```

Voici la version mise à jour de la classe du processeur d'enregistrements.

```
package com.amazonaws.kcl;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class TestRecordProcessor implements ShardRecordProcessor {
    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
```

```
public void processRecords(ProcessRecordsInput processRecordsInput) {

}

@Override
public void leaseLost(LeaseLostInput leaseLostInput) {

}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        //
        // Swallow the exception
        //
        e.printStackTrace();
    }
}
}
```

Migrer l'usine de traitement des enregistrements

La fabrique de processeurs d'enregistrements est responsable de la création des processeurs d'enregistrements lorsqu'un bail est acquis. Voici un exemple d'usine KCL 1.x.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
```

```
import
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

Pour migrer la fabrique de processeurs d'enregistrements

1. Modifiez l'interface implémentée de

`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory` vers `software.amazon.kinesis.processor.ShardRecordProcessorFactory`, comme suit.

```
// import
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
  com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
```

2. Modifiez la signature de retour pour `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Voici un exemple de fabrique de processeurs d'enregistrements dans 2.0 :

```
package com.amazonaws.kcl;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
```

```
@Override
public ShardRecordProcessor shardRecordProcessor() {
    return new TestRecordProcessor();
}
}
```

Migrer le travailleur

Dans la version 2.0 de KCL, une nouvelle classe, appelée `Scheduler`, remplace la `Worker` classe. Voici un exemple de worker KCL 1.x.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Pour migrer l'application de travail

1. Modifiez la déclaration `import` de la classe `Worker` pour les instructions d'importation pour les classes `Scheduler` et `ConfigsBuilder`.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Créez le `ConfigsBuilder` et un `Scheduler` comme illustré dans l'exemple suivant.

Il est recommandé d'utiliser `KinesisClientUtil` pour créer `KinesisAsyncClient` et configurer `maxConcurrency` dans `KinesisAsyncClient`.

Important

Le client Amazon Kinesis peut voir une augmentation significative de la latence, sauf si vous configurez `KinesisAsyncClient` pour avoir un `maxConcurrency` suffisamment élevée pour autoriser tous les baux et les utilisations supplémentaires de `KinesisAsyncClient`.


```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
    KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
    DynamoDbAsyncClient.builder().region(region).build();
CloudWatchAsyncClient cloudWatchClient =
    CloudWatchAsyncClient.builder().region(region).build();

ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, applicationName,
    kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
    SampleRecordProcessorFactory());

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

Configuration du client Amazon Kinesis

Avec la version 2.0 de la bibliothèque client Kinesis, la configuration du client est passée d'une seule classe de configuration (`KinesisClientLibConfiguration`) à six classes de configuration. Le tableau suivant décrit la migration.

Champs de configuration et leurs nouvelles classes

Champ d'origine	Nouvelle classe de configuration	Description
<code>applicationName</code>	<code>ConfigsBuilder</code>	Le nom de cette KCL application est. Utilisé par défaut pour le <code>tableName</code> et le <code>consumerName</code> .
<code>tableName</code>	<code>ConfigsBuilder</code>	Permet de remplacer le nom du tableau utilisé par le tableau des baux Amazon DynamoDB.
<code>streamName</code>	<code>ConfigsBuilder</code>	Nom du flux à partir duquel cette application traite les enregistrements.
<code>kinesisEndpoint</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>dynamoDBEndpoint</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>initialPositionInStreamExtended</code>	<code>RetrievalConfig</code>	Emplacement de la partition à partir duquel KCL commence à extraire les enregistrements, en commençant par l'exécution initiale de l'application.
<code>kinesisCredentialsProvider</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>dynamoDBCredentialsProvider</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>cloudWatchCredentialsProvider</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.

Champ d'origine	Nouvelle classe de configuration	Description
<code>failoverTimeMillis</code>	<code>LeaseManagementConfig</code>	Nombre de millisecondes qui doivent s'écouler avant que vous puissiez considérer qu'un bail propriétaire a échoué.
<code>workerIdentifier</code>	<code>ConfigsBuilder</code>	Identifiant unique qui représente cette instanciation du processeur d'applications. Il doit être unique.
<code>shardSyncIntervalMillis</code>	<code>LeaseManagementConfig</code>	Délai entre les appels de synchronisation des partitions.
<code>maxRecords</code>	<code>PollingConfig</code>	Permet de définir le nombre maximum d'enregistrements renvoyés par Kinesis.
<code>idleTimeBetweenReadsInMillis</code>	<code>CoordinatorConfig</code>	Cette option a été supprimée. Consultez Suppression du temps d'inactivité .
<code>callProcessorRecordsEvenForEmptyRecordList</code>	<code>ProcessorConfig</code>	Lorsqu'il est défini, le processeur d'enregistrements est appelé même si aucun enregistrement n'a été fourni depuis Kinesis.
<code>parentShardPollIntervalMillis</code>	<code>CoordinatorConfig</code>	À quelle fréquence un processeur d'enregistrements doit-il interroger pour voir si la partition parent est terminée.
<code>cleanupLeasesUponShardCompletion</code>	<code>LeaseManagementConfig</code>	Lorsqu'ils sont définis, les baux sont supprimés dès que les baux enfant ont commencé le traitement.

Champ d'origine	Nouvelle classe de configuration	Description
<code>ignoreUnexpectedChildShards</code>	<code>LeaseManagementConfig</code>	Lorsqu'elles sont définies, les partitions enfant ont une partition ouverte qui est ignorée. Cela concerne principalement DynamoDB Streams.
<code>kinesisClientConfig</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>dynamoDBClientConfig</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>cloudWatchClientConfig</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppressions des configurations du client.
<code>taskBackoffTimeMillis</code>	<code>LifecycleConfig</code>	Durée d'attente pour relancer des tâches ayant échoué.
<code>metricsBufferTimeMillis</code>	<code>MetricsConfig</code>	Contrôle la publication des CloudWatch métriques.
<code>metricsMaxQueueSize</code>	<code>MetricsConfig</code>	Contrôle la publication des CloudWatch métriques.
<code>metricsLevel</code>	<code>MetricsConfig</code>	Contrôle la publication des CloudWatch métriques.
<code>metricsEnabledDimensions</code>	<code>MetricsConfig</code>	Contrôle la publication des CloudWatch métriques.
<code>validateSequenceNumberBeforeCheckpointing</code>	<code>CheckpointConfig</code>	Cette option a été supprimée. Consultez Validation du numéro de séquence des points de contrôle.

Champ d'origine	Nouvelle classe de configuration	Description
<code>regionName</code>	<code>ConfigsBuilder</code>	Cette option a été supprimée. Consultez Suppression des configurations du client .
<code>maxLeasesForWorker</code>	<code>LeaseManagementConfig</code>	Nombre maximum de baux qu'une instance unique de l'application doit accepter.
<code>maxLeasesToStealAtOneTime</code>	<code>LeaseManagementConfig</code>	Nombre maximum de baux qu'une application doit tenter de voler à la fois.
<code>initialLeaseTableReadCapacity</code>	<code>LeaseManagementConfig</code>	La IOPs lecture DynamoDB utilisée si la bibliothèque cliente Kinesis doit créer une nouvelle table de bail DynamoDB.
<code>initialLeaseTableWriteCapacity</code>	<code>LeaseManagementConfig</code>	La IOPs lecture DynamoDB utilisée si la bibliothèque cliente Kinesis doit créer une nouvelle table de bail DynamoDB.
<code>initialPositionInStreamExtended</code>	<code>LeaseManagementConfig</code>	La position initiale dans le flux à laquelle l'application doit commencer. Elle est utilisée uniquement lors de la création de bail initiale.
<code>skipShardSyncAtWorkerInitializationIfLeasesExist</code>	<code>CoordinatorConfig</code>	Désactivez la synchronisation des données de partition si la table des baux contient des baux existants. TODO: KinesisEco -438
<code>shardPrioritization</code>	<code>CoordinatorConfig</code>	Définition des priorités de partition à utiliser.
<code>shutdownGraceMillis</code>	N/A	Cette option a été supprimée. Voir MultiLang Suppressions .

Champ d'origine	Nouvelle classe de configuration	Description
<code>timeoutInSeconds</code>	N/A	Cette option a été supprimée. Voir <code>MultiLangSuppressions</code> .
<code>retryGetRecordsInSeconds</code>	<code>PollingConfig</code>	Configure le délai entre les <code>GetRecords</code> tentatives d'échec.
<code>maxGetRecordsThreadPool</code>	<code>PollingConfig</code>	La taille du pool de threads utilisée pour <code>GetRecords</code> .
<code>maxLeaseRenewalThreads</code>	<code>LeaseManagementConfig</code>	Contrôle la taille du pool de threads des renouvellements de baux. Plus votre application accepte de baux, plus la taille du pool doit être importante.
<code>recordsFetcherFactory</code>	<code>PollingConfig</code>	Permet de remplacer la fabrique utilisée pour créer des extracteurs qui effectuent la récupération à partir de flux.
<code>logWarningForTaskAfterMillis</code>	<code>LifecycleConfig</code>	Temps d'attente avec la consignation d'un avertissement si une tâche n'a pas été terminée.
<code>listShardsBackoffTimeInMillis</code>	<code>RetrievalConfig</code>	Nombre de millisecondes à attendre entre les appels de <code>ListShards</code> en cas de défaillance.
<code>maxListShardsRetryAttempts</code>	<code>RetrievalConfig</code>	Nombre maximum de nouvelles tentatives par <code>ListShards</code> avant l'abandon.

Suppression des temps d'inactivité

Dans la version 1.x du KCL, cela `idleTimeBetweenReadsInMillis` correspondait à deux quantités :

- Durée entre les vérifications de répartition des tâches. Vous pouvez maintenant configurer cette durée entre les tâches en définissant `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis`.
- Durée de veille lorsqu'aucun enregistrement n'a été renvoyé à partir de Kinesis Data Streams. Dans la version 2.0, les enregistrements de diffusion améliorée sont transmis à partir de leur extracteur respectif. Les activités sur l'application consommateur de la partition ont lieu uniquement lorsqu'une demande push arrive.

Suppressions de configurations clientes

Dans la version 2.0, le KCL ne crée plus de clients. Il incombe à l'utilisateur de fournir un client valide. Avec cette modification, tous les paramètres de configuration qui contrôlaient la configuration du client ont été supprimés. Si vous avez besoin de ces paramètres, vous pouvez les définir sur les clients avant de fournir les clients à `ConfigsBuilder`.

Champ supprimé	Configuration équivalente
<code>kinesisEndpoint</code>	Configurez le SDK <code>KinesisAsyncClient</code> avec le point de terminaison préféré <code>:KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()</code> .
<code>dynamoDBEndpoint</code>	Configurez le SDK <code>DynamoDbAsyncClient</code> avec le point de terminaison préféré <code>:DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()</code> .
<code>kinesisClientConfig</code>	Configurez le SDK <code>KinesisAsyncClient</code> avec la configuration requise <code>:KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()</code> .
<code>dynamoDBClientConfig</code>	Configurez le SDK <code>DynamoDbAsyncClient</code> avec la configuration requise <code>:DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()</code> .
<code>cloudWatchClientConfig</code>	Configurez le SDK <code>CloudWatchAsyncClient</code> avec la configuration requise <code>:CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()</code> .

Champ supprimé	Configuration équivalente
<code>regionName</code>	Configurez le SDK avec la région préférée. Il en va de même pour tous les SDK clients. Par exemple, <code>KinesisAsyncClient.builder().region(Region.US_WEST_2).build()</code> .

Lisez les données de Kinesis Data Streams à l'aide d'autres services AWS

Les AWS services suivants peuvent s'intégrer directement à Amazon Kinesis Data Streams pour lire les données des flux de données Kinesis. Passez en revue les informations relatives à chaque service qui vous intéresse et reportez-vous aux références fournies.

Rubriques

- [Lisez les données de Kinesis Data Streams à l'aide d'Amazon EMR](#)
- [Lisez les données de Kinesis Data Streams à l'aide d'Amazon Pipes EventBridge](#)
- [Lisez les données de Kinesis Data Streams à l'aide de AWS Glue](#)
- [Lisez les données de Kinesis Data Streams à l'aide d'Amazon Redshift](#)

Lisez les données de Kinesis Data Streams à l'aide d'Amazon EMR

EMR Les clusters Amazon peuvent lire et traiter les flux Kinesis directement à l'aide d'outils familiers de l'écosystème Hadoop tels que Hive, Pig MapReduce, Hadoop Streaming et Cascading. API Vous pouvez également associer les données en temps réel de Kinesis Data Streams aux données existantes sur Amazon S3, Amazon DynamoDB HDFS et dans un cluster en cours d'exécution. Vous pouvez charger directement les données d'Amazon EMR vers Amazon S3 ou DynamoDB pour les activités de post-traitement.

Pour plus d'informations, consultez [Amazon Kinesis](#) dans le Amazon EMR Release Guide.

Lisez les données de Kinesis Data Streams à l'aide d'Amazon Pipes EventBridge

Amazon EventBridge Pipes prend en charge les flux de données Kinesis en tant que source. Amazon EventBridge Pipes vous aide à créer des point-to-point intégrations entre les producteurs d'événements et les consommateurs grâce à des étapes facultatives de transformation, de filtrage et d'enrichissement. Vous pouvez utiliser EventBridge Pipes pour recevoir des enregistrements dans un flux de données Kinesis et éventuellement filtrer ou améliorer ces enregistrements avant de les envoyer vers l'une des destinations disponibles pour traitement, y compris Kinesis Data Streams.

Pour plus d'informations, consultez [Amazon Kinesis Stream en tant que source](#) dans le Amazon EventBridge Release Guide.

Lisez les données de Kinesis Data Streams à l'aide de AWS Glue

Grâce au AWS Glue streamingETL, vous pouvez créer des tâches d'extraction, de transformation et de chargement (ETL) en streaming qui s'exécutent en continu et consomment les données d'Amazon Kinesis Data Streams. Les tâches nettoient et transforment les données, puis chargent les résultats dans des lacs de données ou des magasins de JDBC données Amazon S3.

Pour plus d'informations, consultez la section [Streaming ETL des jobs AWS Glue dans](#) le Guide AWS Glue de publication.

Lisez les données de Kinesis Data Streams à l'aide d'Amazon Redshift

Amazon Redshift prend en charge l'ingestion en flux continu à partir d'Amazon Kinesis Data Streams. La fonctionnalité d'ingestion en flux continu Amazon Redshift garantit une ingestion à faible latence et à haute vitesse des données en flux continu provenant d'Amazon Kinesis Data Streams dans une vue matérialisée Amazon Redshift. L'ingestion du streaming par Amazon Redshift élimine le besoin de trier les données dans Amazon S3 avant de les ingérer dans Amazon Redshift.

Pour plus d'informations, consultez la rubrique [Ingestion en flux continu](#) dans le Guide de mise à jour d'Amazon Redshift (français non garanti).

Lisez depuis Kinesis Data Streams à l'aide d'intégrations tierces

Vous pouvez lire les données des flux de données Amazon Kinesis Data Streams à l'aide de l'une des options tierces suivantes qui s'intègrent à Kinesis Data Streams. Sélectionnez l'option

sur laquelle vous souhaitez en savoir plus et recherchez des ressources et des liens vers la documentation pertinente.

Rubriques

- [Apache Flink](#)
- [Adobe Experience Platform](#)
- [Apache Druid](#)
- [Apache Spark](#)
- [Databricks](#)
- [Plateforme Confluent Kafka](#)
- [Kinesumer](#)
- [Talend](#)

Apache Flink

Apache Flink est un environnement et un moteur de traitement distribué pour les calculs avec état sur des flux de données illimités et limités. Pour plus d'informations sur la consommation de Kinesis Data Streams à l'aide d'Apache Flink, consultez la rubrique [Connecteur Amazon Kinesis Data Streams](#) (français non garanti).

Adobe Experience Platform

Adobe Experience Platform permet aux entreprises de centraliser et de standardiser les données des consommateurs depuis n'importe quel système. Elle applique ensuite la science des données et le machine learning pour améliorer considérablement la conception et la fourniture d'expériences riches et personnalisées. Pour plus d'informations sur la consommation de flux de données Kinesis à l'aide d'Adobe Experience Platform, consultez le connecteur [Amazon Kinesis](#).

Apache Druid

Druid est une base de données analytique en temps réel très performante qui permet d'effectuer des requêtes en moins d'une seconde sur des données en flux continu et en lots, à l'échelle et sous charge. Pour plus d'informations sur l'ingestion de flux de données Kinesis à l'aide d'Apache Druid, consultez Amazon [Kinesis](#) ingestion.

Apache Spark

Apache Spark est un moteur analytique unifié, pour le traitement des données à grande échelle. Il fournit un haut niveau APIs en Java, Scala, Python et R, ainsi qu'un moteur optimisé qui prend en charge les graphes d'exécution généraux. Vous pouvez utiliser Apache Spark pour créer des applications de traitement de flux qui consomment les données de vos flux de données Kinesis.

[Pour utiliser des flux de données Kinesis à l'aide d'Apache Spark Structured Streaming, utilisez le connecteur Amazon Kinesis Data Streams.](#) Ce connecteur prend en charge la consommation grâce à Enhanced Fan-Out, qui fournit à votre application un débit de lecture dédié pouvant atteindre 2 Mo de données par seconde et par partition. Pour plus d'informations, voir [Développement de consommateurs personnalisés avec un débit dédié \(Fan-Out amélioré\)](#).

Pour utiliser les flux de données Kinesis à l'aide de Spark Streaming, consultez [Spark Streaming + Kinesis Integration](#).

Databricks

Databricks est une plateforme basée sur le cloud qui fournit un environnement collaboratif pour l'ingénierie des données, la science des données et le machine learning. Pour plus d'informations sur la consommation de flux de données Kinesis à l'aide de Databricks, consultez [Connect to Amazon Kinesis](#).

Plateforme Confluent Kafka

La plateforme Confluent, basée sur Kafka, offre des fonctionnalités supplémentaires qui aident les entreprises à créer et à gérer des pipelines de données en temps réel et des applications de streaming. Pour plus d'informations sur la consommation de flux de données Kinesis à l'aide de la plateforme Confluent, consultez [Amazon Kinesis Source Connector pour Confluent Platform](#).

Kinesumer

Kinesumer est un client Go qui implémente un client de groupe de consommateurs distribué côté client pour les flux de données Kinesis. Pour plus d'informations, consultez le [référentiel GitHub Kinesumer](#) (français non garanti).

Talend

Talend est un logiciel d'intégration et de gestion des données qui permet aux utilisateurs de collecter, de transformer et de connecter des données provenant de différentes sources de manière évolutive

et efficace. Pour plus d'informations sur la consommation de flux de données Kinesis à l'aide de Talend, consultez [Connect Talend à un flux Amazon Kinesis](#).

Résoudre les problèmes des utilisateurs de Kinesis Data Streams

Les rubriques suivantes proposent des solutions aux problèmes courants rencontrés par les utilisateurs d'Amazon Kinesis Data Streams :

- [Certains enregistrements Kinesis Data Streams sont ignorés lors de l'utilisation de la bibliothèque cliente Kinesis](#)
- [Les enregistrements appartenant à la même partition sont traités par différents processeurs d'enregistrements en même temps](#)
- [L'application destinée au consommateur lit plus lentement que prévu](#)
- [GetRecords renvoie un tableau d'enregistrements vide même s'il y a des données dans le flux](#)
- [L'utilisateur de partition expire de façon inattendue](#)
- [Le traitement des dossiers des consommateurs prend du retard](#)
- [Erreur d'autorisation de clé KMS principale non autorisée](#)
- [Résoudre d'autres problèmes courants rencontrés par les consommateurs](#)

Certains enregistrements Kinesis Data Streams sont ignorés lors de l'utilisation de la bibliothèque cliente Kinesis

La cause la plus courante des enregistrements ignorés est une exception non gérée émise par `processRecords`. La bibliothèque cliente Kinesis (KCL) s'appuie sur votre `processRecords` code pour gérer les exceptions liées au traitement des enregistrements de données. Toute exception émise par `processRecords` est absorbée par le KCL. Pour éviter un nombre infini de tentatives en cas d'échec récurrent, le lot d'enregistrements traité au moment de l'exception KCL n'est pas renvoyé. Il appelle KCL `processRecords` ensuite le lot d'enregistrements de données suivant sans redémarrer le processeur d'enregistrements. Il en résulte que des enregistrements sont ignorés dans les applications consommateur. Pour éviter d'avoir des enregistrements ignorés, gérez toutes les exceptions dans `processRecords` de manière appropriée.

Les enregistrements appartenant à la même partition sont traités par différents processeurs d'enregistrements en même temps

Pour toute application Kinesis Client Library (KCL) en cours d'exécution, une partition n'a qu'un seul propriétaire. Toutefois, plusieurs processeurs d'enregistrements peuvent traiter temporairement la même partition. Dans le cas d'une instance de travail qui perd sa connectivité réseau, elle KCL suppose que le travailleur inaccessible ne traite plus les enregistrements, une fois le délai de basculement expiré, et demande aux autres instances de travail de prendre le relais. Pendant une courte période, les nouveaux processeurs d'enregistrements et ceux de l'application de travail inaccessible peuvent traiter les données depuis la même partition.

Vous devez définir un délai de basculement qui est approprié pour votre application. Pour les applications à faible latence, la valeur par défaut de 10 secondes peut représenter l'intervalle de temps maximal pendant lequel vous voulez attendre. Cependant, dans les cas où vous prévoyez des problèmes de connectivité liés, par exemple lorsque les appels sont effectués dans des zones géographiques où la connectivité peut être perdue plus fréquemment, ce nombre peut être trop faible.

Votre application doit anticiper et gérer ce scénario, sachant notamment que la connectivité réseau est généralement restaurée pour l'application de travail précédemment inaccessible. Si un processeur d'enregistrements se voit prendre ses partitions par un autre processeur d'enregistrements, il doit gérer les deux cas suivants pour effectuer un arrêt approprié :

1. Une fois l'appel en cours `processRecords` terminé, KCL invoque la méthode d'arrêt sur le processeur d'enregistrements avec la raison d'arrêt « ZOMBIE ». Vos processeurs d'enregistrements sont censés nettoyer toutes les ressources de façon appropriée, puis se fermer.
2. Lorsque vous essayez de vous faire repérer par un travailleur « zombie », les KCL lancent `ShutdownException`. Après avoir reçu cette exception, votre code est censé fermer proprement la méthode en cours.

Pour plus d'informations, consultez [Gérer les enregistrements dupliqués](#).

L'application destinée au consommateur lit plus lentement que prévu

Les raisons principales les plus courantes liées au débit de lecture plus lent que prévu sont les suivantes :

1. Plusieurs applications consommateur ont un nombre total de lectures qui dépasse les limites par partition. Pour plus d'informations, consultez [Quotas et limites](#). Dans ce cas, augmentez le nombre de partitions dans votre flux de données Kinesis.
2. Le nombre [limite](#) qui spécifie le nombre maximal de `GetRecords` par appel peut avoir été configuré avec une valeur faible. Si vous utilisez le KCL, vous avez peut-être configuré le travailleur avec une faible valeur pour la `maxRecords` propriété. En général, nous recommandons d'utiliser les valeurs système par défaut pour cette propriété.
3. La logique de votre `processRecords` appel peut prendre plus de temps que prévu pour plusieurs raisons : elle peut être CPU intensive, bloquer les E/S ou être bloquée lors de la synchronisation. Pour tester si cela est vrai, testez les processeurs enregistrements vides et comparez le débit de lecture. Pour plus d'informations sur la façon de faire face aux données entrantes, consultez la page [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).

Si vous n'avez qu'une seule application consommateur, il est toujours possible de lire au moins deux fois plus vite que la vitesse de placement. En effet, vous pouvez écrire jusqu'à 1 000 enregistrements par seconde pour les écritures, jusqu'à une vitesse d'écriture totale des données maximale de 1 Mo par seconde (y compris les clés de partition). Chaque partition ouverte peut prendre en charge jusqu'à 5 transactions par seconde pour la lecture, jusqu'à une vitesse de lecture totale des données maximale de 2 Mo par seconde. Notez que chaque lecture (appel de `GetRecords`) extrait un lot d'enregistrements. La taille des données renvoyées par appel de `GetRecords` varie en fonction de l'utilisation de la partition. La taille maximale des données que `GetRecords` peut renvoyer est de 10 Mo. Si un appel retourne cette valeur limite, les appels suivants effectués dans les 5 secondes suivantes lèvent l'exception `ProvisionedThroughputExceededException`.

GetRecords renvoie un tableau d'enregistrements vide même s'il y a des données dans le flux

L'utilisation ou l'obtention d'enregistrements est un modèle d'extraction. Les développeurs sont tenus d'effectuer [GetRecords](#) des appels en boucle continue, sans interruption. Chaque appel de `GetRecords` renvoie également une valeur `ShardIterator`, qui doit être utilisée dans la prochaine itération de la boucle.

L'opération `GetRecords` ne se bloque pas. Elle renvoie immédiatement un résultat : des enregistrements de données appropriés ou un élément `Records` vide. Un élément `Records` vide est renvoyé dans deux conditions :

1. La partition ne contient plus de données pour le moment.
2. Il n'y a pas de données près de la partie de la partition vers laquelle pointe le `ShardIterator`.

Cette dernière condition est subtile, mais constitue un compromis de conception nécessaire pour éviter un temps de recherche illimité (latence) lors de l'extraction des enregistrements. Ainsi, l'application qui utilise le flux doit être exécutée en boucle et appeler `GetRecords`, en traitant les enregistrements vides comme une évidence.

Dans un scénario de production, la seule fois où la boucle continue doit être fermée est lorsque la valeur `NextShardIterator` est `NULL`. Lorsque `NextShardIterator` est `NULL`, cela signifie que la partition actuelle a été fermée et que la valeur `ShardIterator` pointerait autrement après le dernier enregistrement. Si l'application consommateur n'appelle jamais `SplitShard` ou `MergeShards`, la partition reste ouverte et les appels de `GetRecords` ne renvoient jamais de valeur `NextShardIterator` égale à `NULL`.

Si vous utilisez la bibliothèque cliente Kinesis (KCL), le modèle de consommation ci-dessus est résumé pour vous. Cela inclut la gestion automatique d'un ensemble de partitions qui changent dynamiquement. Avec le KCL, le développeur fournit uniquement la logique pour traiter les enregistrements entrants. Cela est rendu possible par le fait que la bibliothèque effectue des appels continus de `GetRecords` à votre place.

L'itérateur de partition expire de façon inattendue

Un nouvel itérateur de partition est renvoyé par chaque demande de `GetRecords` (en tant que `NextShardIterator`), que vous utilisez ensuite dans la demande `GetRecords` suivante (en tant que `ShardIterator`). En règle générale, cet itérateur de partition n'expire pas avant d'être utilisé. Cependant, vous pouvez constater que les itérateurs de partition expirent lorsque vous n'avez pas appelé `GetRecords` pendant plus de 5 minutes ou que vous avez redémarré votre application consommateur.

Si l'itérateur de partition expire immédiatement avant d'être utilisé, cela peut indiquer que la table DynamoDB utilisée par Kinesis n'a pas suffisamment de capacité pour stocker les données de bail. Cette situation est plus susceptible de se produire si vous avez un grand nombre de partitions. Pour résoudre ce problème, augmentez la capacité d'écriture attribuée à la table de partition. Pour plus d'informations, consultez [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#).

Le traitement des dossiers des consommateurs prend du retard

Dans la plupart des cas d'utilisation, les applications consommateur lisent les données les plus récentes dans le flux. Dans certains cas, les lectures de l'application consommateur peuvent être en retard, ce qui n'est pas souhaitable. Une fois que vous avez identifié quel est le retard dans les lectures des applications consommateur, examinez les motifs les plus courants du retard des applications consommateur.

Commencez par la métrique `GetRecords.IteratorAgeMilliseconds`, qui suit la position de lecture dans toutes les partitions et les applications consommateur dans le flux. Notez que si l'ancienneté de l'itérateur dépasse 50 % de la période de conservation (par défaut 24 heures, configurable jusqu'à 365 jours), il y a un risque de perte de données suite à l'expiration des enregistrements. Une solution rapide provisoire consiste à augmenter la période de conservation. Cela arrête la perte de données importantes pendant que vous continuez à résoudre le problème. Pour plus d'informations, consultez [Surveillez le service Amazon Kinesis Data Streams avec Amazon CloudWatch](#). Identifiez ensuite le retard pris par votre application client pour lire chaque partition à l'aide d'une CloudWatch métrique personnalisée émise par la bibliothèque cliente Kinesis KCL (`MillisBehindLatest`). Pour plus d'informations, consultez [Surveillez la bibliothèque cliente Kinesis avec Amazon CloudWatch](#).

Voici les raisons les plus courantes pour lesquelles les applications consommateur sont en retard :

- Les fortes augmentations soudaines `GetRecords.IteratorAgeMilliseconds` ou indiquent `MillisBehindLatest` généralement un problème transitoire, tel que des échecs de API fonctionnement d'une application en aval. Vous devez examiner ces augmentations soudaines si l'une des métriques affiche systématiquement ce comportement.
- Une augmentation progressive de ces métriques indique qu'une application consommateur est en retard sur le flux, car elle ne traite pas les enregistrements assez vite. Les causes les plus courantes de ce comportement proviennent de ressources physiques insuffisantes ou d'une logique de traitement des enregistrements qui n'a pas été mise à l'échelle après une augmentation du débit du flux. Vous pouvez vérifier ce comportement en examinant les autres CloudWatch métriques personnalisées KCL émises par l'opération `processTask`, notamment `RecordProcessor.processRecords.TimeSuccess`, et `RecordsProcessed`.
- Si vous constatez une augmentation de la métrique `processRecords.Time` qui correspond à un débit plus élevé, vous devez analyser votre logique de traitement des enregistrements afin de déterminer pourquoi elle n'est pas mise à l'échelle alors que le débit augmente.

- Si vous constatez une hausse des valeurs `processRecords.Time` qui n'est pas liée à l'augmentation du débit, vérifiez si vous effectuez des appels bloquants dans le chemin critique, lesquels sont souvent responsables des ralentissements du traitement des enregistrements. Une autre approche consiste à augmenter votre parallélisme en augmentant le nombre de partitions. Enfin, vérifiez que vous disposez d'une quantité suffisante de ressources physiques (mémoire, CPU utilisation, etc.) sur les nœuds de traitement sous-jacents pendant les pics de demande.

Erreur d'autorisation de clé KMS principale non autorisée

Cette erreur se produit lorsqu'une application grand public lit un flux chiffré sans autorisation sur la clé KMS principale. Pour attribuer à une application des autorisations d'accès à une KMS clé, consultez les sections [Utilisation de politiques clés dans AWS KMS](#) et [Utilisation de IAM politiques avec AWS KMS](#).

Résoudre d'autres problèmes courants rencontrés par les consommateurs

- [Pourquoi le déclencheur Kinesis Data Streams ne parvient-il pas à invoquer ma fonction Lambda ?](#)
- [Comment détecter et résoudre les `ReadProvisionedThroughputExceeded` exceptions dans Kinesis Data Streams ?](#)
- [Pourquoi est-ce que je rencontre des problèmes de latence élevée avec Kinesis Data Streams ?](#)
- [Pourquoi mon flux de données Kinesis renvoie-t-il une erreur interne de serveur 500 ?](#)
- [Comment résoudre les problèmes liés au blocage ou au blocage d'une KCL application pour Kinesis Data Streams ?](#)
- [Puis-je utiliser différentes applications Amazon Kinesis Client Library avec la même table Amazon DynamoDB ?](#)

Optimisez les utilisateurs d'Amazon Kinesis Data Streams

Vous pouvez optimiser davantage votre client Amazon Kinesis Data Streams en fonction du comportement spécifique que vous observez.

Consultez les rubriques suivantes pour identifier des solutions.

Rubriques

- [Améliorez le traitement à faible latence](#)

- [Traitez les données sérialisées à l' AWS Lambda aide de la bibliothèque Kinesis Producer](#)
- [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#)
- [Gérer les enregistrements dupliqués](#)
- [Gérez le démarrage, l'arrêt et la régulation](#)

Améliorez le traitement à faible latence

Le délai de propagation est défini comme la end-to-end latence entre le moment où un enregistrement est écrit dans le flux et celui où il est lu par une application grand public. Ce délai varie en fonction d'un certain nombre de facteurs, mais il est surtout modifié par l'intervalle d'interrogation des applications consommateur.

Pour la plupart des applications, nous recommandons d'interroger chaque partition une fois par seconde par application. Ainsi, plusieurs applications consommateur peuvent traiter un flux simultanément sans atteindre les limites de 5 appels `GetRecords` par seconde de Amazon Kinesis Data Streams. En outre, traiter de gros lots de données a tendance à être plus efficace pour réduire les latences réseau et les autres latences en aval dans votre application.

Les KCL valeurs par défaut sont définies de manière à suivre la meilleure pratique consistant à effectuer un sondage toutes les secondes. Elles génère des délais de propagation normalement inférieurs à 1 seconde.

Les enregistrements Kinesis Data Streams sont disponibles pour la lecture immédiatement après avoir été écrits. Il existe quelques cas d'utilisation qui ont besoin de tirer parti de cet avantage et d'utiliser les données du flux dès qu'elles sont disponibles. Vous pouvez réduire considérablement le délai de propagation en remplaçant les paramètres KCL par défaut pour interroger plus fréquemment, comme le montrent les exemples suivants.

Code de configuration KCL Java :

```
kinesisClientLibConfiguration = new
    KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Paramétrage du fichier de propriétés pour Python et Ruby KCL :

```
idleTimeBetweenReadsInMillis = 250
```

Note

Étant donné que Kinesis Data Streams est limité à 5 appels `GetRecords` par seconde et par partition, le fait de définir la propriété `idleTimeBetweenReadsInMillis` sur une valeur inférieure à 200 ms peut faire que votre application émet l'exception `ProvisionedThroughputExceededException`. Une trop grande partie de ces exceptions peuvent engendrer des interruptions exponentielles et provoquer ainsi des latences de traitement inattendues significatives. Si vous définissez cette propriété sur une valeur égale ou juste supérieure à 200 ms et que vous avez plusieurs applications de traitement, vous constaterez une limitation similaire.

Traitez les données sérialisées à l' AWS Lambda aide de la bibliothèque Kinesis Producer

La [Kinesis Producer Library](#) (KPL) agrège les petits enregistrements formatés par l'utilisateur en enregistrements plus grands jusqu'à 1 Mo afin de mieux utiliser le débit d'Amazon Kinesis Data Streams. Bien que Java KCL prenne en charge la désagrégation de ces enregistrements, vous devez utiliser un module spécial pour désagréger les enregistrements lorsque vous les utilisez en AWS Lambda tant que consommateur de vos flux. Vous pouvez obtenir le code de projet et les instructions nécessaires auprès GitHub des modules de [désagrégation de la bibliothèque Kinesis Producer pour Lambda](#). AWS Les composants de ce projet vous permettent de traiter des données KPL sérialisées dans AWS Lambda, en Java, Node.js et Python. Ces composants peuvent également être utilisés dans le cadre d'une [KCLapplication multilingue](#).

Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions

Le repartitionnement vous permet d'augmenter ou de diminuer le nombre de partitions dans un flux afin de faire face aux modifications de la vitesse de circulation des données dans le flux. En général, le repartitionnement est effectué par une application administrative qui surveille les métriques de traitement des données de partition. Bien qu'il ne lance pas KCL lui-même les opérations de repartage, il est conçu pour s'adapter aux variations du nombre de partitions résultant du repartage.

Comme indiqué dans [Utilisez un tableau des baux pour suivre les partitions traitées par l'application KCL client](#), il KCL suit les partitions du flux à l'aide d'une table Amazon DynamoDB. Lorsque de nouvelles partitions sont créées à la suite d'un redécoupage, les KCL nouvelles partitions sont découvertes et renseigne de nouvelles lignes dans le tableau. Les applications de travail détectent automatiquement les nouvelles partitions et créent des processeurs pour traiter les données qu'elles contiennent. KCL distribue également les fragments du flux entre tous les opérateurs et processeurs de disques disponibles.

Cela garantit que toutes les données qui existaient dans des partitions avant le repartage sont traitées en premier. Une fois que les données ont été traitées, les données issues des nouvelles partitions sont envoyées aux processeurs d'enregistrements. De cette façon, l'ordre dans lequel les enregistrements de données ont été ajoutés au flux pour une clé de partition donnée est KCL préservé.

Exemple : repartage, mise à l'échelle et traitement parallèle

L'exemple suivant montre comment vous pouvez gérer KCL le redimensionnement et le repartage :

- Par exemple, si votre application s'exécute sur une EC2 instance et traite un flux de données Kinesis comportant quatre partitions. Cette instance possède un KCL travailleur et quatre processeurs d'enregistrements (un processeur d'enregistrement pour chaque partition). Ces quatre processeurs d'enregistrements s'exécutent en parallèle dans le même processus.
- Ensuite, si vous mettez à l'échelle l'application pour utiliser une autre instance, vous avez deux instances qui traitent un flux qui contient quatre partitions. Lorsque le KCL travailleur démarre sur la deuxième instance, il équilibre la charge avec la première instance, de sorte que chaque instance traite désormais deux partitions.
- Si vous décidez ensuite de fractionner les quatre partitions en cinq partitions. KCL à encore, le traitement est coordonné entre les instances : une instance traite trois partitions et l'autre deux partitions. Une coordination similaire se produit lorsque vous fusionnez des partitions.

Généralement, lorsque vous utilisez le KCL, vous devez vous assurer que le nombre d'instances ne dépasse pas le nombre de partitions (sauf en cas de panne en veille). Chaque partition est traitée par un seul KCL opérateur et possède exactement un processeur d'enregistrement correspondant. Vous n'avez donc jamais besoin de plusieurs instances pour traiter une partition. Cependant, une seule application de travail peut traiter tout nombre de partitions, et il est donc approprié que le nombre de partitions dépasse le nombre d'instances.

Pour faire monter le traitement en puissance dans votre application, vous devez tester une combinaison des méthodes suivantes :

- Augmentation de la taille de l'instance (car tous les processeurs d'enregistrements s'exécutent en parallèle dans un processus)
- Augmentation du nombre d'instances jusqu'au nombre maximal de partitions ouvertes (car les partitions peuvent être traitées de façon indépendante)
- Augmentation du nombre de partitions (ce qui augmente le niveau de parallélisme)

Notez que vous pouvez utiliser autoscaling pour mettre à l'échelle automatiquement vos instances sur la base des métriques appropriées. Pour plus d'informations, consultez le [guide de l'utilisateur d'Amazon EC2 Auto Scaling](#).

Lorsque le repartage augmente le nombre de partitions dans le flux, l'augmentation correspondante du nombre de processeurs d'enregistrements augmente la charge sur les EC2 instances qui les hébergent. Si les instances font partie d'un groupe Auto Scaling et que la charge augmente suffisamment, le groupe Auto Scaling ajoute des instances pour gérer l'augmentation de la charge. Vous devez configurer vos instances pour lancer votre Amazon Kinesis Data Streams au démarrage, afin que les applications de travail et processeurs d'enregistrements supplémentaires deviennent actifs immédiatement sur la nouvelle instance.

Pour en savoir plus sur le repartitionnement, référez-vous à la section [Revisiionner un stream](#).

Gérer les enregistrements dupliqués

Il y a deux raisons principales pour lesquelles les enregistrements peuvent être distribués plusieurs fois à votre Amazon Kinesis Data Streams : nouvelles tentatives de l'application producteur et nouvelles tentatives de l'application consommateur. Votre application doit anticiper et gérer adéquatement le traitement des enregistrements plusieurs fois.

Réessaie le producteur

Prenez l'exemple d'une application producteur qui connaît un délai d'attente lié au réseau après avoir effectué un appel vers `PutRecord`, mais avant de recevoir un accusé de réception de Amazon Kinesis Data Streams. L'application producteur ne peut pas être sûre que l'enregistrement a été distribué à Kinesis Data Streams. En supposant que chaque enregistrement est important pour l'application, l'application producteur aura été écrite de façon à renouveler l'appel avec les mêmes données. Si les deux appels vers `PutRecord` sur ces mêmes données ont été validés avec succès

pour Kinesis Data Streams, il y aura deux enregistrements Kinesis Data Streams. Même si les deux enregistrements comportent des données identiques, ils ont aussi des numéros de séquence uniques. Les applications qui ont besoin de garanties strictes doivent intégrer une clé primaire dans l'enregistrement pour supprimer les doubles ultérieurement lors du traitement. Notez que le nombre de doubles dû aux tentatives de l'application producteur est généralement faible par rapport au nombre de doubles dû aux tentatives de l'application consommateur.

Note

Si vous utilisez le `AWS SDKPutRecord`, découvrez le [comportement de la SDK nouvelle tentative](#) dans le guide de l'utilisateur de AWS SDKs and Tools.

Réessaie par le consommateur

Les tentatives de l'application consommateur (l'application qui traite les données) ont lieu lorsque les processeurs d'enregistrements redémarrent. Les processeurs d'enregistrements d'une même partition redémarrent dans les cas suivants :

1. Une application de travail s'arrête de manière inattendue
2. Des instances de travail sont ajoutées ou supprimées
3. Des partitions sont fusionnées ou fractionnées
4. L'application est déployée

Dans tous ces cas, le mappage shards-to-worker-to-record-processor est continuellement mis à jour pour le traitement de l'équilibrage de charge. Les processeurs de partition qui ont été migrés vers d'autres instances recommencent à traiter les enregistrements depuis le dernier point de contrôle. Il en résulte un traitement double des enregistrements, comme l'illustre l'exemple ci-dessous. Pour plus d'informations sur l'équilibrage de charge, consultez [Utilisez le redécoupage, le dimensionnement et le traitement parallèle pour modifier le nombre de partitions](#).

Exemple : une nouvelle tentative du client aboutit à une nouvelle livraison d'enregistrements

Dans cet exemple, vous avez une application qui lit continuellement les enregistrements à partir d'un flux, regroupe les enregistrements dans un fichier local et charge ce fichier dans Amazon S3. Pour simplifier, supposons qu'il y a une seule partition et une application de travail qui traite la partition. Prenons l'exemple de la séquence d'événements suivante, en supposant que le dernier point de contrôle était au numéro d'enregistrement 10 000 :

1. Une application de travail lit le prochain lot d'enregistrements à partir de la partition, soit les enregistrements 10 001 à 20 000.
2. L'application de travail passe le lot d'enregistrements au processeur d'enregistrements associé.
3. Le processeur d'enregistrements regroupe les données, crée un fichier Amazon S3 et charge ce fichier dans Amazon S3 avec succès.
4. L'application de travail s'arrête de façon inattendue avant l'arrivée d'un nouveau point de contrôle.
5. L'application, l'application de travail et le processeur d'enregistrements redémarrent.
6. L'application de travail commence maintenant à lire depuis le dernier point de contrôle, ici 10 001.

Donc les enregistrements 10 001 à 20 000 sont utilisés plusieurs fois.

Faire preuve de résilience face aux nouvelles tentatives des consommateurs

Même si les enregistrements peuvent être traités plusieurs fois, votre application peut vouloir présenter les effets secondaires en indiquant que les enregistrements ont été traités une seule fois (traitement idempotent). Les solutions à ce problème varient en complexité et précision. Si la destination des données finales peut gérer les doubles correctement, nous vous recommandons de vous appuyer sur la destination finale pour effectuer un traitement idempotent. Par exemple, avec [Opensearch](#), vous pouvez utiliser une combinaison de versionnement et d'unique IDs pour éviter le traitement dupliqué.

Dans l'exemple de la section précédente, l'application lit en permanence les enregistrements à partir d'un flux, regroupe les enregistrements dans un fichier local et charge ce fichier dans Amazon S3. Comme illustré, les enregistrements 10 001 à 20 000 sont utilisés plusieurs fois, ce qui donne plusieurs fichiers Amazon S3 contenant les mêmes données. Une façon d'atténuer les doubles constatés dans cet exemple est de faire en sorte que l'étape 3 utilise le schéma suivant :

1. Le processeur d'enregistrements utilise un nombre fixe d'enregistrements par fichier Amazon S3, par exemple 5 000.
2. Le nom de fichier utilise le schéma suivant : préfixe Amazon S3, id de partition et `First-Sequence-Num`. Dans ce cas, ce peut être quelque chose comme `sample-shard000001-10001`.
3. Une fois que vous avez chargé le fichier Amazon S3, effectuez un point de contrôle en spécifiant `Last-Sequence-Num`. Dans ce cas, vous effectuez le point de contrôle au numéro d'enregistrement 15 000.

Avec ce schéma, même si les enregistrements sont traités plusieurs fois, le fichier Amazon S3 résultant a le même nom et contient les mêmes données. Les tentatives se traduisent uniquement par l'écriture des mêmes données dans le même fichier plusieurs fois.

Dans le cas d'une opération de repartitionnement, le nombre d'enregistrements laissés dans la partition peut être inférieur au nombre fixe voulu. Dans ce cas, votre méthode `shutdown()` doit vider le fichier dans Amazon S3 et effectuer un point de contrôle sur le dernier numéro de séquence. Le schéma ci-dessus est aussi compatible avec les opérations de repartitionnement.

Gérez le démarrage, l'arrêt et la régulation

Voici quelques éléments de réflexion supplémentaires à intégrer dans la conception de votre Amazon Kinesis Data Streams.

Rubriques

- [Producteurs et consommateurs de données en démarrage](#)
- [Arrêter une application Amazon Kinesis Data Streams](#)
- [Limitation de lecture](#)

Producteurs et consommateurs de données en démarrage

Par défaut, la lecture des enregistrements KCL commence à partir de la pointe du flux, qui est le dernier enregistrement ajouté. Dans cette configuration, si une application productrice de données ajoute des enregistrements avant le démarrage des processeurs d'enregistrements, ces enregistrements ne sont pas lus par les processeurs d'enregistrements une fois qu'ils ont démarré.

Pour modifier le comportement des processeurs d'enregistrements afin qu'ils lisent toujours les données à partir du début du flux, définissez la valeur suivante dans le fichier de propriétés de votre application Amazon Kinesis Data Streams :

```
initialPositionInStream = TRIM_HORIZON
```

Par défaut, Amazon Kinesis Data Streams stocke toutes les données pendant 24 heures. Il prend également en charge la conservation étendue jusqu'à 7 jours et la conservation à long terme jusqu'à 365 jours. Ce laps de temps est appelé la période de conservation. Si vous définissez la position de début sur `TRIM_HORIZON`, le processeur d'enregistrements démarre par les données les plus anciennes du flux, telles qu'elles sont définies par la période de conservation. Même avec la valeur

TRIM_HORIZON, si un processeur d'enregistrements démarre après un délai supérieur à la période de conservation, certains enregistrements du flux ne sont plus disponibles. C'est pourquoi vous devez toujours faire en sorte que les applications grand public lisent à partir du flux et utiliser la CloudWatch métrique `GetRecords.IteratorAgeMilliseconds` pour vérifier que les applications suivent le rythme des données entrantes.

Dans certains scénarios, il peut être très bien pour les processeurs d'enregistrements de manquer les quelques premiers enregistrements du flux. Par exemple, vous pouvez exécuter certains enregistrements initiaux dans le flux pour vérifier que le flux fonctionne end-to-end comme prévu. Après avoir fait cette vérification initiale, démarrez vos applications de travail et commencez à placer des données de production dans le flux.

Pour plus d'informations sur le paramètre TRIM_HORIZON, consultez [Utiliser des itérateurs de partitions](#).

Arrêter une application Amazon Kinesis Data Streams

Lorsque votre application Amazon Kinesis Data Streams a terminé la tâche prévue, vous devez l'arrêter en mettant fin EC2 aux instances sur lesquelles elle s'exécute. Vous pouvez arrêter les instances à l'aide de la [AWS Management Console](#) ou de la [AWS CLI](#).

Après avoir arrêté votre application Amazon Kinesis Data Streams, vous devez supprimer la table Amazon DynamoDB utilisée pour suivre l'état de KCL l'application.

Limitation de lecture

Le débit d'un flux est configuré au niveau de la partition. Chaque partition a un débit de lecture allant jusqu'à 5 transactions par seconde pour les lectures, jusqu'à un taux total de lecture de données maximal de 2 Mo par seconde. Si une application (ou un groupe d'applications fonctionnant sur le même flux) tente d'extraire des données à partir d'une partition à une vitesse supérieure ; Kinesis Data Streams limite les opérations Get correspondantes.

Dans une application Amazon Kinesis Data Streams, si un processeur d'enregistrements traite les données plus vite que la limite, comme dans le cas d'un basculement, la limitation se produit. Dans la KCL mesure où il gère les interactions entre l'application et Kinesis Data Streams, les exceptions de limitation se produisent dans KCL le code plutôt que dans le code de l'application. Cependant, étant donné que KCL ces exceptions sont enregistrées, vous les voyez dans les journaux.

Si vous trouvez que votre application est systématiquement limitée, vous devez penser à augmenter le nombre de partitions du flux.

Surveillez Kinesis Data Streams

Vous pouvez surveiller vos flux de données dans Amazon Kinesis Data Streams en utilisant les fonctionnalités suivantes :

- [CloudWatch métriques](#) — Kinesis Data Streams envoie à CloudWatch Amazon des métriques personnalisées avec un suivi détaillé pour chaque flux.
- [Kinesis Agent : l'agent](#) Kinesis publie des CloudWatch métriques personnalisées pour aider à évaluer si l'agent fonctionne comme prévu.
- [APIjournalisation](#) : Kinesis Data Streams permet de AWS CloudTrail API consigner les appels et de stocker les données dans un compartiment Amazon S3.
- [Bibliothèque cliente Kinesis : la bibliothèque cliente](#) Kinesis (KCL) fournit des mesures par partition, par utilisateur et par application. KCL
- [Kinesis Producer Library](#) : Kinesis Producer Library (KPL) fournit des métriques par partition, par utilisateur et par application. KPL

Pour obtenir plus d'informations sur les problèmes, les questions et le dépannage courants en matière de surveillance, consultez les ressources suivantes :

- [Quels indicateurs utiliser pour surveiller et résoudre les problèmes liés à Kinesis Data Streams ?](#)
- [Pourquoi la IteratorAgeMilliseconds valeur de Kinesis Data Streams continue-t-elle d'augmenter ?](#)

Surveillez le service Amazon Kinesis Data Streams avec Amazon CloudWatch

Amazon Kinesis Data Streams et CloudWatch Amazon sont intégrés afin que vous puissiez collecter, consulter et CloudWatch analyser les métriques de vos flux de données Kinesis. Par exemple, pour suivre l'utilisation de la partition, vous pouvez surveiller les métriques IncomingBytes et OutgoingBytes, et les comparer au nombre de partitions dans le flux.

Les métriques de stream et les métriques au niveau des partitions que vous configurez sont automatiquement collectées et transmises à CloudWatch chaque minute. Les métriques sont archivées pendant deux semaines ; après cette période, les données sont supprimées.

Le tableau suivant décrit la surveillance de base au niveau des flux et la surveillance avancée au niveau des partitions pour les flux de données Kinesis.

Type	Description
Basique (au niveau du flux)	Les données au niveau des flux sont envoyées automatiquement et gratuitement toutes les minutes.
Avancée (au niveau des partitions)	<p>Les données au niveau des flux sont envoyées toutes les minutes avec un coût additionnel. Pour obtenir ce niveau de données, vous devez l'activer spécifiquement pour le flux à l'aide de l'EnableEnhancedMonitoring opération.</p> <p>Pour plus d'informations sur les tarifs, consultez la page CloudWatch produit d'Amazon.</p>

Dimensions et statistiques d'Amazon Kinesis Data Streams

Kinesis Data Streams envoie des métriques CloudWatch à deux niveaux : le niveau du flux et, éventuellement, le niveau de la partition. Les métriques de niveau flux sont destinées aux cas d'utilisation de surveillance les plus courants dans des conditions normales. Les métriques au niveau des partitions concernent des tâches de surveillance spécifiques, généralement liées au dépannage, et sont activées à l'aide de l'[EnableEnhancedMonitoring](#) opération.

Pour une explication des statistiques collectées à partir des CloudWatch métriques, consultez la section [CloudWatch Statistiques](#) du guide de CloudWatch l'utilisateur Amazon.

Rubriques

- [Indicateurs de base au niveau du stream](#)
- [Indicateurs améliorés au niveau des partitions](#)
- [Dimensions des métriques Amazon Kinesis Data Streams](#)
- [Mesures Amazon Kinesis Data Streams recommandées](#)

Indicateurs de base au niveau du stream

L'espace de noms `AWS/Kinesis` inclut les métriques au niveau des flux suivantes.

Kinesis Data Streams envoie ces mesures CloudWatch au niveau du flux à chaque minute. Ces métriques sont toujours disponibles.

Métrique	Description
<code>GetRecords.Bytes</code>	<p>Nombre d'octets extraits du flux Kinesis mesurés pendant la période spécifiée. Les statistiques Minimum, Maximum et Moyenne représentent le nombre d'octets dans une même opération <code>GetRecords</code> pour le flux dans la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>OutgoingBytes</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : octets</p>
<code>GetRecords.IteratorAge</code>	<p>Cette métrique est obsolète. Utilisez <code>GetRecords.IteratorAgeMilliseconds</code>.</p>
<code>GetRecords.IteratorAgeMilliseconds</code>	<p>L'âge du dernier enregistrement parmi tous les appels <code>GetRecords</code> effectués sur un flux Kinesis, mesuré pendant la période spécifiée. L'âge est la différence entre l'heure actuelle et le moment où le dernier enregistrement de l'appel <code>GetRecords</code> a été écrit dans le flux. Les statistiques Minimum et Maximum peuvent être utilisées pour suivre l'avancement des applications consommateur Kinesis. Une valeur de zéro indique que les enregistrements en cours de lecture sont totalement absorbés par le flux.</p>

Métrique	Description
	<p>Nom de la métrique au niveau des partitions : <code>IteratorAgeMilliseconds</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Exemples</p> <p>Unités : millisecondes</p>
<code>GetRecords.Latency</code>	<p>Délai nécessaire par opération <code>GetRecords</code> , mesurée pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, moyenne</p> <p>Unités : millisecondes</p>
<code>GetRecords.Records</code>	<p>Nombre d'enregistrements extraits de la partition, mesurés pendant la période spécifiée. Les statistiques Minimum, Maximum et Moyenne représentent les enregistrements dans une même opération <code>GetRecords</code> pour le flux dans la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>OutgoingRecords</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>GetRecords.Success</code>	<p>Nombre d'opérations <code>GetRecords</code> réussies par flux mesurées pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>IncomingBytes</code>	<p>Nombre d'octets placés avec succès dans le flux Kinesis au cours de la période spécifiée. Cette métrique inclut le nombre d'octets des opérations <code>PutRecord</code> et <code>PutRecords</code>. Les statistiques <code>Minimum</code>, <code>Maximum</code> et <code>Moyenne</code> représentent le nombre d'octets dans une même opération <code>put</code> pour le flux dans la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>IncomingBytes</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : <code>Minimum</code>, <code>Maximum</code>, <code>Moyenne</code>, <code>Somme</code>, <code>Exemples</code></p> <p>Unités : octets</p>

Métrique	Description
IncomingRecords	<p>Nombre d'enregistrements placés avec succès dans le flux Kinesis au cours de la période spécifiée. Cette métrique inclut le nombre d'enregistrements des opérations <code>PutRecord</code> et <code>PutRecords</code>. Les statistiques Minimum, Maximum et Moyenne représentent les enregistrements dans une même opération put pour le flux dans la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>IncomingRecords</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
PutRecord.Bytes	<p>Nombre d'octets placés dans le flux Kinesis à l'aide de l'opération <code>PutRecord</code> pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : octets</p>
PutRecord.Latency	<p>Délai nécessaire par opération <code>PutRecord</code>, mesurée pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, moyenne</p> <p>Unités : millisecondes</p>

Métrique	Description
<code>PutRecord.Success</code>	<p>Nombre d'opérations <code>PutRecord</code> réussies par flux Kinesis mesurées pendant la période spécifiée. La moyenne représente le pourcentage d'opérations d'écriture réussies dans un flux.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>PutRecords.Bytes</code>	<p>Nombre d'octets placés dans le flux Kinesis à l'aide de l'opération <code>PutRecords</code> pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : octets</p>
<code>PutRecords.Latency</code>	<p>Délai nécessaire par opération <code>PutRecords</code>, mesurée pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, moyenne</p> <p>Unités : millisecondes</p>
<code>PutRecords.Records</code>	<p>Cette métrique est obsolète. Utilisez <code>PutRecords.SuccessfulRecords</code>.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>PutRecords.Success</code>	<p>Nombre d'opérations <code>PutRecords</code> où au moins un enregistrement a abouti, par flux Kinesis, mesurées pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>PutRecords.TotalRecords</code>	<p>Nombre total d'enregistrements envoyés à une opération <code>PutRecords</code> par flux de données Kinesis, mesuré sur la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>PutRecords.SuccessfulRecords</code>	<p>Nombre d'enregistrements réussis dans une opération <code>PutRecords</code> par flux de données Kinesis mesurés pendant la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>PutRecords.FailedRecords</code>	<p>Nombre d'enregistrements rejetés à cause de défaillances internes d'une opération <code>PutRecords</code> par flux de données Kinesis, mesuré sur la période spécifiée. Des défaillances internes occasionnelles peuvent survenir et doivent être réessayées.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>PutRecords.ThrottledRecords</code>	<p>Nombre d'enregistrements rejetés à cause de limitations dans une opération <code>PutRecords</code> par flux de données Kinesis, mesuré sur la période spécifiée.</p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>ReadProvisionedThroughputExceeded</code>	<p>Nombre d'appels <code>GetRecords</code> limités pour le flux au cours de la période spécifiée. La statistique la plus couramment utilisée pour cette métrique est Moyenne.</p> <p>Lorsque la statistique Minimum a une valeur de 1, tous les enregistrements ont été limités pour le flux au cours de la période spécifiée.</p> <p>Lorsque la statistique Maximum a une valeur de 0 (zéro), aucun enregistrement n'a été limité pour le flux au cours de la période spécifiée.</p> <p>Nom de la métrique au niveau des partitions : <code>ReadProvisionedThroughputExceeded</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>SubscribeToShardRateExceeded</code>	<p>Cette métrique est émise lorsqu'une nouvelle tentative d'abonnement échoue, car il existe déjà un abonnement actif par le même consommateur ou car vous dépassez le nombre d'appels par seconde autorisés pour cette opération.</p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p>
<code>SubscribeToShard.Success</code>	<p>Cette métrique indique si l'abonnement a été établi avec succès. L'abonnement n'est actif que pendant 5 minutes maximum. Par conséquent, cette métrique est émise au moins une fois toutes les 5 minutes.</p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p>

Métrique	Description
<code>SubscribeToShardEvent.Bytes</code>	<p>Nombre d'octets reçus de la partition, mesuré pendant la période spécifiée. Les statistiques Minimum, Maximum et Moyenne représentent les octets publiés dans un seul événement pour la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>OutgoingBytes</code></p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : octets</p>
<code>SubscribeToShardEvent.MillisBehindLatest</code>	<p>Le nombre de millisecondes pendant lesquelles les enregistrements lus se situent à partir de la fin du flux, indiquant à quel point le consommateur est en retard sur l'heure actuelle.</p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Exemples</p> <p>Unités : millisecondes</p>

Métrique	Description
<code>SubscribeToShardEvent.Records</code>	<p>Nombre d'enregistrements reçus de la partition, mesurés pendant la période spécifiée. Les statistiques Minimum, Maximum et Moyenne représentent les enregistrements dans un seul événement pour la période déterminée.</p> <p>Nom de la métrique au niveau des partitions : <code>OutgoingRecords</code></p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>
<code>SubscribeToShardEvent.Success</code>	<p>Cette métrique est émise chaque fois qu'un événement est publié avec succès. Elle est émise uniquement lorsqu'il existe un abonnement actif.</p> <p>Dimensions : <code>StreamName</code>, <code>ConsumerName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>WriteProvisionedThroughputExceeded</code>	<p>Nombre d'enregistrements rejetés à cause de la limitation du flux au cours de la période spécifiée. Cette métrique inclut la limitation des opérations <code>PutRecord</code> et <code>PutRecords</code>. La statistique la plus couramment utilisée pour cette métrique est Moyenne.</p> <p>Lorsque la statistique Minimum a une valeur différente de zéro, les enregistrements ont été limités pour le flux au cours de la période spécifiée.</p> <p>Lorsque la statistique Maximum a une valeur de 0 (zéro), aucun enregistrement n'a été limité pour le flux au cours de la période spécifiée.</p> <p>Nom de la métrique au niveau des partitions : <code>WriteProvisionedThroughputExceeded</code></p> <p>Dimensions : <code>StreamName</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Indicateurs améliorés au niveau des partitions

L'espace de noms `AWS/Kinesis` inclut les métriques suivantes au niveau des partitions.

Kinesis envoie les mesures suivantes au niveau des partitions par minute. CloudWatch Chaque dimension métrique crée une CloudWatch métrique et effectue environ 43 200 `PutMetricData` API appels par mois. Ces métriques ne sont pas activées par défaut. Des frais sont associés aux métriques améliorées émises depuis Kinesis. Pour plus d'informations, consultez [Amazon CloudWatch Pricing](#) sous la rubrique Amazon CloudWatch Custom Metrics. Les frais sont fournis par partition par métrique par mois.

Métrique	Description
IncomingBytes	<p>Nombre d'octets placés avec succès dans la partition au cours de la période spécifiée. Cette métrique inclut le nombre d'octets des opérations <code>PutRecord</code> et <code>PutRecords</code> . Les statistiques Minimum, Maximum et Moyenne représentent le nombre d'octets dans une même opération put pour la partition dans la période déterminée.</p> <p>Nom de la métrique au niveau des flux : <code>IncomingBytes</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : octets</p>
IncomingRecords	<p>Nombre d'enregistrements placés avec succès dans la partition au cours de la période spécifiée. Cette métrique inclut le nombre d'enregistrements des opérations <code>PutRecord</code> et <code>PutRecords</code> . Les statistiques Minimum, Maximum et Moyenne représentent les enregistrements dans une même opération put pour la partition dans la période déterminée.</p> <p>Nom de la métrique au niveau des flux : <code>IncomingRecords</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>IteratorAgeMilliseconds</code>	<p>Age du dernier enregistrement parmi tous les appels <code>GetRecords</code> effectués sur une partition, mesuré pendant la période spécifiée. L'âge est la différence entre l'heure actuelle et le moment où le dernier enregistrement de l'appel <code>GetRecords</code> a été écrit dans le flux. Les statistiques <code>Minimum</code> et <code>Maximum</code> peuvent être utilisées pour suivre l'avancement des applications consommateur Kinesis. Une valeur de 0 (zéro) indique que les enregistrements en cours de lecture sont totalement absorbés par le flux.</p> <p>Nom de la métrique au niveau des flux : <code>GetRecords.IteratorAgeMilliseconds</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : <code>Minimum</code>, <code>Maximum</code>, <code>Moyenne</code>, <code>Exemples</code></p> <p>Unités : millisecondes</p>
<code>OutgoingBytes</code>	<p>Nombre d'octets extraits de la partition, mesuré pendant la période spécifiée. Les statistiques <code>Minimum</code>, <code>Maximum</code> et <code>Moyenne</code> représentent le nombre d'octets renvoyés dans une même opération <code>GetRecords</code> ou publiés dans un seul événement <code>SubscribeToShard</code> pour la partition dans la période déterminée.</p> <p>Nom de la métrique au niveau des flux : <code>GetRecords.Bytes</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : <code>Minimum</code>, <code>Maximum</code>, <code>Moyenne</code>, <code>Somme</code>, <code>Exemples</code></p> <p>Unités : octets</p>

Métrique	Description
OutgoingRecords	<p>Nombre d'enregistrements extraits de la partition, mesurés pendant la période spécifiée. Les statistiques Minimum, Maximum et Moyenne représentent le nombre d'enregistrements renvoyés dans une même opération <code>GetRecords</code> ou publiés dans un seul événement <code>SubscribeToShard</code> pour la partition dans la période déterminée.</p> <p>Nom de la métrique au niveau des flux : <code>GetRecords.Records</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
ReadProvisionedThroughputExceeded	<p>Nombre d'appels GetRecords limités pour la partition au cours de la période spécifiée. Ce nombre d'exceptions couvre toutes les dimensions des limites suivantes : 5 lectures par partition par seconde ou 2 Mo par seconde par partition. La statistique la plus couramment utilisée pour cette métrique est Moyenne.</p> <p>Lorsque la statistique Minimum a une valeur de 1, tous les enregistrements ont été limités pour la partition au cours de la période spécifiée.</p> <p>Lorsque la statistique Maximum a une valeur de 0 (zéro), aucun enregistrement n'a été limité pour la partition au cours de la période spécifiée.</p> <p>Nom de la métrique au niveau des flux : ReadProvisionedThroughputExceeded</p> <p>Dimensions : StreamName, ShardId</p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Métrique	Description
<code>WriteProvisionedThroughputExceeded</code>	<p>Nombre d'enregistrements rejetés à cause de la limitation de la partition au cours de la période spécifiée. Cette métrique inclut la limitation des opérations <code>PutRecord</code> et <code>PutRecords</code> et couvre toutes les dimensions des limites suivantes : 1 000 enregistrements par seconde par partition ou 1 Mo par seconde par partition. La statistique la plus couramment utilisée pour cette métrique est Moyenne.</p> <p>Lorsque la statistique Minimum a une valeur différente de zéro, les enregistrements ont été limités pour la partition au cours de la période spécifiée.</p> <p>Lorsque la statistique Maximum a une valeur de 0 (zéro), aucun enregistrement n'a été limité pour la partition au cours de la période spécifiée.</p> <p>Nom de la métrique au niveau des flux : <code>WriteProvisionedThroughputExceeded</code></p> <p>Dimensions : <code>StreamName</code>, <code>ShardId</code></p> <p>Statistiques : Minimum, Maximum, Moyenne, Somme, Exemples</p> <p>Unités : nombre</p>

Dimensions des métriques Amazon Kinesis Data Streams

Dimension	Description
<code>StreamName</code>	Nom du flux Kinesis. Toutes les statistiques disponibles sont filtrées par <code>StreamName</code> .

Mesures Amazon Kinesis Data Streams recommandées

Certains indicateurs Amazon Kinesis Data Streams peuvent particulièrement intéresser les clients de Kinesis Data Streams. La liste suivante contient les métriques recommandées et leurs utilisations.

Métrique	Notes d'utilisation
<code>GetRecords.IteratorAgeMilliseconds</code>	Suit la position de lecture sur toutes les partitions et applications consommateur du flux. Si l'ancienneté de l'itérateur dépasse 50 % de la période de conservation (par défaut 24 heures, configurable jusqu'à 7 jours), il y a un risque de perte de données suite à l'expiration des enregistrements. Nous vous recommandons d'utiliser des CloudWatch alarmes sur la statistique Maximum pour vous avertir avant que cette perte ne constitue un risque. Pour obtenir un exemple de scénario qui utilise cette métrique, consultez la page Le traitement des dossiers des consommateurs prend du retard .
<code>ReadProvisionedThroughputExceeded</code>	Lorsque le traitement de votre enregistrement côté consommateur est en retard, il est parfois difficile de savoir où se situe le goulot d'étranglement. Utilisez cette métrique pour déterminer si vos lectures sont limitées suite au dépassement de vos limites de débit de lecture. La statistique la plus couramment utilisée pour cette métrique est Moyenne.
<code>WriteProvisionedThroughputExceeded</code>	Elle a le même objectif que la métrique <code>ReadProvisionedThroughputExceeded</code> mais du côté producteur (put) du flux. La statistique la plus couramment utilisée pour cette métrique est Moyenne.
<code>PutRecords.Success</code> , <code>PutRecords.Success</code>	Nous vous recommandons d'utiliser des CloudWatch alarmes sur la statistique Average pour indiquer quand les enregistrements ne parviennent pas au flux. Sélectionnez un type put ou les deux en fonction de ce que votre producteur utilise. Si vous utilisez la bibliothèque Kinesis Producer (KPL), utilisez <code>PutRecords.Success</code> .
<code>GetRecords.Success</code>	Nous vous recommandons d'utiliser des CloudWatch alarmes sur la statistique Average pour indiquer quand les enregistrements du flux échouent.

Accédez aux CloudWatch métriques Amazon pour Kinesis Data Streams

Vous pouvez surveiller les métriques de Kinesis Data Streams à l'aide de CloudWatch la console, de la ligne de commande ou CloudWatch API du. Les procédures suivantes vous montrent comment accéder aux métriques à l'aide de ces différentes méthodes.

Pour accéder aux métriques à l'aide de la CloudWatch console

1. Ouvrez la CloudWatch console à l'adresse <https://console.aws.amazon.com/cloudwatch/>.
2. Sélectionnez une région dans la barre de navigation.
3. Dans le panneau de navigation, sélectionnez Métriques.
4. Dans le volet CloudWatch Metrics by Category, sélectionnez Kinesis Metrics.
5. Cliquez sur la ligne correspondante pour afficher les statistiques pour les valeurs spécifiées MetricName et StreamName.

Remarque : La plupart des noms de statistiques de console correspondent aux noms de CloudWatch métriques correspondants répertoriés ci-dessus, à l'exception du débit de lecture et du débit d'écriture. Ces statistiques sont calculées sur des intervalles de 5 minutes : le débit d'écriture surveille la IncomingBytes CloudWatch métrique et le débit de lecture surveille le débit de lecture. `GetRecords.Bytes`

6. (Facultatif) Dans le volet graphique, sélectionnez une statistique et une période, puis créez une CloudWatch alarme à l'aide de ces paramètres.

Pour accéder aux métriques à l'aide du AWS CLI

Utilisez les [métriques et get-metric-statistics](#) les commandes de liste.

Pour accéder aux métriques à l'aide du CloudWatch CLI

Utilisez les [mon-get-stats](#) commandes [mon-list-metric](#) set.

Pour accéder aux métriques à l'aide du CloudWatch API

Utilisez les [GetMetricStatistics](#) opérations [ListMetric](#) set.

Surveillez l'état de santé des agents Kinesis Data Streams avec Amazon CloudWatch

L'agent publie des CloudWatch métriques personnalisées avec un espace de noms de AWS KinesisAgent. Ces indicateurs vous aident à déterminer si l'agent envoie des données à Kinesis Data Streams conformément aux spécifications, si elles sont saines et si elles consomment la quantité et les ressources de mémoire appropriées CPU de la part du producteur de données. Les métriques comme le nombre d'enregistrements et d'octets envoyés sont très utiles pour comprendre la vitesse à laquelle l'agent soumet des données au flux. Lorsque ces métriques deviennent inférieures d'un certain pourcentage aux seuils attendus ou nulles, elles peuvent traduire des problèmes de configuration, des erreurs réseau ou des problèmes d'état de l'agent. Des indicateurs tels que la consommation sur l'hôte CPU et la consommation de mémoire et les compteurs d'erreurs des agents indiquent l'utilisation des ressources par le producteur de données et fournissent des informations sur les erreurs de configuration ou d'hôte potentielles. Enfin, l'agent consigne également les exceptions au niveau du service pour mieux examiner les problèmes de l'agent. Ces métriques sont présentées dans la région spécifiée dans le paramètre de configuration d'agent `cloudwatch.endpoint`. Les métriques Cloudwatch qui sont publiées par plusieurs agents Kinesis sont agrégées ou combinées. Pour en savoir plus sur la configuration de l'agent, consultez [Spécifiez les paramètres de configuration de l'agent](#).

Moniteur avec CloudWatch

L'agent Kinesis Data Streams envoie les métriques CloudWatch suivantes à.

Métrique	Description
BytesSent	Nombre d'octets envoyés à Kinesis Data Streams au cours de la période spécifiée. Unités : octets
RecordSendAttempts	Nombre d'enregistrements tentés (une première fois ou lors d'une nouvelle tentative) dans un appel de <code>PutRecords</code> au cours de la période spécifiée. Unités : nombre

Métrique	Description
RecordSendErrors	Nombre d'enregistrements qui ont renvoyé un état d'échec dans un appel de <code>PutRecords</code> , y compris les nouvelles tentatives, au cours de la période spécifiée. Unités : nombre
ServiceErrors	Nombre d'appels de <code>PutRecords</code> qui ont provoqué une erreur de service (autre qu'une erreur de limitation) au cours de la période spécifiée. Unités : nombre

Enregistrez les appels Amazon Kinesis Data API Streams avec AWS CloudTrail

Amazon Kinesis Data Streams est intégré à AWS CloudTrail, un service qui fournit un enregistrement des actions effectuées par un utilisateur, un rôle ou AWS un service dans Kinesis Data Streams. CloudTrail capture tous les API appels pour Kinesis Data Streams sous forme d'événements. Les appels capturés incluent des appels provenant de la console Kinesis Data Streams et des appels de code vers les opérations Kinesis Data Streams. API Si vous créez un suivi, vous pouvez activer la diffusion continue d'événements CloudTrail vers un compartiment Amazon S3, y compris des événements pour Kinesis Data Streams. 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 Kinesis Data Streams, l'adresse IP à partir de laquelle la demande a été effectuée, l'auteur de la demande, la date à laquelle elle a été faite, ainsi que des informations supplémentaires.

Pour en savoir plus CloudTrail, notamment comment le configurer et l'activer, consultez le [guide de AWS CloudTrail l'utilisateur](#).

Informations sur Kinesis Data Streams dans CloudTrail

CloudTrail est activé sur votre AWS compte lorsque vous le créez. Lorsqu'une activité événementielle prise en charge se produit dans Kinesis Data Streams, cette activité est enregistrée dans CloudTrail un événement avec AWS d'autres événements de service dans l'historique des événements. Vous

pouvez consulter, rechercher et télécharger les événements récents dans votre AWS compte. Pour plus d'informations, consultez la section [Affichage des événements à l'aide de l'historique des CloudTrail événements](#).

Pour un enregistrement continu des événements de votre AWS compte, y compris des événements liés à Kinesis Data Streams, créez une trace. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un parcours dans la console, celui-ci s'applique à toutes les AWS régions. 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 configurer d'autres AWS services 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 SNS notifications Amazon pour CloudTrail](#)
- [Réception de fichiers CloudTrail journaux de plusieurs régions](#) et [réception de fichiers CloudTrail journaux de plusieurs comptes](#)

Kinesis Data Streams prend en charge l'enregistrement des actions suivantes sous forme d'événements CloudTrail dans des fichiers journaux :

- [AddTagsToStream](#)
- [CreateStream](#)
- [DecreaseStreamRetentionPeriod](#)
- [DeleteStream](#)
- [DeregisterStreamConsumer](#)
- [DescribeStream](#)
- [DescribeStreamConsumer](#)
- [DisableEnhancedMonitoring](#)
- [EnableEnhancedMonitoring](#)
- [GetRecords](#)
- [GetShardIterator](#)
- [IncreaseStreamRetentionPeriod](#)

- [ListStreamConsumers](#)
- [ListStreams](#)
- [ListTagsForStream](#)
- [MergeShards](#)
- [PutRecord](#)
- [PutRecords](#)
- [RegisterStreamConsumer](#)
- [RemoveTagsFromStream](#)
- [SplitShard](#)
- [StartStreamEncryption](#)
- [StopStreamEncryption](#)
- [SubscribeToShard](#)
- [UpdateShardCount](#)
- [UpdateStreamMode](#)

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 faite avec les informations d'identification de l'utilisateur root ou AWS Identity and Access Management (IAM).
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Pour plus d'informations, consultez l'[CloudTrail userIdentity élément](#).

Exemple : entrées du fichier journal Kinesis Data Streams

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 comprend 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 montre une entrée de CloudTrail journal qui illustre les MergeShards actions CreateStream DescribeStream ListStreams, DeleteStream, SplitShard,, et.

```
{
  "Records": [
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-04-19T00:16:31Z",
      "eventSource": "kinesis.amazonaws.com",
      "eventName": "CreateStream",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "127.0.0.1",
      "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
      "requestParameters": {
        "shardCount": 1,
        "streamName": "GoodStream"
      },
      "responseElements": null,
      "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
      "eventID": "b7acfc0-6ca9-4ee1-a3d7-c4e8d420d99b"
    },
    {
      "eventVersion": "1.01",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
      },
      "eventTime": "2014-04-19T00:17:06Z",
```

```
"eventSource": "kinesis.amazonaws.com",
"eventName": "DescribeStream",
"awsRegion": "us-east-1",
"sourceIPAddress": "127.0.0.1",
"userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
"requestParameters": {
  "streamName": "GoodStream"
},
"responseElements": null,
"requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
"eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
},
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::012345678910:user/Alice",
    "accountId": "012345678910",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  },
  "eventTime": "2014-04-19T00:15:02Z",
  "eventSource": "kinesis.amazonaws.com",
  "eventName": "ListStreams",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
  "requestParameters": {
    "limit": 10
  },
  "responseElements": null,
  "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
  "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
},
{
  "eventVersion": "1.01",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::012345678910:user/Alice",
    "accountId": "012345678910",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  }
```

```

    },
    "eventTime": "2014-04-19T00:17:07Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "DeleteStream",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
      "streamName": "GoodStream"
    },
    "responseElements": null,
    "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
  },
  {
    "eventVersion": "1.01",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::012345678910:user/Alice",
      "accountId": "012345678910",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:15:03Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "SplitShard",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
      "shardToSplit": "shardId-000000000000",
      "streamName": "GoodStream",
      "newStartingHashKey": "11111111"
    },
    "responseElements": null,
    "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
    "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
  },
  {
    "eventVersion": "1.01",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",

```

```
        "arn": "arn:aws:iam::012345678910:user/Alice",
        "accountId": "012345678910",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice"
    },
    "eventTime": "2014-04-19T00:16:56Z",
    "eventSource": "kinesis.amazonaws.com",
    "eventName": "MergeShards",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
    "requestParameters": {
        "streamName": "GoodStream",
        "adjacentShardToMerge": "shardId-000000000002",
        "shardToMerge": "shardId-000000000001"
    },
    "responseElements": null,
    "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
    "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
}
]
```

Surveillez la bibliothèque cliente Kinesis avec Amazon CloudWatch

La [bibliothèque client Kinesis](#) (KCL) pour Amazon Kinesis Data Streams publie des CloudWatch métriques Amazon personnalisées en votre nom, en utilisant le nom de votre KCL application comme espace de noms. Vous pouvez consulter ces statistiques en accédant à la [CloudWatch console](#) et en choisissant Mesures personnalisées. Pour plus d'informations sur les métriques personnalisées, consultez [Publier des métriques personnalisées](#) dans le guide de CloudWatch l'utilisateur Amazon.

Des frais nominaux sont facturés pour les statistiques téléchargées CloudWatch par le KCL ; en particulier, les frais Amazon CloudWatch Custom Metrics et Amazon CloudWatch API Requests s'appliquent. Pour plus d'informations, consultez [Amazon CloudWatch Pricing](#).

Rubriques

- [Métriques et espace de noms](#)
- [Niveaux métriques et dimensions](#)
- [Configuration métrique](#)
- [Liste des métriques](#)

Métriques et espace de noms

L'espace de noms utilisé pour télécharger les métriques est le nom de l'application que vous spécifiez lorsque vous lancez leKCL.

Niveaux métriques et dimensions

Il existe deux options pour contrôler les métriques à télécharger CloudWatch :

niveaux de métriques

Chaque métrique se voit attribuer un niveau individuel. Lorsque vous définissez un niveau de rapport de mesures, les mesures dont le niveau individuel est inférieur au niveau de rapport ne sont pas envoyées à CloudWatch. Les niveaux sont : NONE, SUMMARY et DETAILED. Le paramètre par défaut estDETAILED, c'est-à-dire que toutes les métriques sont envoyées à CloudWatch. Un niveau de création de rapport NONE signifie qu'aucune métrique n'est envoyée. Pour plus d'informations sur les niveaux attribués aux métriques, consultez la page [Liste des métriques](#).

dimensions activées

Chaque KCL métrique est associée à des dimensions qui sont également envoyées à CloudWatch. Dans la KCL version 2.x, s'KCLil est configuré pour traiter un seul flux de données, toutes les dimensions des métriques (Operation,ShardId, etWorkerIdentifier) sont activées par défaut. De plus, dans la KCL version 2.x, si elle KCL est configurée pour traiter un seul flux de données, la Operation dimension ne peut pas être désactivée. Dans la KCL version 2.x, si elle KCL est configurée pour traiter plusieurs flux de données, toutes les dimensions des métriques (OperationShardId,StreamId, etWorkerIdentifier) sont activées par défaut. De plus, dans la KCL version 2.x, si elle KCL est configurée pour traiter plusieurs flux de données, les StreamId dimensions Operation et ne peuvent pas être désactivées. StreamIdla dimension n'est disponible que pour les métriques par partition.

Dans la KCL version 1.x, seules les ShardId dimensions Operation et sont activées par défaut, et la WorkerIdentifier dimension est désactivée. Dans la KCL version 1.x, la Operation dimension ne peut pas être désactivée.

Pour plus d'informations sur les dimensions CloudWatch métriques, consultez la section [Dimensions](#) de la rubrique Amazon CloudWatch Concepts, dans le guide de CloudWatch l'utilisateur Amazon.

Lorsque la `WorkerIdentifier` dimension est activée, si une valeur différente est utilisée pour la propriété `Worker ID` chaque fois qu'un KCL worker spécifique redémarre, de nouveaux ensembles de métriques avec de nouvelles valeurs de `WorkerIdentifier` dimension sont envoyés à CloudWatch. Si vous avez besoin que la valeur de `WorkerIdentifier` dimension soit la même lors des redémarrages de KCL travailleurs spécifiques, vous devez spécifier explicitement la même valeur d'ID de travail lors de l'initialisation pour chaque travailleur. Notez que la valeur de l'identifiant de chaque KCL travailleur actif doit être unique pour tous les KCL travailleurs.

Configuration métrique

Les niveaux métriques et les dimensions activées peuvent être configurés à l'aide de l'`KinesisClientLibConfiguration` instance, qui est transmise à `Worker` lors du lancement de l'`KCLApplication`. `MultiLangDaemon` Dans ce cas, les `metricsEnabledDimensions` propriétés `metricsLevel` et peuvent être spécifiées dans le fichier `.properties` utilisé pour lancer l'`MultiLangDaemon KCLApplication`.

Les niveaux métriques peuvent se voir attribuer l'une des trois valeurs suivantes : `NONESUMMARY`, `ouDETAILED`. Les valeurs de dimensions activées doivent être des chaînes séparées par des virgules avec la liste des dimensions autorisées pour les CloudWatch métriques. Les dimensions utilisées par l'`KCLApplication` sont `OperationShardId`, et `WorkerIdentifier`.

Liste des métriques

Les tableaux suivants répertorient les KCL métriques, regroupées par étendue et par opération.

Rubriques

- [Métriques par KCL application](#)
- [Indicateurs par travailleur](#)
- [Métriques par partition](#)

Métriques par KCL application

Ces métriques sont agrégées pour tous les KCL collaborateurs concernés par l'application, tels que définis par l'espace de CloudWatch noms Amazon.

Rubriques

- [InitializeTask](#)

- [ShutdownTask](#)
- [ShardSyncTask](#)
- [BlockOnParentTask](#)
- [PeriodicShardSyncManager](#)
- [MultistreamTracker](#)

InitializeTask

L'`InitializeTask` opération est responsable de l'initialisation du processeur d'enregistrement pour l'`KCL` application. La logique de cette opération implique d'extraire un itérateur de partition de Kinesis Data Streams et d'initialiser le processeur d'enregistrements.

Métrique	Description
<code>KinesisDataFetcher.getIterator.Succès</code>	<p>Nombre d'<code>GetShardIterator</code> opérations réussies par <code>KCL</code> application.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
<code>KinesisDataFetcher.getIterator.Heure</code>	<p>Temps nécessaire à chaque <code>GetShardIterator</code> opération pour l'<code>KCL</code> application donnée.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>
<code>RecordProcessor.Initialiser.time</code>	<p>Délai nécessaire pour la méthode d'initialisation du processeur d'enregistrements.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>
Réussite	<p>Nombre d'initialisations de processeur d'enregistrements réussies.</p> <p>Niveau de métrique : Summary</p>

Métrique	Description
	Unités : nombre
Heure	<p>Temps nécessaire au KCL travailleur pour l'initialisation du processeur d'enregistrement.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>

ShutdownTask

L'opération ShutdownTask initie la séquence de fermeture pour le traitement des partitions. Elle peut avoir lieu lorsqu'une partition est fractionnée ou fusionnée ou que le bail de la partition est perdu par l'application de travail. Dans les deux cas, la fonction shutdown() du processeur d'enregistrements est appelée. De nouvelles partitions sont également détectées si une partition a été fractionnée ou fusionnée, ce qui entraîne la création d'une ou de deux nouvelles partitions.

Métrique	Description
CreateLease. Succès	<p>Nombre de fois où de nouvelles partitions enfants sont ajoutées avec succès dans la table DynamoDB de l'KCLapplication après la fermeture de la partition parent.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
CreateLease.Heure	<p>Temps nécessaire pour ajouter de nouvelles informations de partition enfant dans la table DynamoDB de l'KCLapplication.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>
UpdateLease. Succès	<p>Nombre de points de contrôle finaux réussis au cours de l'arrêt du processeur d'enregistrements.</p> <p>Niveau de métrique : Detailed</p>

Métrique	Description
	Unités : nombre
UpdateLease.Heure	Délai nécessaire pour l'opération de point de contrôle au cours de l'arrêt du processeur d'enregistrements. Niveau de métrique : Detailed Unités : millisecondes
RecordProcessor.Heure d'arrêt	Délai nécessaire pour la méthode de fermeture du processeur d'enregistrements. Niveau de métrique : Summary Unités : millisecondes
Réussite	Nombre de tâches de fermeture réussies. Niveau de métrique : Summary Unités : nombre
Heure	Temps consacré par le KCL travailleur à la tâche d'arrêt. Niveau de métrique : Summary Unités : millisecondes

ShardSyncTask

L'opération `ShardSyncTask` découvre les modifications apportées aux informations relatives aux partitions pour le flux de données Kinesis, afin que les nouvelles partitions puissent être traitées par l'application. KCL

Métrique	Description
CreateLease. Succès	Nombre de tentatives réussies pour ajouter de nouvelles informations de partition dans la table DynamoDB de l'application KCL.

Métrique	Description
	Niveau de métrique : Detailed Unités : nombre
CreateLease.Heure	Temps nécessaire pour ajouter de nouvelles informations de partition dans la table DynamoDB de l'KCLapplication. Niveau de métrique : Detailed Unités : millisecondes
Réussite	Nombre d'opérations de synchronisation de partitions réussies. Niveau de métrique : Summary Unités : nombre
Heure	Délai nécessaire pour l'opération de synchronisation des partitions. Niveau de métrique : Summary Unités : millisecondes

BlockOnParentTask

Si la partition est fractionnée ou fusionnée avec d'autres partitions, de nouvelles partitions enfant sont créées. L'BlockOnParentTaskopération garantit que le traitement des enregistrements pour les nouvelles partitions ne démarre pas tant que les partitions parents ne sont pas complètement traitées par le KCL

Métrique	Description
Réussite	Nombre de contrôles réussis pour l'achèvement de la partition parent. Niveau de métrique : Summary Unités : nombre
Heure	Temps nécessaire à la réalisation de partitions parent.

Métrique	Description
	Niveau de métrique : Summary Unité : millisecondes

PeriodicShardSyncManager

`PeriodicShardSyncManager` Il est chargé d'examiner les flux de données traités par l'application KCL client, d'identifier les flux de données faisant l'objet de baux partiels et de les transférer pour synchronisation.

Les métriques suivantes sont disponibles lorsqu'il KCL est configuré pour traiter un seul flux de données (alors la valeur de `NumStreamsToSync` et `NumStreamsWithPartialLeases` est définie sur 1) et également lorsqu'il KCL est configuré pour traiter plusieurs flux de données.

Métrique	Description
<code>NumStreamsToSync</code>	Nombre de flux de données (par AWS compte) traités par l'application client qui contiennent des baux partiels et qui doivent être transférés pour synchronisation. Niveau de métrique : Summary Unités : nombre
<code>NumStreamsWithPartialLeases</code>	Nombre de flux de données (par AWS compte) traités par l'application client qui contiennent des baux partiels. Niveau de métrique : Summary Unités : nombre
Réussite	Nombre de fois où <code>PeriodicShardSyncManager</code> a réussi à identifier des baux partiels dans les flux de données traités par l'application client. Niveau de métrique : Summary Unités : nombre

Métrique	Description
Heure	<p>Temps (en millisecondes) nécessaire à <code>PeriodicShardSyncManager</code> pour examiner les flux de données traités par l'application client afin de déterminer quels flux de données nécessitent une synchronisation des partitions.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>

MultistreamTracker

L'`MultistreamTracker` interface vous permet de créer KCL des applications grand public capables de traiter plusieurs flux de données en même temps.

Métrique	Description
<code>DeletedStreams</code> . Compter	<p>Nombre de flux de données supprimés au cours de cette période.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>
<code>ActiveStreams</code> . Compter	<p>Nombre de flux de données actifs en cours de traitement.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>
<code>StreamsPendingDeletion</code> . Compter	<p>Nombre de flux de données en attente de suppression en fonction de <code>FormerStreamsLeasesDeletionStrategy</code>.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>

Indicateurs par travailleur

Ces mesures sont agrégées entre tous les processeurs d'enregistrements consommant des données issues d'un flux de données Kinesis, tel qu'une instance AmazonEC2.

Rubriques

- [RenewAllLeases](#)
- [TakeLeases](#)

RenewAllLeases

L'opération `RenewAllLeases` renouvelle périodiquement les baux de partition appartenant à une instance d'application de travail spécifique.

Métrique	Description
RenewLease.Succès	Nombre de renouvellements de bail réussis par application de travail. Niveau de métrique : Detailed Unités : nombre
RenewLease.Heure	Délai nécessaire pour l'opération de renouvellement de bail. Niveau de métrique : Detailed Unités : millisecondes
CurrentLeases	Nombre de baux de partition appartenant à l'application de travail après le renouvellement de tous les baux. Niveau de métrique : Summary Unités : nombre
LostLeases	Nombre de baux de partition qui ont été perdus suite à une tentative de renouvellement de tous les baux appartenant à l'application de travail. Niveau de métrique : Summary

Métrique	Description
	Unités : nombre
Réussite	<p>Nombre de fois que l'opération de renouvellement des baux a abouti pour l'application de travail.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>
Heure	<p>Délai nécessaire pour le renouvellement de tous les baux d'une application de travail.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>

TakeLeases

L'opération `TakeLeases` équilibre le traitement des dossiers entre tous les KCL travailleurs. Si le KCL travailleur actuel dispose d'un nombre de baux de partage inférieur à ce qui est requis, il prend les baux de partage d'un autre travailleur surchargé.

Métrique	Description
ListLeases.Succès	<p>Nombre de fois où tous les baux de partitions ont été correctement extraits de la table KCL DynamoDB de l'application.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
ListLeases.Heure	<p>Temps nécessaire pour récupérer tous les baux de partitions depuis la table KCL DynamoDB de l'application.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>

Métrique	Description
TakeLease.Succès	<p>Nombre de fois où le travailleur a obtenu avec succès des contrats de location collective avec d'autres KCL travailleurs.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
TakeLease.Heure	<p>Délai nécessaire pour la mise à jour de la table des baux avec les baux extraits par l'application de travail.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>
NumWorkers	<p>Nombre total d'applications de travail identifiées par une application de travail spécifique.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>
NeededLeases	<p>Nombre de baux de partition dont l'application de travail actuelle a besoin pour que la charge de traitement des partitions soit équilibrée.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
LeasesToTake	<p>Nombre de baux que l'application de travail va tenter de prendre.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
TakenLeases	<p>Nombre de baux pris avec succès par l'application de travail.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>

Métrique	Description
TotalLeases	Nombre total de partitions traitées par l'KCLapplication. Niveau de métrique : Detailed Unités : nombre
ExpiredLeases	Nombre total de partitions qui ne sont pas traitées par une application de travail, comme identifié par l'application de travail spécifique. Niveau de métrique : Summary Unités : nombre
Réussite	Nombre de fois que l'opération TakeLeases a été effectuée avec succès. Niveau de métrique : Summary Unités : nombre
Heure	Délai nécessaire pour l'opération TakeLeases effectuée par une application de travail. Niveau de métrique : Summary Unités : millisecondes

Métriques par partition

Ces métriques sont regroupées sur un processeur d'enregistrements unique.

ProcessTask

L'opération `ProcessTask` fait appel [GetRecords](#) à la position actuelle de l'itérateur pour récupérer les enregistrements du flux et invoque la fonction du processeur `processRecords` d'enregistrements.

Métrique	Description
KinesisDataFetcher.getRecords.Succès	<p>Nombre d'opérations <code>GetRecords</code> réussies par partition de flux de données Kinesis.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
KinesisDataFetcher.getRecords.Heure	<p>Délai nécessaire par opération <code>GetRecords</code> pour la partition de flux de données Kinesis.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>
UpdateLease.Succès	<p>Nombre de points de contrôle réussis effectués par le processeur d'enregistrements pour la partition donnée.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : nombre</p>
UpdateLease.Heure	<p>Délai nécessaire pour chaque opération de point de contrôle pour la partition donnée.</p> <p>Niveau de métrique : Detailed</p> <p>Unités : millisecondes</p>
DataBytesProcessed	<p>Taille totale des enregistrements traités, en octets, à chaque invocation de <code>ProcessTask</code> .</p> <p>Niveau de métrique : Summary</p> <p>Unités : octet</p>
RecordsProcessed	<p>Nombre d'enregistrements traités à chaque invocation de <code>ProcessTask</code> .</p> <p>Niveau de métrique : Summary</p>

Métrique	Description
	Unités : nombre
ExpiredIterator	<p>Numéro ExpiredIteratorException reçu lors de l'appel <code>GetRecords</code> .</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>
MillisBehindLatest	<p>Retard pris par l'itérateur actuel depuis le dernier enregistrement (extrémité) de la partition. Cette valeur est inférieure ou égale à la différence de temps entre le dernier enregistrement figurant dans une réponse et l'heure actuelle. Cette méthode donne des résultats plus précis pour connaître la distance parcourue depuis le début d'une partition que celle qui consiste à comparer les horodatages dans le dernier enregistrement de réponse. Cette valeur s'applique au dernier lot d'enregistrements et pas à une moyenne de tous les horodatages de chaque enregistrement.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>
RecordProcessor. processRecords.Heure	<p>Délai nécessaire pour la méthode <code>processRecords</code> du processeur d'enregistrements.</p> <p>Niveau de métrique : Summary</p> <p>Unités : millisecondes</p>
Réussite	<p>Nombre d'opérations de traitement réussies.</p> <p>Niveau de métrique : Summary</p> <p>Unités : nombre</p>

Métrique	Description
Heure	Délai nécessaire pour l'opération de traitement. Niveau de métrique : Summary Unités : millisecondes

Surveillez la bibliothèque Kinesis Producer avec Amazon CloudWatch

La [Kinesis Producer Library](#) (KPL) pour Amazon Kinesis Data Streams publie des CloudWatch métriques Amazon personnalisées en votre nom. Vous pouvez consulter ces statistiques en accédant à la [CloudWatch console](#) et en choisissant Mesures personnalisées. Pour plus d'informations sur les métriques personnalisées, consultez [Publier des métriques personnalisées](#) dans le guide de CloudWatch l'utilisateur Amazon.

Des frais nominaux sont facturés pour les statistiques téléchargées CloudWatch par le KPL ; en particulier, les frais Amazon CloudWatch Custom Metrics et Amazon CloudWatch API Requests s'appliquent. Pour plus d'informations, consultez [Amazon CloudWatch Pricing](#). La collecte de statistiques locales n'entraîne aucun CloudWatch frais.

Rubriques

- [Métriques, dimensions et espaces de noms](#)
- [Niveau métrique et granularité](#)
- [Accès local et CloudWatch téléchargement sur Amazon](#)
- [Liste des métriques](#)

Métriques, dimensions et espaces de noms

Vous pouvez spécifier un nom d'application lors du lancement du KPL, qui est ensuite utilisé dans le cadre de l'espace de noms lors du téléchargement des métriques. Ceci est facultatif ; KPL fournit une valeur par défaut si aucun nom d'application n'est défini.

Vous pouvez également configurer le KPL pour ajouter des dimensions supplémentaires arbitraires aux métriques. Cela est utile si vous souhaitez intégrer des données plus précises à vos indicateurs.

CloudWatch Par exemple, vous pouvez ajouter le nom d'hôte comme dimension pour identifier les répartitions de charge inégales dans votre flotte. Tous les paramètres de KPL configuration étant immuables, vous ne pouvez pas modifier ces dimensions supplémentaires une fois l'KPLinstance initialisée.

Niveau métrique et granularité

Il existe deux options pour contrôler le nombre de métriques téléchargées vers CloudWatch :

niveau de métrique

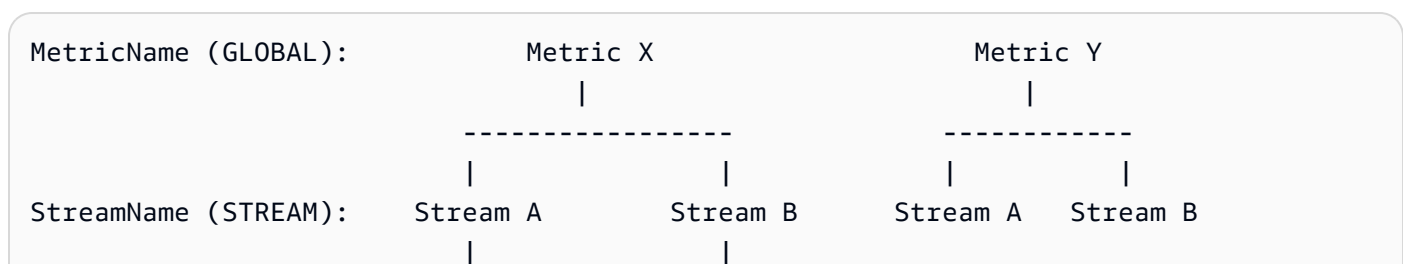
Le niveau de métrique sert à mesurer approximativement l'importance d'une métrique. Chaque métrique se voit attribuer un niveau. Lorsque vous définissez un niveau, les métriques dont les niveaux sont inférieurs ne sont pas envoyées à CloudWatch. Les niveaux sont : NONE, SUMMARY et DETAILED. Le paramètre par défaut est DETAILED ; c'est-à-dire, toutes les métriques. NONE signifie qu'il n'y a pas de métriques du tout, donc aucune métrique n'est réellement affectée à ce niveau.

granularité

La granularité contrôle si la même métrique est émise à des niveaux de granularité supplémentaires. Les niveaux sont : GLOBAL, STREAM et SHARD. La valeur par défaut est SHARD, qui contient les mesures les plus granulaires.

Lorsque SHARD est choisi, les métriques sont émises avec le nom du flux et l'ID de partition comme dimensions. En outre, la même métrique est également émise avec uniquement la dimension nom du flux et sans le nom du flux. Cela signifie que, pour une métrique donnée, deux flux contenant chacun deux partitions produiront sept CloudWatch métriques : une pour chaque partition, une pour chaque flux et une pour l'ensemble ; toutes décrivant les mêmes statistiques mais à des niveaux de granularité différents. Ces métriques sont illustrées dans le diagramme ci-dessous.

Les différents niveaux de granularité forment une hiérarchie et toutes les métriques du système forment des arborescences, partant des noms de métrique :



Liste des métriques

Métrique	Description
UserRecordsReceived	<p>Nombre d'enregistrements utilisateur logiques reçus par le KPL noyau pour les opérations de vente. Non disponible au niveau de la partition.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>
UserRecordsPending	<p>Echantillon périodique indiquant le nombre d'enregistrements utilisateur actuellement en attente. Un enregistrement est en attente s'il est actuellement en mémoire tampon et attend d'être envoyé, ou envoyé et en cours vers le service principal. Non disponible au niveau de la partition.</p> <p>KPL fournit une méthode dédiée pour récupérer cette métrique au niveau mondial afin que les clients puissent gérer leur taux de vente.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>
UserRecordsPut	<p>Nombre d'enregistrements utilisateur logiques placés avec succès.</p> <p>Ne compte KPL pas les enregistrements ayant échoué pour cette métrique. Ainsi, la moyenne correspond au taux de réussite, le nombre correspond au nombre total de tentatives, et la différence entre le nombre et la somme correspond au nombre d'échecs.</p> <p>Niveau de métrique : Summary</p> <p>Unité : nombre</p>
UserRecordsDataPut	<p>Nombre d'octets des enregistrements utilisateur logique placés avec succès.</p> <p>Niveau de métrique : Detailed</p>

Métrique	Description
	Unité : octets
KinesisRecordsPut	<p>Nombre d'enregistrements Kinesis Data Streams placés avec succès (chaque enregistrement Kinesis Data Streams peut contenir plusieurs enregistrements utilisateur).</p> <p>La KPL sortie est un zéro pour les enregistrements ayant échoué. Ainsi, la moyenne correspond au taux de réussite, le nombre correspond au nombre total de tentatives, et la différence entre le nombre et la somme correspond au nombre d'échecs.</p> <p>Niveau de métrique : Summary</p> <p>Unité : nombre</p>
KinesisRecordsDataPut	<p>Octets au sein des enregistrements Kinesis Data Streams.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : octets</p>
ErrorsByCode	<p>Nombre de chaque type de code d'erreur. C'est une autre dimension de <code>ErrorCode</code>, en plus des dimensions normales comme <code>StreamName</code> et <code>ShardId</code>. Les erreurs ne peuvent pas toutes être remontées dans une partition. Les erreurs qui ne peuvent pas être remontées ne sont émises qu'au niveau du flux ou au niveau général. Cette métrique capture les informations sur les éléments comme les limites, les modifications de mappage de partition, les défaillances internes, l'indisponibilité de service, les dépassements de délais; etc.</p> <p>Les erreurs Kinesis Data API Streams sont comptabilisées une fois par enregistrement Kinesis Data Streams. Les enregistrements utilisateur multiples dans un enregistrement Kinesis Data Streams ne génèrent pas de nombres multiples.</p> <p>Niveau de métrique : Summary</p> <p>Unité : nombre</p>

Métrique	Description
AllErrors	<p>Cette métrique est déclenchée par les mêmes erreurs que Errors by Code (Erreurs par code), mais ne fait pas la distinction entre les différents types. Elle est très utile pour la surveillance générale du taux d'erreurs sans nécessiter la somme manuelle des chiffres des différents types d'erreurs.</p> <p>Niveau de métrique : Summary</p> <p>Unité : nombre</p>
RetriesPerRecord	<p>Nombre de tentatives effectuées par enregistrement utilisateur. Zéro est la valeur émise pour les enregistrements qui ont réussi dès la première tentative.</p> <p>Les données sont émises au moment où un enregistrement utilisateur se termine (lorsqu'il a réussi ou qu'il ne peut plus être réessayé). Si l'enregistrement time-to-live est une valeur importante, cette métrique peut être retardée de manière significative.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>
BufferingTime	<p>Le délai entre l'arrivée d'un enregistrement utilisateur KPL et son départ pour le backend. Ces informations sont renvoyées à l'utilisateur pour chaque enregistrement par, mais sont également disponibles sous forme de statistique groupée.</p> <p>Niveau de métrique : Summary</p> <p>Unité : millisecondes</p>
Request Time	<p>Délai nécessaire pour exécuter PutRecordsRequests .</p> <p>Niveau de métrique : Detailed</p> <p>Unité : millisecondes</p>

Métrique	Description
User Records per Kinesis Record	<p>Nombre d'enregistrements utilisateur logiques regroupés en un seul enregistrement Kinesis Data Streams.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>
Amazon Kinesis Records per PutRecord sRequest	<p>Nombre d'enregistrements Kinesis Data Streams logiques regroupés en un seul PutRecordsRequest . Non disponible au niveau de la partition .</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>
User Records per PutRecord sRequest	<p>Nombre total d'enregistrements utilisateur contenus dans une PutRecordsRequest . Il équivaut à peu près au produit des deux métriques précédentes. Non disponible au niveau de la partition.</p> <p>Niveau de métrique : Detailed</p> <p>Unité : nombre</p>

Sécurité dans Amazon Kinesis Data Streams

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficierez d'un centre de données et d'une architecture 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 cette notion par les termes sécurité du cloud et 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. L'efficacité de notre sécurité est régulièrement testée et vérifiée par des auditeurs tiers dans le cadre des [programmes de conformité AWS](#). Pour en savoir plus sur les programmes de conformité qui s'appliquent à Kinesis Data Streams, consultez la rubrique [Services AWS concernés par le programme de conformité](#).
- 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 organisation ainsi que les lois et réglementations applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de Kinesis Data Streams. Les rubriques suivantes expliquent comment configurer Kinesis Data Streams pour répondre à vos objectifs de sécurité et de conformité. Vous découvrirez également comment utiliser d'autres AWS services qui peuvent vous aider à surveiller et à sécuriser vos ressources Kinesis Data Streams.

Rubriques

- [Protection des données dans Amazon Kinesis Data Streams](#)
- [Contrôler l'accès aux ressources Amazon Kinesis Data Streams à l'aide de IAM](#)
- [Validation de conformité pour Amazon Kinesis Data Streams](#)
- [Résilience dans Amazon Kinesis Data Streams](#)
- [Sécurité de l'infrastructure dans Kinesis Data Streams](#)
- [Bonnes pratiques en matière de sécurité pour Kinesis Data Streams](#)

Protection des données dans Amazon Kinesis Data Streams

Le chiffrement côté serveur à l'aide de clés AWS Key Management Service (AWS KMS) vous permet de répondre facilement à des exigences strictes en matière de gestion des données en chiffrant vos données au repos dans Amazon Kinesis Data Streams.

Note

Si vous avez besoin de FIPS 140 à 2 modules cryptographiques validés pour accéder AWS via une interface de ligne de commande ou un API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les FIPS points de terminaison disponibles, voir [Federal Information Processing Standard \(FIPS\) 140-2](#).

Rubriques

- [Qu'est-ce que le chiffrement côté serveur pour Kinesis Data Streams ?](#)
- [Considérations relatives aux coûts, aux régions et aux performances](#)
- [Comment démarrer avec le chiffrement côté serveur ?](#)
- [Création et utilisation de clés générées par l'utilisateur KMS](#)
- [Autorisations d'utilisation des clés générées par l'utilisateur KMS](#)
- [Vérifier et résoudre les problèmes liés aux autorisations KMS clés](#)
- [Utiliser Amazon Kinesis Data Streams avec les points de terminaison d'interface VPC](#)

Qu'est-ce que le chiffrement côté serveur pour Kinesis Data Streams ?

Le chiffrement côté serveur est une fonctionnalité d'Amazon Kinesis Data Streams qui chiffre automatiquement les données avant qu'elles ne soient inactives à l'aide de la clé principale du AWS KMS client () que vous spécifiez. Les données sont chiffrées avant leur écriture sur la couche de stockage du flux Kinesis et déchiffrées après leur extraction de l'espace de stockage. Par conséquent, vos données sont chiffrées au repos dans le service Kinesis Data Streams. Cela vous permet de respecter des exigences réglementaires strictes et d'améliorer la sécurité de vos données.

Avec le chiffrement côté serveur, les producteurs et les consommateurs de votre flux Kinesis n'ont pas besoin de gérer les clés principales ni les opérations de chiffrement. Vos données sont automatiquement cryptées lorsqu'elles entrent dans le service Kinesis Data Streams et en sortent. Vos données au repos sont donc cryptées. AWS KMS fournit toutes les clés principales utilisées par

la fonction de chiffrement côté serveur. AWS KMS facilite l'utilisation d'une clé CMK pour Kinesis gérée par AWS AWS KMS CMK, spécifiée par l'utilisateur ou importée dans le service. AWS KMS

Note

Le chiffrement côté serveur chiffre les données entrantes seulement après l'activation du chiffrement. Les données préexistantes dans un flux de données non chiffré ne sont pas chiffrées après l'activation du chiffrement côté serveur.

Lorsque vous chiffrez vos flux de données et que vous partagez l'accès avec d'autres principaux, vous devez accorder une autorisation à la fois dans la politique clé pour la AWS KMS clé et dans les IAM politiques du compte externe. Pour plus d'informations, voir [Autoriser les utilisateurs d'autres comptes à utiliser une KMS clé](#).

Si vous avez activé le chiffrement côté serveur pour un flux de données avec KMS clé AWS gérée et que vous souhaitez partager l'accès via une politique de ressources, vous devez passer à l'utilisation de la clé gérée par le client (CMK), comme indiqué ci-dessous :

Edit encryption for test_encryption

Encryption [Info](#)

Enable server-side encryption
Kinesis Data Stream uses AWS Key Management Service (KMS) to encrypt your data. You can choose the AWS managed customer master key (CMK) to encrypt your data or specify a customer-managed CMK.

Use AWS managed CMK
The AWS managed CMK (aws/kinesis) in your account is created, managed, and used on your behalf by Kinesis Data Streams.

Use customer-managed CMK
Customer-managed CMKs in your AWS account are created, owned, and managed by you.

Customer-managed CMK in KMS

En outre, vous devez autoriser vos entités principales de partage à accéder à votre compte CMK, en utilisant KMS les fonctionnalités de partage entre comptes. Assurez-vous de modifier également les

IAM politiques relatives au partage des entités principales. Pour plus d'informations, voir [Autoriser les utilisateurs d'autres comptes à utiliser une KMS clé](#).

Considérations relatives aux coûts, aux régions et aux performances

Lorsque vous appliquez le chiffrement côté serveur, vous êtes soumis à des coûts d' AWS KMS API utilisation et de clé. Contrairement aux clés KMS principales personnalisées, la clé principale du (Default) `aws/kinesis` client (CMK) est proposée gratuitement. Toutefois, vous devez toujours payer les frais API d'utilisation engagés par Amazon Kinesis Data Streams en votre nom.

API Les frais d'utilisation s'appliquent à tous CMK, y compris les frais personnalisés. Kinesis Data Streams appelle AWS KMS toutes les cinq minutes environ lors de la rotation de la clé des données. Dans un mois de 30 jours, le coût total des AWS KMS API appels initiés par un flux Kinesis devrait être inférieur à quelques dollars. Ce coût varie en fonction du nombre d'informations d'identification utilisateur que vous utilisez pour vos producteurs et consommateurs de données, car chaque identifiant d'utilisateur nécessite un API appel unique. AWS KMS Lorsque vous utilisez un IAM rôle pour l'authentification, chaque appel à assumer un rôle génère des informations d'identification utilisateur uniques. Pour réduire KMS les coûts, vous souhaitez peut-être mettre en cache les informations d'identification utilisateur renvoyées par l'appel d'acceptation du rôle.

Voici une description des coûts par ressource :

Clés

- Le CMK for Kinesis géré par AWS (alias =`aws/kinesis`) est gratuit.
- Les KMS clés générées par l'utilisateur sont soumises à des coûts KMS clés. Pour plus d'informations, consultez [Tarification de AWS Key Management Service](#) (français non garanti).

API Les frais d'utilisation s'appliquent à tous CMK, y compris les frais personnalisés. Kinesis Data Streams KMS appelle environ toutes les 5 minutes lorsqu'il fait pivoter la clé de données. Dans un mois de 30 jours, le coût total des KMS API appels initiés par un flux de données Kinesis devrait être inférieur à quelques dollars. Veuillez noter que ce coût varie en fonction du nombre d'informations d'identification utilisateur que vous utilisez pour vos producteurs et consommateurs de données, car chaque identifiant d'utilisateur nécessite un API appel unique. AWS KMS Lorsque vous utilisez IAM le rôle pour l'authentification, chacune d'entre elles `assume-role-call` se traduit par des informations d'identification utilisateur uniques et vous souhaitez peut-être mettre en cache les informations d'identification utilisateur renvoyées par le `assume-role-call` pour réduire KMS les coûts.

KMSAPIutilisation

Pour chaque flux chiffré, lors de la lecture TIP et de l'utilisation d'un seul IAM compte ou d'une clé d'accès utilisateur pour les lecteurs et les rédacteurs, le service Kinesis appelle le AWS KMS service environ 12 fois toutes les 5 minutes. Le fait de ne pas lire TIP pourrait entraîner une augmentation du nombre d'appels au AWS KMS service. API Les demandes de génération de nouvelles clés de chiffrement des données sont soumises à des coûts AWS KMS d'utilisation. Pour plus d'informations, consultez [Tarification de AWS Key Management Service Pricing : Utilisation](#) (français non garanti).

Disponibilité du chiffrement côté serveur par région

À l'heure actuelle, le chiffrement des flux Kinesis côté serveur est disponible dans toutes les régions prises en charge par Kinesis Data Streams, AWS GovCloud notamment dans l'ouest des États-Unis et en Chine. Pour plus d'informations sur les régions prises en charge pour Kinesis Data Streams, <https://docs.aws.amazon.com/general/consultez> le fichier latest/gr/ak.html.

Considérations sur les performances

En raison de la surcharge du service liée au chiffrement, l'application du chiffrement côté serveur augmente la latence habituelle de PutRecord, PutRecords et GetRecords de moins de 100µs.

Comment démarrer avec le chiffrement côté serveur ?

Le moyen le plus simple de démarrer avec le chiffrement côté serveur est d'utiliser la clé de service Amazon Kinesis AWS Management Console et la clé de service Amazon KMS Kinesis. `aws/kinesis`

La procédure suivante explique comment activer le chiffrement côté serveur pour un flux Kinesis.

Activer le chiffrement côté serveur pour un flux Kinesis

1. Connectez-vous à la console [Amazon Kinesis Data Streams AWS Management Console](#) et ouvrez-la.
2. Créez ou sélectionnez un flux Kinesis dans la AWS Management Console.
3. Choisissez l'onglet Détails.
4. Dans Server-side encryption (Chiffrement côté serveur), choisissez Modifier.
5. À moins que vous ne souhaitiez utiliser une clé KMS principale générée par l'utilisateur, assurez-vous que la clé principale `aws/kinesis` KMS (par défaut) est sélectionnée. Il s'agit de la clé KMS principale générée par le service Kinesis. Choisissez Enabled, puis Save.

Note

La clé principale du service Kinesis par défaut est gratuite, mais les API appels passés par Kinesis vers le AWS KMS service sont soumis à des frais d'utilisation. KMS

6. Le flux passe à l'état pending (en attente). Une fois que le flux est revenu à un état actif avec le chiffrement activé, toutes les données entrantes écrites dans le flux sont chiffrées à l'aide de la clé KMS principale que vous avez sélectionnée.
7. Pour désactiver le chiffrement côté serveur, choisissez Désactivé dans le chiffrement côté serveur dans le AWS Management Console, puis sélectionnez Enregistrer.

Création et utilisation de clés générées par l'utilisateur KMS

Cette section explique comment créer et utiliser vos propres KMS clés, au lieu d'utiliser la clé principale administrée par Amazon Kinesis.

Création de clés générées par l'utilisateur KMS

Pour obtenir des instructions sur la création de vos propres clés, consultez la section [Création de clés](#) dans le guide du AWS Key Management Service développeur. Une fois que vous avez créé les clés de votre compte, le service Kinesis Data Streams les renvoie dans KMS la liste des clés principales.

Utilisation de clés générées par l'utilisateur KMS

Une fois que les autorisations appropriées ont été appliquées à vos consommateurs, producteurs et administrateurs, vous pouvez utiliser des KMS clés personnalisées dans votre propre AWS compte ou dans un autre AWS compte. Toutes les clés KMS principales de votre compte apparaissent dans la liste des clés KMS principales du AWS Management Console.

Pour utiliser des clés KMS principales personnalisées situées dans un autre compte, vous devez disposer d'autorisations pour utiliser ces clés. Vous devez également spécifier ARN la clé KMS principale dans la zone de ARN saisie du AWS Management Console.

Autorisations d'utilisation des clés générées par l'utilisateur KMS

Avant de pouvoir utiliser le chiffrement côté serveur avec une KMS clé générée par l'utilisateur, vous devez configurer des politiques AWS KMS clés pour autoriser le chiffrement des flux ainsi

que le chiffrement et le déchiffrement des enregistrements des flux. Pour des exemples et plus d'informations sur AWS KMS les autorisations, voir [AWS KMS API Autorisations : référence sur les actions et les ressources](#).

Note

L'utilisation de la clé de service par défaut pour le chiffrement ne nécessite pas l'application d'IAM autorisations personnalisées.

Avant d'utiliser des clés KMS principales générées par les utilisateurs, assurez-vous que les producteurs et les consommateurs de vos flux Kinesis IAM (principaux) sont des utilisateurs visés par KMS la politique des clés principales. Si ce n'est pas le cas, les écritures et les lectures à partir d'un flux échoueront, ce qui pourrait entraîner la perte de données, des retards de traitement ou la suspension d'applications. Vous pouvez gérer les autorisations relatives aux KMS clés à l'aide IAM de politiques. Pour plus d'informations, consultez la section [Utilisation IAM de politiques avec AWS KMS](#).

Exemple d'autorisations pour les producteurs

Vos applications producteur de flux Kinesis doivent disposer de l'autorisation `kms:GenerateDataKey`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": "arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
    }
  ]
}
```

```
    "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
  }
]
}
```

Exemples d'autorisations accordées aux consommateurs

Vos consommateurs de flux Kinesis doivent disposer de l'autorisation `kms:Decrypt`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:GetRecords",
        "kinesis:DescribeStream"
      ],
      "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Amazon Managed Service pour Apache Flink et AWS Lambda utilisent des rôles pour consommer des flux Kinesis. Assurez-vous d'ajouter l'autorisation `kms:Decrypt` aux rôles que ces consommateurs utilisent.

Autorisations d'administrateur du stream

Les administrateurs de flux Kinesis doivent être autorisés à appeler `kms:List*` et `kms:DescribeKey*`.

Vérifier et résoudre les problèmes liés aux autorisations KMS clés

Après avoir activé le chiffrement sur un flux Kinesis, nous vous recommandons de contrôler le succès de vos `getRecords` appels et de vos `putRecord` appels à l'aide des indicateurs Amazon CloudWatch suivants : `putRecords`

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Pour plus d'informations, consultez [Surveillez Kinesis Data Streams](#).

Utiliser Amazon Kinesis Data Streams avec les points de terminaison d'interface VPC

Vous pouvez utiliser un point de VPC terminaison d'interface pour empêcher le trafic entre Amazon VPC et Kinesis Data Streams de quitter le réseau Amazon. VPC Les points de terminaison de l'interface ne nécessitent pas de passerelle Internet, NAT d'appareil, de VPN connexion ou de AWS Direct Connect connexion. Les VPC points de terminaison d'interface sont AWS PrivateLink alimentés par une AWS technologie qui permet une communication privée entre les AWS services à l'aide d'une interface réseau élastique avec private IPs in your AmazonVPC. Pour plus d'informations, consultez [Amazon Virtual Private Cloud](#) et [Interface VPC Endpoints \(AWS PrivateLink\)](#).

Rubriques

- [Utiliser les VPC points de terminaison de l'interface pour Kinesis Data Streams](#)
- [Contrôlez l'accès aux VPC points de terminaison pour Kinesis Data Streams](#)
- [Disponibilité des politiques relatives aux VPC terminaux pour Kinesis Data Streams](#)

Utiliser les VPC points de terminaison de l'interface pour Kinesis Data Streams

Pour démarrer, vous n'avez pas besoin de modifier les paramètres de vos flux, producteurs ou consommateurs. Créez simplement un point de VPC terminaison d'interface pour que votre trafic Kinesis Data Streams en provenance et à destination de vos ressources VPC Amazon commence à passer par le point de terminaison de l'VPCinterface. Pour plus d'informations, consultez [Création d'un point de terminaison d'interface](#).

La Kinesis Producer Library (KPL) et la Kinesis Consumer Library (KCL) appellent des services AWS tels qu'Amazon et Amazon CloudWatch DynamoDB en utilisant des points de terminaison publics ou des points de terminaison d'interface privés, selon ceux utilisés. VPC Par exemple, si votre KCL application s'exécute dans une interface VPC DynamoDB VPC avec des points de terminaison activés, les appels entre DynamoDB et KCL votre application passent par le point de terminaison de l'interface. VPC

Contrôlez l'accès aux VPC points de terminaison pour Kinesis Data Streams

VPCles politiques de point de terminaison vous permettent de contrôler l'accès en attachant une politique à un VPC point de terminaison ou en utilisant des champs supplémentaires dans une politique attachée à un IAM utilisateur, à un groupe ou à un rôle afin de restreindre l'accès uniquement via le point de VPC terminaison spécifié. Ces politiques peuvent être utilisées pour restreindre l'accès à des flux spécifiques à un point de VPC terminaison spécifique lorsqu'elles sont utilisées conjointement avec les IAM politiques visant à accorder l'accès aux actions de flux de données Kinesis uniquement via le point de terminaison spécifiéVPC.

Voici des exemples de politiques de point de terminaison pour accéder aux flux de données Kinesis.

- VPCexemple de stratégie : accès en lecture seule : cet exemple de politique peut être attaché à un VPC point de terminaison. (Pour plus d'informations, consultez la section [Contrôle de l'accès aux VPC ressources Amazon](#)). Il limite les actions à la seule liste et à la description d'un flux de données Kinesis via VPC le point de terminaison auquel il est connecté.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- VPCexemple de politique : restreindre l'accès à un flux de données Kinesis spécifique. Cet exemple de politique peut être associé à un VPC point de terminaison. Il restreint l'accès à un flux de données spécifique via le VPC point de terminaison auquel il est rattaché.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- IAMexemple de politique : restreindre l'accès à un flux spécifique à partir d'un point de VPC terminaison spécifique uniquement - cet exemple de politique peut être attaché à un IAM utilisateur, à un rôle ou à un groupe. Il restreint l'accès à un flux de données Kinesis spécifique uniquement à partir d'un VPC point de terminaison spécifié.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AccessFromSpecificEndpoint",
      "Action": "kinesis:*",
      "Effect": "Deny",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream",
      "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-11aa22bb" } }
    }
  ]
}
```

Disponibilité des politiques relatives aux VPC terminaux pour Kinesis Data Streams

Les points de terminaison de VPC l'interface Kinesis Data Streams dotés de politiques sont pris en charge dans les régions suivantes :

- Europe (Paris)
- Europe (Irlande)
- USA Est (Virginie du Nord)
- Europe (Stockholm)
- USA Est (Ohio)
- Europe (Francfort)
- Amérique du Sud (São Paulo)
- Europe (Londres)
- Asie-Pacifique (Tokyo)
- USA Ouest (Californie du Nord)
- Asie-Pacifique (Singapour)
- Asie-Pacifique (Sydney)
- Chine (Beijing)
- Chine (Ningxia)
- Asie-Pacifique (Hong Kong)
- Moyen-Orient (Bahreïn)
- Moyen-Orient (UAE)
- Europe (Milan)
- Afrique (Le Cap)
- Asie-Pacifique (Mumbai)
- Asie-Pacifique (Séoul)
- Canada (Centre)
- USA Ouest (Oregon) sauf usw2-az4
- AWS GovCloud (USA Est)
- AWS GovCloud (US-Ouest)

- Asie-Pacifique (Osaka)
- Europe (Zurich)
- Asie-Pacifique (Hyderabad)

Contrôler l'accès aux ressources Amazon Kinesis Data Streams à l'aide de IAM

AWS Identity and Access Management (IAM) vous permet d'effectuer les opérations suivantes :

- Créez des utilisateurs et des groupes sous votre AWS compte
- Attribuez des informations de sécurité uniques à chaque utilisateur de votre AWS compte
- Contrôlez les autorisations de chaque utilisateur pour effectuer des tâches à l'aide de AWS ressources
- Autoriser les utilisateurs d'un autre AWS compte à partager vos AWS ressources
- Créez des rôles pour votre AWS compte et définissez les utilisateurs ou les services qui peuvent les assumer
- Utilisez les identités existantes pour que votre entreprise accorde des autorisations pour effectuer des tâches à l'aide de AWS ressources

IAM Grâce à Kinesis Data Streams, vous pouvez contrôler si les utilisateurs de votre organisation peuvent effectuer une tâche à l'aide d'actions Kinesis Data API Streams spécifiques et s'ils peuvent utiliser des ressources spécifiques. AWS

Si vous développez une application à l'aide de la bibliothèque cliente Kinesis (KCL), votre politique doit inclure des autorisations pour Amazon DynamoDB et Amazon CloudWatch ; DynamoDB KCL utilise DynamoDB pour suivre les informations d'état de l'application et pour envoyer des métriques en votre nom. CloudWatch KCL CloudWatch Pour plus d'informations sur le KCL, consultez [Développez des KCL consommateurs 1.x](#).

Pour plus d'informations sur IAM, consultez les rubriques suivantes :

- [AWS Identity and Access Management \(IAM\)](#)
- [Prise en main](#)
- [Guide de l'utilisateur IAM](#)

Pour plus d'informations sur Amazon DynamoDB IAM et sur Amazon DynamoDB, [consultez la section IAM Utiliser pour contrôler l'accès aux ressources Amazon DynamoDB dans le manuel du développeur Amazon DynamoDB](#).

Pour plus d'informations sur Amazon IAM et sur Amazon CloudWatch, consultez la section [Contrôle de l'accès des utilisateurs à votre AWS compte](#) dans le guide de CloudWatch l'utilisateur Amazon.

Table des matières

- [Syntaxe d'une politique](#)
- [Actions pour Kinesis Data Streams](#)
- [Amazon Resource Names \(ARNs\) pour Kinesis Data Streams](#)
- [Exemples de politiques pour Kinesis Data Streams](#)
- [Partagez votre flux de données avec un autre compte](#)
- [Configurer une AWS Lambda fonction pour lire depuis Kinesis Data Streams dans un autre compte](#)
- [Partagez l'accès à l'aide de politiques basées sur les ressources](#)

Syntaxe d'une politique

Une IAM politique est un JSON document composé d'une ou de plusieurs déclarations. Chaque déclaration est structurée comme suit :

```
{
  "Statement": [{
    "Effect": "effect",
    "Action": "action",
    "Resource": "arn",
    "Condition": {
      "condition": {
        "key": "value"
      }
    }
  ]
}
```

Une déclaration se compose de différents éléments :

- **Effect** : effect peut avoir la valeur Allow ou Deny. Par défaut, IAM les utilisateurs ne sont pas autorisés à utiliser les ressources et les API actions. Toutes les demandes sont donc refusées. Une autorisation explicite remplace l'autorisation par défaut. Un refus explicite remplace toute autorisation.
- **Action** : L'action est l'APIaction spécifique pour laquelle vous accordez ou refusez l'autorisation.
- **Resource** : la ressource affectée par l'action. Pour spécifier une ressource dans l'instruction, vous devez utiliser son Amazon Resource Name (ARN).
- **Condition** : les conditions sont facultatives. Elles permettent de contrôler à quel moment votre stratégie sera effective.

Lorsque vous créez et gérez des IAM politiques, vous souhaitez peut-être utiliser le [générateur de IAM politiques](#) et le [simulateur IAM de politiques](#).

Actions pour Kinesis Data Streams

Dans une déclaration IAM de politique, vous pouvez spécifier n'importe quelle API action à partir de n'importe quel service compatibleIAM. Pour Kinesis Data Streams, utilisez le préfixe suivant avec le nom de API l'action : `kinesis:` Par exemple : `kinesis:CreateStream`, `kinesis:ListStreams` et `kinesis:DescribeStreamSummary`.

Pour spécifier plusieurs actions dans une seule déclaration, séparez-les par des virgules comme suit :

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Vous pouvez aussi spécifier plusieurs actions à l'aide de caractères génériques. Par exemple, vous pouvez spécifier toutes les actions dont le nom commence par le mot « Get » comme suit :

```
"Action": "kinesis:Get*"
```

Pour spécifier toutes les opérations Kinesis Data Streams, utilisez le caractère générique `*` comme suit :

```
"Action": "kinesis:*"
```

Pour obtenir la liste complète des actions Kinesis Data API Streams, consultez [le API Amazon Kinesis Reference](#).

Amazon Resource Names (ARNs) pour Kinesis Data Streams

Chaque déclaration IAM de politique s'applique aux ressources que vous spécifiez à l'aide de leur ARNs.

Utilisez le format de ARN ressource suivant pour les flux de données Kinesis :

```
arn:aws:kinesis:region:account-id:stream/stream-name
```

Par exemple :

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Exemples de politiques pour Kinesis Data Streams

Les exemples de politiques suivants montrent comment vous pouvez contrôler l'accès utilisateur à vos flux de données Kinesis.

Exemple 1: Allow users to get data from a stream

Exemple

Cette stratégie permet à un utilisateur ou un groupe d'effectuer les opérations `DescribeStreamSummary`, `GetShardIterator` et `GetRecords` sur le flux spécifié, et `ListStreams` sur n'importe quel flux. Cette stratégie peut être appliquée aux utilisateurs qui doivent pouvoir extraire des données d'un flux spécifique.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:Get*",
        "kinesis:DescribeStreamSummary"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
      ]
    },
    {
```

```

    "Effect": "Allow",
    "Action": [
        "kinesis:ListStreams"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

Example 2: Allow users to add data to any stream in the account

Example

Cette stratégie permet à un utilisateur ou un groupe d'utiliser l'opération PutRecord avec tous les flux du compte. Cette stratégie peut être appliquée aux utilisateurs qui doivent pouvoir ajouter des enregistrements à tous les flux d'un compte.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecord"
      ],
      "Resource": [
        "arn:aws:kinesis:us-east-1:111122223333:stream/*"
      ]
    }
  ]
}

```

Example 3: Allow any Kinesis Data Streams action on a specific stream

Example

Cette politique permet à un utilisateur ou un groupe d'utiliser une opération Kinesis Data Streams sur le flux spécifié. Cette stratégie peut être appliquée aux utilisateurs qui doivent disposer d'un contrôle administratif sur un flux spécifique.

```

{

```

```

    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": "kinesis:*",
        "Resource": [
          "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
        ]
      }
    ]
  }
}

```

Example 4: Allow any Kinesis Data Streams action on any stream

Example

Cette politique permet à un utilisateur ou un groupe d'utiliser une opération Kinesis Data Streams sur un flux d'un compte. Etant donné que cette stratégie accorde un accès complet à tous vos flux, vous devez la limiter aux administrateurs.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kinesis:*",
      "Resource": [
        "arn:aws:kinesis:*:111122223333:stream/*"
      ]
    }
  ]
}

```

Partagez votre flux de données avec un autre compte

Note

La bibliothèque Kinesis Producer ne prend actuellement pas en charge la spécification d'un flux ARN lors de l'écriture dans un flux de données. Utilisez le AWS SDK si vous souhaitez écrire dans un flux de données entre comptes.

Associez une [politique basée sur les ressources](#) à votre flux de données pour accorder l'accès à un autre compte, IAM utilisateur ou IAM rôle. Les politiques basées sur les ressources sont des documents de JSON stratégie que vous attachez à une ressource telle qu'un flux de données. Ces politiques accordent au [principal spécifié](#) l'autorisation d'effectuer des actions spécifiques sur cette ressource et définissent sous quelles conditions cela s'applique. Une politique peut comporter plusieurs instructions. Vous devez spécifier un principal dans une politique basée sur les ressources. Les principaux peuvent inclure des comptes, des utilisateurs, des rôles, des utilisateurs fédérés ou AWS des services. Vous pouvez configurer des politiques dans la console Kinesis Data StreamsAPI, SDK ou.

Notez que le partage de l'accès aux consommateurs enregistrés, tel que [Enhanced Fan Out](#), nécessite une politique à la fois concernant le flux de données ARN et le consommateurARN.

Activer l'accès entre comptes

Pour activer l'accès entre comptes, vous pouvez spécifier un compte entier ou IAM des entités d'un autre compte comme principal dans une politique basée sur les ressources. L'ajout d'un principal entre comptes à une politique basée sur les ressources ne représente qu'une partie de l'instauration de la relation d'approbation. Lorsque le principal et la ressource se trouvent dans des AWS comptes distincts, vous devez également utiliser une politique basée sur l'identité pour accorder au principal l'accès à la ressource. Toutefois, si une politique basée sur des ressources accorde l'accès à un principal dans le même compte, aucune autre politique basée sur l'identité n'est requise.

Pour plus d'informations sur l'utilisation de politiques basées sur les ressources pour l'accès entre comptes, voir Accès aux [ressources entre comptes](#) dans IAM

Les administrateurs de flux de données peuvent utiliser des AWS Identity and Access Management politiques pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions. L'Actionélément d'une JSON politique décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès dans une politique. Les actions de stratégie portent généralement le même nom que l' AWS APIopération associée.

Actions Kinesis Data Streams pouvant être partagées :

Action	Niveau d'accès
DescribeStreamConsumer	Consommateur

Action	Niveau d'accès
DescribeStreamSummary	Flux de données
GetRecords	Flux de données
GetShardIterator	Flux de données
ListShards	Flux de données
PutRecord	Flux de données
PutRecords	Flux de données
SubscribeToShard	Consommateur

Vous trouverez ci-dessous des exemples d'utilisation d'une politique basée sur les ressources pour accorder l'accès intercompte à votre flux de données ou à votre consommateur enregistré.

Pour effectuer une action entre comptes, vous devez spécifier le flux ARN pour l'accès au flux de données et le consommateur ARN pour l'accès client enregistré.

Exemples de politiques basées sur les ressources pour les flux de données Kinesis

Le partage d'un consommateur enregistré implique à la fois une politique de flux de données et une politique des consommateurs en raison des actions nécessaires.

Note

Vous trouverez ci-dessous des exemples de valeurs valides pour le Principal:

- {"AWS": "123456789012"}
- IAMUtilisateur — {"AWS": "arn:aws:iam::123456789012:user/user-name"}
- IAMRôle — {"AWS": ["arn:aws:iam::123456789012:role/role-name"]}
- Principaux multiples (peut être une combinaison de compte, d'utilisateur, de rôle) – {"AWS": ["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}

Example 1: Write access to the data stream

Example

```
{
  "Version": "2012-10-17",
  "Id": "__default_write_policy_ID",
  "Statement": [
    {
      "Sid": "writestatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "Account12345"
      },
      "Action": [
        "kinesis:DescribeStreamSummary",
        "kinesis:ListShards",
        "kinesis:PutRecord",
        "kinesis:PutRecords"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
  ]
}
```

Example 2: Read access to the data stream

Example

```
{
  "Version": "2012-10-17",
  "Id": "__default_sharedthroughput_read_policy_ID",
  "Statement": [
    {
      "Sid": "sharedthroughputreadstatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "Account12345"
      },
      "Action": [
        "kinesis:DescribeStreamSummary",
        "kinesis:ListShards",
        "kinesis:GetRecords",

```

```

        "kinesis:GetShardIterator"
    ],
    "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
]
}

```

Example 3: Share enhanced fan-out read access to a registered consumer

Example

Déclaration de politique de flux de données :

```

{
  "Version": "2012-10-17",
  "Id": "__default_sharedthroughput_read_policy_ID",
  "Statement": [
    {
      "Sid": "consumerreadstatement",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::Account12345:role/role-name"
      },
      "Action": [
        "kinesis:DescribeStreamSummary",
        "kinesis:ListShards"
      ],
      "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
    }
  ]
}

```

Déclaration de politique des consommateurs :

```

{
  "Version": "2012-10-17",
  "Id": "__default_efo_read_policy_ID",
  "Statement": [
    {
      "Sid": "eforeadstatement",
      "Effect": "Allow",

```



```
    "Principal": {
      "AWS": "arn:aws:iam::Account12345:role/role-name"
    },
    "Action": [
      "kinesis:DescribeStreamConsumer",
      "kinesis:SubscribeToShard"
    ],
    "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
  ]
}
```

Le caractère générique (*) n'est pas pris en charge pour les actions ou le champ principal afin de maintenir le principe du moindre privilège.

Gérez la politique de votre flux de données de manière programmatique

En dehors de AWS Management Console, Kinesis Data Streams en propose APIS trois pour gérer votre politique de flux de données :

- [PutResourcePolicy](#)
- [GetResourcePolicy](#)
- [DeleteResourcePolicy](#)

Utilisez `PutResourcePolicy` pour attacher ou remplacer une politique pour un flux de données ou un consommateur. Utilisez `GetResourcePolicy` pour vérifier et afficher une politique pour le flux de données ou le consommateur spécifié. Utilisez `DeleteResourcePolicy` pour supprimer une politique pour le flux de données ou le consommateur spécifié.

Limites de la politique

Les politiques relatives aux ressources de Kinesis Data Streams comportent les restrictions suivantes :

- Les caractères génériques (*) ne sont pas pris en charge pour empêcher l'octroi d'un accès étendu par le biais des politiques de ressources directement associées à un flux de données ou à un consommateur enregistré. En outre, examinez attentivement les politiques suivantes pour vous assurer qu'elles n'accordent pas un accès étendu :

- Politiques basées sur l'identité associées aux AWS principaux associés (par exemple, les rôles IAM)
- Politiques basées sur les AWS ressources associées (par exemple, AWS Key Management Service KMS clés)
- AWS Les directeurs de service ne sont pas pris en charge pour éviter toute [confusion](#) potentielle entre les adjoints.
- Les principaux fédérés ne sont pas pris en charge.
- IDs Les utilisateurs canoniques ne sont pas pris en charge.
- La taille de la politique ne peut pas dépasser 20 Ko.

Partage de l'accès aux données chiffrées

Si vous avez activé le chiffrement côté serveur pour un flux de données avec KMS clé AWS gérée et que vous souhaitez partager l'accès via une politique de ressources, vous devez passer à l'utilisation de la clé gérée par le client (). CMK Pour de plus amples informations, veuillez consulter [Qu'est-ce que le chiffrement côté serveur pour Kinesis Data Streams ?](#). En outre, vous devez autoriser vos entités principales de partage à accéder à votre compte CMK, en utilisant KMS les fonctionnalités de partage entre comptes. Assurez-vous de modifier également les IAM politiques relatives au partage des entités principales. Pour plus d'informations, voir [Autoriser les utilisateurs d'autres comptes à utiliser une KMS clé](#).

Configurer une AWS Lambda fonction pour lire depuis Kinesis Data Streams dans un autre compte

Pour un exemple de configuration d'une fonction Lambda pour qu'elle puisse lire à partir de Kinesis Data Streams dans un autre compte, consultez [Partage de l'accès grâce à des fonctions multi-comptes AWS Lambda](#).

Partagez l'accès à l'aide de politiques basées sur les ressources

Note

La mise à jour d'une politique existante basée sur les ressources implique le remplacement de la politique existante. Assurez-vous donc d'inclure toutes les informations nécessaires dans votre nouvelle politique.

Partage de l'accès grâce à des fonctions multi-comptes AWS Lambda

Opérateur Lambda

1. Accédez à la [IAMconsole](#) pour créer un IAM rôle qui sera utilisé comme rôle d'[exécution Lambda pour votre AWS Lambda fonction](#). Ajoutez la IAM politique gérée `AWSLambdaKinesisExecutionRole` qui dispose des autorisations d'invocation Kinesis Data Streams et Lambda requises. Cette politique accorde également l'accès à toutes les ressources Kinesis Data Streams auxquelles vous pourriez avoir accès.
2. Dans la [AWS Lambda console](#), créez une AWS Lambda fonction [pour traiter les enregistrements d'un flux de données Kinesis Data Streams](#) et, lors de la configuration du rôle d'exécution, choisissez le rôle que vous avez créé à l'étape précédente.
3. Fournissez le rôle d'exécution au propriétaire de la ressource Kinesis Data Streams pour configurer la politique de ressources.
4. Terminez la configuration de la fonction Lambda.

Propriétaire de la ressource Kinesis Data Streams

1. Obtenez le rôle d'exécution Lambda entre comptes qui appellera la fonction Lambda.
2. Sur la console Amazon Kinesis Data Streams, choisissez le flux de données. Choisissez l'onglet Partage de flux de données, puis le bouton Créer une politique de partage pour démarrer l'éditeur visuel de politique. Pour partager un consommateur enregistré dans un flux de données, choisissez le consommateur, puis choisissez Créer une politique de partage. Vous pouvez également rédiger la JSON politique directement.
3. Spécifiez le rôle d'exécution Lambda entre comptes comme principal et les actions Kinesis Data Streams exactes auxquelles vous partagez l'accès. Veillez à inclure l'action `kinesis:DescribeStream`. Pour plus d'informations sur les exemples de politiques de ressources pour Kinesis Data Streams, consultez [Exemples de politiques basées sur les ressources pour les flux de données Kinesis](#).
4. Choisissez Créer une politique ou utilisez le [PutResourcePolicy](#) pour associer la politique à votre ressource.

Partagez l'accès avec les clients ayant plusieurs comptes KCL

- Si vous utilisez la version KCL 1.x, assurez-vous d'utiliser la version KCL 1.15.0 ou une version ultérieure.

- Si vous utilisez la version KCL 2.x, assurez-vous d'utiliser la version KCL 2.5.3 ou supérieure.

KCLopérateur

1. Indiquez IAM l'utilisateur ou IAM le rôle qui exécutera l'KCLapplication au propriétaire de la ressource.
2. Demandez au propriétaire de la ressource le flux de données ou le consommateurARN.
3. Assurez-vous de spécifier le flux fourni ARN dans le cadre de votre KCL configuration.
 - Pour KCL 1.x : utilisez le [KinesisClientLibConfiguration](#) constructeur et fournissez le flux. ARN
 - Pour la KCL version 2.x : vous pouvez uniquement fournir le flux ARN ou la [StreamTracker](#) bibliothèque cliente Kinesis. [ConfigsBuilder](#) Pour StreamTracker, fournissez le flux ARN et l'époque de création à partir de la table de location DynamoDB générée par la bibliothèque. Si vous souhaitez lire un article d'un consommateur enregistré partagé, comme Enhanced Fan-Out, utilisez StreamTracker et fournissez également le consommateur. ARN

Propriétaire de la ressource Kinesis Data Streams

1. Choisissez l'IAMutilisateur ou le IAM rôle multi-comptes qui exécutera l'KCLapplication.
2. Sur la console Amazon Kinesis Data Streams, choisissez le flux de données. Choisissez l'onglet Partage de flux de données, puis le bouton Créer une politique de partage pour démarrer l'éditeur visuel de politique. Pour partager un consommateur enregistré dans un flux de données, choisissez le consommateur, puis choisissez Créer une politique de partage. Vous pouvez également rédiger la JSON politique directement.
3. Spécifiez l'IAMutilisateur ou le IAM rôle de l'KCLapplication multicompte en tant que principal et les actions Kinesis Data Streams exactes auxquelles vous partagez l'accès. Pour plus d'informations sur les exemples de politiques de ressources pour Kinesis Data Streams, consultez [Exemples de politiques basées sur les ressources pour les flux de données Kinesis](#).
4. Choisissez Créer une politique ou utilisez le [PutResourcePolicy](#) pour associer la politique à votre ressource.

Partage de l'accès aux données chiffrées

Si vous avez activé le chiffrement côté serveur pour un flux de données avec KMS clé AWS gérée et que vous souhaitez partager l'accès via une politique de ressources, vous devez passer à l'utilisation de la clé gérée par le client (). CMK Pour de plus amples informations, veuillez consulter [Qu'est-ce](#)

[que le chiffrement côté serveur pour Kinesis Data Streams ?](#). En outre, vous devez autoriser vos entités principales de partage à accéder à votre compte CMK, en utilisant KMS les fonctionnalités de partage entre comptes. Assurez-vous de modifier également les IAM politiques relatives au partage des entités principales. Pour plus d'informations, voir [Autoriser les utilisateurs d'autres comptes à utiliser une KMS clé](#).

Validation de conformité pour Amazon Kinesis Data Streams

Des auditeurs tiers évaluent la sécurité et la conformité d'Amazon Kinesis Data Streams dans le cadre de AWS plusieurs programmes de conformité. Il s'agit SOC PCI notamment de RAMP la Fed HIPAA et d'autres.

Pour une liste des AWS services concernés par des programmes de conformité spécifiques, voir [AWS Services concernés par programme de conformité](#). Pour obtenir des informations générales, consultez [Programmes de conformité AWS](#).

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, consultez la section [Téléchargement de rapports dans AWS Artifact](#).

Votre responsabilité de conformité lors de l'utilisation de Kinesis Data Streams est déterminée par la sensibilité de vos données, les objectifs de conformité de votre entreprise, ainsi que par la législation et la réglementation applicables. Si votre utilisation de Kinesis Data Streams est soumise au respect de normes HIPAA telles que PCI, ou si la RAMP Fed AWS fournit des ressources pour vous aider à :

- [Guides de démarrage rapide sur la sécurité et la conformité](#) : ces guides de déploiement abordent les considérations architecturales et indiquent les étapes à suivre pour déployer des environnements de base axés sur la sécurité et la conformité sur AWS
- [Livre blanc sur l'architecture au service de la HIPAA sécurité et de la conformité](#) : ce livre blanc décrit comment les entreprises peuvent créer des applications AWS conformes. HIPAA
- [AWS Ressources de conformité](#) — Cette collection de classeurs et de guides susceptibles de s'appliquer à votre secteur d'activité et à votre région
- [AWS Config](#) — Ce AWS service évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- [AWS Security Hub](#) — Ce AWS service fournit une vue complète de l'état de votre sécurité interne, AWS ce qui vous permet de vérifier votre conformité aux normes et aux meilleures pratiques du secteur de la sécurité.

Résilience dans Amazon Kinesis Data Streams

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité. Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone de disponibilité à l'autre sans interruption. Les zones de disponibilité sont plus hautement disponibles, tolérantes aux pannes et évolutives que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section [Infrastructure AWS mondiale](#).

Outre l'infrastructure AWS mondiale, Kinesis Data Streams propose plusieurs fonctionnalités pour répondre à vos besoins en matière de résilience et de sauvegarde des données.

Reprise après sinistre dans Amazon Kinesis Data Streams

Un incident peut se produire aux niveaux suivants lorsque vous utilisez une Amazon Kinesis Data Streams pour traiter des données à partir d'un flux :

- Un processeur d'enregistrements peut échouer
- Une application de travail peut échouer, ou l'instance de l'application qui a instancié l'application de travail peut échouer
- Une EC2 instance hébergeant une ou plusieurs instances de l'application peut échouer

Défaillance du processeur d'enregistrement

Le travailleur invoque les méthodes du processeur d'enregistrements à l'aide de [ExecutorService](#) tâches Java. En cas d'échec d'une tâche, l'application de travail garde le contrôle de la partition que le processeur d'enregistrements était en train de traiter. L'application de travail démarre une nouvelle tâche de processeur d'enregistrements pour traiter cette partition. Pour de plus amples informations, veuillez consulter [Limitation de lecture](#).

Défaillance du travailleur ou de l'application

Si une application de travail ou une instance de Amazon Kinesis Data Streams échoue, vous devez détecter et gérer la situation. Par exemple, si la méthode `Worker.run` lève une exception, vous devez l'intercepter et la traiter.

En cas de défaillance de l'application elle-même, vous devez détecter cette défaillance et redémarrer l'application. Lorsque l'application démarre, elle instancie une nouvelle application de travail, qui à son tour instancie de nouveaux processeurs d'enregistrements auxquels sont attribuées automatiquement des partitions à traiter. Il peut s'agir des mêmes partitions que ces processeurs d'enregistrements traitaient avant la défaillance ou de partitions qui sont nouvelles pour ces processeurs.

Lorsque le programme de travail ou l'application échoue, que la défaillance n'est pas détectée et que d'autres instances de l'application s'exécutent sur d'autres EC2 instances, les travailleurs de ces autres instances gèrent l'échec. Elles créent d'autres processeurs d'enregistrements pour traiter les partitions qui ne sont plus traitées par l'application de travail en échec. La charge sur ces autres EC2 instances augmente en conséquence.

Le scénario décrit ici suppose que même si le programme de travail ou l'application a échoué, l'EC2 instance d'hébergement est toujours en cours d'exécution et n'est donc pas redémarrée par un groupe Auto Scaling.

EC2 Défaillance de l'instance Amazon

Nous vous recommandons d'exécuter les EC2 instances de votre application dans un groupe Auto Scaling. Ainsi, si l'une des EC2 instances échoue, le groupe Auto Scaling lance automatiquement une nouvelle instance pour la remplacer. Vous devez configurer les instances pour lancer votre application Amazon Kinesis Data Streams au démarrage.

Sécurité de l'infrastructure dans Kinesis Data Streams

En tant que service géré, Amazon Kinesis Data Streams est protégé par AWS les procédures de sécurité du réseau mondial décrites dans [le livre blanc Amazon Web Services : présentation des processus de sécurité](#).

Vous utilisez les API appels AWS publiés pour accéder à Kinesis Data Streams via le réseau. Les clients doivent prendre en charge Transport Layer Security (TLS) 1.2 ou version ultérieure. Les clients doivent également prendre en charge les suites de chiffrement parfaitement confidentielles (), telles que Ephemeral Diffie-Hellman (PFS) ou Elliptic Curve Ephemeral Diffie-Hellman (DHE). ECDHE La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un identifiant de clé d'accès et d'une clé d'accès secrète associés à un IAM principal. Vous pouvez également utiliser [AWS Security Token](#)

[Service](#) (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Bonnes pratiques en matière de sécurité pour Kinesis Data Streams

Amazon Kinesis Data Streams fournit différentes fonctions de sécurité à prendre en compte lorsque vous développez et implémentez vos propres politiques de sécurité. Les bonnes pratiques suivantes doivent être considérées comme des instructions générales et ne représentent pas une solution de sécurité complète. Étant donné que ces bonnes pratiques peuvent ne pas être appropriées ou suffisantes pour votre environnement, considérez-les comme des remarques utiles plutôt que comme des recommandations.

Implémentation d'un accès sur la base du moindre privilège

Lorsque vous accordez des autorisations, vous sélectionnez qui obtient les autorisations pour telles ou telles ressources Kinesis Data Streams. Vous activez des actions spécifiques que vous souhaitez autoriser sur ces ressources. Par conséquent, vous devez accorder uniquement les autorisations qui sont requises pour exécuter une tâche. L'implémentation d'un accès sur la base du moindre privilège est fondamentale pour réduire les risques en matière de sécurité et l'impact que pourraient avoir des d'erreurs ou des actes de malveillance.

Utiliser IAM les rôles

Les applications client et de type producteur doivent disposer d'informations d'identification valides pour accéder aux flux de données Kinesis. Vous ne devez pas stocker les AWS informations d'identification directement dans une application cliente ou dans un compartiment Amazon S3. Il s'agit d'autorisations à long terme qui ne font pas automatiquement l'objet d'une rotation et qui pourraient avoir un impact commercial important si elles étaient compromises.

Vous devez plutôt utiliser un IAM rôle pour gérer les informations d'identification temporaires permettant à vos applications productrices et clientes d'accéder aux flux de données Kinesis. Lorsque vous utilisez un rôle, vous n'avez pas à utiliser d'informations d'identification à long terme (par exemple, un nom d'utilisateur et un mot de passe ou des clés d'accès) pour accéder à d'autres ressources.

Pour plus d'informations, consultez les rubriques suivantes du guide de l'IAM utilisateur :

- [IAM Rôles](#)
- [Scénarios courants pour les rôles : utilisateurs, applications et services.](#)

Implémenter le chiffrement côté serveur dans les ressources dépendantes

Les données au repos et les données en transit peuvent être chiffrées dans Kinesis Data Streams. Pour de plus amples informations, veuillez consulter [Protection des données dans Amazon Kinesis Data Streams](#).

CloudTrail À utiliser pour surveiller les API appels

Kinesis Data Streams est intégré AWS CloudTrail à un service qui fournit un enregistrement des actions effectuées par un utilisateur, un rôle ou AWS un service dans Kinesis Data Streams.

À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à Kinesis Data Streams, l'adresse IP à partir de laquelle la demande a été effectuée, l'auteur de la demande, la date à laquelle elle a été faite, ainsi que des informations supplémentaires.

Pour de plus amples informations, veuillez consulter [the section called “Enregistrez les appels Amazon Kinesis Data API Streams avec AWS CloudTrail”](#).

Historique du document

Le tableau suivant décrit les modifications importantes apportées à la documentation dans cette version d'Amazon Kinesis Data Streams.

Modification	Description	Date de modification
Ajout de la prise en charge du partage des flux de données entre les comptes.	Ajouté Partagez votre flux de données avec un autre compte.	22 novembre 2023
Ajout de la prise en charge des modes de capacité de flux de données à la demande et provisionnés.	Ajouté Choisissez le mode de capacité du flux de données.	29 novembre 2021
Nouveau contenu pour le chiffrement côté serveur.	Ajouté Protection des données dans Amazon Kinesis Data Streams.	7 juillet 2017
Nouveau contenu pour des CloudWatch indicateurs améliorés.	Mis à jour Surveillez Kinesis Data Streams.	19 avril 2016
Nouveautés pour agent Kinesis amélioré.	Mis à jour Écrire sur Amazon Kinesis Data Streams à l'aide de Kinesis Agent.	11 avril 2016

Modification	Description	Date de modification
Nouveautés pour l'utilisation d'agents Kinesis.	Ajouté Écrire sur Amazon Kinesis Data Streams à l'aide de Kinesis Agent.	2 octobre 2015
KPLContenu de mise à jour pour la version 0.10.0.	Ajouté Développez des producteurs à l'aide de la bibliothèque Amazon Kinesis Producer () KPL.	15 juillet 2015
Mettre à jour la rubrique KCL des métriques pour les métriques configurables.	Ajouté Surveillez la bibliothèque cliente Kinesis avec Amazon CloudWatch.	9 juillet 2015
Contenu réorganisé.	Rubriques au contenu réorganisé offrant une vue d'arborescence plus concise et un regroupement plus logique.	01 juillet 2015
Nouveau sujet du guide du KPL développeur.	Ajouté Développez des producteurs à l'aide de la bibliothèque Amazon Kinesis Producer () KPL.	02 juin 2015
Nouveau sujet KCL sur les métriques.	Ajouté Surveillez la bibliothèque cliente Kinesis avec Amazon CloudWatch.	19 mai 2015
Support pourKCL.NET	Ajouté Développez un client de la bibliothèque cliente Kinesis dans .NET.	1 mai 2015
Support pour KCL Node.js	Ajouté Développez un client de bibliothèque cliente Kinesis dans Node.js.	26 mars 2015
Support pour KCL Ruby	Ajout de liens vers la bibliothèque KCL Ruby.	12 janvier 2015

Modification	Description	Date de modification
Nouveau API PutRecords	Ajout d'informations sur les nouveautés PutRecords API de the section called "Ajoutez plusieurs enregistrements avec PutRecords" .	15 décembre 2014
Prise en charge du balisage	Ajouté Ajoutez des balises à vos flux dans Amazon Kinesis Data Streams .	11 septembre 2014
Nouvelle CloudWatch métrique	Métrique GetRecords.IteratorAgeMilli seconds ajoutée à Dimensions et statistiques d'Amazon Kinesis Data Streams .	3 septembre 2014
Nouveau chapitre sur la surveillance	Ajout de Surveillez Kinesis Data Streams et de Surveillez le service Amazon Kinesis Data Streams avec Amazon CloudWatch .	30 juillet 2014
Nombre limite de partitions par défaut	Mis à jour Quotas et limites : le nombre limite de partitions a été porté de 5 à 10.	25 février 2014
Nombre limite de partitions par défaut	Mis à jour Quotas et limites : le nombre limite de partitions a été porté de 2 à 5.	28 janvier 2014
API mises à jour de version	Mises à jour pour la version 2013-12-02 de Kinesis Data Streams. API	12 décembre 2013
Première version	Première version du Guide du développeur Amazon Kinesis.	14 novembre 2013

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.