OpenShift Container Platform 4.12

Sans serveur

Installation, utilisation et notes de version d'OpenShift Serverless

Last Updated: 2023-05-03

OpenShift Container Platform 4.12 Sans serveur

Installation, utilisation et notes de version d'OpenShift Serverless

Notice légale

Résumé

Ce document fournit des informations sur l'utilisation d'OpenShift Serverless dans OpenShift Container Platform.

Table des matières

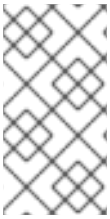
CHAPITRE 1. NOTES DE MISE À JOUR

NOTE

Pour plus d'informations sur le cycle de vie d'OpenShift Serverless et les plateformes prises en charge, consultez la Politique relative au cycle de vie des plateformes .

Les notes de version contiennent des informations sur les fonctionnalités nouvelles et obsolètes, les changements de rupture et les problèmes connus. Les notes de version suivantes s'appliquent aux versions les plus récentes d'OpenShift Serverless sur OpenShift Container Platform.

Pour une vue d'ensemble des fonctionnalités d'OpenShift Serverless, voir À propos d'OpenShift Serverless.



NOTE

OpenShift Serverless est basé sur le projet open source Knative.

Pour plus de détails sur les dernières versions des composants Knative, consultez le blog Knative.

1.1. À PROPOS DES VERSIONS DE L'API

Les versions d'API sont une mesure importante de l'état de développement de certaines fonctionnalités et ressources personnalisées dans OpenShift Serverless. La création de ressources sur votre cluster qui n'utilisent pas la bonne version d'API peut entraîner des problèmes dans votre déploiement.

L'OpenShift Serverless Operator met automatiquement à jour les anciennes ressources qui utilisent des versions obsolètes d'API pour utiliser la dernière version. Par exemple, si vous avez créé des ressources sur votre cluster qui utilisent des versions plus anciennes de l'API **ApiServerSource**, telles que **v1beta1**, l'OpenShift Serverless Operator met automatiquement à jour ces ressources pour utiliser la version **v1** de l'API lorsqu'elle est disponible et que la version **v1beta1** est obsolète.

Une fois qu'elles sont devenues obsolètes, les anciennes versions des API peuvent être supprimées dans une prochaine version. L'utilisation de versions obsolètes des API n'entraîne pas l'échec des ressources. Toutefois, si vous essayez d'utiliser une version d'une API qui a été supprimée, les ressources échoueront. Assurez-vous que vos manifestes sont mis à jour pour utiliser la dernière version afin d'éviter tout problème.

1.2. FONCTIONNALITÉS DISPONIBLES EN GÉNÉRAL ET EN AVANT-PREMIÈRE TECHNOLOGIQUE

Les fonctionnalités qui sont généralement disponibles (GA) sont entièrement prises en charge et conviennent à une utilisation en production. Les fonctionnalités de Technology Preview (TP) sont des fonctionnalités expérimentales et ne sont pas destinées à une utilisation en production. Consultez l'étendue de l'assistance de l'aperçu technologique sur le portail client de Red Hat pour plus d'informations sur les fonctionnalités TP.

Le tableau suivant fournit des informations sur les fonctionnalités OpenShift Serverless qui sont GA et celles qui sont TP :

Tableau 1.1. Suivi des caractéristiques de la version générale et de la version préliminaire de la technologie

Fonctionnalité	1.26	1.27	1.28
kn func	GA	GA	GA
Fonctions de Quarkus	GA	GA	GA
Fonctions Node.js	TP	TP	GA
Fonctions TypeScript	TP	TP	GA
Fonctions Python	-	-	TP
Service Mesh mTLS	GA	GA	GA
emptyDir volumes	GA	GA	GA
Redirection HTTPS	GA	GA	GA
Courtier Kafka	GA	GA	GA
Puits Kafka	GA	GA	GA
Support des conteneurs Init pour les services Knative	GA	GA	GA
Soutien du PVC aux services Knative	GA	GA	GA
TLS pour le trafic interne	TP	TP	TP
Courtiers à espace de nommage	-	TP	TP
multi-container support	-	-	TP

1.3. FONCTIONNALITÉS OBSOLÈTES ET SUPPRIMÉES

Certaines fonctionnalités qui étaient généralement disponibles (GA) ou un aperçu technologique (TP) dans les versions précédentes ont été dépréciées ou supprimées. Les fonctionnalités dépréciées sont toujours incluses dans OpenShift Serverless et continuent d'être prises en charge ; cependant, elles seront supprimées dans une prochaine version de ce produit et ne sont pas recommandées pour les nouveaux déploiements.

Pour la liste la plus récente des principales fonctionnalités dépréciées et supprimées dans OpenShift Serverless, reportez-vous au tableau suivant :

Tableau 1.2. Suivi des fonctionnalités obsolètes et supprimées

Fonctionnalité	1.20	1.21	1.22 à 1.26	1.27	1.28
KafkaBinding API	Déclassé	Déclassé	Supprimé	Supprimé	Supprimé
kn func emit (**kn func invoke** dans 1.21)	Déclassé	Supprimé	Supprimé	Supprimé	Supprimé
Serving and Eventing **v1alpha1** API	-	-	-	Déclassé	Déclassé
enable-secret-informer-filtering annotation	-	-	-	-	Déclassé

1.4. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.28

OpenShift Serverless 1.28 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.4.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.7.
- OpenShift Serverless utilise désormais Knative Eventing 1.7.
- OpenShift Serverless utilise désormais Kourier 1.7.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.7.
- OpenShift Serverless utilise désormais l'implémentation du courtier Knative pour Apache Kafka 1.7.
- Le plug-in **kn func** CLI utilise désormais la version 1.9.1 de **func**.
- Les moteurs d'exécution Node.js et TypeScript pour OpenShift Serverless Functions sont désormais généralement disponibles (GA).
- Le runtime Python pour OpenShift Serverless Functions est désormais disponible en tant qu'aperçu technologique.
- Le support multi-conteneurs pour Knative Serving est maintenant disponible en tant qu'aperçu technologique. Cette fonctionnalité vous permet d'utiliser un seul service Knative pour déployer un pod multi-conteneurs.
- Dans OpenShift Serverless 1.29 ou plus récent, les composants suivants de Knative Eventing seront réduits de deux pods à un seul :
  - **imc-controller**
  - **imc-dispatcher**
  - **mt-broker-controller**

- **mt-broker-filter**
- **mt-broker-ingress**

- L'annotation **serverless.openshift.io/enable-secret-informer-filtering** pour le CR Serving est maintenant obsolète. L'annotation n'est valable que pour Istio, et non pour Kourier.
  Avec OpenShift Serverless 1.28, l'opérateur OpenShift Serverless permet d'injecter la variable d'environnement **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** pour **net-istio** et **net-kourier**.

  Si vous activez le filtrage des secrets, tous vos secrets doivent être étiquetés avec **networking.internal.knative.dev/certificate-uid: "<id>"**. Sinon, Knative Serving ne les détecte pas, ce qui entraîne des échecs. Vous devez étiqueter à la fois les nouveaux secrets et les secrets existants.

  Dans l'une des prochaines versions d'OpenShift Serverless, le filtrage des secrets sera activé par défaut. Pour éviter les échecs, étiquetez vos secrets à l'avance.

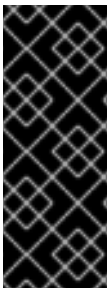
1.4.2. Problèmes connus

- Actuellement, les runtimes pour Python ne sont pas pris en charge pour OpenShift Serverless Functions sur IBM Power, IBM zSystems et IBM® LinuxONE.
  Les fonctions Node.js, TypeScript et Quarkus sont prises en charge sur ces architectures.

- Sur la plateforme Windows, les fonctions Python ne peuvent pas être construites, exécutées ou déployées localement à l'aide du constructeur Source-to-Image en raison des autorisations du fichier **app.sh**.
  Pour contourner ce problème, utilisez le sous-système Windows pour Linux.

1.5. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.27

OpenShift Serverless 1.27 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.



IMPORTANT

OpenShift Serverless 1.26 est la première version entièrement supportée par OpenShift Container Platform 4.12. OpenShift Serverless 1.25 et les versions antérieures ne se déploient pas sur OpenShift Container Platform 4.12.

Pour cette raison, avant de mettre à niveau OpenShift Container Platform vers la version 4.12, il faut d'abord mettre à niveau OpenShift Serverless vers la version 1.26 ou 1.27.

1.5.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.6.
- OpenShift Serverless utilise désormais Knative Eventing 1.6.
- OpenShift Serverless utilise désormais Kourier 1.6.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.6.
- OpenShift Serverless utilise désormais Knative Kafka 1.6.

- Le plug-in CLI **kn func** utilise désormais **func** 1.8.1.
- Les courtiers à espace de nommage sont désormais disponibles en tant qu'aperçu technologique. Ces courtiers peuvent être utilisés, par exemple, pour mettre en œuvre des politiques de contrôle d'accès basées sur les rôles (RBAC).
- **KafkaSink** utilise désormais par défaut le mode de contenu binaire **CloudEvent**. Le mode de contenu binaire est plus efficace que le mode structuré car il utilise des en-têtes dans son corps au lieu d'un **CloudEvent**. Par exemple, pour le protocole HTTP, il utilise les en-têtes HTTP.
- Vous pouvez désormais utiliser le framework gRPC sur le protocole HTTP/2 pour le trafic externe en utilisant OpenShift Route sur OpenShift Container Platform 4.10 et plus. Cela améliore l'efficacité et la vitesse des communications entre le client et le serveur.
- La version de l'API **v1alpha1** des CRDs Knative Operator Serving et Eventings est obsolète dans la version 1.27. Elle sera supprimée dans les versions futures. Red Hat recommande fortement d'utiliser la version **v1beta1** à la place. Cela n'affecte pas les installations existantes, car les CRD sont mis à jour automatiquement lors de la mise à niveau du Serverless Operator.
- La fonction de délai de livraison est désormais activée par défaut. Elle vous permet de spécifier le délai d'attente pour chaque requête HTTP envoyée. Cette fonctionnalité reste un aperçu technologique.

1.5.2. Problèmes corrigés

- Auparavant, les services Knative ne passaient pas toujours à l'état **Ready**, signalant qu'ils attendaient que l'équilibreur de charge soit prêt. Ce problème a été corrigé.

1.5.3. Problèmes connus

- L'intégration d'OpenShift Serverless avec Red Hat OpenShift Service Mesh fait en sorte que le pod **net-kourier** manque de mémoire au démarrage lorsque trop de secrets sont présents sur le cluster.
- Les courtiers à espace de noms peuvent laisser **ClusterRoleBindings** dans l'espace de noms de l'utilisateur même après la suppression des courtiers à espace de noms.
  Si cela se produit, supprimez le site **ClusterRoleBinding** nommé **rbac-proxy-reviews-prom-rb-knative-kafka-broker-data-plane-{{.Namespace}}** dans l'espace de noms de l'utilisateur.
- Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.
  Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

1.6. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.26

OpenShift Serverless 1.26 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.6.1. Nouvelles fonctionnalités

- OpenShift Serverless Functions with Quarkus est maintenant GA.
- OpenShift Serverless utilise désormais Knative Serving 1.5.
- OpenShift Serverless utilise désormais Knative Eventing 1.5.
- OpenShift Serverless utilise désormais Kourier 1.5.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.5.
- OpenShift Serverless utilise désormais Knative Kafka 1.5.
- OpenShift Serverless utilise désormais Knative Operator 1.3.
- Le plugin CLI **kn func** utilise désormais **func** 1.8.1.
- Les réclamations de volumes persistants (PVC) sont désormais des GA. Les PVC fournissent un stockage permanent des données pour vos services Knative.
- La nouvelle fonctionnalité des filtres de déclenchement est désormais disponible en avant-première pour les développeurs. Elle permet aux utilisateurs de spécifier un ensemble d'expressions de filtrage, où chaque expression est évaluée comme vraie ou fausse pour chaque événement.
  Pour activer les nouveaux filtres de déclenchement, ajoutez l'entrée **new-trigger-filters: enabled** dans la section du type **KnativeEventing** dans la carte de configuration de l'opérateur :

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
...
...
spec:
  config:
    features:
      new-trigger-filters: enabled
...
```

- Knative Operator 1.3 ajoute la version mise à jour de **v1beta1** de l'API pour **operator.knative.dev**.
  Pour passer de **v1alpha1** à **v1beta1** dans vos cartes de configuration des ressources personnalisées **KnativeServing** et **KnativeEventing**, modifiez la clé **apiVersion**:

  Exemple KnativeServing carte de configuration des ressources personnalisées

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
...
```

  Exemple KnativeEventing carte de configuration des ressources personnalisées

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
...
```

1.6.2. Problèmes corrigés

- Auparavant, le mode FIPS (Federal Information Processing Standards) était désactivé pour le courtier Kafka, la source Kafka et le puits Kafka. Ce problème a été corrigé et le mode FIPS est désormais disponible.

1.6.3. Problèmes connus

- Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.
  Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

Ressources supplémentaires

- Documentation Knative sur les nouveaux filtres de déclenchement

1.7. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.25.0

OpenShift Serverless 1.25.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.7.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.4.
- OpenShift Serverless utilise désormais Knative Eventing 1.4.
- OpenShift Serverless utilise désormais Kourier 1.4.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.4.
- OpenShift Serverless utilise désormais Knative Kafka 1.4.
- Le plugin CLI **kn func** utilise désormais **func** 1.7.0.
- Des plugins d'environnement de développement intégré (IDE) pour la création et le déploiement de fonctions sont désormais disponibles pour Visual Studio Code et IntelliJ.
- Knative Kafka broker est maintenant GA. Knative Kafka broker est une implémentation très performante de l'API Knative broker, ciblant directement Apache Kafka.
  Il est recommandé de ne pas utiliser le MT-Channel-Broker, mais plutôt le Knative Kafka Broker.
- Le Knative Kafka sink est maintenant GA. Un **KafkaSink** prend un **CloudEvent** et l'envoie à un sujet Apache Kafka. Les événements peuvent être spécifiés en mode de contenu structuré ou binaire.
- L'activation de TLS pour le trafic interne est désormais disponible en tant qu'aperçu technologique.

1.7.2. Problèmes corrigés

- Auparavant, Knative Serving avait un problème où la sonde de préparation échouait si le conteneur était redémarré après un échec de la sonde de durée de vie. Ce problème a été corrigé.

1.7.3. Problèmes connus

- Le mode Federal Information Processing Standards (FIPS) est désactivé pour le courtier Kafka, la source Kafka et le puits Kafka.
- L'objet **SinkBinding** ne prend pas en charge les noms de révision personnalisés pour les services.
- Le pod Knative Serving Controller ajoute un nouvel informateur pour surveiller les secrets dans le cluster. L'informateur inclut les secrets dans le cache, ce qui augmente la consommation de mémoire du pod contrôleur.
  Si le module manque de mémoire, vous pouvez contourner le problème en augmentant la limite de mémoire pour le déploiement.
- Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.
  Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

Ressources supplémentaires

- Configuration de l'authentification TLS

1.8. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.24.0

OpenShift Serverless 1.24.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.8.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.3.
- OpenShift Serverless utilise désormais Knative Eventing 1.3.
- OpenShift Serverless utilise désormais Kourier 1.3.
- OpenShift Serverless utilise désormais Knative **kn** CLI 1.3.
- OpenShift Serverless utilise désormais Knative Kafka 1.3.
- Le plugin CLI **kn func** utilise désormais **func** 0.24.
- La prise en charge des services Knative par les conteneurs Init est désormais disponible de manière générale (GA).
- OpenShift Serverless logic est désormais disponible en tant que Developer Preview. Elle permet de définir des modèles de flux de travail déclaratifs pour gérer les applications sans serveur.

- Vous pouvez désormais utiliser le service de gestion des coûts avec OpenShift Serverless.

1.8.2. Problèmes corrigés

- L'intégration d'OpenShift Serverless avec Red Hat OpenShift Service Mesh fait en sorte que le pod **net-istio-controller** manque de mémoire au démarrage lorsque trop de secrets sont présents sur le cluster.
  Il est désormais possible d'activer le filtrage des secrets, ce qui permet à **net-istio-controller** de ne prendre en compte que les secrets portant une étiquette **networking.internal.knative.dev/certificate-uid**, réduisant ainsi la quantité de mémoire nécessaire.

- L'aperçu technologique d'OpenShift Serverless Functions utilise désormais par défaut les Buildpacks Cloud Native pour construire des images de conteneurs.

1.8.3. Problèmes connus

- Le mode Federal Information Processing Standards (FIPS) est désactivé pour le courtier Kafka, la source Kafka et le puits Kafka.

- Dans OpenShift Serverless 1.23, le support pour KafkaBindings et le webhook **kafka-binding** ont été supprimés. Cependant, un **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** existant peut subsister, pointant vers le service **kafka-source-webhook**, qui n'existe plus.
  Pour certaines spécifications de KafkaBindings sur le cluster, **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** peut être configuré pour transmettre tout événement de création et de mise à jour à diverses ressources, telles que Deployments, Knative Services ou Jobs, par le biais du webhook, qui échouerait alors.

  Pour contourner ce problème, supprimez manuellement **kafkabindings.webhook.kafka.sources.knative.dev MutatingWebhookConfiguration** du cluster après avoir mis à jour OpenShift Serverless 1.23 :

  ```
  $ oc delete mutatingwebhookconfiguration kafkabindings.webhook.kafka.sources.knative.dev
  ```

- Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.
  Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

1.9. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.23.0

OpenShift Serverless 1.23.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.9.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.2.

- OpenShift Serverless utilise désormais Knative Eventing 1.2.

- OpenShift Serverless utilise désormais Kourier 1.2.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.2.
- OpenShift Serverless utilise désormais Knative Kafka 1.2.
- Le plugin CLI **kn func** utilise désormais **func** 0.24.
- Il est désormais possible d'utiliser l'annotation **kafka.eventing.knative.dev/external.topic** avec le courtier Kafka. Cette annotation permet d'utiliser un topic existant géré en externe au lieu que le broker crée son propre topic interne.
- Les composants Kafka **kafka-ch-controller** et **kafka-webhook** n'existent plus. Ils ont été remplacés par le composant **kafka-webhook-eventing**.
- L'aperçu technologique d'OpenShift Serverless Functions utilise désormais Source-to-Image (S2I) par défaut pour construire des images de conteneurs.

1.9.2. Problèmes connus

- Le mode Federal Information Processing Standards (FIPS) est désactivé pour le courtier Kafka, la source Kafka et le puits Kafka.
- Si vous supprimez un espace de noms qui inclut un courtier Kafka, la suppression du finalisateur de l'espace de noms peut échouer si le secret **auth.secret.ref.name** du courtier est supprimé avant le courtier.
- L'exécution d'OpenShift Serverless avec un grand nombre de services Knative peut faire en sorte que les pods activateurs Knative s'approchent de leur limite de mémoire par défaut de 600 Mo. Ces pods peuvent être redémarrés si la consommation de mémoire atteint cette limite. Les requêtes et les limites pour le déploiement de l'activateur peuvent être configurées en modifiant la ressource personnalisée **KnativeServing**:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  deployments:
  - name: activator
    resources:
    - container: activator
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
```

- Si vous utilisez Cloud Native Buildpacks comme stratégie de construction locale pour une fonction, **kn func** n'est pas en mesure de démarrer automatiquement podman ou d'utiliser un tunnel SSH vers un démon distant. Pour résoudre ces problèmes, il faut qu'un démon Docker ou podman soit déjà en cours d'exécution sur l'ordinateur de développement local avant de déployer une fonction.

- Les constructions de fonctions sur le cluster échouent actuellement pour les systèmes d'exécution Quarkus et Golang. Elles fonctionnent correctement pour les exécutions Node, Typescript, Python et Springboot.

- Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.
  Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

Ressources supplémentaires

- De la source à l'image

1.10. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.22.0

OpenShift Serverless 1.22.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.10.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.1.

- OpenShift Serverless utilise désormais Knative Eventing 1.1.

- OpenShift Serverless utilise désormais Kourier 1.1.

- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.1.

- OpenShift Serverless utilise désormais Knative Kafka 1.1.

- Le plugin CLI **kn func** utilise désormais **func** 0.23.

- Le support des conteneurs Init pour les services Knative est maintenant disponible en tant qu'aperçu technologique.

- La prise en charge des réclamations de volume persistantes (PVC) pour les services Knative est désormais disponible en tant qu'aperçu technologique.

- Les espaces de noms des systèmes **knative-serving**, **knative-serving-ingress**, **knative-eventing** et **knative-kafka** ont désormais l'étiquette **knative.openshift.io/part-of: "openshift-serverless"** par défaut.

- Le tableau de bord **Knative Eventing - Kafka Broker/Trigger** a été ajouté, ce qui permet de visualiser le courtier Kafka et les métriques de déclenchement dans la console web.

- Le tableau de bord **Knative Eventing - KafkaSink** a été ajouté, ce qui permet de visualiser les métriques de KafkaSink dans la console web.

- Le tableau de bord **Knative Eventing - Broker/Trigger** s'appelle désormais **Knative Eventing - Channel-based Broker/Trigger**.

- Le label **knative.openshift.io/part-of: "openshift-serverless"** a remplacé le label **knative.openshift.io/system-namespace**.

- Le style de nommage dans les fichiers de configuration YAML de Knative Serving est passé de la casse camel (**ExampleName**) au style trait d'union (**example-name**). À partir de cette version, utilisez la notation de style trait d'union lors de la création ou de l'édition des fichiers de configuration YAML de Knative Serving.

1.10.2. Problèmes connus

- Le mode Federal Information Processing Standards (FIPS) est désactivé pour le courtier Kafka, la source Kafka et le puits Kafka.

1.11. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.21.0

OpenShift Serverless 1.21.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.11.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 1.0
- OpenShift Serverless utilise désormais Knative Eventing 1.0.
- OpenShift Serverless utilise désormais Kourier 1.0.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 1.0.
- OpenShift Serverless utilise désormais Knative Kafka 1.0.
- Le plugin CLI **kn func** utilise désormais **func** 0.21.
- Le puits Kafka est maintenant disponible en tant qu'aperçu technologique.
- Le projet open source Knative a commencé à déprécier les clés de configuration à base de camel en faveur de l'utilisation cohérente de clés à base de kebab. Par conséquent, la clé **defaultExternalScheme**, précédemment mentionnée dans les notes de version 1.18.0 d'OpenShift Serverless, est maintenant obsolète et remplacée par la clé **default-external-scheme**. Les instructions d'utilisation de la clé restent les mêmes.

1.11.2. Problèmes corrigés

- Dans OpenShift Serverless 1.20.0, il y avait un problème de livraison d'événement affectant l'utilisation de **kn event send** pour envoyer des événements à un service. Ce problème est maintenant corrigé.
- Dans OpenShift Serverless 1.20.0 (**func** 0.20), les fonctions TypeScript créées avec le modèle **http** ne parvenaient pas à se déployer sur le cluster. Ce problème est maintenant corrigé.
- Dans OpenShift Serverless 1.20.0 (**func** 0.20), le déploiement d'une fonction utilisant le registre **gcr.io** échouait avec une erreur. Ce problème est maintenant corrigé.
- Dans OpenShift Serverless 1.20.0 (**func** 0.20), la création d'un répertoire de projet de fonction Springboot avec la commande **kn func create** et l'exécution de la commande **kn func build** ont échoué avec un message d'erreur. Ce problème est maintenant corrigé.

- Dans OpenShift Serverless 1.19.0 (**func** 0.19), certains runtimes étaient incapables de construire une fonction en utilisant podman. Ce problème est maintenant corrigé.

1.11.3. Problèmes connus

- Actuellement, le contrôleur de mappage de domaine ne peut pas traiter l'URI d'un courtier qui contient un chemin qui n'est actuellement pas pris en charge.
  Cela signifie que si vous souhaitez utiliser une ressource personnalisée (CR) **DomainMapping** pour mapper un domaine personnalisé à un courtier, vous devez configurer la CR **DomainMapping** avec le service d'entrée du courtier et ajouter le chemin d'accès exact du courtier au domaine personnalisé :

  Exemple DomainMapping CR

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: <domain-name>
  namespace: knative-eventing
spec:
  ref:
    name: broker-ingress
    kind: Service
    apiVersion: v1
```

  L'URI du courtier est alors **<domain-name>/<broker-namespace>/<broker-name>**.

1.12. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.20.0

OpenShift Serverless 1.20.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.12.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 0.26.
- OpenShift Serverless utilise désormais Knative Eventing 0.26.
- OpenShift Serverless utilise désormais Kourier 0.26.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 0.26.
- OpenShift Serverless utilise désormais Knative Kafka 0.26.
- Le plugin CLI **kn func** utilise désormais **func** 0.20.
- Le courtier Kafka est désormais disponible en tant qu'aperçu technologique.

IMPORTANT

Le courtier Kafka, qui est actuellement en avant-première technologique, n'est pas pris en charge par FIPS.

- Le plugin **kn event** est désormais disponible en tant qu'aperçu technologique.
- Les drapeaux **--min-scale** et **--max-scale** de la commande **kn service create** sont obsolètes. Utilisez plutôt les drapeaux **--scale-min** et **--scale-max**.

1.12.2. Problèmes connus

- OpenShift Serverless déploie les services Knative avec une adresse par défaut qui utilise HTTPS. Lors de l'envoi d'un événement à une ressource à l'intérieur du cluster, l'expéditeur n'a pas l'autorité de certification (CA) du cluster configurée. Cela entraîne l'échec de la livraison de l'événement, à moins que le cluster n'utilise des certificats globalement acceptés.
  Par exemple, la livraison d'un événement à une adresse accessible au public fonctionne :

  ```
  $ kn event send --to-url https://ce-api.foo.example.com/
  ```

  En revanche, cette livraison échoue si le service utilise une adresse publique avec un certificat HTTPS délivré par une autorité de certification personnalisée :

  ```
  $ kn event send --to Service:serving.knative.dev/v1:event-display
  ```

  L'envoi d'un événement à d'autres objets adressables, tels que les courtiers ou les canaux, n'est pas concerné par ce problème et fonctionne comme prévu.
- Le courtier Kafka ne fonctionne pas actuellement sur un cluster avec le mode Federal Information Processing Standards (FIPS) activé.
- Si vous créez un répertoire de projet de fonction Springboot avec la commande **kn func create**, l'exécution ultérieure de la commande **kn func build** échoue avec ce message d'erreur :

  ```
  [analyzer] no stack metadata found at path ''
  [analyzer] ERROR: failed to : set API for buildpack 'paketo-buildpacks/ca-certificates@3.0.2':
  buildpack API version '0.7' is incompatible with the lifecycle
  ```

  Pour contourner le problème, vous pouvez remplacer la propriété **builder** par **gcr.io/paketo-buildpacks/builder:base** dans le fichier de configuration de la fonction **func.yaml**.
- Le déploiement d'une fonction à l'aide du registre **gcr.io** échoue avec ce message d'erreur :

  ```
  Error: failed to get credentials: failed to verify credentials: status code: 404
  ```

  Pour contourner le problème, utilisez un registre différent de celui de **gcr.io**, tel que **quay.io** ou **docker.io**.
- Les fonctions TypeScript créées avec le modèle **http** ne se déploient pas sur le cluster.
  Pour contourner le problème, remplacez la section suivante dans le fichier **func.yaml**:

  ```
  buildEnvs: []
  ```

  avec ceci :

  ```
  buildEnvs:
  - name: BP_NODE_RUN_SCRIPTS
    value: build
  ```

- Dans **func** version 0.20, certains runtimes peuvent être incapables de construire une fonction en utilisant podman. Vous pouvez voir un message d'erreur similaire au suivant :

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - La solution suivante permet de résoudre ce problème :

    a. Mettez à jour le service podman en ajoutant **--time=0** à la définition du service **ExecStart**:

    Exemple de configuration de service

```
ExecStart=/usr/bin/podman $LOGGING system service --time=0
```

    b. Redémarrez le service podman en exécutant les commandes suivantes :

```
$ systemctl --user daemon-reload
```

```
$ systemctl restart --user podman.socket
```

  - Vous pouvez également exposer l'API podman en utilisant TCP :

```
$ podman system service --time=0 tcp:127.0.0.1:5534 &
export DOCKER_HOST=tcp://127.0.0.1:5534
```

1.13. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.19.0

OpenShift Serverless 1.19.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.13.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 0.25.
- OpenShift Serverless utilise désormais Knative Eventing 0.25.
- OpenShift Serverless utilise désormais Kourier 0.25.
- OpenShift Serverless utilise désormais Knative (**kn**) CLI 0.25.
- OpenShift Serverless utilise désormais Knative Kafka 0.25.
- Le plugin CLI **kn func** utilise désormais **func** 0.19.
- L'API **KafkaBinding** est obsolète dans OpenShift Serverless 1.19.0 et sera supprimée dans une prochaine version.
- La redirection HTTPS est maintenant supportée et peut être configurée soit globalement pour un cluster, soit pour chaque service Knative.

1.13.2. Problèmes corrigés

- Dans les versions précédentes, le répartiteur du canal Kafka n'attendait que le succès du commit local avant de répondre, ce qui pouvait entraîner la perte d'événements en cas de défaillance d'un nœud Apache Kafka. Le répartiteur du canal Kafka attend désormais que toutes les répliques synchronisées s'engagent avant de répondre.

1.13.3. Problèmes connus

- Dans **func** version 0.19, certains runtimes peuvent être incapables de construire une fonction en utilisant podman. Vous pouvez voir un message d'erreur similaire au suivant :

```
ERROR: failed to image: error during connect: Get
"http://%2Fvar%2Frun%2Fdocker.sock/v1.40/info": EOF
```

  - La solution suivante permet de résoudre ce problème :

    a. Mettez à jour le service podman en ajoutant **--time=0** à la définition du service **ExecStart**:

    Exemple de configuration de service

    ```
    ExecStart=/usr/bin/podman $LOGGING system service --time=0
    ```

    b. Redémarrez le service podman en exécutant les commandes suivantes :

    ```
    $ systemctl --user daemon-reload
    ```

    ```
    $ systemctl restart --user podman.socket
    ```

  - Vous pouvez également exposer l'API podman en utilisant TCP :

    ```
    $ podman system service --time=0 tcp:127.0.0.1:5534 &
    export DOCKER_HOST=tcp://127.0.0.1:5534
    ```

1.14. NOTES DE VERSION POUR RED HAT OPENSHIFT SERVERLESS 1.18.0

OpenShift Serverless 1.18.0 est maintenant disponible. Les nouvelles fonctionnalités, les changements et les problèmes connus qui concernent OpenShift Serverless sur OpenShift Container Platform sont inclus dans cette rubrique.

1.14.1. Nouvelles fonctionnalités

- OpenShift Serverless utilise désormais Knative Serving 0.24.0.
- OpenShift Serverless utilise désormais Knative Eventing 0.24.0.
- OpenShift Serverless utilise désormais Kourier 0.24.0.
- OpenShift Serverless utilise maintenant Knative (**kn**) CLI 0.24.0.
- OpenShift Serverless utilise désormais Knative Kafka 0.24.7.

- Le plugin CLI **kn func** utilise désormais **func** 0.18.0.

- Dans la prochaine version OpenShift Serverless 1.19.0, le schéma d'URL des routes externes sera par défaut HTTPS pour une sécurité accrue.
  Si vous ne souhaitez pas que cette modification s'applique à vos charges de travail, vous pouvez remplacer le paramètre par défaut avant de passer à la version 1.19.0, en ajoutant le fichier YAML suivant à votre ressource personnalisée (CR) **KnativeServing**:

```
...
spec:
  config:
    network:
      defaultExternalScheme: "http"
...
```

  Si vous voulez que le changement s'applique déjà à la version 1.18.0, ajoutez le YAML suivant :

```
...
spec:
  config:
    network:
      defaultExternalScheme: "https"
...
```

- Dans la prochaine version OpenShift Serverless 1.19.0, le type de service par défaut par lequel la passerelle Kourier est exposée sera **ClusterIP** et non **LoadBalancer**.
  Si vous ne souhaitez pas que cette modification s'applique à vos charges de travail, vous pouvez remplacer le paramètre par défaut avant la mise à niveau vers la version 1.19.0, en ajoutant le fichier YAML suivant à votre ressource personnalisée (CR) **KnativeServing**:

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

- Vous pouvez désormais utiliser les volumes **emptyDir** avec OpenShift Serverless. Voir la documentation OpenShift Serverless sur Knative Serving pour plus de détails.

- Les modèles Rust sont désormais disponibles lorsque vous créez une fonction à l'aide de **kn func**.

1.14.2. Problèmes corrigés

- La version précédente 1.4 de Camel-K n'était pas compatible avec OpenShift Serverless 1.17.0. Le problème dans Camel-K a été corrigé, et la version 1.4.1 de Camel-K peut être utilisée avec OpenShift Serverless 1.17.0.

- Auparavant, si vous créiez un nouvel abonnement pour un canal Kafka ou une nouvelle source Kafka, il était possible que le plan de données Kafka ne soit pas prêt à distribuer les messages après que l'abonnement ou le puits nouvellement créé ait signalé un statut prêt.
  Par conséquent, les messages envoyés pendant la période où le plan de données n'indiquait pas qu'il était prêt peuvent ne pas avoir été transmis à l'abonné ou au destinataire.

Dans OpenShift Serverless 1.18.0, le problème est corrigé et les messages initiaux ne sont plus perdus. Pour plus d'informations sur ce problème, voir l'article #6343981 de la base de connaissances.

1.14.3. Problèmes connus

- Les anciennes versions du CLI Knative **kn** peuvent utiliser d'anciennes versions des API Knative Serving et Knative Eventing. Par exemple, la version 0.23.2 du CLI **kn** utilise la version de l'API **v1alpha1**.
  D'autre part, les nouvelles versions d'OpenShift Serverless peuvent ne plus prendre en charge les anciennes versions de l'API. Par exemple, OpenShift Serverless 1.18.0 ne prend plus en charge la version **v1alpha1** de l'API **kafkasources.sources.knative.dev**.

  Par conséquent, l'utilisation d'une ancienne version de la CLI Knative **kn** avec une version plus récente d'OpenShift Serverless peut produire une erreur parce que **kn** ne peut pas trouver l'API obsolète. Par exemple, la version 0.23.2 du CLI **kn** ne fonctionne pas avec OpenShift Serverless 1.18.0.

  Pour éviter tout problème, utilisez la dernière version de **kn** CLI disponible pour votre version d'OpenShift Serverless. Pour OpenShift Serverless 1.18.0, utilisez Knative **kn** CLI 0.24.0.

CHAPITRE 2. À PROPOS DE SERVERLESS

2.1. APERÇU D'OPENSHIFT SERVERLESS

OpenShift Serverless fournit des blocs de construction natifs de Kubernetes qui permettent aux développeurs de créer et de déployer des applications sans serveur et axées sur les événements sur OpenShift Container Platform. OpenShift Serverless est basé sur le projet open source Knative, qui assure la portabilité et la cohérence pour les environnements hybrides et multicloud en permettant une plateforme serverless de niveau entreprise.

NOTE

Parce qu'OpenShift Serverless est publié à une cadence différente de celle d'OpenShift Container Platform, la documentation d'OpenShift Serverless ne maintient pas de jeux de documentation distincts pour les versions mineures du produit. La documentation actuelle s'applique à toutes les versions d'OpenShift Serverless actuellement prises en charge, à moins que des limitations spécifiques à une version ne soient indiquées dans un sujet particulier ou pour une fonctionnalité particulière.

Pour plus d'informations sur le cycle de vie d'OpenShift Serverless et les plateformes prises en charge, consultez la Politique relative au cycle de vie des plateformes .

2.1.1. Ressources supplémentaires

- Extension de l'API Kubernetes avec des définitions de ressources personnalisées
- Gestion des ressources à partir de définitions de ressources personnalisées
- Qu'est-ce que le "sans serveur" ?

2.2. SERVIR KNATIVE

Knative Serving soutient les développeurs qui souhaitent créer, déployer et gérer des applications cloud-natives. Il fournit un ensemble d'objets en tant que définitions de ressources personnalisées Kubernetes (CRD) qui définissent et contrôlent le comportement des charges de travail sans serveur sur un cluster OpenShift Container Platform.

Les développeurs utilisent ces CRD pour créer des instances de ressources personnalisées (CR) qui peuvent être utilisées comme blocs de construction pour répondre à des cas d'utilisation complexes. Par exemple :

- Déployer rapidement des conteneurs sans serveur.
- Mise à l'échelle automatique des pods.

2.2.1. Ressources de service Knative

Service

Le CRD **service.serving.knative.dev** gère automatiquement le cycle de vie de votre charge de travail pour s'assurer que l'application est déployée et accessible via le réseau. Il crée une route, une configuration et une nouvelle révision pour chaque changement apporté à un service créé par l'utilisateur ou à une ressource personnalisée. La plupart des interactions des développeurs dans Knative sont effectuées en modifiant les services.

Révision

Le CRD **revision.serving.knative.dev** est un instantané du code et de la configuration pour chaque modification apportée à la charge de travail. Les révisions sont des objets immuables et peuvent être conservées aussi longtemps que nécessaire.

Itinéraire

Le CRD **route.serving.knative.dev** établit une correspondance entre un point d'extrémité du réseau et une ou plusieurs révisions. Vous pouvez gérer le trafic de plusieurs manières, y compris le trafic fractionné et les itinéraires nommés.

Configuration

Le CRD **configuration.serving.knative.dev** maintient l'état souhaité pour votre déploiement. Il fournit une séparation nette entre le code et la configuration. La modification d'une configuration crée une nouvelle révision.

2.3. CONCOURS COMPLET D'ÉQUITATION

Knative Eventing sur OpenShift Container Platform permet aux développeurs d'utiliser une architecture pilotée par les événements avec des applications sans serveur. Une architecture pilotée par les événements est basée sur le concept de relations découplées entre les producteurs et les consommateurs d'événements.

Les producteurs d'événements créent des événements, et les événements *sinks*, ou consommateurs, reçoivent des événements. Knative Eventing utilise des requêtes HTTP POST standard pour envoyer et recevoir des événements entre les producteurs et les récepteurs d'événements. Ces événements sont conformes aux spécifications CloudEvents, qui permettent de créer, d'analyser, d'envoyer et de recevoir des événements dans n'importe quel langage de programmation.

Knative Eventing prend en charge les cas d'utilisation suivants :

Publier un événement sans créer de consommateur

Vous pouvez envoyer des événements à un courtier sous la forme d'un HTTP POST et utiliser la liaison pour découpler la configuration de destination de votre application qui produit les événements.

Consommer un événement sans créer d'éditeur

Vous pouvez utiliser un déclencheur pour consommer des événements à partir d'un courtier en fonction des attributs de l'événement. L'application reçoit les événements sous forme de HTTP POST.

Pour permettre la livraison à plusieurs types de puits, Knative Eventing définit les interfaces génériques suivantes qui peuvent être mises en œuvre par plusieurs ressources Kubernetes :

Ressources adressables

Capable de recevoir et d'accuser réception d'un événement transmis par HTTP à une adresse définie dans le champ **status.address.url** de l'événement. La ressource Kubernetes **Service** satisfait également à l'interface adressable.

Ressources appelables

Capable de recevoir un événement transmis par HTTP et de le transformer, en renvoyant **0** ou **1** nouveaux événements dans la charge utile de la réponse HTTP. Ces événements renvoyés peuvent être traités de la même manière que les événements provenant d'une source externe.

2.3.1. Utiliser le courtier Knative pour Apache Kafka

L'implémentation du courtier Knative pour Apache Kafka fournit des options d'intégration vous

permettant d'utiliser les versions prises en charge de la plateforme de streaming de messages Apache Kafka avec OpenShift Serverless. Kafka fournit des options pour la source d'événement, le canal, le courtier et les capacités de puits d'événement.

NOTE

L'implémentation du courtier Knative pour Apache Kafka n'est pas actuellement prise en charge pour les systèmes IBM zSystems et IBM Power.

Le courtier Knative pour Apache Kafka offre des options supplémentaires, telles que :

- Source Kafka
- Canal Kafka
- Courtier Kafka
- Puits Kafka

2.3.2. Ressources supplémentaires

- Installation de la ressource personnalisée **KnativeKafka** .
- Documentation de Red Hat AMQ Streams
- Documentation de Red Hat AMQ Streams TLS et SASL sur Apache Kafka
- Livraison de l'événement

2.4. À PROPOS D'OPENSHIFT SERVERLESS FUNCTIONS

OpenShift Serverless Functions permet aux développeurs de créer et de déployer des fonctions sans état et pilotées par des événements en tant que service Knative sur OpenShift Container Platform. Le CLI **kn func** est fourni en tant que plugin pour le CLI Knative **kn**. Vous pouvez utiliser la CLI **kn func** pour créer, construire et déployer l'image du conteneur en tant que service Knative sur le cluster.

2.4.1. Temps d'exécution inclus

OpenShift Serverless Functions fournit des modèles qui peuvent être utilisés pour créer des fonctions de base pour les runtimes suivants :

- Quarkus
- Node.js
- TypeScript

2.4.2. Prochaines étapes

- Commencer avec les fonctions.

CHAPITRE 3. INSTALLATION DE SERVERLESS

3.1. PRÉPARER L'INSTALLATION D'OPENSHIFT SERVERLESS

Lisez les informations suivantes sur les configurations prises en charge et les prérequis avant d'installer OpenShift Serverless.

- OpenShift Serverless est pris en charge pour une installation dans un environnement réseau restreint.
- OpenShift Serverless ne peut actuellement pas être utilisé dans une configuration multi-tenant sur un seul cluster.

3.1.1. Configurations prises en charge

L'ensemble des fonctionnalités, configurations et intégrations prises en charge pour OpenShift Serverless, versions actuelles et antérieures, est disponible sur la page Configurations prises en charge .

3.1.2. Évolutivité et performance

OpenShift Serverless a été testé avec une configuration de 3 nœuds principaux et 3 nœuds de travail, qui disposent chacun de 64 CPU, 457 Go de mémoire et 394 Go de stockage.

Le nombre maximal de services Knative pouvant être créés à l'aide de cette configuration est de 3 000. Cela correspond à la limite de 10 000 services Kubernetes de OpenShift Container Platform , puisque 1 service Knative crée 3 services Kubernetes.

Le temps de réponse moyen à partir de zéro était d'environ 3,4 secondes, avec un temps de réponse maximum de 8 secondes, et un 99,9ème percentile de 4,5 secondes pour une application Quarkus simple. Ces temps peuvent varier en fonction de l'application et de sa durée d'exécution.

3.1.3. Définition des exigences en matière de taille des grappes

Pour installer et utiliser OpenShift Serverless, le cluster OpenShift Container Platform doit être correctement dimensionné.

NOTE

Les exigences suivantes ne concernent que le pool de machines de travail du cluster OpenShift Container Platform. Les nœuds du plan de contrôle ne sont pas utilisés pour l'ordonnancement général et ne sont pas pris en compte dans les exigences.

Le minimum requis pour utiliser OpenShift Serverless est un cluster avec 10 CPUs et 40GB de mémoire. Par défaut, chaque pod demande ~400m de CPU, les exigences minimales sont donc basées sur cette valeur.

La taille totale requise pour exécuter OpenShift Serverless dépend des composants installés et des applications déployées, et peut varier en fonction de votre déploiement.

3.1.4. Mise à l'échelle de votre cluster à l'aide d'ensembles de machines de calcul

Vous pouvez utiliser l'API d'OpenShift Container Platform **MachineSet** pour augmenter manuellement la taille de votre cluster. Les exigences minimales signifient généralement que vous devez augmenter

l'un des ensembles de machines de calcul par défaut de deux machines supplémentaires. Voir Mise à l'échelle manuelle d'un ensemble de machines de calcul.

3.1.4.1. Exigences supplémentaires pour les cas d'utilisation avancés

Pour des cas d'utilisation plus avancés tels que le logging ou le metering sur OpenShift Container Platform, vous devez déployer plus de ressources. Les exigences recommandées pour de tels cas d'utilisation sont 24 CPU et 96 Go de mémoire.

Si vous avez activé la haute disponibilité (HA) sur votre cluster, cela nécessite entre 0,5 et 1,5 cœurs et entre 200 Mo et 2 Go de mémoire pour chaque réplique du plan de contrôle de Knative Serving. HA est activé par défaut pour certains composants de Knative Serving. Vous pouvez désactiver HA en suivant la documentation sur "Configuring high availability replicas".

3.1.5. Ressources supplémentaires

- Using Operator Lifecycle Manager on restricted networks
- Comprendre OperatorHub
- Capacités des clusters

3.2. INSTALLATION DE L'OPÉRATEUR OPENSHIFT SERVERLESS

L'installation de l'OpenShift Serverless Operator vous permet d'installer et d'utiliser Knative Serving, Knative Eventing et le courtier Knative pour Apache Kafka sur un cluster OpenShift Container Platform. OpenShift Serverless Operator gère les définitions de ressources personnalisées (CRD) Knative pour votre cluster et vous permet de les configurer sans modifier directement les cartes de configuration individuelles pour chaque composant.

3.2.1. Installer l'opérateur OpenShift Serverless depuis la console web

Vous pouvez installer l'Opérateur OpenShift Serverless depuis l'OperatorHub en utilisant la console web d'OpenShift Container Platform. L'installation de cet opérateur vous permet d'installer et d'utiliser des composants Knative.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Votre cluster a la capacité Marketplace activée ou la source du catalogue Red Hat Operator configurée manuellement.
- Vous vous êtes connecté à la console web de OpenShift Container Platform.

Procédure

1. Dans la console web d'OpenShift Container Platform, naviguez jusqu'à la page **Operators → OperatorHub**.
2. Faites défiler ou tapez le mot-clé **Serverless** dans la boîte **Filter by keyword** pour trouver l'OpenShift Serverless Operator.
3. Examinez les informations relatives à l'opérateur et cliquez sur **Install**.

4. Sur la page **Install Operator**:

    a. L'adresse **Installation Mode** est **All namespaces on the cluster (default)** Ce mode installe l'opérateur dans l'espace de noms par défaut **openshift-serverless** afin qu'il soit surveillé et mis à la disposition de tous les espaces de noms du cluster.

    b. Le site **Installed Namespace** est **openshift-serverless**.

    c. Sélectionnez le canal **stable** comme **Update Channel**. Le canal **stable** permettra l'installation de la dernière version stable d'OpenShift Serverless Operator.

    d. Sélectionnez la stratégie d'approbation **Automatic** ou **Manual**.

5. Cliquez sur **Install** pour rendre l'opérateur disponible pour les espaces de noms sélectionnés sur ce cluster OpenShift Container Platform.

6. Depuis la page **Catalog → Operator Management**, vous pouvez surveiller la progression de l'installation et de la mise à niveau de l'abonnement OpenShift Serverless Operator.

    a. Si vous avez sélectionné une stratégie d'approbation **Manual**, l'état de mise à niveau de l'abonnement restera **Upgrading** jusqu'à ce que vous examiniez et approuviez son plan d'installation. Après approbation sur la page **Install Plan**, le statut de mise à niveau de l'abonnement passe à **Up to date**.

    b. Si vous avez sélectionné une stratégie d'approbation **Automatic**, le statut du surclassement devrait être résolu à **Up to date** sans intervention.

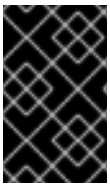
Vérification

Une fois que l'état de mise à niveau de l'abonnement est **Up to date**, sélectionnez **Catalog → Installed Operators** pour vérifier que l'opérateur OpenShift Serverless finit par apparaître et que son **Status** se résout finalement en **InstallSucceeded** dans l'espace de noms concerné.

Si ce n'est pas le cas :

1. Passez à la page **Catalog → Operator Management** et inspectez les onglets **Operator Subscriptions** et **Install Plans** pour voir s'il n'y a pas de défaillance ou d'erreur sous **Status**.

2. Vérifiez les journaux de tous les pods du projet **openshift-serverless** sur la page **Workloads → Pods** qui signalent des problèmes afin de les résoudre.



IMPORTANT

Si vous souhaitez utiliser Red Hat OpenShift distributed tracing avec OpenShift Serverless, vous devez installer et configurer Red Hat OpenShift distributed tracing avant d'installer Knative Serving ou Knative Eventing.

3.2.2. Installer l'opérateur OpenShift Serverless depuis le CLI

Vous pouvez installer l'Opérateur OpenShift Serverless depuis le OperatorHub en utilisant le CLI. L'installation de cet opérateur vous permet d'installer et d'utiliser des composants Knative.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.

- Votre cluster a la capacité Marketplace activée ou la source du catalogue Red Hat Operator configurée manuellement.
- Vous vous êtes connecté au cluster OpenShift Container Platform.

Procédure

1. Créez un fichier YAML contenant les objets **Namespace**, **OperatorGroup**, et **Subscription** pour abonner un espace de noms à l'opérateur OpenShift Serverless. Par exemple, créez le fichier **serverless-subscription.yaml** avec le contenu suivant :

   #### Exemple d'abonnement

   ```
   ---
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-serverless
   ---
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: serverless-operators
     namespace: openshift-serverless
   spec: {}
   ---
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: serverless-operator
     namespace: openshift-serverless
   spec:
     channel: stable (1)
     name: serverless-operator (2)
     source: redhat-operators (3)
     sourceNamespace: openshift-marketplace (4)
   ```

   (1) Le nom du canal de l'opérateur. Le canal **stable** permet l'installation de la version stable la plus récente de l'Opérateur OpenShift Serverless.

   (2) Le nom de l'opérateur auquel s'abonner. Pour l'opérateur OpenShift Serverless, il s'agit toujours de **serverless-operator**.

   (3) Le nom du CatalogSource qui fournit l'opérateur. Utilisez **redhat-operators** pour les sources de catalogue par défaut d'OperatorHub.

   (4) L'espace de noms du CatalogSource. Utilisez **openshift-marketplace** pour les sources de catalogue par défaut d'OperatorHub.

2. Créer l'objet **Subscription**:

   ```
   $ oc apply -f serverless-subscription.yaml
   ```

Vérification

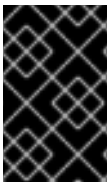
Vérifiez que la version du service de cluster (CSV) a atteint la phase **Succeeded**:

Example command

```
$ oc get csv
```

Exemple de sortie

```
NAME                                DISPLAY                       VERSION   REPLACES                    PHASE
serverless-operator.v1.25.0   Red Hat OpenShift Serverless   1.25.0    serverless-operator.v1.24.0
Succeeded
```



IMPORTANT

Si vous souhaitez utiliser Red Hat OpenShift distributed tracing avec OpenShift Serverless, vous devez installer et configurer Red Hat OpenShift distributed tracing avant d'installer Knative Serving ou Knative Eventing.

3.2.3. Configuration globale

L'opérateur OpenShift Serverless gère la configuration globale d'une installation Knative, y compris la propagation des valeurs à partir de **KnativeServing** et **KnativeEventing** ressources personnalisées aux cartes de configuration du système. Toute mise à jour des cartes de configuration appliquée manuellement est écrasée par l'Opérateur. Cependant, la modification des ressources personnalisées Knative vous permet de définir des valeurs pour ces cartes de configuration.

Knative dispose de plusieurs cartes de configuration qui sont nommées avec le préfixe **config-**. Toutes les cartes de configuration Knative sont créées dans le même espace de noms que la ressource personnalisée à laquelle elles s'appliquent. Par exemple, si la ressource personnalisée **KnativeServing** est créée dans l'espace de noms **knative-serving**, toutes les cartes de configuration Knative Serving sont également créées dans cet espace de noms.

Les ressources personnalisées Knative **spec.config** ont une entrée **<name>** pour chaque carte de configuration, nommée **config-<name>**, avec une valeur qui sera utilisée pour la carte de configuration **data**.

3.2.4. Ressources supplémentaires

- Gestion des ressources à partir de définitions de ressources personnalisées
- Comprendre le stockage persistant
- Configuration d'une ICP personnalisée

3.2.5. Prochaines étapes

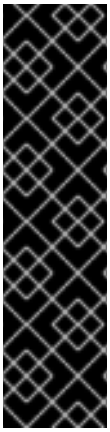
- Une fois l'OpenShift Serverless Operator installé, vous pouvez installer Knative Serving ou Knative Eventing.

3.3. INSTALLATION DU CLI KNATIVE

Le CLI de Knative (**kn**) n'a pas son propre mécanisme de connexion. Pour vous connecter au cluster, vous devez installer la CLI OpenShift (**oc**) et utiliser la commande **oc login**. Les options d'installation des CLI peuvent varier en fonction de votre système d'exploitation.

Pour plus d'informations sur l'installation de l'OpenShift CLI (**oc**) pour votre système d'exploitation et la connexion avec **oc**, consultez la documentation de démarrage de l'OpenShift CLI .

OpenShift Serverless ne peut pas être installé à l'aide de la CLI Knative (**kn**). Un administrateur de cluster doit installer l'opérateur OpenShift Serverless et configurer les composants Knative, comme décrit dans la documentation Installation de l'opérateur OpenShift Serverless .



IMPORTANT

Si vous essayez d'utiliser une ancienne version du CLI Knative (**kn**) avec une version plus récente d'OpenShift Serverless, l'API n'est pas trouvée et une erreur se produit.

Par exemple, si vous utilisez la version 1.23.0 du CLI Knative (**kn**), qui utilise la version 1.2, avec la version 1.24.0 d'OpenShift Serverless, qui utilise les versions 1.3 des API Knative Serving et Knative Eventing, le CLI ne fonctionne pas parce qu'il continue à rechercher les versions dépassées de l'API 1.2.

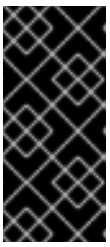
Assurez-vous que vous utilisez la dernière version du CLI Knative (**kn**) pour votre version OpenShift Serverless afin d'éviter les problèmes.

3.3.1. Installer le CLI Knative à l'aide de la console web de OpenShift Container Platform

L'utilisation de la console web d'OpenShift Container Platform fournit une interface utilisateur rationalisée et intuitive pour installer le CLI Knative (**kn**). Après l'installation d'OpenShift Serverless Operator, vous verrez un lien pour télécharger la CLI Knative (**kn**) pour Linux (amd64, s390x, ppc64le), macOS ou Windows à partir de la page **Command Line Tools** dans la console web d'OpenShift Container Platform.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster OpenShift Container Platform.



IMPORTANT

Si **libc** n'est pas disponible, vous risquez d'obtenir l'erreur suivante lorsque vous exécutez des commandes CLI :

```
$ kn: No such file or directory
```

- Si vous souhaitez utiliser les étapes de vérification pour cette procédure, vous devez installer le CLI OpenShift (**oc**).

Procédure

1. Téléchargez le CLI Knative (**kn**) à partir de la page **Command Line Tools**. Vous pouvez accéder à la page **Command Line Tools** en cliquant sur le  dans le coin supérieur droit de la console web et en sélectionnant **Command Line Tools** dans la liste.

2. Décompressez l'archive :

```
tar -xf <fichier>
```

3. Déplacez le fichier binaire **kn** dans un répertoire de votre site **PATH**.

4. Pour vérifier votre **PATH**, exécutez :

```
$ echo $PATH
```

Vérification

- Exécutez les commandes suivantes pour vérifier que les ressources et la route Knative CLI correctes ont été créées :

```
$ oc get ConsoleCLIDownload
```

Exemple de sortie

```
NAME                DISPLAY NAME                                      AGE
kn                  kn - OpenShift Serverless Command Line Interface (CLI)   2022-09-
20T08:41:18Z
oc-cli-downloads    oc - OpenShift Command Line Interface (CLI)              2022-09-
20T08:00:20Z
```

```
$ oc get route -n openshift-serverless
```

Exemple de sortie

```
NAME   HOST/PORT                          PATH   SERVICES                  PORT
TERMINATION     WILDCARD
kn     kn-openshift-serverless.apps.example.com          knative-openshift-metrics-3   http-cli
edge/Redirect   None
```

3.3.2. Installer le CLI Knative pour Linux à l'aide d'un gestionnaire de paquets RPM

Pour Red Hat Enterprise Linux (RHEL), vous pouvez installer le CLI Knative (**kn**) en tant que RPM à l'aide d'un gestionnaire de paquets, tel que **yum** ou **dnf**. Cela permet au système de gérer automatiquement la version du CLI Knative. Par exemple, l'utilisation d'une commande telle que **dnf upgrade** met à jour tous les paquets, y compris **kn**, si une nouvelle version est disponible.

Conditions préalables

- Vous disposez d'un abonnement OpenShift Container Platform actif sur votre compte Red Hat.

Procédure

1. S'inscrire auprès du gestionnaire d'abonnements Red Hat :

```
# subscription-manager register
```

2. Extraire les données d'abonnement les plus récentes :

```
# subscription-manager refresh
```

3. Attachez l'abonnement au système enregistré :

```
# subscription-manager attach --pool=<pool_id> 1
```

1 ID du pool pour un abonnement actif à OpenShift Container Platform

4. Activer les dépôts requis par le CLI Knative (**kn**) :

- Linux (x86_64, amd64)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-x86_64-rpms"
```

- Linux sur IBM zSystems et IBM® LinuxONE (s390x)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-s390x-rpms"
```

- Linux sur IBM Power (ppc64le)

```
# subscription-manager repos --enable="openshift-serverless-1-for-rhel-8-ppc64le-rpms"
```

5. Installez le CLI Knative (**kn**) en tant que RPM à l'aide d'un gestionnaire de paquets :

Exemple de commande yum

```
# yum install openshift-serverless-clients
```

3.3.3. Installation du CLI Knative pour Linux

Si vous utilisez une distribution Linux qui ne dispose pas de RPM ou d'un autre gestionnaire de paquets, vous pouvez installer le CLI Knative (**kn**) sous la forme d'un fichier binaire. Pour ce faire, vous devez télécharger et décompresser une archive **tar.gz** et ajouter le fichier binaire à un répertoire de votre **PATH**.

Conditions préalables

- Si vous n'utilisez pas RHEL ou Fedora, assurez-vous que **libc** est installé dans un répertoire de votre chemin de bibliothèque.

IMPORTANT

Si **libc** n'est pas disponible, vous risquez d'obtenir l'erreur suivante lorsque vous exécutez des commandes CLI :

```
$ kn: No such file or directory
```

Procédure

1. Téléchargez l'archive Knative (**kn**) CLI **tar.gz**:
   - Linux (x86_64, amd64)
   - Linux sur IBM zSystems et IBM® LinuxONE (s390x)
   - Linux sur IBM Power (ppc64le)

   Vous pouvez également télécharger n'importe quelle version de **kn** en naviguant vers le répertoire correspondant à cette version dans le miroir de téléchargement du client Serverless .
2. Décompressez l'archive :

```
tar -xf <filename>
```

3. Déplacez le fichier binaire **kn** dans un répertoire de votre site **PATH**.
4. Pour vérifier votre **PATH**, exécutez :

```
$ echo $PATH
```

3.3.4. Installation de la CLI Knative pour macOS

Si vous utilisez macOS, vous pouvez installer le CLI Knative (**kn**) sous forme de fichier binaire. Pour ce faire, vous devez télécharger et décompresser une archive **tar.gz** et ajouter le fichier binaire à un répertoire de votre **PATH**.

Procédure

1. Téléchargez l'archive du CLI Knative (**kn**) **tar.gz**.
   Vous pouvez également télécharger n'importe quelle version de **kn** en naviguant vers le répertoire correspondant à cette version dans le miroir de téléchargement du client Serverless .
2. Décompressez et extrayez l'archive.
3. Déplacez le fichier binaire **kn** dans un répertoire de votre site **PATH**.
4. Pour vérifier votre **PATH**, ouvrez une fenêtre de terminal et exécutez :

```
$ echo $PATH
```

3.3.5. Installation du CLI Knative pour Windows

Si vous utilisez Windows, vous pouvez installer le CLI Knative (**kn**) sous forme de fichier binaire. Pour ce faire, vous devez télécharger et décompresser une archive ZIP et ajouter le fichier binaire à un répertoire de votre site **PATH**.

Procédure

1. Télécharger l'archive ZIP du CLI Knative (**kn**).
   Vous pouvez également télécharger n'importe quelle version de **kn** en naviguant vers le répertoire correspondant à cette version dans le miroir de téléchargement du client Serverless .

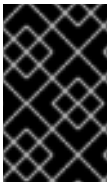
2. Extraire l'archive à l'aide d'un programme ZIP.
3. Déplacez le fichier binaire **kn** dans un répertoire de votre site **PATH**.
4. Pour vérifier votre **PATH**, ouvrez l'invite de commande et exécutez la commande suivante :

```
C:\N> path
```

3.4. INSTALLATION DE KNATIVE SERVING

L'installation de Knative Serving vous permet de créer des services et des fonctions Knative sur votre cluster. Elle vous permet également d'utiliser des fonctionnalités supplémentaires telles que l'autoscaling et les options de mise en réseau pour vos applications.

Après avoir installé OpenShift Serverless Operator, vous pouvez installer Knative Serving en utilisant les paramètres par défaut ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeServing**. Pour plus d'informations sur les options de configuration de la CR **KnativeServing**, voir Configuration globale.



IMPORTANT

Si vous souhaitez utiliser Red Hat OpenShift distributed tracing avec OpenShift Serverless, vous devez installer et configurer Red Hat OpenShift distributed tracing avant d'installer Knative Serving.

3.4.1. Installer Knative Serving en utilisant la console web

Après avoir installé l'OpenShift Serverless Operator, installez Knative Serving en utilisant la console web OpenShift Container Platform. Vous pouvez installer Knative Serving en utilisant les paramètres par défaut ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeServing**.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Operators → Installed Operators**.
2. Vérifiez que le menu déroulant **Project** en haut de la page est bien réglé sur **Project: knative-serving**.
3. Cliquez sur **Knative Serving** dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator afin d'accéder à l'onglet **Knative Serving**.
4. Cliquez sur **Create Knative Serving**

5. Dans la page **Create Knative Serving** vous pouvez installer Knative Serving en utilisant les paramètres par défaut en cliquant sur **Create**.
   Vous pouvez également modifier les paramètres de l'installation de Knative Serving en éditant l'objet **KnativeServing** à l'aide du formulaire fourni ou en éditant le YAML.

   - L'utilisation du formulaire est recommandée pour les configurations plus simples qui ne nécessitent pas un contrôle total de la création des objets **KnativeServing**.

   - Il est recommandé d'éditer le YAML pour les configurations plus complexes qui nécessitent un contrôle total de la création des objets **KnativeServing**. Vous pouvez accéder au YAML en cliquant sur le lien **edit YAML** en haut à droite de la page **Create Knative Serving**
     Une fois que vous avez rempli le formulaire ou que vous avez fini de modifier le YAML, cliquez sur **Create**.

NOTE

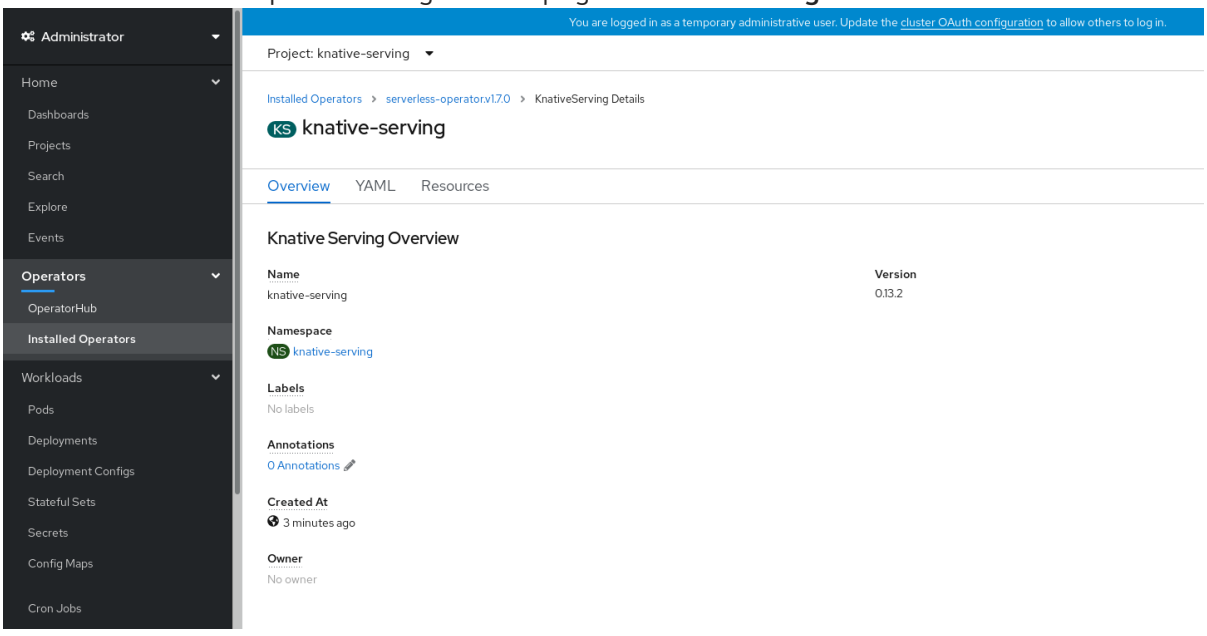
Pour plus d'informations sur les options de configuration de la définition des ressources personnalisées KnativeServing, voir la documentation sur *Advanced installation configuration options*.

6. Après avoir installé Knative Serving, l'objet **KnativeServing** est créé et vous êtes automatiquement dirigé vers l'onglet **Knative Serving**. Vous verrez la ressource personnalisée **knative-serving** dans la liste des ressources.

Vérification

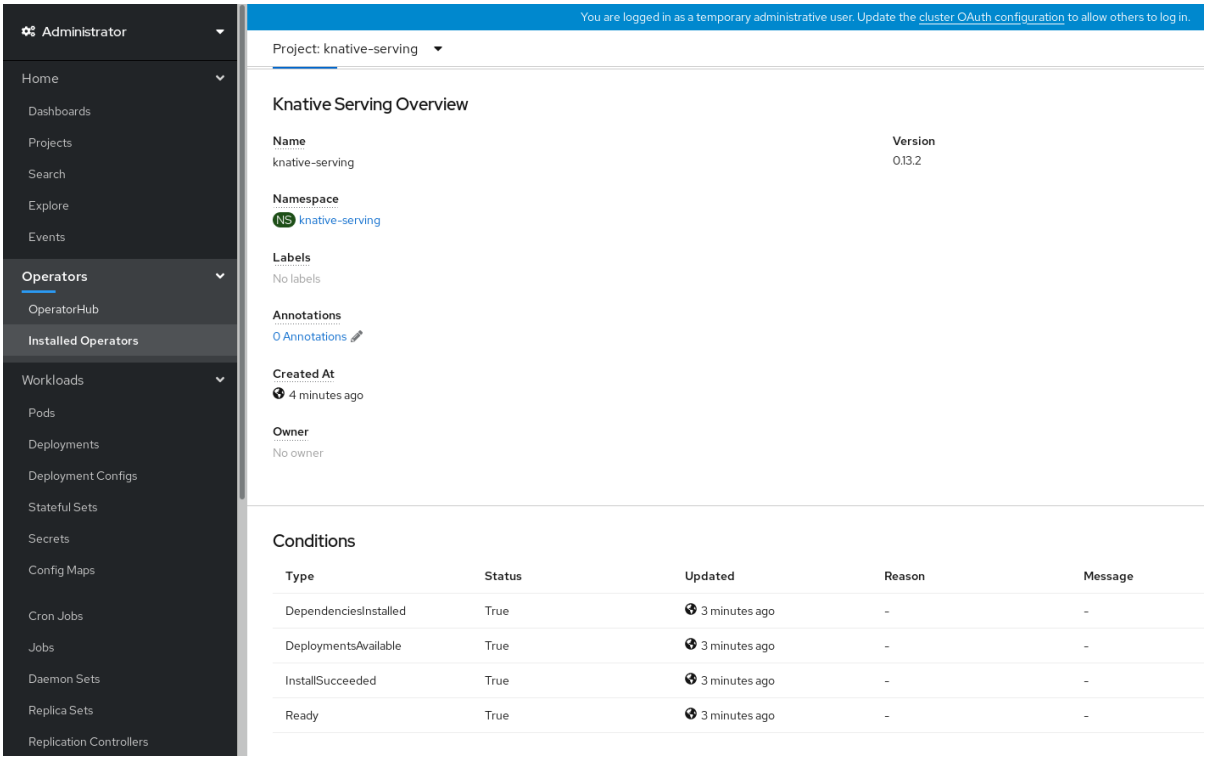
1. Cliquez sur **knative-serving** custom resource dans l'onglet **Knative Serving**.

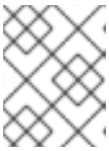
2. Vous serez automatiquement dirigé vers la page **Knative Serving Overview**.



3. Faites défiler la page vers le bas pour consulter la liste de **Conditions**.

4. La liste des conditions dont l'état est **True** s'affiche, comme le montre l'image d'exemple.





NOTE

La création des ressources Knative Serving peut prendre quelques secondes. Vous pouvez vérifier leur statut dans l'onglet **Resources**.

5. Si les conditions ont un statut de **Unknown** ou **False**, attendez quelques instants et vérifiez à nouveau après avoir confirmé que les ressources ont été créées.

3.4.2. Installer Knative Serving en utilisant YAML

Après avoir installé OpenShift Serverless Operator, vous pouvez installer Knative Serving en utilisant les paramètres par défaut, ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeServing**. Vous pouvez utiliser la procédure suivante pour installer Knative Serving en utilisant des fichiers YAML et le CLI **oc**.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez installé OpenShift Serverless Operator.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez un fichier nommé **serving.yaml** et copiez-y l'exemple YAML suivant :

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
```

```
    name: knative-serving
    namespace: knative-serving
```

2. Appliquer le fichier **serving.yaml**:

```
$ oc apply -f serving.yaml
```

Vérification

1. Pour vérifier que l'installation est terminée, entrez la commande suivante :

```
$ oc get knativeserving.operator.knative.dev/knative-serving -n knative-serving --
template='{{range .status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
```

Exemple de sortie

```
DependenciesInstalled=True
DeploymentsAvailable=True
InstallSucceeded=True
Ready=True
```

NOTE

La création des ressources Knative Serving peut prendre quelques secondes.

Si les conditions ont un statut de **Unknown** ou **False**, attendez quelques instants et vérifiez à nouveau après avoir confirmé que les ressources ont été créées.

2. Vérifier que les ressources Knative Serving ont été créées :

```
$ oc get pods -n knative-serving
```

Exemple de sortie

```
NAME                                                READY   STATUS      RESTARTS   AGE
activator-67ddf8c9d7-p7rm5                          2/2     Running     0          4m
activator-67ddf8c9d7-q84fz                          2/2     Running     0          4m
autoscaler-5d87bc6dbf-6nqc6                         2/2     Running     0          3m59s
autoscaler-5d87bc6dbf-h64rl                         2/2     Running     0          3m59s
autoscaler-hpa-77f85f5cc4-lrts7                     2/2     Running     0          3m57s
autoscaler-hpa-77f85f5cc4-zx7hl                     2/2     Running     0          3m56s
controller-5cfc7cb8db-nlccl                         2/2     Running     0          3m50s
controller-5cfc7cb8db-rmv7r                         2/2     Running     0          3m18s
domain-mapping-86d84bb6b4-r746m                     2/2     Running     0          3m58s
domain-mapping-86d84bb6b4-v7nh8                     2/2     Running     0          3m58s
domainmapping-webhook-769d679d45-bkcnj              2/2     Running     0          3m58s
domainmapping-webhook-769d679d45-fff68              2/2     Running     0          3m58s
storage-version-migration-serving-serving-0.26.0--1-6qlkb   0/1     Completed   0          3m56s
webhook-5fb774f8d8-6bqrt                            2/2     Running     0          3m57s
webhook-5fb774f8d8-b8lt5                            2/2     Running     0          3m57s
```

3. Vérifiez que les composants réseau nécessaires ont été installés dans l'espace de noms **knative-serving-ingress** créé automatiquement :

```
$ oc get pods -n knative-serving-ingress
```

Exemple de sortie

```
NAME                                    READY   STATUS    RESTARTS   AGE
net-kourier-controller-7d4b6c5d95-62mkf   1/1     Running   0          76s
net-kourier-controller-7d4b6c5d95-qmgm2   1/1     Running   0          76s
3scale-kourier-gateway-6688b49568-987qz   1/1     Running   0          75s
3scale-kourier-gateway-6688b49568-b5tnp   1/1     Running   0          75s
```

3.4.3. Prochaines étapes

- Si vous souhaitez utiliser l'architecture événementielle de Knative, vous pouvez installer Knative Eventing.

3.5. INSTALLATION DE KNATIVE EVENTING

Pour utiliser une architecture pilotée par les événements sur votre cluster, installez Knative Eventing. Vous pouvez créer des composants Knative tels que des sources d'événements, des courtiers et des canaux, puis les utiliser pour envoyer des événements à des applications ou à des systèmes externes.

Après avoir installé OpenShift Serverless Operator, vous pouvez installer Knative Eventing en utilisant les paramètres par défaut ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeEventing**. Pour plus d'informations sur les options de configuration de la CR **KnativeEventing**, voir Configuration globale.

IMPORTANT

Si vous souhaitez utiliser Red Hat OpenShift distributed tracing avec OpenShift Serverless, vous devez installer et configurer Red Hat OpenShift distributed tracing avant d'installer Knative Eventing.

3.5.1. Installation de Knative Eventing à l'aide de la console web

Après avoir installé l'OpenShift Serverless Operator, installez Knative Eventing en utilisant la console web OpenShift Container Platform. Vous pouvez installer Knative Eventing en utilisant les paramètres par défaut ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeEventing**.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Operators → Installed Operators**.

2. Vérifiez que le menu déroulant **Project** en haut de la page est bien réglé sur **Project: knative-eventing**.

3. Cliquez sur **Knative Eventing** dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator afin d'accéder à l'onglet **Knative Eventing**.

4. Cliquez sur **Create Knative Eventing**.

5. Dans la page **Create Knative Eventing**, vous pouvez choisir de configurer l'objet **KnativeEventing** en utilisant le formulaire par défaut fourni ou en modifiant le YAML.

   - L'utilisation du formulaire est recommandée pour les configurations plus simples qui ne nécessitent pas un contrôle total de la création des objets **KnativeEventing**.
     Facultatif. Si vous configurez l'objet **KnativeEventing** à l'aide du formulaire, apportez les modifications que vous souhaitez mettre en œuvre pour votre déploiement Knative Eventing.

6. Cliquez sur **Create**.

   - Il est recommandé d'éditer le YAML pour les configurations plus complexes qui nécessitent un contrôle total de la création des objets **KnativeEventing**. Vous pouvez accéder au YAML en cliquant sur le lien **edit YAML** en haut à droite de la page **Create Knative Eventing**.
     Facultatif. Si vous configurez l'objet **KnativeEventing** en modifiant le YAML, apportez toutes les modifications au YAML que vous souhaitez mettre en œuvre pour votre déploiement Knative Eventing.

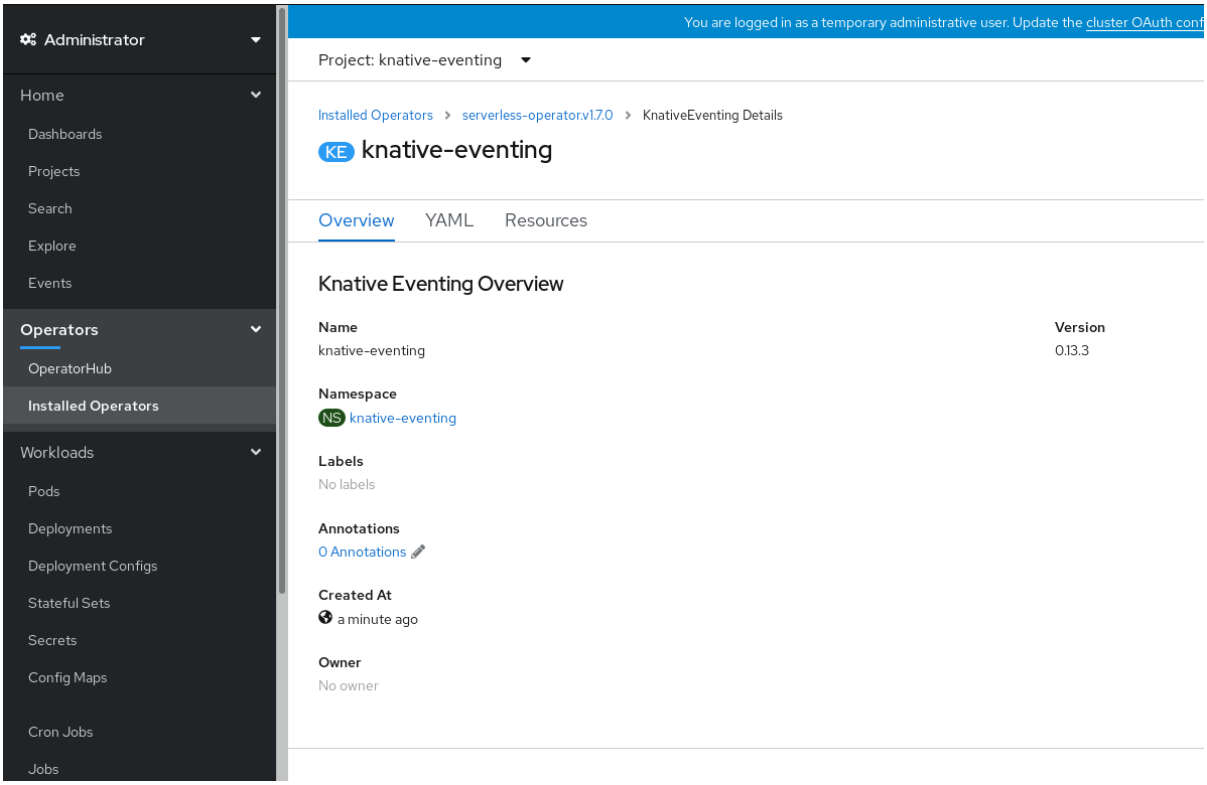
7. Cliquez sur **Create**.

8. Après avoir installé Knative Eventing, l'objet **KnativeEventing** est créé et vous êtes automatiquement dirigé vers l'onglet **Knative Eventing**. Vous verrez la ressource personnalisée **knative-eventing** dans la liste des ressources.

Vérification

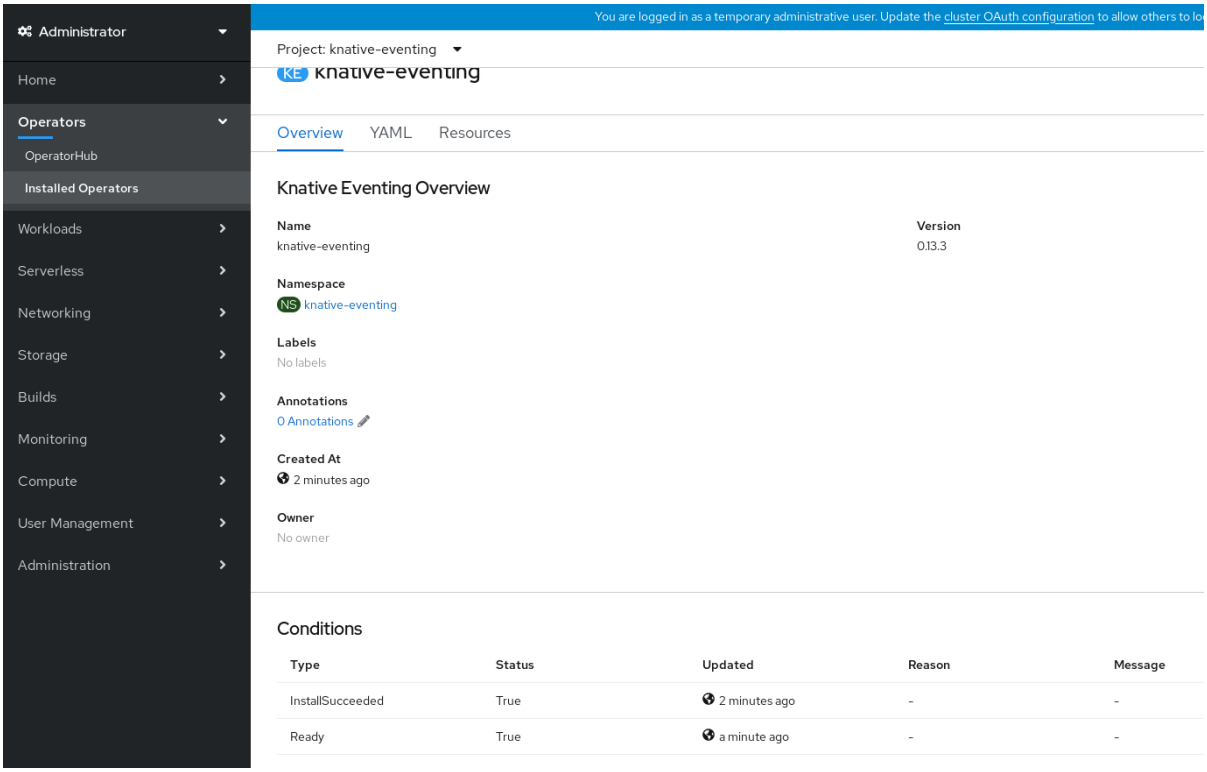
1. Cliquez sur la ressource personnalisée **knative-eventing** dans l'onglet **Knative Eventing**.

2. Vous êtes automatiquement dirigé vers la page **Knative Eventing Overview**.



3. Faites défiler la page vers le bas pour consulter la liste de **Conditions**.

4. La liste des conditions dont l'état est **True** s'affiche, comme le montre l'image d'exemple.



NOTE

La création des ressources Knative Eventing peut prendre quelques secondes. Vous pouvez vérifier leur statut dans l'onglet **Resources**.

5. Si les conditions ont un statut de **Unknown** ou **False**, attendez quelques instants et vérifiez à nouveau après avoir confirmé que les ressources ont été créées.

3.5.2. Installer Knative Eventing en utilisant YAML

Après avoir installé OpenShift Serverless Operator, vous pouvez installer Knative Eventing en utilisant les paramètres par défaut, ou configurer des paramètres plus avancés dans la ressource personnalisée (CR) **KnativeEventing**. Vous pouvez utiliser la procédure suivante pour installer Knative Eventing en utilisant des fichiers YAML et le CLI **oc**.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez installé OpenShift Serverless Operator.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez un fichier nommé **eventing.yaml**.
2. Copiez l'exemple suivant de YAML dans **eventing.yaml**:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
```

3. Facultatif. Apportez toutes les modifications au YAML que vous souhaitez mettre en œuvre pour votre déploiement Knative Eventing.
4. Appliquez le fichier **eventing.yaml** en entrant :

```
$ oc apply -f eventing.yaml
```

Vérification

1. Vérifiez que l'installation est terminée en entrant la commande suivante et en observant la sortie :

```
$ oc get knativeeventing.operator.knative.dev/knative-eventing \
  -n knative-eventing \
  --template='{{range .status.conditions}}{{printf "%s=%s\n" .type .status}}{{end}}'
```

Exemple de sortie

```
InstallSucceeded=True
Ready=True
```

NOTE

La création des ressources Knative Eventing peut prendre quelques secondes.

2. Si les conditions ont un statut de **Unknown** ou **False**, attendez quelques instants et vérifiez à nouveau après avoir confirmé que les ressources ont été créées.

3. Vérifier que les ressources Knative Eventing ont été créées en entrant :

```
$ oc get pods -n knative-eventing
```

Exemple de sortie

```
NAME                                  READY   STATUS    RESTARTS   AGE
broker-controller-58765d9d49-g9zp6    1/1     Running   0          7m21s
eventing-controller-65fdd66b54-jw7bh  1/1     Running   0          7m31s
eventing-webhook-57fd74b5bd-kvhlz     1/1     Running   0          7m31s
imc-controller-5b75d458fc-ptvm2       1/1     Running   0          7m19s
imc-dispatcher-64f6d5fccb-kkc4c       1/1     Running   0          7m18s
```

3.5.3. Installation du courtier Knative pour Apache Kafka

L'implémentation du courtier Knative pour Apache Kafka fournit des options d'intégration qui vous permettent d'utiliser les versions prises en charge de la plateforme de streaming de messages Apache Kafka avec OpenShift Serverless. La fonctionnalité Knative broker for Apache Kafka est disponible dans une installation OpenShift Serverless si vous avez installé la ressource personnalisée **KnativeKafka**.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Eventing sur votre cluster.
- Vous avez accès à un cluster Red Hat AMQ Streams.
- Installez le CLI OpenShift (**oc**) si vous souhaitez utiliser les étapes de vérification.
- Vous disposez des droits d'administrateur de cluster sur OpenShift Container Platform.
- Vous êtes connecté à la console web de OpenShift Container Platform.

Procédure

1. Dans la perspective **Administrator**, naviguez vers **Operators → Installed Operators**.
2. Vérifiez que le menu déroulant **Project** en haut de la page est bien réglé sur **Project: knative-eventing**.
3. Dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator, trouvez la boîte **Knative Kafka** et cliquez sur **Create Instance**.
4. Configurez l'objet **KnativeKafka** dans la page **Create Knative Kafka**

IMPORTANT

Pour utiliser le canal, la source, le courtier ou le puits Kafka sur votre cluster, vous devez basculer le commutateur **enabled** pour les options que vous souhaitez utiliser sur **true**. Ces commutateurs sont réglés sur **false** par défaut. En outre, pour utiliser le canal, le courtier ou le puits Kafka, vous devez spécifier les serveurs d'amorçage.

Exemple KnativeKafka ressource personnalisée

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  channel:
    enabled: true (1)
    bootstrapServers: <bootstrap_servers> (2)
  source:
    enabled: true (3)
  broker:
    enabled: true (4)
    defaultConfig:
      bootstrapServers: <bootstrap_servers> (5)
      numPartitions: <num_partitions> (6)
      replicationFactor: <replication_factor> (7)
  sink:
    enabled: true (8)
```

1. Permet aux développeurs d'utiliser le type de canal **KafkaChannel** dans le cluster.
2. Une liste de serveurs d'amorçage de votre cluster AMQ Streams, séparés par des virgules.
3. Permet aux développeurs d'utiliser le type de source d'événement **KafkaSource** dans le cluster.
4. Permet aux développeurs d'utiliser l'implémentation du courtier Knative pour Apache Kafka dans le cluster.
5. Une liste de serveurs d'amorçage de votre cluster Red Hat AMQ Streams, séparée par des virgules.
6. Définit le nombre de partitions des sujets Kafka, soutenus par les objets **Broker**. La valeur par défaut est **10**.
7. Définit le facteur de réplication des sujets Kafka, soutenu par les objets **Broker**. La valeur par défaut est **3**.
8. Permet aux développeurs d'utiliser un puits Kafka dans le cluster.

NOTE

La valeur **replicationFactor** doit être inférieure ou égale au nombre de nœuds de votre cluster Red Hat AMQ Streams.

a. L'utilisation du formulaire est recommandée pour les configurations plus simples qui ne nécessitent pas un contrôle total de la création des objets **KnativeKafka**.

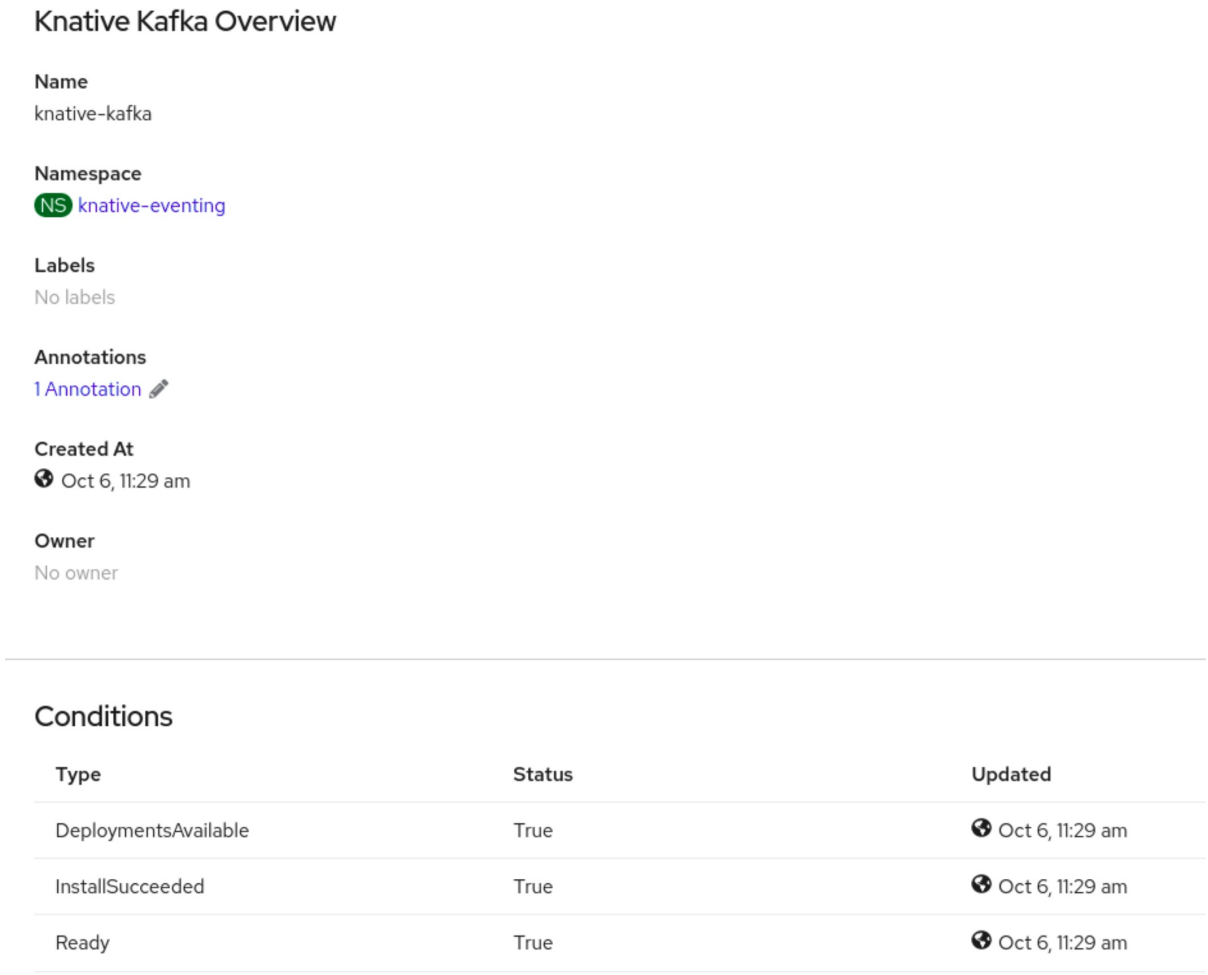
b. Il est recommandé d'éditer le YAML pour les configurations plus complexes qui nécessitent un contrôle total de la création des objets **KnativeKafka**. Vous pouvez accéder au YAML en cliquant sur le lien **Edit YAML** en haut à droite de la page **Create Knative Kafka**

5. Cliquez sur **Create** après avoir effectué l'une des configurations optionnelles pour Kafka. Vous êtes automatiquement dirigé vers l'onglet **Knative Kafka** où se trouve **knative-kafka** dans la liste des ressources.

Vérification

1. Cliquez sur la ressource **knative-kafka** dans l'onglet **Knative Kafka**. Vous êtes automatiquement dirigé vers la page **Knative Kafka Overview**.

2. Consulter la liste de **Conditions** pour la ressource et confirmer qu'ils ont un statut de **True**.



Si les conditions ont un statut de **Unknown** ou **False**, attendez quelques instants pour rafraîchir la page.

3. Vérifier que les ressources Knative broker for Apache Kafka ont été créées :

```
$ oc get pods -n knative-eventing
```

Exemple de sortie

```
NAME                                  READY   STATUS    RESTARTS   AGE
kafka-broker-dispatcher-7769fbbcbb-xgffn    2/2     Running   0          44s
kafka-broker-receiver-5fb56f7656-fhq8d      2/2     Running   0          44s
kafka-channel-dispatcher-84fd6cb7f9-k2tjv   2/2     Running   0          44s
```

```
kafka-channel-receiver-9b7f795d5-c76xr      2/2     Running   0          44s
kafka-controller-6f95659bf6-trd6r           2/2     Running   0          44s
kafka-source-dispatcher-6bf98bdfff-8bcsn    2/2     Running   0          44s
kafka-webhook-eventing-68dc95d54b-825xs     2/2     Running   0          44s
```

3.5.4. Prochaines étapes

- Si vous souhaitez utiliser les services Knative, vous pouvez installer Knative Serving.

3.6. CONFIGURATION DU COURTIER KNATIVE POUR APACHE KAFKA

L'implémentation du courtier Knative pour Apache Kafka fournit des options d'intégration vous permettant d'utiliser les versions prises en charge de la plateforme de streaming de messages Apache Kafka avec OpenShift Serverless. Kafka fournit des options pour la source d'événement, le canal, le courtier et les capacités de puits d'événement.

En plus des composants Knative Eventing qui sont fournis dans le cadre d'une installation de base d'OpenShift Serverless, les administrateurs de cluster peuvent installer la ressource personnalisée (CR) **KnativeKafka**.

NOTE

Knative broker for Apache Kafka n'est pas actuellement pris en charge pour IBM zSystems et IBM Power.

Le CR **KnativeKafka** offre aux utilisateurs des options supplémentaires, telles que

- Source Kafka
- Canal Kafka
- Courtier Kafka
- Puits Kafka

3.7. CONFIGURATION DES FONCTIONS OPENSHIFT SERVERLESS

Pour améliorer le processus de déploiement du code de votre application, vous pouvez utiliser OpenShift Serverless pour déployer des fonctions sans état et pilotées par les événements en tant que service Knative sur OpenShift Container Platform. Si vous souhaitez développer des fonctions, vous devez effectuer les étapes de configuration.

3.7.1. Conditions préalables

Pour activer l'utilisation d'OpenShift Serverless Functions sur votre cluster, vous devez effectuer les étapes suivantes :

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.

NOTE

Les fonctions sont déployées en tant que service Knative. Si vous souhaitez utiliser une architecture pilotée par les événements avec vos fonctions, vous devez également installer Knative Eventing.

- Vous avez installé le CLI**oc** .
- Vous avez installé le CLI Knative (**kn**). L'installation du CLI Knative permet d'utiliser les commandes **kn func** pour créer et gérer des fonctions.
- Vous avez installé Docker Container Engine ou Podman version 3.4.7 ou supérieure.
- Vous avez accès à un registre d'images disponible, tel que le OpenShift Container Registry.
- Si vous utilisez Quay.io comme registre d'images, vous devez vous assurer que le dépôt n'est pas privé, ou que vous avez suivi la documentation d'OpenShift Container Platform sur l'autorisation des pods à référencer des images à partir d'autres registres sécurisés.
- Si vous utilisez le OpenShift Container Registry, un administrateur de cluster doit exposer le registre.

3.7.2. Mise en place de Podman

Pour utiliser des fonctionnalités avancées de gestion de conteneurs, vous pourriez vouloir utiliser Podman avec OpenShift Serverless Functions. Pour ce faire, vous devez démarrer le service Podman et configurer le CLI Knative (**kn**) pour vous y connecter.

Procédure

1. Démarrez le service Podman qui sert l'API Docker sur un socket UNIX à l'adresse **${XDG_RUNTIME_DIR}/podman/podman.sock**:

```
$ systemctl start --user podman.socket
```

NOTE

Sur la plupart des systèmes, cette prise est située à l'adresse **/run/user/$(id -u)/podman/podman.sock**.

2. Établir la variable d'environnement qui est utilisée pour construire une fonction :

```
$ export DOCKER_HOST="unix://${XDG_RUNTIME_DIR}/podman/podman.sock"
```

3. Exécutez la commande build dans le répertoire de votre projet de fonction avec le drapeau **-v** pour voir la sortie verbose. Vous devriez voir une connexion à votre socket UNIX local :

```
$ kn func build -v
```

3.7.3. Installation de Podman sur macOS

Pour utiliser des fonctionnalités avancées de gestion de conteneurs, vous pourriez vouloir utiliser Podman avec OpenShift Serverless Functions. Pour ce faire sur macOS, vous devez démarrer la machine Podman et configurer le CLI Knative (**kn**) pour vous y connecter.

Procédure

1. Créer la machine Podman :

```
$ podman machine init --memory=8192 --cpus=2 --disk-size=20
```

2. Démarrez la machine Podman, qui sert l'API Docker sur un socket UNIX :

```
$ podman machine start
Starting machine "podman-machine-default"
Waiting for VM ...
Mounting volume... /Users/myuser:/Users/user

[...truncated output...]

You can still connect Docker API clients by setting DOCKER_HOST using the
following command in your terminal session:

 export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-
machine-default/podman.sock'

Machine "podman-machine-default" started successfully
```

NOTE

Sur la plupart des systèmes macOS, cette prise se trouve à l'adresse **/Users/myuser/.local/share/containers/podman/machine/podman-machine-default/podman.sock**.

3. Établir la variable d'environnement qui est utilisée pour construire une fonction :

```
$ export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-
machine-default/podman.sock'
```

4. Exécutez la commande build dans le répertoire de votre projet de fonction avec le drapeau **-v** pour voir la sortie verbose. Vous devriez voir une connexion à votre socket UNIX local :

```
$ kn func build -v
```

3.7.4. Prochaines étapes

- Pour plus d'informations sur Docker Container Engine ou Podman, voir Options d'outils de construction de conteneurs.
- Voir la section Démarrer avec les fonctions.

3.8. MISES À NIVEAU SANS SERVEUR

OpenShift Serverless doit être mis à niveau sans sauter de version. Cette section montre comment résoudre les problèmes de mise à niveau.

3.8.1. Résoudre un échec de mise à niveau de l'opérateur OpenShift Serverless

Vous pouvez rencontrer une erreur lors de la mise à niveau d'OpenShift Serverless Operator, par exemple, lors de désinstallations et réinstallations manuelles. Si vous rencontrez une erreur, vous devez réinstaller manuellement OpenShift Serverless Operator.

Procédure

1. Identifier la version d'OpenShift Serverless Operator qui a été installée à l'origine en recherchant dans les notes de version d'OpenShift Serverless.
   Par exemple, le message d'erreur lors d'une tentative de mise à niveau peut contenir la chaîne suivante :

   ```
   The installed KnativeServing version is v1.5.0.
   ```

   Dans cet exemple, la version de KnativeServing **MAJOR.MINOR** est **1.5**, qui est couverte par les notes de version d'OpenShift Serverless 1.26 : *OpenShift Serverless now uses Knative Serving 1.5* .

2. Désinstaller OpenShift Serverless Operator et tous ses plans d'installation.

3. Installez manuellement la version d'OpenShift Serverless Operator que vous avez découverte à la première étape. Pour l'installation, créez d'abord un fichier **serverless-subscription.yaml** comme indiqué dans l'exemple suivant :

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: serverless-operator
     namespace: openshift-serverless
   spec:
     channel: stable
     name: serverless-operator
     source: redhat-operators
     sourceNamespace: openshift-marketplace
     installPlanApproval: Manual
     startingCSV: serverless-operator.v1.26.0
   ```

4. Ensuite, installez l'abonnement en exécutant la commande suivante :

   ```
   $ oc apply -f serverless-subscription.yaml
   ```

5. Mettre à niveau en approuvant manuellement les plans d'installation de la mise à niveau au fur et à mesure qu'ils apparaissent.

Ressources supplémentaires

- Notes de version d'OpenShift Serverless
- Suppression d'opérateurs d'une grappe à l'aide de la console web

- Installer l'opérateur OpenShift Serverless depuis la console web

CHAPITRE 4. SERVIR

4.1. DÉMARRER AVEC KNATIVE SERVING

4.1.1. Applications sans serveur

Les applications sans serveur sont créées et déployées en tant que services Kubernetes, définies par une route et une configuration, et contenues dans un fichier YAML. Pour déployer une application sans serveur à l'aide d'OpenShift Serverless, vous devez créer un objet Knative **Service**.

Exemple de fichier YAML de l'objet Knative Service

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello 1
  namespace: default 2
spec:
  template:
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift 3
          env:
            - name: RESPONSE 4
              value: "Hello Serverless!"
```

1. Le nom de l'application.
2. L'espace de noms utilisé par l'application.
3. L'image de l'application.
4. Variable d'environnement imprimée par l'exemple d'application.

Vous pouvez créer une application sans serveur en utilisant l'une des méthodes suivantes :

- Créez un service Knative à partir de la console web d'OpenShift Container Platform. Pour plus d'informations, voir Création d'applications à l'aide de la perspective du développeur .
- Créer un service Knative en utilisant le CLI Knative (**kn**).
- Créer et appliquer un objet Knative **Service** sous la forme d'un fichier YAML, en utilisant le CLI **oc**.

4.1.1.1. Créer des applications sans serveur en utilisant le CLI Knative

L'utilisation de la CLI Knative (**kn**) pour créer des applications sans serveur offre une interface utilisateur plus rationalisée et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn service create** pour créer une application sans serveur de base.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un service Knative :

```
$ kn service create <service-name> --image <image> --tag <tag-value>
```

Où ?

- **--image** est l'URI de l'image pour l'application.
- **--tag** est un indicateur facultatif qui peut être utilisé pour ajouter une étiquette à la révision initiale créée avec le service.

Example command

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

Exemple de sortie

```
Creating service 'event-display' in namespace 'default':

  0.271s The Route is still working to reflect the latest desired specification.
  0.580s Configuration "event-display" is waiting for a Revision to become ready.
  3.857s ...
  3.861s Ingress has not yet been reconciled.
  4.270s Ready to serve.

Service 'event-display' created with latest revision 'event-display-bxshg-1' and URL:
http://event-display-default.apps-crc.testing
```

4.1.1.2. Créer des applications sans serveur à l'aide de YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire des applications de manière déclarative et reproductible. Pour créer une application sans serveur à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet Knative **Service**, puis l'appliquer à l'aide de **oc apply**.

Une fois le service créé et l'application déployée, Knative crée une révision immuable pour cette version de l'application. Knative effectue également la programmation du réseau pour créer une route, une entrée, un service et un équilibreur de charge pour votre application et fait automatiquement évoluer vos pods vers le haut ou vers le bas en fonction du trafic.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.

- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez un fichier YAML contenant l'exemple de code suivant :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-delivery
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
          env:
            - name: RESPONSE
              value: "Hello Serverless!"
```

2. Naviguez jusqu'au répertoire où se trouve le fichier YAML et déployez l'application en appliquant le fichier YAML :

```
$ oc apply -f <filename>
```

Si vous ne souhaitez pas passer à la perspective **Developer** dans la console web d'OpenShift Container Platform ou utiliser le CLI Knative (**kn**) ou les fichiers YAML, vous pouvez créer des composants Knative en utilisant la perspective **Administator** de la console web d'OpenShift Container Platform.

4.1.1.3. Créer des applications sans serveur en utilisant la perspective de l'administrateur

Les applications sans serveur sont créées et déployées en tant que services Kubernetes, définies par une route et une configuration, et contenues dans un fichier YAML. Pour déployer une application sans serveur à l'aide d'OpenShift Serverless, vous devez créer un objet Knative **Service**.

Exemple de fichier YAML de l'objet Knative Service

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: hello (1)
  namespace: default (2)
spec:
  template:
    spec:
      containers:
        - image: docker.io/openshift/hello-openshift (3)
          env:
            - name: RESPONSE (4)
              value: "Hello Serverless!"
```

1. Le nom de l'application.
2. L'espace de noms utilisé par l'application.
3. L'image de l'application.
4. Variable d'environnement imprimée par l'exemple d'application.

Une fois le service créé et l'application déployée, Knative crée une révision immuable pour cette version de l'application. Knative effectue également la programmation du réseau pour créer une route, une entrée, un service et un équilibreur de charge pour votre application et fait automatiquement évoluer vos pods vers le haut ou vers le bas en fonction du trafic.

Conditions préalables

Pour créer des applications sans serveur à l'aide de la perspective **Administrator**, assurez-vous d'avoir effectué les étapes suivantes.

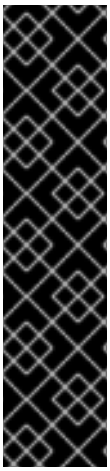
- L'opérateur OpenShift Serverless et Knative Serving sont installés.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.

Procédure

1. Naviguez jusqu'à la page **Serverless → Serving**.
2. Dans la liste **Create**, sélectionnez **Service**.
3. Saisir manuellement des définitions YAML ou JSON, ou glisser-déposer un fichier dans l'éditeur.
4. Cliquez sur **Create**.

4.1.1.4. Création d'un service en mode hors ligne

Vous pouvez exécuter les commandes **kn service** en mode déconnecté, de sorte qu'aucune modification n'est apportée au cluster et que le fichier de descripteur de service est créé sur votre machine locale. Une fois le fichier descripteur créé, vous pouvez le modifier avant de propager les changements au cluster.



IMPORTANT

Le mode hors ligne de la CLI Knative est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, permettant aux clients de tester les fonctionnalités et de fournir un retour d'information au cours du processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. En mode déconnecté, créez un fichier de descripteurs de service Knative local :

```
$ kn service create event-display \
   --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest \
   --target ./ \
   --namespace test
```

Exemple de sortie

```
Service 'event-display' created in namespace 'test'.
```

- L'option **--target ./** active le mode hors ligne et spécifie **./** comme répertoire de stockage de la nouvelle arborescence.
  Si vous n'indiquez pas de répertoire existant, mais que vous utilisez un nom de fichier, tel que **--target my-service.yaml**, aucune arborescence n'est créée. Seul le fichier de descripteurs de service **my-service.yaml** est créé dans le répertoire actuel.

  Le nom de fichier peut avoir l'extension **.yaml**, **.yml**, ou **.json**. Le choix de **.json** crée le fichier du descripteur de service au format JSON.

- L'option **--namespace test** place le nouveau service dans l'espace de noms **test**.
  Si vous n'utilisez pas **--namespace**, et que vous êtes connecté à un cluster OpenShift Container Platform, le fichier de descripteurs est créé dans l'espace de noms actuel. Sinon, le fichier de descripteurs est créé dans l'espace de noms **default**.

2. Examinez la structure de répertoire créée :

```
$ tree ./
```

Exemple de sortie

```
./
└── test
    └── ksvc
        └── event-display.yaml

2 directories, 1 file
```

- Le répertoire actuel **./** spécifié avec **--target** contient le nouveau répertoire **test/** qui porte le nom de l'espace de noms spécifié.
- Le répertoire **test/** contient le répertoire **ksvc**, nommé d'après le type de ressource.
- Le répertoire **ksvc** contient le fichier descripteur **event-display.yaml**, nommé d'après le nom du service spécifié.

3. Examinez le fichier de descripteurs de service généré :

```
$ cat test/ksvc/event-display.yaml
```

Exemple de sortie

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  creationTimestamp: null
  name: event-display
  namespace: test
spec:
  template:
    metadata:
      annotations:
        client.knative.dev/user-image: quay.io/openshift-knative/knative-eventing-sources-event-
display:latest
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
        name: ""
        resources: {}
status: {}
```

4. Liste des informations sur le nouveau service :

```
$ kn service describe event-display --target ./ --namespace test
```

Exemple de sortie

```
Name:       event-display
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- L'option **--target ./** spécifie le répertoire racine de la structure de répertoires contenant les sous-répertoires de l'espace de noms.
  Vous pouvez également spécifier directement un nom de fichier YAML ou JSON à l'aide de l'option **--target**. Les extensions de fichier acceptées sont **.yaml**, **.yml**, et **.json**.

- L'option **--namespace** spécifie l'espace de noms, qui communique à **kn** le sous-répertoire contenant le fichier de descripteur de service nécessaire.
  Si vous n'utilisez pas **--namespace** et que vous êtes connecté à un cluster OpenShift Container Platform, **kn** recherche le service dans le sous-répertoire portant le nom de l'espace de noms actuel. Sinon, **kn** effectue la recherche dans le sous-répertoire **default/**.

5. Utilisez le fichier descripteur de service pour créer le service sur le cluster :

```
$ kn service create -f test/ksvc/event-display.yaml
```

-

Exemple de sortie

```
Creating service 'event-display' in namespace 'test':

 0.058s The Route is still working to reflect the latest desired specification.
 0.098s ...
 0.168s Configuration "event-display" is waiting for a Revision to become ready.
23.377s ...
23.419s Ingress has not yet been reconciled.
23.534s Waiting for load balancer to be ready
23.723s Ready to serve.

Service 'event-display' created to latest revision 'event-display-00001' is available at URL:
http://event-display-test.apps.example.com
```

4.1.1.5. Ressources supplémentaires

- Commandes CLI de Knative Serving
- Configuration de l'authentification par jeton Web JSON pour les services Knative

4.1.2. Vérifier le déploiement de votre application sans serveur

Pour vérifier que votre application serverless a été déployée avec succès, vous devez obtenir l'URL de l'application créée par Knative, puis envoyer une requête à cette URL et observer la sortie. OpenShift Serverless prend en charge l'utilisation d'URL HTTP et HTTPS, mais la sortie de **oc get ksvc** imprime toujours les URL en utilisant le format **http:**//.

4.1.2.1. Vérifier le déploiement de votre application sans serveur

Pour vérifier que votre application serverless a été déployée avec succès, vous devez obtenir l'URL de l'application créée par Knative, puis envoyer une requête à cette URL et observer la sortie. OpenShift Serverless prend en charge l'utilisation d'URL HTTP et HTTPS, mais la sortie de **oc get ksvc** imprime toujours les URL en utilisant le format **http:**//.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI **oc**.
- Vous avez créé un service Knative.

Conditions préalables

- Installez le CLI OpenShift (**oc**).

Procédure

1. Recherchez l'URL de l'application :

```
oc get ksvc -YRFFGUNA nom_du_service>
```

Exemple de sortie

```
NAME URL LATESTCREATED LATESTREADY
READY REASON
event-delivery http://event-delivery-default.example.com event-delivery-4wsd2 event-
delivery-4wsd2 True
```

2. Envoyez une requête à votre cluster et observez le résultat.

Exemple de demande HTTP

```
$ curl http://event-delivery-default.example.com
```

Exemple de demande HTTPS

```
$ curl https://event-delivery-default.example.com
```

Exemple de sortie

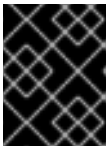
```
Hello Serverless!
```

3. Facultatif. Si vous recevez une erreur concernant un certificat auto-signé dans la chaîne de certificats, vous pouvez ajouter le drapeau **--insecure** à la commande curl pour ignorer l'erreur :

```
$ curl https://event-delivery-default.example.com --insecure
```

Exemple de sortie

```
Hello Serverless!
```



IMPORTANT

Les certificats auto-signés ne doivent pas être utilisés dans un déploiement de production. Cette méthode ne doit être utilisée qu'à des fins de test.

4. Facultatif. Si votre cluster OpenShift Container Platform est configuré avec un certificat signé par une autorité de certification (CA) mais pas encore configuré globalement pour votre système, vous pouvez le spécifier avec la commande **curl**. Le chemin d'accès au certificat peut être transmis à la commande curl à l'aide de l'indicateur **--cacert**:

```
$ curl https://event-delivery-default.example.com --cacert <file>
```

Exemple de sortie

```
Hello Serverless!
```

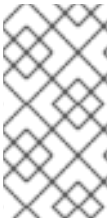
4.2. MISE À L'ÉCHELLE AUTOMATIQUE

4.2.1. Mise à l'échelle automatique

Knative Serving fournit une mise à l'échelle automatique, ou *autoscaling*, pour les applications afin de répondre à la demande entrante. Par exemple, si une application ne reçoit aucun trafic et que l'option scale-to-zero est activée, Knative Serving réduit l'application à zéro réplicas. Si l'option scale-to-zero est désactivée, l'application est réduite au nombre minimum de répliques configuré pour les applications sur le cluster. Les répliques peuvent également être mises à l'échelle pour répondre à la demande si le trafic de l'application augmente.

Les paramètres de mise à l'échelle automatique des services Knative peuvent être des paramètres globaux configurés par les administrateurs du cluster, ou des paramètres par révision configurés pour des services individuels.

Vous pouvez modifier les paramètres par révision de vos services en utilisant la console web d'OpenShift Container Platform, en modifiant le fichier YAML de votre service, ou en utilisant le CLI Knative (**kn**).



NOTE

Toutes les limites ou tous les objectifs que vous définissez pour un service sont mesurés par rapport à une instance unique de votre application. Par exemple, la définition de l'annotation **target** sur **50** configure l'autoscaler pour qu'il mette l'application à l'échelle de manière à ce que chaque révision traite 50 demandes à la fois.

4.2.2. Limites d'échelle

Les limites d'échelle déterminent les nombres minimum et maximum de répliques pouvant servir une application à un moment donné. Vous pouvez définir des limites d'échelle pour une application afin d'éviter les démarrages à froid ou de contrôler les coûts informatiques.

4.2.2.1. Limites minimales de l'échelle

Le nombre minimum de répliques pouvant servir une application est déterminé par l'annotation **min-scale**. Si la mise à zéro n'est pas activée, la valeur de **min-scale** est par défaut **1**.

La valeur **min-scale** est remplacée par défaut par des répliques **0** si les conditions suivantes sont remplies :

- L'annotation **min-scale** n'est pas définie
- La mise à l'échelle à zéro est activée
- La classe **KPA** est utilisée

Exemple de spécification de service avec l'annotation min-scale

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/min-scale: "0"
...
```

4.2.2.1.1. Définir l'annotation min-scale en utilisant le CLI Knative

L'utilisation de l'interface de programmation Knative (**kn**) pour définir l'annotation **min-scale** offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn service** avec l'indicateur **--scale-min** pour créer ou modifier la valeur **min-scale** pour un service.

Conditions préalables

- Knative Serving est installé sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Définissez le nombre minimum de répliques pour le service en utilisant l'option **--scale-min**:

```
$ kn service create <service_name> --image <image_uri> --scale-min <integer>
```

Example command

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --scale-min 2
```

4.2.2.2. Limites maximales de l'échelle

Le nombre maximum de répliques pouvant servir une application est déterminé par l'annotation **max-scale**. Si l'annotation **max-scale** n'est pas définie, il n'y a pas de limite supérieure pour le nombre de répliques créées.

Exemple de spécification de service avec l'annotation max-scale

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/max-scale: "10"
...
```

4.2.2.2.1. Définir l'annotation max-scale en utilisant le CLI Knative

L'utilisation de l'interface de programmation Knative (**kn**) pour définir l'annotation **max-scale** offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn service** avec l'indicateur **--scale-max** pour créer ou modifier la valeur **max-scale** pour un service.

Conditions préalables

- Knative Serving est installé sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Définissez le nombre maximum de répliques pour le service en utilisant l'option **--scale-max**:

```
$ kn service create <service_name> --image <image_uri> --scale-max <integer>
```

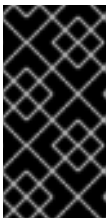
Example command

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --scale-max 10
```

4.2.3. Concurrence

La concurence détermine le nombre de requêtes simultanées qui peuvent être traitées par chaque réplique d'une application à un moment donné. La simultanéité peut être configurée comme *soft limit* ou *hard limit*:

- Une limite souple est une limite de demande ciblée, plutôt qu'une limite strictement appliquée. Par exemple, en cas d'augmentation soudaine du trafic, la limite souple peut être dépassée.
- Une limite stricte est une limite supérieure de demandes strictement appliquée. Si la concurrence atteint la limite supérieure, les demandes excédentaires sont mises en mémoire tampon et doivent attendre qu'il y ait suffisamment de capacité libre pour les exécuter.



IMPORTANT

L'utilisation d'une configuration de limite stricte n'est recommandée que s'il existe un cas d'utilisation clair pour votre application. Le fait de spécifier une limite basse et stricte peut avoir un impact négatif sur le débit et la latence d'une application, et peut provoquer des démarrages à froid.

L'ajout d'une cible souple et d'une limite stricte signifie que l'autoscaler vise la cible souple du nombre de requêtes simultanées, mais impose une limite stricte de la valeur de la limite stricte pour le nombre maximum de requêtes.

Si la valeur de la limite dure est inférieure à la valeur de la limite souple, la valeur de la limite souple est réduite, car il n'est pas nécessaire de cibler plus de demandes que le nombre qui peut réellement être traité.

4.2.3.1. Configuration d'une cible de concurrence douce

Une limite souple est une limite de demande ciblée, plutôt qu'une limite strictement appliquée. Par exemple, s'il y a une explosion soudaine du trafic, la cible de la limite souple peut être dépassée. Vous pouvez spécifier un objectif de concurrence souple pour votre service Knative en définissant l'annotation **autoscaling.knative.dev/target** dans la spécification, ou en utilisant la commande **kn service** avec les drapeaux corrects.

Procédure

- Facultatif : Définissez l'annotation **autoscaling.knative.dev/target** pour votre service Knative dans la spécification de la ressource personnalisée **Service**:

  Exemple de cahier des charges

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target: "200"
```

- Facultatif : Utilisez la commande **kn service** pour spécifier l'indicateur **--concurrency-target**:

```
$ kn service create <service_name> --image <image_uri> --concurrency-target <integer>
```

  Exemple de commande pour créer un service avec un objectif de concurrence de 50 requêtes

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --concurrency-target 50
```

4.2.3.2. Configuration d'une limite de concurrence stricte

Une limite de concurrence stricte est une limite supérieure de demandes strictement appliquée. Si la concurrence atteint la limite dure, les demandes excédentaires sont mises en mémoire tampon et doivent attendre qu'il y ait suffisamment de capacité libre pour les exécuter. Vous pouvez spécifier une limite dure de concurrence pour votre service Knative en modifiant la spécification **containerConcurrency**, ou en utilisant la commande **kn service** avec les drapeaux corrects.

Procédure

- Facultatif : Définissez la spécification **containerConcurrency** pour votre service Knative dans la spécification de la ressource personnalisée **Service**:

  Exemple de cahier des charges

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    spec:
      containerConcurrency: 50
```

  La valeur par défaut est **0**, ce qui signifie qu'il n'y a pas de limite au nombre de requêtes simultanées autorisées dans une réplique du service à la fois.

Une valeur supérieure à **0** spécifie le nombre exact de requêtes autorisées à circuler dans une réplique du service à la fois. Dans cet exemple, la limite de simultanéité est fixée à 50 requêtes.

- Facultatif : Utilisez la commande **kn service** pour spécifier l'indicateur **--concurrency-limit**:

```
$ kn service create <service_name> --image <image_uri> --concurrency-limit <integer>
```

Exemple de commande pour créer un service avec une limite de concurrence de 50 requêtes

```
$ kn service create example-service --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest --concurrency-limit 50
```

4.2.3.3. Utilisation de l'objectif de simultanéité

Cette valeur indique le pourcentage de la limite de concurrence qui est effectivement ciblé par l'autoscaler. Il s'agit également de spécifier la valeur *hotness* à laquelle un réplica s'exécute, ce qui permet à l'autoscaler d'augmenter sa capacité avant que la limite définie ne soit atteinte.

Par exemple, si la valeur **containerConcurrency** est fixée à 10 et la valeur **target-utilization-percentage** à 70 %, l'autoscaler crée une nouvelle réplique lorsque le nombre moyen de requêtes simultanées dans toutes les répliques existantes atteint 7. Les demandes numérotées de 7 à 10 sont toujours envoyées aux répliques existantes, mais des répliques supplémentaires sont démarrées en prévision d'un besoin lorsque la valeur **containerConcurrency** est atteinte.

Exemple de service configuré à l'aide de l'annotation target-utilization-percentage

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
  template:
    metadata:
      annotations:
        autoscaling.knative.dev/target-utilization-percentage: "70"
...
```

4.2.4. Échelle zéro

Knative Serving permet une mise à l'échelle automatique, ou *autoscaling*, des applications pour répondre à la demande entrante.

4.2.4.1. Permettre le passage à l'échelle zéro

Vous pouvez utiliser la spécification **enable-scale-to-zero** pour activer ou désactiver globalement le passage à l'échelle zéro pour les applications sur le cluster.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Serving sur votre cluster.

- Vous avez des droits d'administrateur de cluster.
- Vous utilisez le Knative Pod Autoscaler par défaut. La fonction de mise à zéro n'est pas disponible si vous utilisez le Kubernetes Horizontal Pod Autoscaler.

Procédure

- Modifier la spécification **enable-scale-to-zero** dans la ressource personnalisée (CR) **KnativeServing**:

  Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    autoscaler:
      enable-scale-to-zero: "false" 1
```

1 La spécification **enable-scale-to-zero** peut être **"true"** ou **"false"**. Si elle vaut true, la fonction scale-to-zero est activée. S'il est défini sur false, les applications sont réduites à la valeur configurée *minimum scale bound*. La valeur par défaut est **"true"**.

4.2.4.2. Configuration du délai de grâce scale-to-zero

Knative Serving fournit une mise à l'échelle automatique jusqu'à zéro pods pour les applications. Vous pouvez utiliser la spécification **scale-to-zero-grace-period** pour définir une limite de temps supérieure pendant laquelle Knative attend que le mécanisme de mise à l'échelle vers zéro soit en place avant que la dernière réplique d'une application soit supprimée.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Vous avez des droits d'administrateur de cluster.
- Vous utilisez le Knative Pod Autoscaler par défaut. La fonctionnalité scale-to-zero n'est pas disponible si vous utilisez le Kubernetes Horizontal Pod Autoscaler.

Procédure

- Modifier la spécification **scale-to-zero-grace-period** dans la ressource personnalisée (CR) **KnativeServing**:

  Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
```

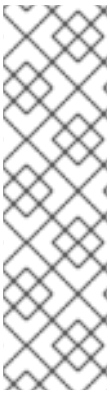
```
config:
  autoscaler:
    scale-to-zero-grace-period: "30s" 1
```

1 Durée du délai de grâce en secondes. La valeur par défaut est de 30 secondes.

4.3. CONFIGURATION DES APPLICATIONS SANS SERVEUR

4.3.1. Remplacer les configurations de déploiement du système Knative Serving

Vous pouvez remplacer les configurations par défaut pour certains déploiements spécifiques en modifiant la spécification **deployments** dans les ressources personnalisées (CR) **KnativeServing**.



NOTE

Vous ne pouvez remplacer que les sondes définies par défaut dans le déploiement.

Tous les déploiements de Knative Serving définissent par défaut une sonde de préparation et une sonde de disponibilité, à ces exceptions près :

- **net-kourier-controller** et **3scale-kourier-gateway** ne définissent qu'une sonde de préparation.
- **net-istio-controller** et **net-istio-webhook** ne définissent pas de sondes.

4.3.1.1. Remplacer les configurations de déploiement du système

Actuellement, le remplacement des paramètres de configuration par défaut est possible pour les champs **resources**, **replicas**, **labels**, **annotations** et **nodeSelector**, ainsi que pour les champs **readiness** et **liveness** pour les sondes.

Dans l'exemple suivant, une CR **KnativeServing** remplace le déploiement **webhook** de sorte que :

- Le délai d'attente de la sonde **readiness** pour **net-kourier-controller** est fixé à 10 secondes.
- Le déploiement a spécifié des limites de ressources de CPU et de mémoire.
- Le déploiement comporte 3 répliques.
- L'étiquette **example-label: label** est ajoutée.
- L'annotation **example-annotation: annotation** est ajoutée.
- Le champ **nodeSelector** est défini pour sélectionner les nœuds portant l'étiquette **disktype: hdd**.

NOTE

Les paramètres d'étiquetage et d'annotation de **KnativeServing** CR remplacent les étiquettes et les annotations du déploiement, tant pour le déploiement lui-même que pour les pods qui en résultent.

Exemple de CR KnativeServing

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: ks
  namespace: knative-serving
spec:
  high-availability:
    replicas: 2
  deployments:
  - name: net-kourier-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
  - name: webhook
    resources:
    - container: webhook
      requests:
        cpu: 300m
        memory: 60Mi
      limits:
        cpu: 1000m
        memory: 1000Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
      example-annotation: annotation
    nodeSelector:
      disktype: hdd
```

1 Vous pouvez utiliser les surcharges de sonde **readiness** et **liveness** pour remplacer tous les champs d'une sonde dans un conteneur d'un déploiement comme spécifié dans l'API Kubernetes, à l'exception des champs liés au gestionnaire de la sonde : **exec**, **grpc**, **httpGet**, et **tcpSocket**.

Ressources supplémentaires

- Section sur la configuration des sondes de la documentation de l'API Kubernetes

4.3.2. Prise en charge de plusieurs conteneurs pour le service

Vous pouvez déployer un pod multi-conteneurs en utilisant un seul service Knative. Cette méthode est utile pour séparer les responsabilités de l'application en parties plus petites et spécialisées.

IMPORTANT

La prise en charge de plusieurs conteneurs pour Serving est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes d'un point de vue fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

4.3.2.1. Configuration d'un service multi-conteneurs

La prise en charge de plusieurs conteneurs est activée par défaut. Vous pouvez créer un pod multi-conteneurs en spécifiant plusieurs conteneurs dans le service.

Procédure

1. Modifiez votre service pour inclure des conteneurs supplémentaires. Un seul conteneur peut traiter les requêtes, il faut donc spécifier **ports** pour un seul conteneur. Voici un exemple de configuration avec deux conteneurs :

Configuration de plusieurs conteneurs

```
apiVersion: serving.knative.dev/v1
kind: Service
...
spec:
  template:
    spec:
      containers:
        - name: first-container 1
          image: gcr.io/knative-samples/helloworld-go
          ports:
            - containerPort: 8080 2
        - name: second-container 3
          image: gcr.io/knative-samples/helloworld-java
```

1. Première configuration du conteneur.
2. Spécification du port pour le premier conteneur.
3. Deuxième configuration du conteneur.

4.3.3. Volumes EmptyDir

emptyDir les volumes **emptyDir** sont des volumes vides créés lors de la création d'un pod et utilisés pour fournir un espace disque de travail temporaire. Les volumes sont supprimés lorsque le pod pour lequel ils ont été créés est supprimé.

4.3.3.1. Configuration de l'extension EmptyDir

L'extension **kubernetes.podspec-volumes-emptydir** contrôle si les volumes **emptyDir** peuvent être utilisés avec Knative Serving. Pour permettre l'utilisation des volumes **emptyDir**, vous devez modifier la ressource personnalisée (CR) **KnativeServing** pour inclure le YAML suivant :

Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-volumes-emptydir: enabled
...
```

4.3.4. Réclamations de volumes persistants pour la mise en service

Certaines applications sans serveur ont besoin d'un stockage permanent des données. Pour y parvenir, vous pouvez configurer des réclamations de volumes persistants (PVC) pour vos services Knative.

4.3.4.1. Activation de la prise en charge du PVC

Procédure

1. Pour permettre à Knative Serving d'utiliser les PVC et d'y écrire, modifiez la ressource personnalisée (CR) **KnativeServing** pour inclure le YAML suivant :

   Activation des PVC avec accès en écriture

   ```
   ...
   spec:
     config:
       features:
         "kubernetes.podspec-persistent-volume-claim": enabled
         "kubernetes.podspec-persistent-volume-write": enabled
   ...
   ```

   - L'extension **kubernetes.podspec-persistent-volume-claim** détermine si les volumes persistants (PV) peuvent être utilisés avec Knative Serving.
   - L'extension **kubernetes.podspec-persistent-volume-write** détermine si les PV sont disponibles pour Knative Serving avec l'accès en écriture.

2. Pour réclamer un PV, modifiez votre service pour inclure la configuration du PV. Par exemple, vous pouvez avoir une demande de volume persistant avec la configuration suivante :

   NOTE

   Utilisez la classe de stockage qui prend en charge le mode d'accès que vous demandez. Par exemple, vous pouvez utiliser la classe **ocs-storagecluster-cephfs** pour le mode d'accès **ReadWriteMany**.

Configuration de PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: example-pv-claim
  namespace: my-ns
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ocs-storagecluster-cephfs
  resources:
    requests:
      storage: 1Gi
```

Dans ce cas, pour réclamer un PV avec accès en écriture, modifiez votre service comme suit :

Configuration du service Knative PVC

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  namespace: my-ns
...
spec:
 template:
   spec:
     containers:
         ...
         volumeMounts: (1)
           - mountPath: /data
             name: mydata
             readOnly: false
     volumes:
       - name: mydata
         persistentVolumeClaim: (2)
           claimName: example-pv-claim
           readOnly: false (3)
```

1. Spécification de montage de volume.
2. Spécification de la demande de volume persistant.
3. Indicateur permettant l'accès en lecture seule.

NOTE

Pour utiliser avec succès le stockage persistant dans les services Knative, vous avez besoin d'une configuration supplémentaire, telle que les autorisations de l'utilisateur du conteneur Knative.

4.3.4.2. Ressources supplémentaires

- Comprendre le stockage persistant

4.3.5. Init containers

Les conteneurs d'initialisation sont des conteneurs spécialisés qui sont exécutés avant les conteneurs d'application dans un pod. Ils sont généralement utilisés pour mettre en œuvre la logique d'initialisation d'une application, ce qui peut inclure l'exécution de scripts d'installation ou le téléchargement des configurations requises. Vous pouvez activer l'utilisation de conteneurs d'initialisation pour les services Knative en modifiant la ressource personnalisée (CR) **KnativeServing**.

NOTE

Les conteneurs Init peuvent entraîner des temps de démarrage d'application plus longs et doivent être utilisés avec prudence pour les applications sans serveur, qui sont censées évoluer fréquemment à la hausse et à la baisse.

4.3.5.1. Activation des conteneurs init

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Vous avez des droits d'administrateur de cluster.

Procédure

- Activez l'utilisation des conteneurs init en ajoutant le drapeau **kubernetes.podspec-init-containers** au CR **KnativeServing**:

  Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    features:
      kubernetes.podspec-init-containers: enabled
...
```

4.3.6. Transformer les balises d'images en condensés

Si le contrôleur Knative Serving a accès au registre des conteneurs, Knative Serving résout les balises d'image en un condensé lorsque vous créez une révision d'un service. Ceci est connu sous le nom de *tag-to-digest resolution*, et permet d'assurer la cohérence des déploiements.

4.3.6.1. Résolution Tag-to-digest

Pour donner au contrôleur l'accès au registre des conteneurs sur OpenShift Container Platform, vous devez créer un secret et ensuite configurer les certificats personnalisés du contrôleur. Vous pouvez configurer les certificats personnalisés du contrôleur en modifiant la spécification **controller-custom-**

certs dans la ressource personnalisée (CR) **KnativeServing**. Le secret doit résider dans le même espace de noms que la CR **KnativeServing**.

Si un secret n'est pas inclus dans le CR **KnativeServing**, ce paramètre utilise par défaut l'infrastructure à clé publique (PKI). Lors de l'utilisation de l'ICP, les certificats de l'ensemble du cluster sont automatiquement injectés dans le contrôleur Knative Serving à l'aide de la carte de configuration **config-service-sa**. L'OpenShift Serverless Operator remplit la carte de configuration **config-service-sa** avec les certificats de l'ensemble du cluster et monte la carte de configuration en tant que volume sur le contrôleur.

4.3.6.1.1. Configuration de la résolution tag-to-digest par l'utilisation d'un secret

Si la spécification **controller-custom-certs** utilise le type **Secret**, le secret est monté en tant que volume secret. Les composants Knative consomment le secret directement, en supposant que le secret possède les certificats requis.

Conditions préalables

- Vous disposez des droits d'administrateur de cluster sur OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Serving sur votre cluster.

Procédure

1. Créez un secret :

   Example command

   ```
   oc -n knative-serving create secret generic custom-secret --from-file=<secret_name>.crt=
   <path_to_certificate>
   ```

2. Configurez la spécification **controller-custom-certs** dans la ressource personnalisée (CR) **KnativeServing** pour utiliser le type **Secret**:

   Exemple KnativeServing CR

   ```
   apiVersion: operator.knative.dev/v1beta1
   kind: KnativeServing
   metadata:
     name: knative-serving
     namespace: knative-serving
   spec:
     controller-custom-certs:
       name: custom-secret
       type: Secret
   ```

4.3.7. Configuration de l'authentification TLS

Vous pouvez utiliser *Transport Layer Security* (TLS) pour crypter le trafic Knative et pour l'authentification.

TLS est la seule méthode de cryptage du trafic prise en charge pour Knative Kafka. Red Hat recommande d'utiliser à la fois SASL et TLS pour le courtier Knative pour les ressources Apache Kafka.

NOTE

Si vous souhaitez activer TLS interne avec une intégration Red Hat OpenShift Service Mesh, vous devez activer Service Mesh avec mTLS au lieu du chiffrement interne expliqué dans la procédure suivante. Voir la documentation pour l'activation des métriques Knative Serving lors de l'utilisation de Service Mesh avec mTLS.

4.3.7.1. Activation de l'authentification TLS pour le trafic interne

OpenShift Serverless prend en charge la terminaison TLS par défaut, de sorte que le trafic HTTPS des utilisateurs finaux est chiffré. Cependant, le trafic interne derrière la route OpenShift est transmis aux applications en utilisant des données en clair. En activant TLS pour le trafic interne, le trafic envoyé entre les composants est chiffré, ce qui rend ce trafic plus sûr.

NOTE

Si vous souhaitez activer TLS interne avec une intégration Red Hat OpenShift Service Mesh, vous devez activer Service Mesh avec mTLS au lieu du chiffrement interne expliqué dans la procédure suivante.

IMPORTANT

La prise en charge du cryptage TLS interne est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Serving.
- Vous avez installé le CLI OpenShift (**oc**).

Procédure

1. Créer un service Knative qui inclut le champ **internal-encryption: "true"** dans la spécification :

```
...
spec:
  config:
    network:
      internal-encryption: "true"
...
```

2. Redémarrez les pods activateurs dans l'espace de noms **knative-serving** pour charger les certificats :

```
$ oc delete pod -n knative-serving --selector app=activator
```

Ressources supplémentaires

- Configuration de l'authentification TLS pour le courtier Knative pour Apache Kafka
- Configuration de l'authentification TLS pour les canaux pour Apache Kafka
- Activation des métriques Knative Serving lors de l'utilisation de Service Mesh avec mTLS

4.3.8. Politiques de réseau restrictives

4.3.8.1. Clusters avec des politiques de réseau restrictives

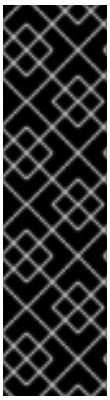
Si vous utilisez un cluster auquel plusieurs utilisateurs ont accès, votre cluster peut utiliser des stratégies de réseau pour contrôler quels pods, services et espaces de noms peuvent communiquer les uns avec les autres sur le réseau. Si votre cluster utilise des stratégies de réseau restrictives, il est possible que les pods du système Knative ne puissent pas accéder à votre application Knative. Par exemple, si votre espace de noms a la politique de réseau suivante, qui refuse toutes les demandes, les pods du système Knative ne peuvent pas accéder à votre application Knative :

Exemple d'objet NetworkPolicy qui refuse toutes les demandes d'accès à l'espace de noms

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: example-namespace
spec:
  podSelector:
  ingress: []
```

4.3.8.2. Permettre la communication avec les applications Knative sur un cluster avec des politiques de réseau restrictives

Pour permettre l'accès à vos applications à partir des pods du système Knative, vous devez ajouter un label à chacun des espaces de noms du système Knative, puis créer un objet **NetworkPolicy** dans l'espace de noms de votre application qui autorise l'accès à l'espace de noms pour d'autres espaces de noms qui ont ce label.



IMPORTANT

Une politique de réseau qui refuse les demandes de services non Knative sur votre cluster empêche toujours l'accès à ces services. Cependant, en autorisant l'accès à votre application Knative à partir des espaces de noms du système Knative, vous autorisez l'accès à votre application Knative à partir de tous les espaces de noms du cluster.

Si vous ne souhaitez pas autoriser l'accès à votre application Knative à partir de tous les espaces de noms du cluster, vous pouvez utiliser *JSON Web Token authentication for Knative services* à la place. L'authentification par jeton Web JSON pour les services Knative nécessite Service Mesh.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.

Procédure

1. Ajoutez l'étiquette **knative.openshift.io/system-namespace=true** à chaque espace de noms du système Knative qui nécessite un accès à votre application :
   a. Étiqueter l'espace de noms **knative-serving**:

   ```
   $ oc label namespace knative-serving knative.openshift.io/system-namespace=true
   ```

   b. Étiqueter l'espace de noms **knative-serving-ingress**:

   ```
   $ oc label namespace knative-serving-ingress knative.openshift.io/system-namespace=true
   ```

   c. Étiqueter l'espace de noms **knative-eventing**:

   ```
   $ oc label namespace knative-eventing knative.openshift.io/system-namespace=true
   ```

   d. Étiqueter l'espace de noms **knative-kafka**:

   ```
   $ oc label namespace knative-kafka knative.openshift.io/system-namespace=true
   ```

2. Créez un objet **NetworkPolicy** dans l'espace de noms de votre application afin d'autoriser l'accès à partir des espaces de noms portant l'étiquette **knative.openshift.io/system-namespace**:

   Exemple d'objet NetworkPolicy

   ```
   apiVersion: networking.k8s.io/v1
   kind: NetworkPolicy
   metadata:
     name: <network_policy_name> 1
     namespace: <namespace> 2
   spec:
     ingress:
     - from:
       - namespaceSelector:
           matchLabels:
             knative.openshift.io/system-namespace: "true"
     podSelector: {}
     policyTypes:
     - Ingress
   ```

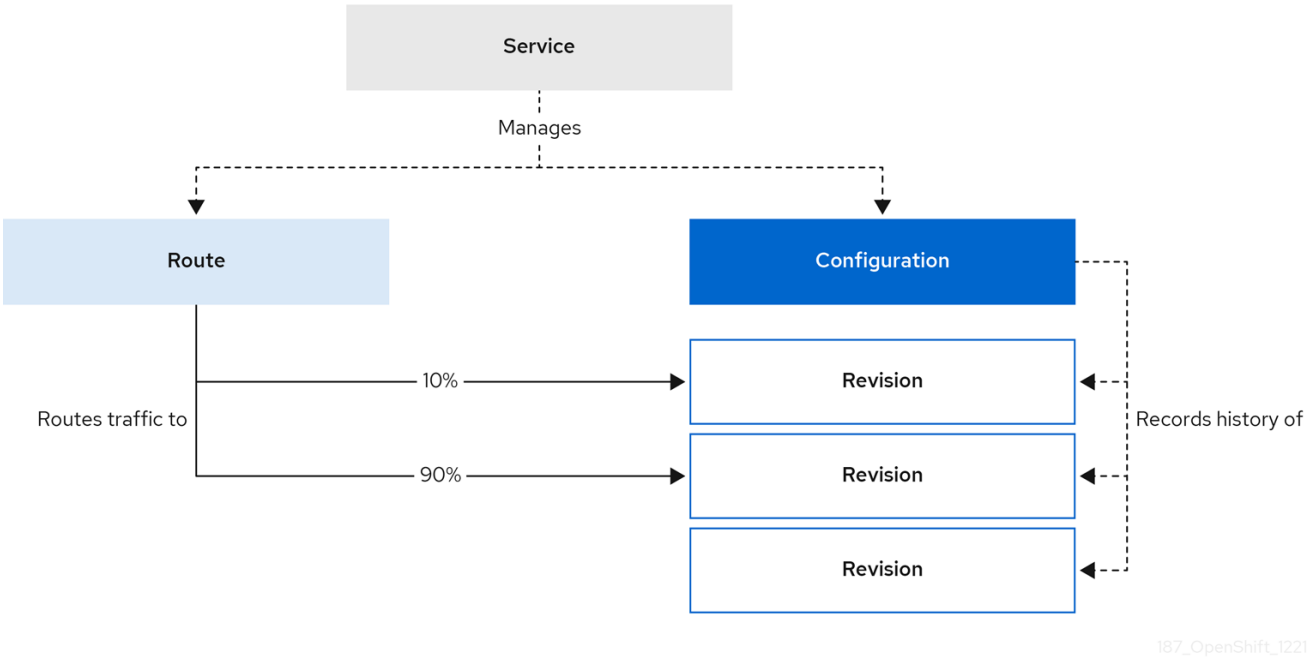
   1 Donnez un nom à votre politique de réseau.

   2 L'espace de noms dans lequel votre application existe.

4.4. FRACTIONNEMENT DU TRAFIC

4.4.1. Vue d'ensemble du fractionnement du trafic

Dans une application Knative, le trafic peut être géré en créant une division du trafic. Un partage de trafic est configuré comme une partie d'une route, qui est gérée par un service Knative.



La configuration d'une route permet d'envoyer des requêtes à différentes révisions d'un service. Ce routage est déterminé par la spécification **traffic** de l'objet **Service**.

Une déclaration de spécification **traffic** consiste en une ou plusieurs révisions, chacune étant responsable de la gestion d'une partie du trafic global. Les pourcentages de trafic acheminés vers chaque révision doivent être égaux à 100 %, ce qui est garanti par une validation Knative.

Les révisions spécifiées dans une spécification **traffic** peuvent être soit une révision fixe et nommée, soit pointer vers la "dernière" révision, qui suit la tête de la liste de toutes les révisions pour le service. La révision "la plus récente" est un type de référence flottante qui se met à jour si une nouvelle révision est créée. Chaque révision peut être associée à une balise qui crée une URL d'accès supplémentaire pour cette révision.

La spécification **traffic** peut être modifiée par :

- Éditer directement le YAML d'un objet **Service**.
- Utilisation de l'indicateur Knative (**kn**) CLI **--traffic**.
- Utilisation de la console web OpenShift Container Platform.

Lorsque vous créez un service Knative, il n'a pas de paramètres par défaut **traffic** spec.

4.4.2. Exemples de spécifications de trafic

L'exemple suivant montre une spécification **traffic** où 100 % du trafic est acheminé vers la dernière révision du service. Sous **status**, vous pouvez voir le nom de la dernière révision à laquelle **latestRevision** se réfère :

```
apiVersion: serving.knative.dev/v1
```

```
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - latestRevision: true
    percent: 100
status:
  ...
  traffic:
  - percent: 100
    revisionName: example-service
```

L'exemple suivant montre une spécification **traffic** où 100 % du trafic est acheminé vers la révision étiquetée **current**, et le nom de cette révision est spécifié comme **example-service**. La révision étiquetée **latest** reste disponible, même si aucun trafic n'est acheminé vers elle :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service
    percent: 100
  - tag: latest
    latestRevision: true
    percent: 0
```

L'exemple suivant montre comment la liste des révisions de la spécification **traffic** peut être étendue pour que le trafic soit réparti entre plusieurs révisions. Cet exemple envoie 50 % du trafic à la révision étiquetée **current**, et 50 % du trafic à la révision étiquetée **candidate**. La révision étiquetée **latest** reste disponible, même si aucun trafic n'est acheminé vers elle :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
  namespace: default
spec:
...
  traffic:
  - tag: current
    revisionName: example-service-1
    percent: 50
  - tag: candidate
    revisionName: example-service-2
    percent: 50
```

```
- tag: latest
  latestRevision: true
  percent: 0
```

4.4.3. Fractionnement du trafic à l'aide de la CLI Knative

L'utilisation de la CLI Knative (**kn**) pour créer des scissions de trafic offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn service update** pour diviser le trafic entre les révisions d'un service.

4.4.3.1. Création d'une division du trafic à l'aide du CLI Knative

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un service Knative.

Procédure

- Spécifiez la révision de votre service et le pourcentage de trafic que vous voulez acheminer vers lui en utilisant la balise **--traffic** avec une commande standard **kn service update**:

  Example command

  ```
  $ kn service update <service_name> --traffic <revision>=<percentage>
  ```

  Où ?

  - **<service_name>** est le nom du service Knative pour lequel vous configurez le routage du trafic.
  - **<revision>** est la révision que vous souhaitez configurer pour recevoir un pourcentage du trafic. Vous pouvez spécifier soit le nom de la révision, soit une étiquette que vous avez attribuée à la révision en utilisant l'indicateur **--tag**.
  - **<percentage>** est le pourcentage de trafic que vous souhaitez envoyer à la révision spécifiée.

- Facultatif : L'indicateur **--traffic** peut être spécifié plusieurs fois dans une commande. Par exemple, si vous avez une révision étiquetée comme **@latest** et une révision nommée **stable**, vous pouvez spécifier le pourcentage de trafic que vous voulez répartir sur chaque révision comme suit :

  Example command

  ```
  $ kn service update example-service --traffic @latest=20,stable=80
  ```

  Si vous avez plusieurs révisions et que vous ne spécifiez pas le pourcentage de trafic qui doit être attribué à la dernière révision, l'indicateur **--traffic** peut le calculer automatiquement. Par exemple, si vous avez une troisième révision nommée **example**, et que vous utilisez la commande suivante :

Example command

```
$ kn service update example-service --traffic @latest=10,stable=60
```

Les 30 % restants du trafic sont répartis sur la révision **example**, même si elle n'a pas été spécifiée.

4.4.4. Indicateurs CLI pour la répartition du trafic

La CLI Knative (**kn**) prend en charge les opérations de trafic sur le bloc de trafic d'un service dans le cadre de la commande **kn service update**.

4.4.4.1. Drapeaux de division du trafic de la CLI Knative

Le tableau suivant présente un résumé des indicateurs de répartition du trafic, des formats de valeur et de l'opération effectuée par l'indicateur. La colonne **Repetition** indique si la répétition de la valeur particulière de l'indicateur est autorisée dans une commande **kn service update**.

Drapeau	Valeur(s)	Fonctionnement	Répétition
--traffic	**RevisionName=Percent**	Donne à **Percent** la possibilité de circuler sur **RevisionName**	Oui
--traffic	**Tag=Percent**	Donne le trafic **Percent** à la révision ayant **Tag**	Oui
--traffic	**@latest=Percent**	**Percent** permet d'accéder à la dernière révision prête à l'emploi	Non
--tag	**RevisionName=Tag**	Donne **Tag** à **RevisionName**	Oui
--tag	**@latest=Tag**	Donne **Tag** à la dernière révision prête	Non
--untag	**Tag**	Supprime **Tag** de la révision	Oui

4.4.4.1.1. Drapeaux multiples et ordre de préséance

Tous les indicateurs relatifs au trafic peuvent être spécifiés à l'aide d'une seule commande **kn service update**. **kn** définit la priorité de ces indicateurs. L'ordre des indicateurs spécifiés lors de l'utilisation de la commande n'est pas pris en compte.

La priorité des drapeaux tels qu'ils sont évalués par **kn** est la suivante :

1. **--untag**: Toutes les révisions référencées avec cet indicateur sont supprimées du bloc de trafic.
2. **--tag**: Les révisions sont étiquetées comme spécifié dans le bloc de trafic.

3. **--traffic**: Les révisions référencées se voient attribuer une partie de la répartition du trafic.

Vous pouvez ajouter des balises aux révisions et diviser le trafic en fonction des balises que vous avez définies.

4.4.4.1.2. URL personnalisés pour les révisions

L'attribution de l'indicateur **--tag** à un service à l'aide de la commande **kn service update** crée une URL personnalisée pour la révision qui est créée lorsque vous mettez à jour le service. L'URL personnalisée suit le modèle **https://<tag>-<service_name>-<namespace>.<domain>** ou **http://<tag>-<service_name>-<namespace>.<domain>**.

Les drapeaux **--tag** et **--untag** utilisent la syntaxe suivante :

- Exiger une valeur.
- Indique une balise unique dans le bloc de trafic du service.
- Peut être spécifié plusieurs fois dans une même commande.

4.4.4.1.2.1. Exemple : Attribuer une étiquette à une révision

L'exemple suivant attribue l'étiquette **latest** à une révision nommée **example-revision**:

```
$ kn service update -YRFFGUNA nom_du_service> --tag @latest=example-tag
```

4.4.4.1.2.2. Exemple : Supprimer une balise d'une révision

Vous pouvez supprimer une balise pour supprimer l'URL personnalisée, en utilisant l'indicateur **--untag**.

NOTE

Si les balises d'une révision sont supprimées et que 0 % du trafic lui est attribué, la révision est entièrement supprimée du bloc de trafic.

La commande suivante supprime toutes les balises de la révision nommée **example-revision**:

```
$ kn service update -YRFFGUNA nom_du_service> --untag exemple-tag
```

4.4.5. Diviser le trafic entre les révisions

Après avoir créé une application sans serveur, l'application est affichée dans la vue **Topology** de la perspective **Developer** dans la console web d'OpenShift Container Platform. La révision de l'application est représentée par le nœud, et le service Knative est indiqué par un quadrilatère autour du nœud.

Toute nouvelle modification du code ou de la configuration du service crée une nouvelle révision, qui est un instantané du code à un moment donné. Pour un service, vous pouvez gérer le trafic entre les révisions du service en le divisant et en l'acheminant vers les différentes révisions selon les besoins.

4.4.5.1. Gérer le trafic entre les révisions en utilisant la console web d'OpenShift Container Platform

Conditions préalables

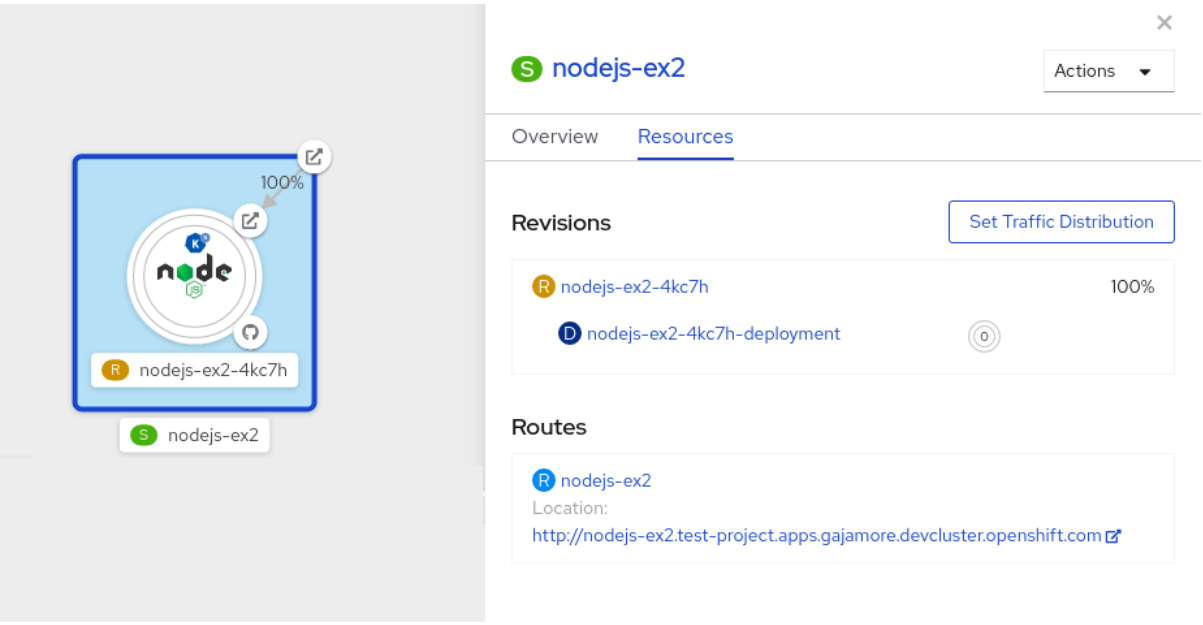
- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Vous vous êtes connecté à la console web de OpenShift Container Platform.

Procédure

Pour répartir le trafic entre plusieurs révisions d'une application dans la vue **Topology**:

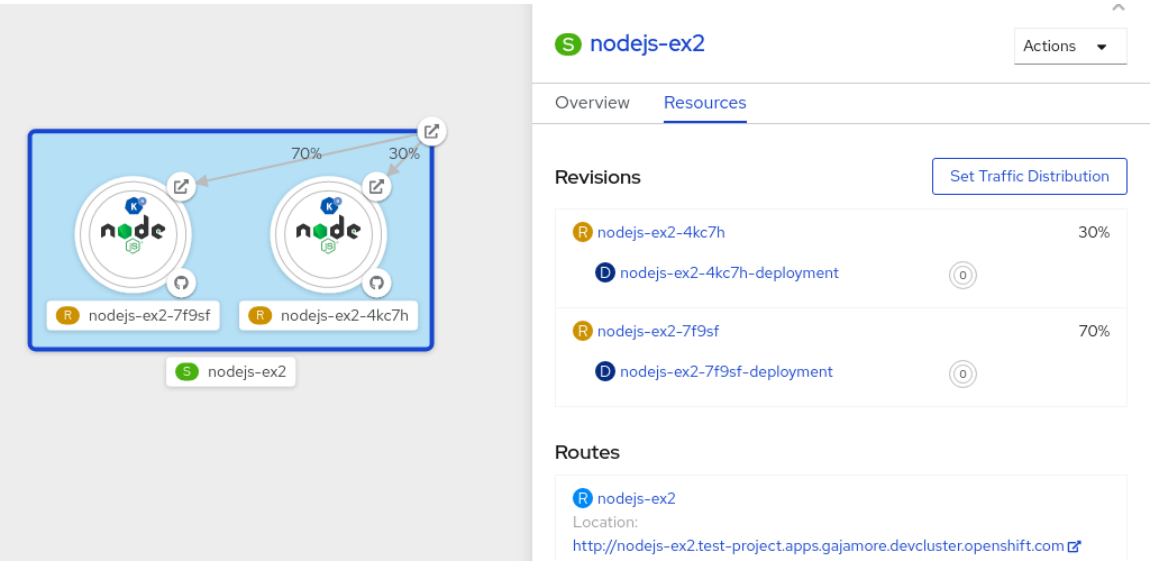
1. Cliquez sur le service Knative pour voir son aperçu dans le panneau latéral.
2. Cliquez sur l'onglet **Resources** pour obtenir une liste de **Revisions** et **Routes** pour le service.

Figure 4.1. Application sans serveur



3. Cliquez sur le service, indiqué par l'icône **S** en haut du panneau latéral, pour obtenir une vue d'ensemble des détails du service.
4. Cliquez sur l'onglet **YAML** et modifiez la configuration du service dans l'éditeur YAML, puis cliquez sur **Save**. Par exemple, changez le **timeoutseconds** de 300 à 301 . Cette modification de la configuration déclenche une nouvelle révision. Dans la vue **Topology**, la dernière révision est affichée et l'onglet **Resources** du service affiche maintenant les deux révisions.
5. Dans l'onglet **Resources**, cliquez sur **Définir la distribution du trafic** pour afficher la boîte de dialogue de distribution du trafic :
   a. Ajoutez le pourcentage de trafic divisé pour les deux révisions dans le champ **Splits**.
   b. Ajoutez des balises pour créer des URL personnalisées pour les deux révisions.
   c. Cliquez sur **Save** pour voir apparaître deux nœuds représentant les deux révisions dans la vue Topologie.

Figure 4.2. Révisions d'applications sans serveur



4.4.6. Reroutage du trafic à l'aide de la stratégie bleu-vert

Vous pouvez réacheminer en toute sécurité le trafic d'une version de production d'une application vers une nouvelle version, en utilisant une stratégie de déploiement bleu-vert .

4.4.6.1. Routage et gestion du trafic grâce à une stratégie de déploiement bleu-vert

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créer et déployer une application en tant que service Knative.
2. Trouvez le nom de la première révision qui a été créée lorsque vous avez déployé le service, en consultant la sortie de la commande suivante :

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

Example command

```
$ oc get ksvc example-service -o=jsonpath='{.status.latestCreatedRevisionName}'
```

Exemple de sortie

```
$ example-service-00001
```

3. Ajoutez le fichier YAML suivant au service **spec** pour envoyer le trafic entrant à la révision :

```
...
spec:
  traffic:
```

```
    - revisionName: <first_revision_name>
      percent: 100 # All traffic goes to this revision
...
```

4. Vérifiez que vous pouvez voir votre application à l'URL que vous obtenez en exécutant la commande suivante :

```
oc get ksvc -YRFFGUNA nom_du_service>
```

5. Déployez une deuxième révision de votre application en modifiant au moins un champ de la spécification **template** du service et en le redéployant. Par exemple, vous pouvez modifier l'adresse **image** du service, ou une variable d'environnement **env**. Vous pouvez redéployer le service en appliquant le fichier YAML du service, ou en utilisant la commande **kn service update** si vous avez installé le CLI Knative (**kn**).

6. Recherchez le nom de la deuxième révision, la plus récente, qui a été créée lorsque vous avez redéployé le service, en exécutant la commande :

```
$ oc get ksvc <service_name> -o=jsonpath='{.status.latestCreatedRevisionName}'
```

   À ce stade, les première et deuxième révisions du service sont déployées et fonctionnent.

7. Mettez à jour votre service existant pour créer un nouveau point d'arrivée test pour la deuxième révision, tout en continuant à envoyer tout le reste du trafic vers la première révision :

   Exemple de spécification de service mise à jour avec point final de test

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 100 # All traffic is still being routed to the first revision
    - revisionName: <second_revision_name>
      percent: 0 # No traffic is routed to the second revision
      tag: v2 # A named route
...
```

   Après avoir redéployé ce service en réappliquant la ressource YAML, la deuxième révision de l'application est maintenant mise en scène. Aucun trafic n'est acheminé vers la deuxième révision à l'URL principale, et Knative crée un nouveau service nommé **v2** pour tester la nouvelle révision déployée.

8. Obtenez l'URL du nouveau service pour la deuxième révision, en exécutant la commande suivante :

```
$ oc get ksvc <service_name> --output jsonpath="{.status.traffic[*].url}"
```

   Vous pouvez utiliser cette URL pour vérifier que la nouvelle version de l'application se comporte comme prévu avant d'y acheminer du trafic.

9. Mettez à nouveau à jour votre service existant, de sorte que 50 % du trafic soit envoyé à la première révision et 50 % à la seconde :

   Exemple de spécification de service actualisée divisant le trafic 50/50 entre les révisions

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 50
    - revisionName: <second_revision_name>
      percent: 50
      tag: v2
...
```

10. Lorsque vous êtes prêt à acheminer tout le trafic vers la nouvelle version de l'application, mettez à nouveau à jour le service pour envoyer 100 % du trafic vers la deuxième révision :

Exemple de spécification de service mise à jour envoyant tout le trafic à la deuxième révision

```
...
spec:
  traffic:
    - revisionName: <first_revision_name>
      percent: 0
    - revisionName: <second_revision_name>
      percent: 100
      tag: v2
...
```

ASTUCE

Vous pouvez supprimer la première révision au lieu de la fixer à 0 % du trafic si vous ne prévoyez pas de revenir en arrière. Les objets de révision non acheminables sont alors mis au rebut.

11. Visitez l'URL de la première révision pour vérifier qu'aucun trafic n'est plus envoyé vers l'ancienne version de l'application.

4.5. ROUTAGE EXTERNE ET ENTRANT

4.5.1. Aperçu du routage

Knative exploite la terminaison TLS de OpenShift Container Platform pour assurer le routage des services Knative. Lorsqu'un service Knative est créé, une route OpenShift Container Platform est automatiquement créée pour le service. Cette route est gérée par l'opérateur OpenShift Serverless. La route OpenShift Container Platform expose le service Knative à travers le même domaine que le cluster OpenShift Container Platform.

Vous pouvez désactiver le contrôle par l'opérateur du routage d'OpenShift Container Platform afin de configurer une route Knative pour utiliser directement vos certificats TLS à la place.

Les routes Knative peuvent également être utilisées avec la route OpenShift Container Platform pour fournir des capacités de routage plus fines, telles que la division du trafic.

4.5.1.1. Ressources supplémentaires

- Annotations spécifiques à l'itinéraire

4.5.2. Personnalisation des étiquettes et des annotations

Les routes OpenShift Container Platform prennent en charge l'utilisation d'étiquettes et d'annotations personnalisées, que vous pouvez configurer en modifiant la spécification **metadata** d'un service Knative. Les labels et annotations personnalisés sont propagés du service à la route Knative, puis à l'ingress Knative et enfin à la route OpenShift Container Platform.

4.5.2.1. Personnaliser les étiquettes et les annotations pour les routes de OpenShift Container Platform

Conditions préalables

- Vous devez avoir l'OpenShift Serverless Operator et Knative Serving installés sur votre cluster OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez un service Knative qui contient l'étiquette ou l'annotation que vous souhaitez propager à la route OpenShift Container Platform :
   - Pour créer un service en utilisant YAML :

     Exemple de service créé à l'aide de YAML

     ```
     apiVersion: serving.knative.dev/v1
     kind: Service
     metadata:
       name: <service_name>
       labels:
         <label_name>: <label_value>
       annotations:
         <annotation_name>: <annotation_value>
     ...
     ```

   - Pour créer un service en utilisant le CLI Knative (**kn**), entrez :

     Exemple de service créé à l'aide d'une commande kn

     ```
     $ kn service create <service_name> \
       --image=<image> \
       --annotation <annotation_name>=<annotation_value> \
       --label <label_value>=<label_value>
     ```

2. Vérifiez que la route OpenShift Container Platform a été créée avec l'annotation ou l'étiquette que vous avez ajoutée en inspectant la sortie de la commande suivante :

   Exemple de commande de vérification

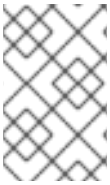
   ```
   $ oc get routes.route.openshift.io \
       -l serving.knative.openshift.io/ingressName=<service_name> \ 1
       -l serving.knative.openshift.io/ingressNamespace=<service_namespace> \ 2
   ```

```
    -n knative-serving-ingress -o yaml \
       | grep -e "<label_name>: \"<label_value>\""  -e "<annotation_name>:
<annotation_value>" 3
```

1. Utilisez le nom de votre service.
2. Utilisez l'espace de noms dans lequel votre service a été créé.
3. Utilisez vos valeurs pour les noms et valeurs de l'étiquette et de l'annotation.

4.5.3. Configuration des routes pour les services Knative

Si vous souhaitez configurer un service Knative pour utiliser votre certificat TLS sur OpenShift Container Platform, vous devez désactiver la création automatique d'une route pour le service par OpenShift Serverless Operator et créer manuellement une route pour le service.



NOTE

Lorsque vous effectuez la procédure suivante, la route par défaut d'OpenShift Container Platform dans l'espace de noms **knative-serving-ingress** n'est pas créée. Cependant, la route Knative pour l'application est toujours créée dans cet espace de noms.

4.5.3.1. Configurer les routes de OpenShift Container Platform pour les services Knative

Conditions préalables

- L'opérateur OpenShift Serverless et le composant Knative Serving doivent être installés sur votre cluster OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créer un service Knative qui inclut l'annotation **serving.knative.openshift.io/disableRoute=true**:

IMPORTANT

L'annotation **serving.knative.openshift.io/disableRoute=true** indique à OpenShift Serverless de ne pas créer automatiquement une route pour vous. Cependant, le service affiche toujours une URL et atteint un statut de **Ready**. Cette URL ne fonctionne pas en externe jusqu'à ce que vous créiez votre propre route avec le même nom d'hôte que le nom d'hôte dans l'URL.

   a. Créer une ressource Knative **Service**:

Exemple de ressource

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
```

```
    annotations:
      serving.knative.openshift.io/disableRoute: "true"
  spec:
    template:
      spec:
        containers:
        - image: <image>
  ...
```

b. Appliquer la ressource **Service**:

```
$ oc apply -f <filename>
```

c. Facultatif. Créez un service Knative à l'aide de la commande **kn service create**:

Exemple de commande kn

```
$ kn service create <service_name> \
  --image=gcr.io/knative-samples/helloworld-go \
  --annotation serving.knative.openshift.io/disableRoute=true
```

2. Vérifiez qu'aucune route OpenShift Container Platform n'a été créée pour le service :

Example command

```
$ $ oc get routes.route.openshift.io \
  -l serving.knative.openshift.io/ingressName=$KSERVICE_NAME \
  -l serving.knative.openshift.io/ingressNamespace=$KSERVICE_NAMESPACE \
  -n knative-serving-ingress
```

Vous obtiendrez le résultat suivant :

```
No resources found in knative-serving-ingress namespace.
```

3. Créer une ressource **Route** dans l'espace de noms **knative-serving-ingress**:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 600s (1)
  name: <route_name> (2)
  namespace: knative-serving-ingress (3)
spec:
  host: <service_host> (4)
  port:
    targetPort: http2
  to:
    kind: Service
    name: kourier
    weight: 100
  tls:
    insecureEdgeTerminationPolicy: Allow
```

```
    termination: edge 5
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE----
  wildcardPolicy: None
```

1. La valeur du délai d'attente pour la route OpenShift Container Platform. Vous devez définir la même valeur que le paramètre **max-revision-timeout-seconds** (**600s** par défaut).
2. Le nom de la route OpenShift Container Platform.
3. L'espace de noms pour la route OpenShift Container Platform. Il doit s'agir de **knative-serving-ingress**.
4. Le nom d'hôte pour l'accès externe. Vous pouvez le définir sur **<service_name>-<service_namespace>.<domain>**.
5. Les certificats que vous souhaitez utiliser. Actuellement, seule la terminaison **edge** est prise en charge.

4. Appliquer la ressource **Route**:

```
$ oc apply -f <filename>
```

4.5.4. Redirection HTTPS globale

La redirection HTTPS permet de rediriger les requêtes HTTP entrantes. Ces requêtes HTTP redirigées sont cryptées. Vous pouvez activer la redirection HTTPS pour tous les services du cluster en configurant la spécification **httpProtocol** pour la ressource personnalisée (CR) **KnativeServing**.

4.5.4.1. Paramètres globaux de la redirection HTTPS

Exemple KnativeServing CR qui active la redirection HTTPS

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    network:
      httpProtocol: "redirected"
...
```

4.5.5. Schéma d'URL pour les routes externes

Le schéma d'URL des itinéraires externes est par défaut HTTPS pour une meilleure sécurité. Ce schéma est déterminé par la clé **default-external-scheme** dans la spécification des ressources personnalisées (CR) **KnativeServing**.

4.5.5.1. Définition du schéma d'URL pour les itinéraires externes

Spécification par défaut

```
...
spec:
  config:
    network:
      default-external-scheme: "https"
...
```

Vous pouvez remplacer la spécification par défaut d'utiliser HTTP en modifiant la clé **default-external-scheme**:

Spécification d'annulation HTTP

```
...
spec:
  config:
    network:
      default-external-scheme: "http"
...
```

4.5.6. Redirection HTTPS par service

Vous pouvez activer ou désactiver la redirection HTTPS pour un service en configurant l'annotation **networking.knative.dev/http-option**.

4.5.6.1. Redirection de HTTPS pour un service

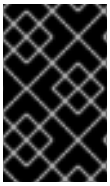
L'exemple suivant montre comment utiliser cette annotation dans un objet YAML Knative **Service**:

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example
  namespace: default
  annotations:
    networking.knative.dev/http-option: "redirected"
spec:
  ...
```

4.5.7. Disponibilité locale du cluster

Par défaut, les services Knative sont publiés sur une adresse IP publique. Le fait d'être publié sur une adresse IP publique signifie que les services Knative sont des applications publiques et qu'ils ont une URL accessible au public.

Les URL accessibles au public sont accessibles depuis l'extérieur du cluster. Cependant, les développeurs peuvent avoir besoin de construire des services back-end qui ne sont accessibles que depuis l'intérieur du cluster, connus sous le nom de *private services*. Les développeurs peuvent étiqueter des services individuels dans le cluster avec l'étiquette **networking.knative.dev/visibility=cluster-local** pour les rendre privés.



IMPORTANT

Pour OpenShift Serverless 1.15.0 et les versions plus récentes, le label **serving.knative.dev/visibility** n'est plus disponible. Vous devez mettre à jour les services existants pour utiliser le label **networking.knative.dev/visibility** à la place.

4.5.7.1. Paramétrage de la disponibilité du cluster sur le cluster local

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez créé un service Knative.

Procédure

- Définissez la visibilité de votre service en ajoutant le label **networking.knative.dev/visibility=cluster-local**:

```
oc label ksvc <service_name> networking.knative.dev/visibility=cluster-local
```

Vérification

- Vérifiez que l'URL de votre service est maintenant au format **http://<service_name>.<namespace>.svc.cluster.local**, en entrant la commande suivante et en examinant la sortie :

```
$ oc get ksvc
```

Exemple de sortie

```
NAME    URL                                        LATESTCREATED   LATESTREADY   READY   REASON
hello   http://hello.default.svc.cluster.local     hello-tx2g7     hello-tx2g7   True
```

4.5.7.2. Activation de l'authentification TLS pour les services locaux du cluster

Pour les services locaux du cluster, la passerelle locale Kourier **kourier-internal** est utilisée. Si vous souhaitez utiliser le trafic TLS avec la passerelle locale Kourier, vous devez configurer vos propres certificats de serveur dans la passerelle locale.

Conditions préalables

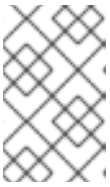
- Vous avez installé OpenShift Serverless Operator et Knative Serving.
- Vous avez des droits d'administrateur.

- Vous avez installé le CLI OpenShift (**oc**).

Procédure

1. Déployer les certificats de serveur dans l'espace de noms **knative-serving-ingress**:

```
$ export san="knative"
```



NOTE

La validation du nom alternatif du sujet (SAN) est nécessaire pour que ces certificats puissent transmettre la demande à **<app_name>.<namespace>.svc.cluster.local**.

2. Générer une clé racine et un certificat :

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \
    -subj '/O=Example/CN=Example' \
    -keyout ca.key \
    -out ca.crt
```

3. Générer une clé de serveur qui utilise la validation SAN :

```
$ openssl req -out tls.csr -newkey rsa:2048 -nodes -keyout tls.key \
  -subj "/CN=Example/O=Example" \
  -addext "subjectAltName = DNS:$san"
```

4. Créer des certificats de serveur :

```
$ openssl x509 -req -extfile <(printf "subjectAltName=DNS:$san") \
  -days 365 -in tls.csr \
  -CA ca.crt -CAkey ca.key -CAcreateserial -out tls.crt
```

5. Configurez un secret pour la passerelle locale de Kourier :

   a. Déployer un secret dans l'espace de noms **knative-serving-ingress** à partir des certificats créés lors des étapes précédentes :

```
$ oc create -n knative-serving-ingress secret tls server-certs \
    --key=tls.key \
    --cert=tls.crt --dry-run=client -o yaml | oc apply -f -
```

   b. Mettre à jour la spécification de la ressource personnalisée (CR) **KnativeServing** pour utiliser le secret créé par la passerelle Kourier :

Exemple KnativeServing CR

```
...
spec:
  config:
    kourier:
      cluster-cert-secret: server-certs
...
```

Le contrôleur Kourier établit le certificat sans redémarrer le service, de sorte qu'il n'est pas nécessaire de redémarrer le pod.

Vous pouvez accéder au service interne de Kourier avec TLS via le port **443** en montant et en utilisant le site **ca.crt** à partir du client.

4.5.8. Type de service de la passerelle Kourier

La passerelle Kourier est exposée par défaut en tant que type de service **ClusterIP**. Ce type de service est déterminé par la spécification d'entrée **service-type** dans la ressource personnalisée (CR) **KnativeServing**.

Spécification par défaut

```
...
spec:
  ingress:
    kourier:
      service-type: ClusterIP
...
```

4.5.8.1. Définition du type de service de la passerelle Kourier

Vous pouvez remplacer le type de service par défaut par un type de service d'équilibreur de charge en modifiant la spécification **service-type**:

Spécification de l'override LoadBalancer

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

4.5.9. Utilisation de HTTP2 et gRPC

OpenShift Serverless ne prend en charge que les itinéraires non sécurisés ou terminés par des bords. Les routes non sécurisées ou terminées par un bord ne supportent pas HTTP2 sur OpenShift Container Platform. Ces itinéraires ne prennent pas non plus en charge gRPC, car gRPC est transporté par HTTP2. Si vous utilisez ces protocoles dans votre application, vous devez appeler l'application en utilisant directement la passerelle d'entrée. Pour ce faire, vous devez trouver l'adresse publique de la passerelle d'entrée et l'hôte spécifique de l'application.

4.5.9.1. Interagir avec une application sans serveur en utilisant HTTP2 et gRPC

IMPORTANT

Cette méthode s'applique à OpenShift Container Platform 4.10 et aux versions ultérieures. Pour les versions antérieures, voir la section suivante.

Conditions préalables

- Installez OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Installez le CLI OpenShift (**oc**).
- Créer un service Knative.
- Mettez à jour OpenShift Container Platform 4.10 ou une version ultérieure.
- Activer HTTP/2 sur le contrôleur OpenShift Ingress.

Procédure

1. Ajoutez l'annotation **serverless.openshift.io/default-enable-http2=true** à la ressource personnalisée **KnativeServing**:

```
oc annotate knativeserving <your_knative_CR> -n knative-serving
serverless.openshift.io/default-enable-http2=true
```

2. Après l'ajout de l'annotation, vous pouvez vérifier que la valeur **appProtocol** du service Kourier est **h2c**:

```
$ oc get svc -n knative-serving-ingress kourier -o jsonpath="{.spec.ports[0].appProtocol}"
```

Exemple de sortie

```
h2c
```

3. Vous pouvez désormais utiliser le cadre gRPC sur le protocole HTTP/2 pour le trafic externe, par exemple :

```
import "google.golang.org/grpc"

grpc.Dial(
  YOUR_URL, (1)
  grpc.WithTransportCredentials(insecure.NewCredentials())), (2)
)
```

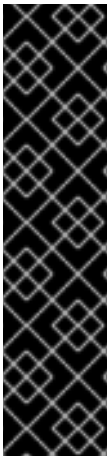
(1) Votre URL **ksvc**.

(2) Votre certificat.

Ressources supplémentaires

- Activation de la connectivité HTTP/2 Ingress

4.5.9.2. Interagir avec une application sans serveur en utilisant HTTP2 et gRPC dans OpenShift Container Platform 4.9 et plus



IMPORTANT

Cette méthode doit exposer la passerelle Kourier en utilisant le type de service **LoadBalancer**. Vous pouvez configurer cela en ajoutant le fichier YAML suivant à votre définition de ressource personnalisée (CRD) **KnativeServing**:

```
...
spec:
  ingress:
    kourier:
      service-type: LoadBalancer
...
```

Conditions préalables

- Installez OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Installez le CLI OpenShift (**oc**).
- Créer un service Knative.

Procédure

1. Trouvez l'hôte de l'application. Voir les instructions dans *Verifying your serverless application deployment*.
2. Recherchez l'adresse publique de la passerelle d'entrée :

```
$ oc -n knative-serving-ingress get svc kourier
```

Exemple de sortie

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)
AGE
kourier   LoadBalancer   172.30.51.103   a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com   80:31380/TCP,443:31390/TCP   67m
```

L'adresse publique apparaît dans le champ **EXTERNAL-IP** et, dans ce cas, il s'agit de **a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com**.

3. Définissez manuellement l'en-tête host de votre requête HTTP sur l'hôte de l'application, mais dirigez la requête elle-même vers l'adresse publique de la passerelle d'entrée.

```
$ curl -H "Host: hello-default.example.com" a83e86291bcdd11e993af02b7a65e514-
33544245.us-east-1.elb.amazonaws.com
```

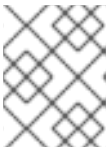
Exemple de sortie

```
Hello Serverless!
```

Vous pouvez également effectuer une requête gRPC directe auprès de la passerelle d'entrée :

```
import "google.golang.org/grpc"

grpc.Dial(
    "a83e86291bcdd11e993af02b7a65e514-33544245.us-east-1.elb.amazonaws.com:80",
    grpc.WithAuthority("hello-default.example.com:80"),
    grpc.WithInsecure(),
)
```



NOTE

Veillez à ajouter le port correspondant, 80 par défaut, aux deux hôtes, comme indiqué dans l'exemple précédent.

4.6. CONFIGURER L'ACCÈS AUX SERVICES KNATIVE

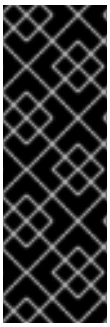
4.6.1. Configuration de l'authentification par jeton Web JSON pour les services Knative

OpenShift Serverless ne dispose pas actuellement de fonctionnalités d'autorisation définies par l'utilisateur. Pour ajouter une autorisation définie par l'utilisateur à votre déploiement, vous devez intégrer OpenShift Serverless à Red Hat OpenShift Service Mesh, puis configurer l'authentification JSON Web Token (JWT) et l'injection sidecar pour les services Knative.

4.6.2. Utilisation de l'authentification par jeton Web JSON avec Service Mesh 2.x

Vous pouvez utiliser l'authentification par jeton Web JSON (JWT) avec les services Knative en utilisant Service Mesh 2.x et OpenShift Serverless. Pour ce faire, vous devez créer des demandes et des politiques d'authentification dans l'espace de noms de l'application qui est membre de l'objet **ServiceMeshMemberRoll**. Vous devez également activer l'injection de sidecar pour le service.

4.6.2.1. Configuration de l'authentification par jeton Web JSON pour Service Mesh 2.x et OpenShift Serverless



IMPORTANT

L'ajout d'une injection de sidecar aux pods dans les espaces de noms du système, tels que **knative-serving** et **knative-serving-ingress**, n'est pas pris en charge lorsque Kourier est activé.

Si vous avez besoin de l'injection de sidecar pour les pods dans ces espaces de noms, consultez la documentation OpenShift Serverless sur *Integrating Service Mesh with OpenShift Serverless natively*.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator, Knative Serving et Red Hat OpenShift Service Mesh sur votre cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Ajoutez l'annotation **sidecar.istio.io/inject="true"** à votre service :

   Exemple de service

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

   1 Ajouter l'annotation **sidecar.istio.io/inject="true"**.

   2 Vous devez définir l'annotation **sidecar.istio.io/rewriteAppHTTPProbers: "true"** dans votre service Knative, car les versions 1.14.0 et supérieures d'OpenShift Serverless utilisent par défaut une sonde HTTP comme sonde de préparation pour les services Knative.

2. Appliquer la ressource **Service**:

```
$ oc apply -f <filename>
```

3. Créer une ressource **RequestAuthentication** dans chaque espace de noms d'application sans serveur qui est un membre de l'objet **ServiceMeshMemberRoll**:

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: jwt-example
  namespace: <namespace>
spec:
  jwtRules:
  - issuer: testing@secure.istio.io
    jwksUri: https://raw.githubusercontent.com/istio/istio/release-
1.8/security/tools/jwt/samples/jwks.json
```

4. Appliquer la ressource **RequestAuthentication**:

```
$ oc apply -f <filename>
```

5. Autoriser l'accès à la ressource **RequestAuthenticaton** à partir des pods système pour chaque espace de noms d'application sans serveur qui est un membre de l'objet **ServiceMeshMemberRoll**, en créant la ressource **AuthorizationPolicy** suivante :

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
```

```
  name: allowlist-by-paths
  namespace: <namespace>
spec:
  action: ALLOW
  rules:
  - to:
    - operation:
        paths:
        - /metrics 1
        - /healthz 2
```

1. Le chemin d'accès à votre application pour collecter les métriques par pod système.
2. Le chemin d'accès à votre application pour sonder le pod du système.

6. Appliquer la ressource **AuthorizationPolicy**:

```
$ oc apply -f <filename>
```

7. Pour chaque espace de noms d'application sans serveur qui est un membre de l'objet **ServiceMeshMemberRoll**, créez la ressource **AuthorizationPolicy** suivante :

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: require-jwt
  namespace: <namespace>
spec:
  action: ALLOW
  rules:
  - from:
    - source:
        requestPrincipals: ["testing@secure.istio.io/testing@secure.istio.io"]
```

8. Appliquer la ressource **AuthorizationPolicy**:

```
$ oc apply -f <filename>
```

Vérification

1. Si vous essayez d'utiliser une requête **curl** pour obtenir l'URL du service Knative, elle est refusée :

Example command

```
$ curl http://hello-example-1-default.apps.mycluster.example.com/
```

Exemple de sortie

```
RBAC: access denied
```

2. Vérifier la demande avec un JWT valide.

a. Obtenir le jeton JWT valide :

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.8/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - | base64 --decode -
```

b. Accédez au service en utilisant le jeton valide dans l'en-tête de la requête **curl**:

```
$ curl -H "Authorization: Bearer $TOKEN"  http://hello-example-1-default.apps.example.com
```

La demande est désormais autorisée :

Exemple de sortie

```
Hello OpenShift!
```

4.6.3. Utilisation de l'authentification par jeton Web JSON avec Service Mesh 1.x

Vous pouvez utiliser l'authentification par jeton Web JSON (JWT) avec les services Knative en utilisant Service Mesh 1.x et OpenShift Serverless. Pour ce faire, vous devez créer une politique dans l'espace de noms de l'application qui est un membre de l'objet **ServiceMeshMemberRoll**. Vous devez également activer l'injection de sidecar pour le service.

4.6.3.1. Configuration de l'authentification par jeton Web JSON pour Service Mesh 1.x et OpenShift Serverless

IMPORTANT

L'ajout d'une injection de sidecar aux pods dans les espaces de noms du système, tels que **knative-serving** et **knative-serving-ingress**, n'est pas pris en charge lorsque Kourier est activé.

Si vous avez besoin de l'injection de sidecar pour les pods dans ces espaces de noms, consultez la documentation OpenShift Serverless sur *Integrating Service Mesh with OpenShift Serverless natively*.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator, Knative Serving et Red Hat OpenShift Service Mesh sur votre cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Ajoutez l'annotation **sidecar.istio.io/inject="true"** à votre service :

Exemple de service

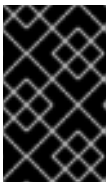
```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 1
        sidecar.istio.io/rewriteAppHTTPProbers: "true" 2
...
```

1. Ajouter l'annotation **sidecar.istio.io/inject="true"**.
2. Vous devez définir l'annotation **sidecar.istio.io/rewriteAppHTTPProbers: "true"** dans votre service Knative, car les versions 1.14.0 et supérieures d'OpenShift Serverless utilisent par défaut une sonde HTTP comme sonde de préparation pour les services Knative.

2. Appliquer la ressource **Service**:

```
$ oc apply -f <filename>
```

3. Créer une politique dans l'espace de noms d'une application sans serveur, qui est un membre de l'objet **ServiceMeshMemberRoll**, qui n'autorise que les demandes avec des jetons Web JSON valides (JWT) :



IMPORTANT

Les chemins **/metrics** et **/healthz** doivent être inclus dans **excludedPaths** car ils sont accessibles à partir des pods du système dans l'espace de noms **knative-serving**.

```
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: default
  namespace: <namespace>
spec:
  origins:
  - jwt:
      issuer: testing@secure.istio.io
      jwksUri: "https://raw.githubusercontent.com/istio/istio/release-
1.6/security/tools/jwt/samples/jwks.json"
      triggerRules:
      - excludedPaths:
        - prefix: /metrics 1
        - prefix: /healthz 2
  principalBinding: USE_ORIGIN
```

1. Le chemin d'accès à votre application pour collecter les métriques par pod système.
2. Le chemin d'accès à votre application pour sonder le pod du système.

4. Appliquer la ressource **Policy**:

```
$ oc apply -f <filename>
```

Vérification

1. Si vous essayez d'utiliser une requête **curl** pour obtenir l'URL du service Knative, elle est refusée :

```
$ curl http://hello-example-default.apps.mycluster.example.com/
```

Exemple de sortie

```
Origin authentication failed.
```

2. Vérifier la demande avec un JWT valide.

   a. Obtenir le jeton JWT valide :

```
$ TOKEN=$(curl https://raw.githubusercontent.com/istio/istio/release-1.6/security/tools/jwt/samples/demo.jwt -s) && echo "$TOKEN" | cut -d '.' -f2 - | base64 --decode -
```

   b. Accédez au service en utilisant le jeton valide dans l'en-tête de la requête **curl**:

```
$ curl http://hello-example-default.apps.mycluster.example.com/ -H "Authorization: Bearer $TOKEN"
```

   La demande est désormais autorisée :

   Exemple de sortie

```
Hello OpenShift!
```

4.7. CONFIGURER DES DOMAINES PERSONNALISÉS POUR LES SERVICES KNATIVE

4.7.1. Configurer un domaine personnalisé pour un service Knative

Les services Knative se voient automatiquement attribuer un nom de domaine par défaut basé sur la configuration de votre cluster. Par exemple, **<service_name>-<namespace>.example.com**. Vous pouvez personnaliser le domaine de votre service Knative en associant un nom de domaine personnalisé que vous possédez à un service Knative.

Pour ce faire, vous pouvez créer une ressource **DomainMapping** pour le service. Vous pouvez également créer plusieurs ressources **DomainMapping** pour associer plusieurs domaines et sous-domaines à un seul service.

4.7.2. Mappage de domaine personnalisé

Vous pouvez personnaliser le domaine de votre service Knative en mappant un nom de domaine

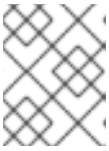
personnalisé que vous possédez à un service Knative. Pour mapper un nom de domaine personnalisé à une ressource personnalisée (CR), vous devez créer une CR **DomainMapping** qui est mappée à une CR cible adressable, telle qu'un service Knative ou une route Knative.

4.7.2.1. Créer un mappage de domaine personnalisé

Vous pouvez personnaliser le domaine de votre service Knative en mappant un nom de domaine personnalisé que vous possédez à un service Knative. Pour mapper un nom de domaine personnalisé à une ressource personnalisée (CR), vous devez créer une CR **DomainMapping** qui est mappée à une CR cible adressable, telle qu'un service Knative ou une route Knative.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez créé un service Knative et contrôlé un domaine personnalisé que vous souhaitez associer à ce service.



NOTE

Votre domaine personnalisé doit pointer vers l'adresse IP du cluster OpenShift Container Platform.

Procédure

1. Créez un fichier YAML contenant le CR **DomainMapping** dans le même espace de noms que le CR cible que vous souhaitez mapper :

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. Le nom de domaine personnalisé que vous souhaitez associer au CR cible.
2. L'espace de noms du CR **DomainMapping** et du CR cible.
3. Le nom du CR cible à mettre en correspondance avec le domaine personnalisé.
4. Le type de CR mappé au domaine personnalisé.

Exemple de mappage de domaine de service

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example-domain
 namespace: default
spec:
 ref:
   name: example-service
   kind: Service
   apiVersion: serving.knative.dev/v1
```

Exemple de mappage de domaines de routes

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: example-domain
 namespace: default
spec:
 ref:
   name: example-route
   kind: Route
   apiVersion: serving.knative.dev/v1
```

2. Appliquer le CR **DomainMapping** sous la forme d'un fichier YAML :

```
$ oc apply -f <filename>
```

4.7.3. Domaines personnalisés pour les services Knative à l'aide du CLI Knative

Vous pouvez personnaliser le domaine de votre service Knative en faisant correspondre un nom de domaine personnalisé que vous possédez à un service Knative. Vous pouvez utiliser la CLI Knative (**kn**) pour créer une ressource personnalisée (CR) **DomainMapping** qui correspond à une CR cible adressable, telle qu'un service Knative ou une route Knative.

4.7.3.1. Créer un mappage de domaine personnalisé en utilisant le CLI Knative

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Vous avez créé un service ou une route Knative, et vous contrôlez un domaine personnalisé que vous souhaitez associer à ce CR.

NOTE

Votre domaine personnalisé doit pointer vers le DNS du cluster OpenShift Container Platform.

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Associe un domaine à un CR dans l'espace de noms actuel :

```
kn domain create <domain_mapping_name> --ref <target_name>
```

Example command

```
$ kn domain create example-domain-map --ref example-service
```

L'indicateur **--ref** spécifie un CR cible adressable pour le mappage de domaine.

Si aucun préfixe n'est fourni lors de l'utilisation de l'indicateur **--ref**, il est supposé que la cible est un service Knative dans l'espace de noms actuel.

- Associer un domaine à un service Knative dans un espace de noms spécifié :

```
kn domain create <domain_mapping_name> --ref <ksvc:service_name:service_namespace>
```

Example command

```
$ kn domain create example-domain-map --ref ksvc:example-service:example-namespace
```

- Associer un domaine à une route Knative :

```
$ kn domain create <domain_mapping_name> --ref <kroute:route_name>
```

Example command

```
$ kn domain create example-domain-map --ref kroute:example-route
```

4.7.4. Cartographie des domaines selon la perspective du développeur

Vous pouvez personnaliser le domaine de votre service Knative en mappant un nom de domaine personnalisé que vous possédez à un service Knative. Vous pouvez utiliser la perspective **Developer** de la console web OpenShift Container Platform pour mapper une ressource personnalisée (CR) **DomainMapping** à un service Knative.

4.7.4.1. Mapper un domaine personnalisé à un service en utilisant la perspective du développeur

Conditions préalables

- Vous vous êtes connecté à la console web.
- Vous êtes dans la perspective **Developer**.
- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster. Cette opération doit être réalisée par un administrateur de cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous avez créé un service Knative et contrôlé un domaine personnalisé que vous souhaitez associer à ce service.

NOTE

Votre domaine personnalisé doit pointer vers l'adresse IP du cluster OpenShift Container Platform.

Procédure

1. Naviguez jusqu'à la page **Topology**.
2. Cliquez avec le bouton droit de la souris sur le service que vous voulez mapper à un domaine, et sélectionnez l'option **Edit** qui contient le nom du service. Par exemple, si le service s'appelle **example-service**, sélectionnez l'option **Edit example-service**.
3. Dans la section **Advanced options**, cliquez sur **Show advanced Routing options**.
   a. Si le CR de mappage de domaine que vous souhaitez mapper au service existe déjà, vous pouvez le sélectionner dans la liste **Domain mapping**.
   b. Si vous voulez créer un nouveau CR de mappage de domaine, tapez le nom de domaine dans la case et sélectionnez l'option **Create**. Par exemple, si vous tapez **example.com**, l'option **Create** est **Create "example.com"**.
4. Cliquez sur **Save** pour enregistrer les modifications apportées à votre service.

Vérification

1. Naviguez jusqu'à la page **Topology**.
2. Cliquez sur le service que vous avez créé.
3. Dans l'onglet **Resources** de la fenêtre d'information sur le service, vous pouvez voir le domaine que vous avez mappé au service listé sous **Domain mappings**.

4.7.5. Mappage de domaines à l'aide de la perspective de l'administrateur

Si vous ne souhaitez pas passer à la perspective **Developer** dans la console web d'OpenShift Container Platform ou utiliser le CLI Knative (**kn**) ou les fichiers YAML, vous pouvez utiliser la perspective **Administator** de la console web d'OpenShift Container Platform.

4.7.5.1. Mapper un domaine personnalisé à un service en utilisant la perspective de l'administrateur

Les services Knative se voient automatiquement attribuer un nom de domaine par défaut basé sur la configuration de votre cluster. Par exemple, **<service_name>-<namespace>.example.com**. Vous pouvez personnaliser le domaine de votre service Knative en associant un nom de domaine personnalisé que vous possédez à un service Knative.

Pour ce faire, vous pouvez créer une ressource **DomainMapping** pour le service. Vous pouvez également créer plusieurs ressources **DomainMapping** pour associer plusieurs domaines et sous-domaines à un seul service.

Si vous disposez des droits d'administrateur de cluster, vous pouvez créer une ressource personnalisée (CR) **DomainMapping** en utilisant la perspective **Administrator** dans la console Web d'OpenShift Container Platform.

Conditions préalables

- Vous vous êtes connecté à la console web.
- Vous êtes dans la perspective **Administrator**.
- Vous avez installé OpenShift Serverless Operator.
- Vous avez installé Knative Serving.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez créé un service Knative et contrôlé un domaine personnalisé que vous souhaitez associer à ce service.

NOTE

Votre domaine personnalisé doit pointer vers l'adresse IP du cluster OpenShift Container Platform.

Procédure

1. Naviguez jusqu'à **CustomResourceDefinitions** et utilisez la boîte de recherche pour trouver la définition de ressource personnalisée (CRD) **DomainMapping**.
2. Cliquez sur le CRD **DomainMapping**, puis naviguez jusqu'à l'onglet **Instances**.
3. Cliquez sur **Create DomainMapping**.
4. Modifiez le YAML pour le CR **DomainMapping** afin qu'il inclue les informations suivantes pour votre instance :

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: <domain_name> 1
 namespace: <namespace> 2
spec:
 ref:
   name: <target_name> 3
   kind: <target_type> 4
   apiVersion: serving.knative.dev/v1
```

1. Le nom de domaine personnalisé que vous souhaitez associer au CR cible.
2. L'espace de noms du CR **DomainMapping** et du CR cible.
3. Le nom du CR cible à mettre en correspondance avec le domaine personnalisé.
4. Le type de CR mappé au domaine personnalisé.

Exemple de mappage d'un domaine vers un service Knative

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
 name: custom-ksvc-domain.example.com
 namespace: default
spec:
 ref:
   name: example-service
   kind: Service
   apiVersion: serving.knative.dev/v1
```

Vérification

- Accédez au domaine personnalisé en utilisant une requête **curl**. Par exemple, il est possible d'accéder au domaine personnalisé en utilisant une requête :

Example command

```
$ curl custom-ksvc-domain.example.com
```

Exemple de sortie

```
Hello OpenShift!
```

4.7.6. Sécurisation d'un service mappé à l'aide d'un certificat TLS

4.7.6.1. Sécuriser un service avec un domaine personnalisé en utilisant un certificat TLS

Après avoir configuré un domaine personnalisé pour un service Knative, vous pouvez utiliser un certificat TLS pour sécuriser le service mappé. Pour ce faire, vous devez créer un secret TLS Kubernetes, puis mettre à jour le CR **DomainMapping** pour utiliser le secret TLS que vous avez créé.

NOTE

Si vous utilisez **net-istio** pour Ingress et activez mTLS via SMCP en utilisant **security.dataPlane.mtls: true**, Service Mesh déploie **DestinationRules** pour l'hôte ***.local**, ce qui n'autorise pas **DomainMapping** pour OpenShift Serverless.

Pour contourner ce problème, activez mTLS en déployant **PeerAuthentication** au lieu d'utiliser **security.dataPlane.mtls: true**.

Conditions préalables

- Vous avez configuré un domaine personnalisé pour un service Knative et vous disposez d'un CR **DomainMapping** fonctionnel.
- Vous disposez d'un certificat TLS délivré par votre autorité de certification ou d'un certificat auto-signé.
- Vous avez obtenu les fichiers **cert** et **key** auprès du fournisseur de votre autorité de certification, ou un certificat auto-signé.

- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez un secret TLS Kubernetes :

```
$ oc create secret tls <tls_secret_name> --cert=<path_to_certificate_file> --key=
<path_to_key_file>
```

2. Ajoutez le label **networking.internal.knative.dev/certificate-uid: <id>`** au secret TLS de Kubernetes :

```
$ oc label secret <tls_secret_name> networking.internal.knative.dev/certificate-uid="<id>"
```

Si vous utilisez un fournisseur de secret tiers tel que cert-manager, vous pouvez configurer votre gestionnaire de secret pour étiqueter automatiquement le secret TLS de Kubernetes. Les utilisateurs de Cert-manager peuvent utiliser le modèle de secret proposé pour générer automatiquement des secrets avec l'étiquette correcte. Dans ce cas, le filtrage des secrets est effectué sur la base de la clé uniquement, mais cette valeur peut contenir des informations utiles telles que l'ID du certificat que le secret contient.

NOTE

L'opérateur cert-manager pour Red Hat OpenShift est une fonctionnalité de l'aperçu technologique. Pour plus d'informations, consultez la documentation **Installing the cert-manager Operator for Red Hat OpenShift**

3. Mettez à jour le CR **DomainMapping** pour utiliser le secret TLS que vous avez créé :

```
apiVersion: serving.knative.dev/v1alpha1
kind: DomainMapping
metadata:
  name: <domain_name>
  namespace: <namespace>
spec:
  ref:
    name: <service_name>
    kind: Service
    apiVersion: serving.knative.dev/v1
# TLS block specifies the secret to be used
  tls:
    secretName: <tls_secret_name>
```

Vérification

1. Vérifiez que l'état de la CR **DomainMapping** est **True** et que la colonne **URL** de la sortie indique le domaine mappé avec le schéma **https**:

```
oc get domainmapping <domain_name>
```

Exemple de sortie

```
NAME                URL                         READY   REASON
example.com         https://example.com         True
```

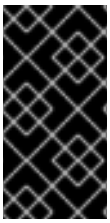
2. Facultatif : si le service est exposé publiquement, vérifiez qu'il est disponible en exécutant la commande suivante :

```
$ curl https://<domain_name>
```

Si le certificat est auto-signé, passez la vérification en ajoutant le drapeau **-k** à la commande **curl**.

4.7.6.2. Amélioration de l'utilisation de la mémoire du net-kourier par l'utilisation du filtrage secret

Par défaut, l'implémentation des informateurs pour la bibliothèque Kubernetes **client-go** récupère toutes les ressources d'un type particulier. Cela peut entraîner une surcharge substantielle lorsque de nombreuses ressources sont disponibles, ce qui peut faire échouer le contrôleur d'entrée Knative **net-kourier** sur les grands clusters en raison de fuites de mémoire. Cependant, un mécanisme de filtrage est disponible pour le contrôleur d'entrée Knative **net-kourier**, ce qui permet au contrôleur de ne récupérer que les secrets liés à Knative. Vous pouvez activer ce mécanisme en définissant une variable d'environnement pour la ressource personnalisée (CR) **KnativeServing**.



IMPORTANT

Si vous activez le filtrage des secrets, tous vos secrets doivent être étiquetés avec **networking.internal.knative.dev/certificate-uid: "<id>"**. Sinon, Knative Serving ne les détecte pas, ce qui entraîne des échecs. Vous devez étiqueter à la fois les nouveaux secrets et les secrets existants.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Un projet que vous avez créé ou pour lequel vous avez des rôles et des permissions afin de créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Installez OpenShift Serverless Operator et Knative Serving.
- Installez le CLI OpenShift (**oc**).

Procédure

- Définissez la variable **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID** à **true** pour **net-kourier-controller** dans le CR **KnativeServing**:

Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
 name: knative-serving
 namespace: knative-serving
spec:
```

```
deployments:
  - env:
    - container: controller
      envVars:
      - name: ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID
        value: 'true'
    name: net-kourier-controller
```

4.8. CONFIGURATION DE LA HAUTE DISPONIBILITÉ POUR LES SERVICES KNATIVE

4.8.1. Haute disponibilité des services Knative

La haute disponibilité (HA) est une fonctionnalité standard des API Kubernetes qui permet de garantir que les API restent opérationnelles en cas de perturbation. Dans un déploiement HA, si un contrôleur actif tombe en panne ou est supprimé, un autre contrôleur est immédiatement disponible. Ce contrôleur prend en charge le traitement des API qui étaient gérées par le contrôleur qui n'est plus disponible.

HA dans OpenShift Serverless est disponible via l'élection de leader, qui est activée par défaut après l'installation du plan de contrôle Knative Serving ou Eventing. Lors de l'utilisation d'un modèle HA d'élection de leader, les instances de contrôleurs sont déjà planifiées et en cours d'exécution dans le cluster avant qu'elles ne soient nécessaires. Ces instances de contrôleurs sont en concurrence pour l'utilisation d'une ressource partagée, connue sous le nom de verrou d'élection du leader. L'instance du contrôleur qui a accès à la ressource de verrouillage de l'élection du leader à un moment donné est appelée le leader.

4.8.2. Haute disponibilité des services Knative

La haute disponibilité (HA) est disponible par défaut pour les composants Knative Serving **activator**, **autoscaler**, **autoscaler-hpa**, **controller**, **webhook**, **kourier-control**, et **kourier-gateway**, qui sont configurés pour avoir deux répliques chacun par défaut. Vous pouvez changer le nombre de répliques pour ces composants en modifiant la valeur **spec.high-availability.replicas** dans la ressource personnalisée (CR) **KnativeServing**.

4.8.2.1. Configuration des répliques de haute disponibilité pour Knative Serving

Pour spécifier trois répliques minimum pour les ressources de déploiement éligibles, définissez la valeur du champ **spec.high-availability.replicas** dans la ressource personnalisée sur **3**.

Conditions préalables

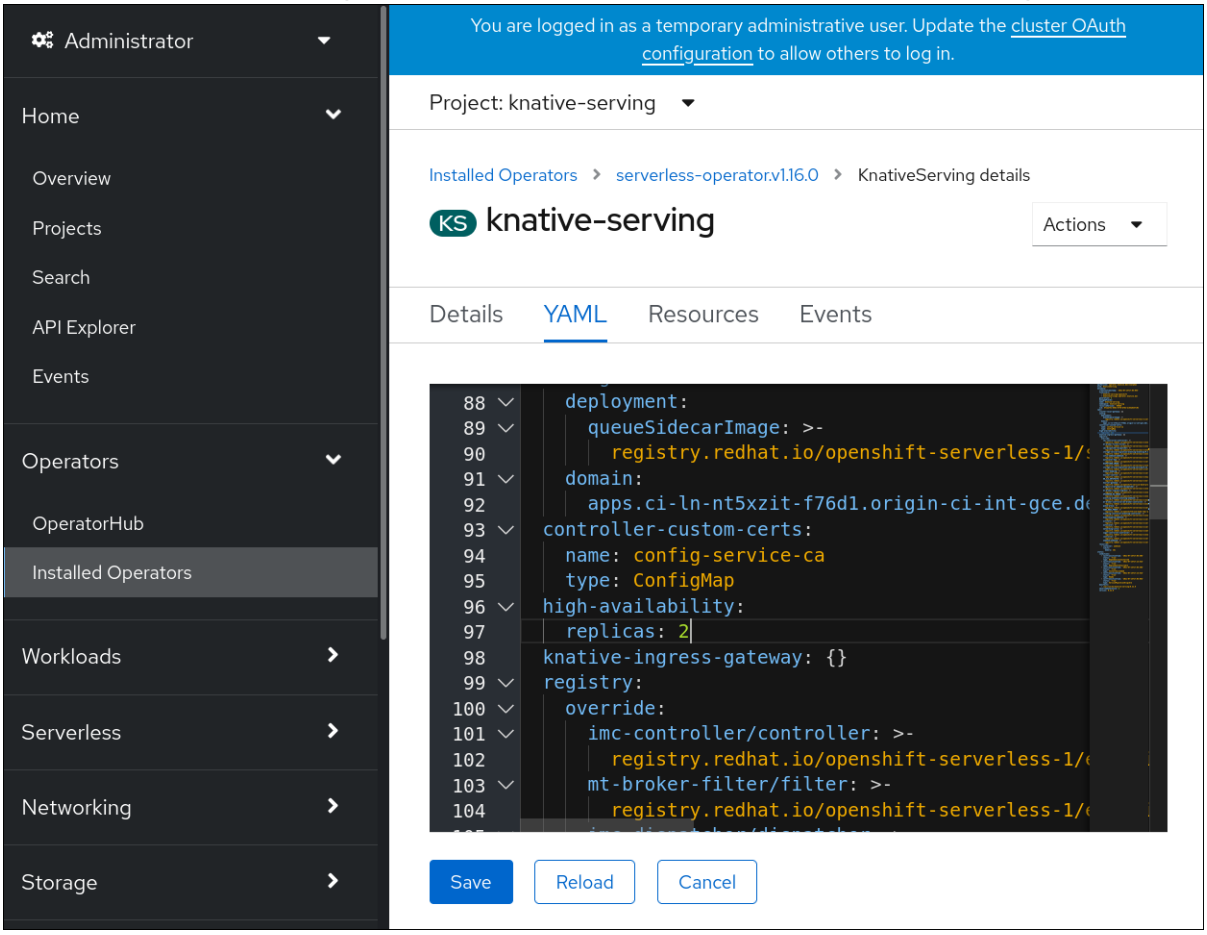
- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.

Procédure

1. Dans la perspective de la console web de OpenShift Container Platform **Administrator**, naviguez vers **OperatorHub → Installed Operators**.
2. Sélectionnez l'espace de noms **knative-serving**.

3. Cliquez sur **Knative Serving** dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator afin d'accéder à l'onglet **Knative Serving**.

4. Cliquez sur **knative-serving**, puis sur l'onglet **YAML** dans la page **knative-serving**.



5. Modifier le nombre de répliques dans le CR **KnativeServing**:

Exemple YAML

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  high-availability:
    replicas: 3
```

CHAPITRE 5. CONCOURS COMPLET

5.1. CONCOURS COMPLET D'ÉQUITATION

Knative Eventing sur OpenShift Container Platform permet aux développeurs d'utiliser une architecture pilotée par les événements avec des applications sans serveur. Une architecture pilotée par les événements est basée sur le concept de relations découplées entre les producteurs et les consommateurs d'événements.

Les producteurs d'événements créent des événements, et les événements *sinks*, ou consommateurs, reçoivent des événements. Knative Eventing utilise des requêtes HTTP POST standard pour envoyer et recevoir des événements entre les producteurs et les récepteurs d'événements. Ces événements sont conformes aux spécifications CloudEvents, qui permettent de créer, d'analyser, d'envoyer et de recevoir des événements dans n'importe quel langage de programmation.

Knative Eventing prend en charge les cas d'utilisation suivants :

Publier un événement sans créer de consommateur

Vous pouvez envoyer des événements à un courtier sous la forme d'un HTTP POST et utiliser la liaison pour découpler la configuration de destination de votre application qui produit les événements.

Consommer un événement sans créer d'éditeur

Vous pouvez utiliser un déclencheur pour consommer des événements à partir d'un courtier en fonction des attributs de l'événement. L'application reçoit les événements sous forme de HTTP POST.

Pour permettre la livraison à plusieurs types de puits, Knative Eventing définit les interfaces génériques suivantes qui peuvent être mises en œuvre par plusieurs ressources Kubernetes :

Ressources adressables

Capable de recevoir et d'accuser réception d'un événement transmis par HTTP à une adresse définie dans le champ **status.address.url** de l'événement. La ressource Kubernetes **Service** satisfait également à l'interface adressable.

Ressources appelables

Capable de recevoir un événement transmis par HTTP et de le transformer, en renvoyant **0** ou **1** nouveaux événements dans la charge utile de la réponse HTTP. Ces événements renvoyés peuvent être traités de la même manière que les événements provenant d'une source externe.

5.2. SOURCES DES ÉVÉNEMENTS

5.2.1. Sources des événements

Un Knative *event source* peut être n'importe quel objet Kubernetes qui génère ou importe des événements cloud, et relaie ces événements vers un autre point d'extrémité, connu sous le nom de *sink*. L'approvisionnement en événements est essentiel pour développer un système distribué qui réagit aux événements.

Vous pouvez créer et gérer des sources d'événements Knative en utilisant la perspective **Developer** dans la console web OpenShift Container Platform, le CLI Knative (**kn**), ou en appliquant des fichiers YAML.

Actuellement, OpenShift Serverless prend en charge les types de sources d'événements suivants :

Source du serveur API

Apporte les événements du serveur API Kubernetes dans Knative. La source du serveur API envoie un nouvel événement chaque fois qu'une ressource Kubernetes est créée, mise à jour ou supprimée.

Source de ping

Produit des événements avec une charge utile fixe selon un calendrier cron spécifié.

Source d'événements Kafka

Connecte un cluster Apache Kafka à un puits en tant que source d'événements.

Vous pouvez également créer une source d'événement personnalisée.

5.2.2. Source d'événement dans la perspective de l'administrateur

L'approvisionnement en événements est essentiel pour développer un système distribué qui réagit aux événements.

5.2.2.1. Création d'une source d'événements à l'aide de la perspective administrateur

Un Knative *event source* peut être n'importe quel objet Kubernetes qui génère ou importe des événements cloud, et relaie ces événements vers un autre point final, connu sous le nom de *sink*.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans la liste **Create**, sélectionnez **Event Source**. Vous serez dirigé vers la page **Event Sources**.
3. Sélectionnez le type de source d'événement que vous souhaitez créer.

5.2.3. Création d'une source de serveur API

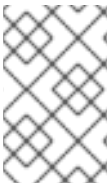
La source du serveur API est une source d'événements qui peut être utilisée pour connecter un récepteur d'événements, tel qu'un service Knative, au serveur API Kubernetes. La source du serveur API surveille les événements Kubernetes et les transmet au courtier Knative Eventing.

5.2.3.1. Création d'une source de serveur API à l'aide de la console web

Une fois Knative Eventing installé sur votre cluster, vous pouvez créer une source de serveur API à l'aide de la console Web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer une source d'événement.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).



PROCÉDURE

Si vous souhaitez réutiliser un compte de service existant, vous pouvez modifier votre ressource **ServiceAccount** existante pour y inclure les autorisations requises au lieu de créer une nouvelle ressource.

1. Créez un compte de service, un rôle et une liaison de rôle pour la source d'événement sous la forme d'un fichier YAML :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default (1)

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default (2)
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default (3)
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default (4)
```

1 2 3 4 Remplacez cet espace de noms par l'espace de noms que vous avez sélectionné pour l'installation de la source d'événements.

2. Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

3. Dans la perspective **Developer**, naviguez vers **Add → Event Source**. La page **Event Sources** s'affiche.

4. Facultatif : si vous avez plusieurs fournisseurs pour vos sources d'événements, sélectionnez le fournisseur requis dans la liste **Providers** pour filtrer les sources d'événements disponibles du fournisseur.

5. Sélectionnez **ApiServerSource** puis cliquez sur **Create Event Source**. La page **Create Event Source** s'affiche.

6. Configurez les paramètres de **ApiServerSource** en utilisant les boutons **Form view** ou **YAML view**:

NOTE

Vous pouvez passer de la vue **Form view** à la vue **YAML view**. Les données sont conservées lors du passage d'une vue à l'autre.

   a. Saisissez **v1** comme **APIVERSION** et **Event** comme **KIND**.

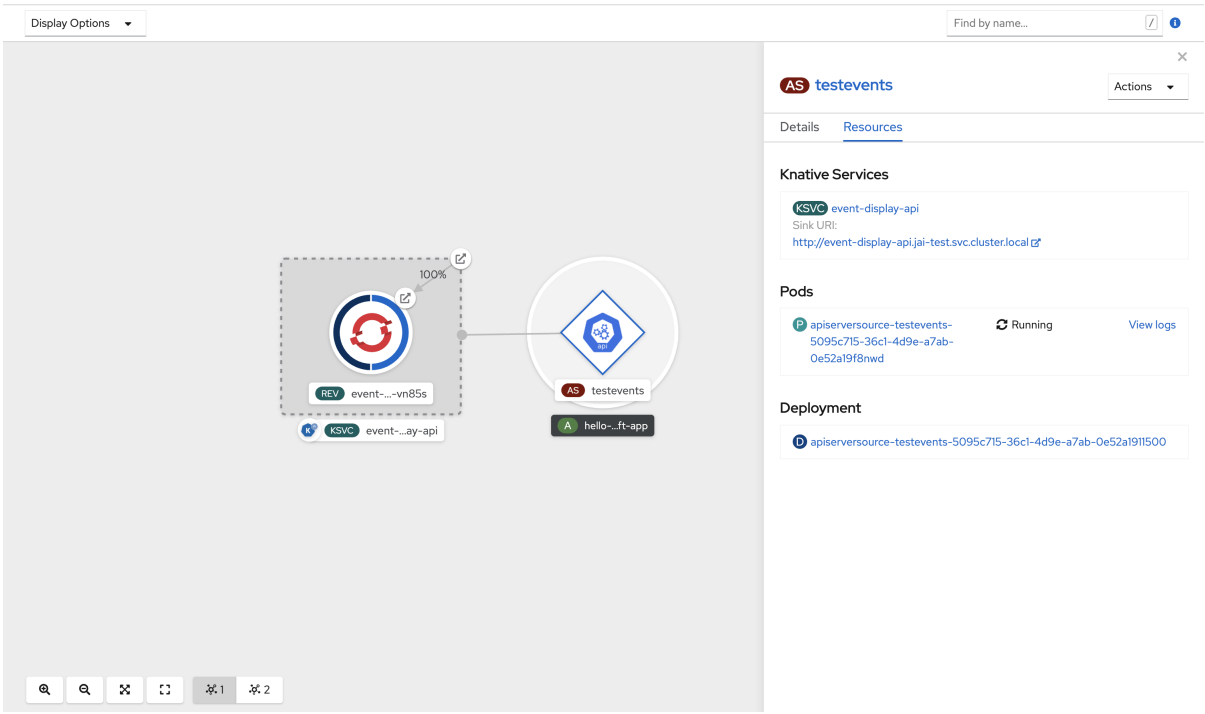
   b. Sélectionnez le site **Service Account Name** pour le compte de service que vous avez créé.

   c. Sélectionnez le site **Sink** pour la source de l'événement. Un **Sink** peut être soit un **Resource**, tel qu'un canal, un courtier ou un service, soit un **URI**.

7. Cliquez sur **Create**.

Vérification

- Après avoir créé la source du serveur API, vous la verrez connectée au service auquel elle est liée dans la vue **Topology**.

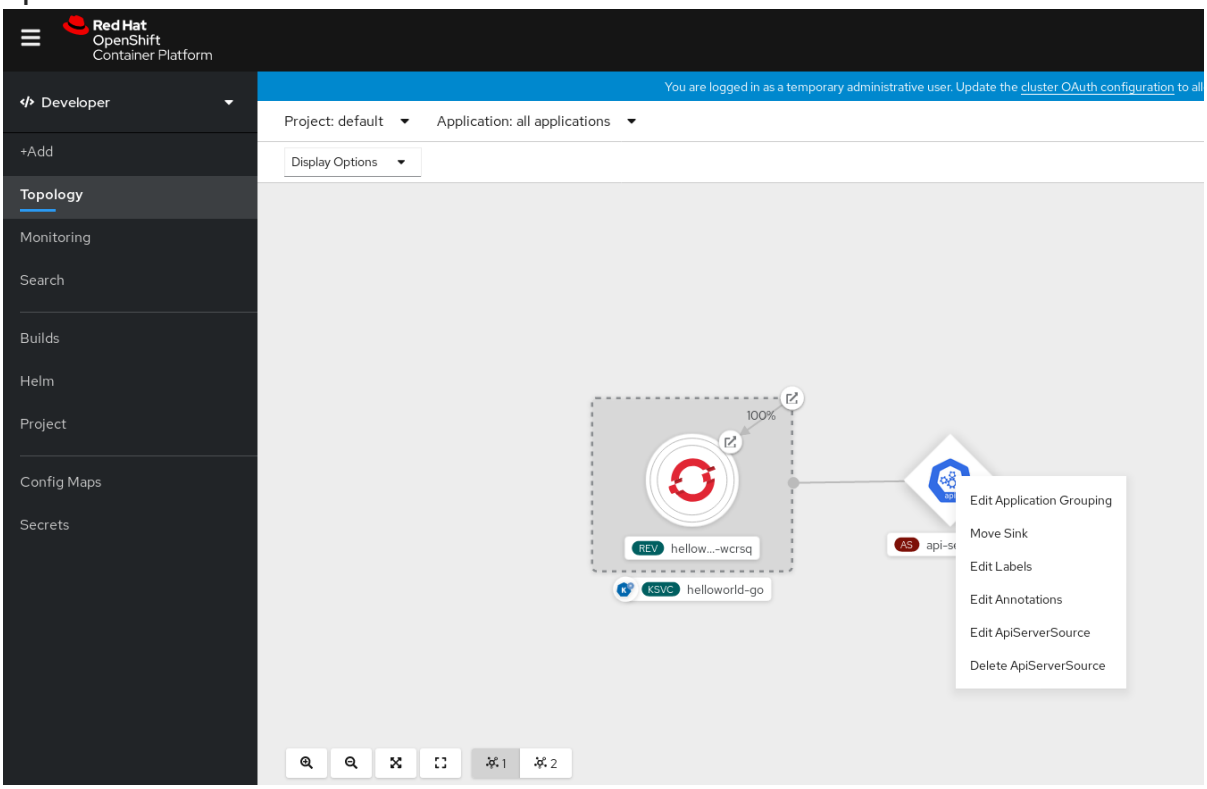


NOTE

Si un puits URI est utilisé, modifiez l'URI en faisant un clic droit sur **URI sink → Edit URI**.

Suppression de la source du serveur API

1. Naviguez jusqu'à la vue **Topology**.
2. Cliquez avec le bouton droit de la souris sur la source du serveur API et sélectionnez **Delete ApiServerSource**.



5.2.3.2. Création d'une source de serveur API à l'aide du CLI Knative

Vous pouvez utiliser la commande **kn source apiserver create** pour créer une source de serveur API à l'aide de l'interface de programmation **kn**. L'utilisation du CLI **kn** pour créer une source de serveur API offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous avez installé le CLI Knative (**kn**).

PROCÉDURE

Si vous souhaitez réutiliser un compte de service existant, vous pouvez modifier votre ressource **ServiceAccount** existante pour y inclure les autorisations requises au lieu de créer une nouvelle ressource.

1. Créez un compte de service, un rôle et une liaison de rôle pour la source d'événement sous la forme d'un fichier YAML :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default (1)

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default (2)
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default (3)
roleRef:
  apiGroup: rbac.authorization.k8s.io
```

```
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default 4
```

1 2 3 4 Remplacez cet espace de noms par l'espace de noms que vous avez sélectionné pour l'installation de la source d'événements.

2. Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

3. Créez une source de serveur API dotée d'un puits d'événements. Dans l'exemple suivant, le puits est un courtier :

```
$ kn source apiserver create <event_source_name> --sink broker:<broker_name> --
resource "event:v1" --service-account <service_account_name> --mode Resource
```

4. Pour vérifier que la source du serveur API est correctement configurée, créez un service Knative qui déverse les messages entrants dans son journal :

```
$ kn service create <service_name> --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

5. Si vous avez utilisé un courtier comme puits d'événements, créez un déclencheur pour filtrer les événements du courtier **default** vers le service :

```
kn trigger create <trigger_name> --sink ksvc:<service_name> $ kn trigger create
<trigger_name>
```

6. Créer des événements en lançant un pod dans l'espace de noms par défaut :

```
$ oc create deployment hello-node --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

7. Vérifiez que le contrôleur est correctement mappé en inspectant la sortie générée par la commande suivante :

```
$ kn source apiserver describe <nom_de_la_source>
```

Exemple de sortie

```
Name:               mysource
Namespace:          default
Annotations:        sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:                3m
ServiceAccountName:  events-sa
Mode:               Resource
Sink:
```

```
  Name:       default
  Namespace:  default
  Kind:       Broker (eventing.knative.dev/v1)
Resources:
  Kind:        event (v1)
  Controller:  false
Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 3m
  ++ Deployed               3m
  ++ SinkProvided           3m
  ++ SufficientPermissions  3m
  ++ EventTypesProvided     3m
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés à Knative en consultant les journaux de la fonction dumper de messages.

1. Obtenez les cosses :

```
$ oc get pods
```

2. Affichez les journaux des fonctions de l'expéditeur de messages pour les modules :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{hello-node}",
      "kind": "Pod",
      "name": "hello-node",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "hello-node.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
```

```
    "reason": "Started",
    ...
}
```

Suppression de la source du serveur API

1. Supprimer le déclencheur :

```
kn trigger delete <trigger_name>
```

2. Supprimer la source de l'événement :

```
$ kn source apiserver delete <nom_de_la_source>
```

3. Supprimez le compte de service, le rôle de cluster et le lien de cluster :

```
$ oc delete -f authentication.yaml
```

5.2.3.2.1. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

5.2.3.3. Création d'un serveur API source à l'aide de fichiers YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les sources d'événements de manière déclarative et reproductible. Pour créer une source de serveur API à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **ApiServerSource**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous avez créé le broker **default** dans le même espace de noms que celui défini dans le fichier YAML source du serveur API.
- Installez le CLI OpenShift (**oc**).

PROCÉDURE

Si vous souhaitez réutiliser un compte de service existant, vous pouvez modifier votre ressource **ServiceAccount** existante pour y inclure les autorisations requises au lieu de créer une nouvelle ressource.

1. Créez un compte de service, un rôle et une liaison de rôle pour la source d'événement sous la forme d'un fichier YAML :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default (1)

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default (2)
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default (3)
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default (4)
```

(1) (2) (3) (4) Remplacez cet espace de noms par l'espace de noms que vous avez sélectionné pour l'installation de la source d'événements.

2. Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

3. Créer une source de serveur API sous la forme d'un fichier YAML :

```
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
  name: testevents
spec:
  serviceAccountName: events-sa
  mode: Resource
  resources:
    - apiVersion: v1
      kind: Event
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1
      kind: Broker
      name: default
```

4. Appliquer le fichier YAML **ApiServerSource**:

```
$ oc apply -f <filename>
```

5. Pour vérifier que la source du serveur API est correctement configurée, créez un service Knative sous la forme d'un fichier YAML qui déverse les messages entrants dans son journal :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
  namespace: default
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

6. Appliquer le fichier YAML **Service**:

```
$ oc apply -f <filename>
```

7. Créez un objet **Trigger** sous la forme d'un fichier YAML qui filtre les événements du courtier **default** vers le service créé à l'étape précédente :

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: event-display-trigger
  namespace: default
spec:
  broker: default
```

```
  subscriber:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

8. Appliquer le fichier YAML **Trigger**:

```
$ oc apply -f <filename>
```

9. Créer des événements en lançant un pod dans l'espace de noms par défaut :

```
$ oc create deployment hello-node --image=quay.io/openshift-knative/knative-eventing-sources-event-display
```

10. Vérifiez que le contrôleur est correctement mappé, en entrant la commande suivante et en inspectant la sortie :

```
$ oc get apiserversource.sources.knative.dev testevents -o yaml
```

Exemple de sortie

```
apiVersion: sources.knative.dev/v1alpha1
kind: ApiServerSource
metadata:
  annotations:
  creationTimestamp: "2020-04-07T17:24:54Z"
  generation: 1
  name: testevents
  namespace: default
  resourceVersion: "62868"
  selfLink:
/apis/sources.knative.dev/v1alpha1/namespaces/default/apiserversources/testevents2
  uid: 1603d863-bb06-4d1c-b371-f580b4db99fa
spec:
  mode: Resource
  resources:
  - apiVersion: v1
    controller: false
    controllerSelector:
      apiVersion: ""
      kind: ""
      name: ""
      uid: ""
    kind: Event
    labelSelector: {}
  serviceAccountName: events-sa
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1
      kind: Broker
      name: default
```

Vérification

Pour vérifier que les événements Kubernetes ont été envoyés à Knative, vous pouvez consulter les journaux de la fonction dumper de messages.

1. Récupérez les pods en entrant la commande suivante :

```
$ oc get pods
```

2. Affichez les journaux de la fonction d'expéditeur de messages pour les modules en entrant la commande suivante :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{hello-node}",
      "kind": "Pod",
      "name": "hello-node",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "hello-node.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
    "reason": "Started",
    ...
  }
```

Suppression de la source du serveur API

1. Supprimer le déclencheur :

```
$ oc delete -f trigger.yaml
```

2. Supprimer la source de l'événement :

```
$ oc delete -f k8s-events.yaml
```

3. Supprimez le compte de service, le rôle de cluster et le lien de cluster :

```
$ oc delete -f authentication.yaml
```

5.2.4. Création d'une source de ping

Une source ping est une source d'événements qui peut être utilisée pour envoyer périodiquement des événements ping avec une charge utile constante à un consommateur d'événements. Une source ping peut être utilisée pour programmer l'envoi d'événements, à l'instar d'une minuterie.

5.2.4.1. Création d'une source de ping à l'aide de la console web

Une fois Knative Eventing installé sur votre cluster, vous pouvez créer une source de ping en utilisant la console web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer une source d'événement.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Pour vérifier que la source de ping fonctionne, créez un service Knative simple qui déverse les messages entrants dans les journaux du service.
   a. Dans la perspective **Developer**, naviguez vers **Add → YAML**.
   b. Copiez l'exemple YAML :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

   c. Cliquez sur **Create**.
2. Créez une source ping dans le même espace de noms que le service créé à l'étape précédente, ou tout autre puits vers lequel vous souhaitez envoyer des événements.
   a. Dans la perspective **Developer**, naviguez vers **Add → Event Source**. La page **Event Sources** s'affiche.
   b. Facultatif : si vous avez plusieurs fournisseurs pour vos sources d'événements, sélectionnez le fournisseur requis dans la liste **Providers** pour filtrer les sources d'événements disponibles du fournisseur.

c. Sélectionnez **Ping Source** puis cliquez sur **Create Event Source** La page **Create Event Source** s'affiche.

NOTE

Vous pouvez configurer les paramètres de **PingSource** en utilisant **Form view** ou **YAML view** et passer d'une vue à l'autre. Les données sont conservées lors du passage d'une vue à l'autre.

d. Enter a value for **Schedule**. In this example, the value is ***/2 * * * ***, which creates a PingSource that sends a message every two minutes.

e. Facultatif : vous pouvez saisir une valeur pour **Data**, qui est la charge utile du message.

f. Sélectionnez un **Sink**. Il peut s'agir d'un **Resource** ou d'un **URI**. Dans cet exemple, le service **event-display** créé à l'étape précédente est utilisé comme puits **Resource**.

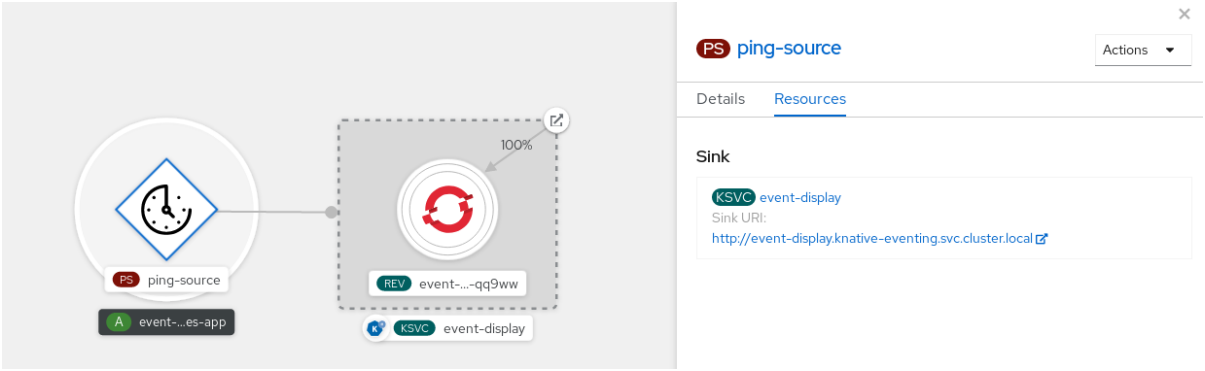
g. Cliquez sur **Create**.

Vérification

Vous pouvez vérifier que la source de ping a été créée et qu'elle est connectée au puits en consultant la page **Topology**.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.

2. Affichez la source et la destination du ping.



Suppression de la source de ping

1. Naviguez jusqu'à la vue **Topology**.

2. Cliquez avec le bouton droit de la souris sur la source du serveur API et sélectionnez **Delete Ping Source**.

5.2.4.2. Création d'une source de ping à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn source ping create** pour créer une source de ping en utilisant le CLI Knative (**kn**). L'utilisation de l'interface de programmation Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Facultatif : Si vous souhaitez utiliser les étapes de vérification pour cette procédure, installez l'OpenShift CLI (**oc**).

Procédure

1. Pour vérifier que la source de ping fonctionne, créez un service Knative simple qui déverse les messages entrants dans les journaux du service :

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

2. Pour chaque ensemble d'événements ping que vous souhaitez demander, créez une source ping dans le même espace de noms que le consommateur d'événements :

```
$ kn source ping create test-ping-source \
    --schedule "*/2 * * * *" \
    --data '{"message": "Hello world!"}' \
    --sink ksvc:event-display
```

3. Vérifiez que le contrôleur est correctement mappé en entrant la commande suivante et en inspectant la sortie :

```
$ kn source ping describe test-ping-source
```

Exemple de sortie

```
Name:         test-ping-source
Namespace:    default
Annotations:  sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:          15s
Schedule:     */2 * * * *
Data:         {"message": "Hello world!"}

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE               AGE REASON
  ++ Ready              8s
  ++ Deployed           8s
  ++ SinkProvided       15s
  ++ ValidSchedule      15s
  ++ EventTypeProvided  15s
  ++ ResourcesCorrect   15s
```

-

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés au puits d'événements Knative en examinant les journaux du pod de puits.

Par défaut, les services Knative terminent leurs pods si aucun trafic n'est reçu pendant une période de 60 secondes. L'exemple présenté dans ce guide crée une source ping qui envoie un message toutes les 2 minutes, chaque message doit donc être observé dans un pod nouvellement créé.

1. Surveillez la création de nouvelles gousses :

```
$ watch oc get pods
```

2. Annulez l'observation des pods en utilisant Ctrl C, puis regardez les journaux du pod créé :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
  time: 2020-04-07T16:16:00.000601161Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

Suppression de la source de ping

- Supprimer la source de ping :

```
$ kn delete pingsources.sources.knative.dev <nom_de_la_source_de_ping>
```

5.2.4.2.1. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
```

```
--subject "Job:batch/v1:app=heartbeat-cron" \
--sink http://event-display.svc.cluster.local \ 1
--ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

5.2.4.3. Création d'une source de ping à l'aide de YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les sources d'événements de manière déclarative et reproductible. Pour créer une source ping sans serveur à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **PingSource**, puis l'appliquer à l'aide de **oc apply**.

Exemple d'objet PingSource

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  name: test-ping-source
spec:
  schedule: "*/2 * * * *" 1
  data: '{"message": "Hello world!"}' 2
  sink: 3
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1 L'horaire de l'événement spécifié à l'aide d'une expression CRON.

2 Le corps du message de l'événement exprimé sous la forme d'une chaîne de données codée en JSON.

3 Il s'agit des détails du consommateur d'événements. Dans cet exemple, nous utilisons un service Knative nommé **event-display**.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Pour vérifier que la source de ping fonctionne, créez un service Knative simple qui déverse les messages entrants dans les journaux du service.
   a. Créer un fichier YAML de service :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

b. Créer le service :

```
$ oc apply -f <filename>
```

2. Pour chaque ensemble d'événements ping que vous souhaitez demander, créez une source ping dans le même espace de noms que le consommateur d'événements.

a. Créer un fichier YAML pour la source ping :

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  name: test-ping-source
spec:
  schedule: "*/2 * * * *"
  data: '{"message": "Hello world!"}'
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

b. Créer la source de ping :

```
$ oc apply -f <filename>
```

3. Vérifiez que le contrôleur est correctement mappé en entrant la commande suivante :

```
$ oc get pingsource.sources.knative.dev <nom_de_la_source_de_ping> -oyaml
```

Exemple de sortie

```
apiVersion: sources.knative.dev/v1
kind: PingSource
metadata:
  annotations:
    sources.knative.dev/creator: developer
    sources.knative.dev/lastModifier: developer
  creationTimestamp: "2020-04-07T16:11:14Z"
  generation: 1
  name: test-ping-source
  namespace: default
  resourceVersion: "55257"
  selfLink: /apis/sources.knative.dev/v1/namespaces/default/pingsources/test-ping-source
```

```
  uid: 3d80d50b-f8c7-4c1b-99f7-3ec00e0a8164
spec:
  data: '{ value: "hello" }'
  schedule: '*/2 * * * *'
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés au puits d'événements Knative en consultant les journaux du pod de puits.

Par défaut, les services Knative terminent leurs pods si aucun trafic n'est reçu pendant une période de 60 secondes. L'exemple présenté dans ce guide crée une PingSource qui envoie un message toutes les 2 minutes, donc chaque message doit être observé dans un pod nouvellement créé.

1. Surveillez la création de nouvelles gousses :

```
$ watch oc get pods
```

2. Annulez l'observation des pods en utilisant Ctrl C, puis regardez les journaux du pod créé :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 042ff529-240e-45ee-b40c-3a908129853e
  time: 2020-04-07T16:22:00.000791674Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

Suppression de la source de ping

- Supprimer la source de ping :

```
oc delete -f <filename>
```

Example command

```
$ oc delete -f ping-source.yaml
```

5.2.5. Source pour Apache Kafka

Vous pouvez créer une source Apache Kafka qui lit des événements à partir d'un cluster Apache Kafka et transmet ces événements à un puits. Vous pouvez créer une source Kafka en utilisant la console web d'OpenShift Container Platform, le CLI Knative (**kn**), ou en créant un objet **KafkaSource** directement en tant que fichier YAML et en utilisant le CLI OpenShift (**oc**) pour l'appliquer.

NOTE

Voir la documentation sur l'installation du courtier Knative pour Apache Kafka .

5.2.5.1. Création d'une source d'événement Apache Kafka à l'aide de la console web

Une fois que l'implémentation du courtier Knative pour Apache Kafka est installée sur votre cluster, vous pouvez créer une source Apache Kafka à l'aide de la console web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer une source Kafka.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster.
- Vous vous êtes connecté à la console web.
- Vous avez accès à un cluster Red Hat AMQ Streams (Kafka) qui produit les messages Kafka que vous souhaitez importer.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

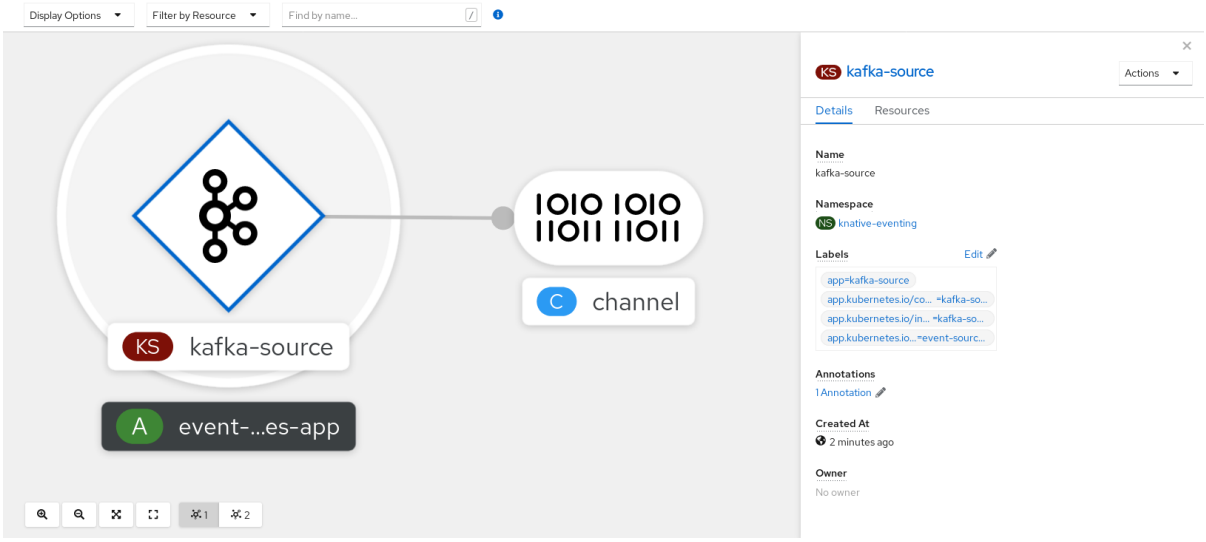
Procédure

1. Dans la perspective **Developer**, naviguez jusqu'à la page **Add** et sélectionnez **Event Source**.
2. Dans la page **Event Sources**, sélectionnez **Kafka Source** dans la section **Type**.
3. Configurez les paramètres de **Kafka Source**:
   a. Ajouter une liste de **Bootstrap Servers** séparée par des virgules.
   b. Ajouter une liste de **Topics** séparée par des virgules.
   c. Ajouter un **Consumer Group**.
   d. Sélectionnez le site **Service Account Name** pour le compte de service que vous avez créé.
   e. Sélectionnez le site **Sink** pour la source de l'événement. Un **Sink** peut être soit un **Resource**, tel qu'un canal, un courtier ou un service, soit un **URI**.
   f. Saisissez une adresse **Name** pour la source d'événements Kafka.
4. Cliquez sur **Create**.

Vérification

Vous pouvez vérifier que la source d'événements Kafka a été créée et qu'elle est connectée au puits en consultant la page **Topology**.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.
2. Afficher la source et le puits d'événements Kafka.



5.2.5.2. Créer une source d'événement Apache Kafka en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn source kafka create** pour créer une source Kafka en utilisant le CLI Knative (**kn**). L'utilisation du CLI Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et intuitive que la modification directe des fichiers YAML.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing, Knative Serving et la ressource personnalisée (CR) **KnativeKafka** sont installés sur votre cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez accès à un cluster Red Hat AMQ Streams (Kafka) qui produit les messages Kafka que vous souhaitez importer.
- Vous avez installé le CLI Knative (**kn**).
- Facultatif : Vous avez installé l'OpenShift CLI (**oc**) si vous voulez utiliser les étapes de vérification dans cette procédure.

Procédure

1. Pour vérifier que la source d'événements Kafka fonctionne, créez un service Knative qui déverse les événements entrants dans les journaux du service :

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display
```

2. Créer un CR **KafkaSource**:

```
$ kn source kafka create <kafka_source_name> \
```

```
    --servers <cluster_kafka_bootstrap>.kafka.svc:9092 \
    --topics <topic_name> --consumergroup my-consumer-group \
    --sink event-display
```

NOTE

Remplacez les valeurs de cette commande par les valeurs de votre nom de source, de vos serveurs d'amorçage et de vos rubriques.

Les options **--servers**, **--topics**, et **--consumergroup** spécifient les paramètres de connexion au cluster Kafka. L'option **--consumergroup** est facultative.

3. Facultatif : Affichez les détails de la CR **KafkaSource** que vous avez créée :

```
$ kn source kafka describe <kafka_source_name>
```

Exemple de sortie

```
Name:              example-kafka-source
Namespace:         kafka
Age:               1h
BootstrapServers:  example-cluster-kafka-bootstrap.kafka.svc:9092
Topics:            example-topic
ConsumerGroup:     example-consumer-group

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE           AGE REASON
  ++ Ready          1h
  ++ Deployed         1h
  ++ SinkProvided     1h
```

Verification steps

1. Déclencher l'envoi par l'instance Kafka d'un message au sujet :

```
$ oc -n kafka run kafka-producer \
    -ti --image=quay.io/strimzi/kafka:latest-kafka-2.7.0 --rm=true \
    --restart=Never -- bin/kafka-console-producer.sh \
    --broker-list <cluster_kafka_bootstrap>:9092 --topic my-topic
```

Saisissez le message dans l'invite. Cette commande suppose que :

- Le cluster Kafka est installé dans l'espace de noms **kafka**.
- L'objet **KafkaSource** a été configuré pour utiliser le sujet **my-topic**.

2. Vérifiez que le message est arrivé en consultant les journaux :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

-

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.kafka.event
  source: /apis/v1/namespaces/default/kafkasources/example-kafka-source#example-topic
  subject: partition:46#0
  id: partition:46/offset:0
  time: 2021-03-10T11:21:49.4Z
Extensions,
  traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
  Hello!
```

5.2.5.2.1. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

5.2.5.3. Créer une source d'événement Apache Kafka en utilisant YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire des applications de manière déclarative et reproductible. Pour créer une source Kafka à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **KafkaSource**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous avez accès à un cluster Red Hat AMQ Streams (Kafka) qui produit les messages Kafka que vous souhaitez importer.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créer un objet **KafkaSource** sous la forme d'un fichier YAML :

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: <source_name>
spec:
  consumerGroup: <group_name> 1
  bootstrapServers:
  - <list_of_bootstrap_servers>
  topics:
  - <list_of_topics> 2
  sink:
  - <list_of_sinks> 3
```

1. Un groupe de consommateurs est un groupe de consommateurs qui utilisent le même identifiant de groupe et consomment des données d'un thème.
2. Un thème fournit une destination pour le stockage des données. Chaque thème est divisé en une ou plusieurs partitions.
3. Un puits indique où les événements sont envoyés à partir d'une source.

IMPORTANT

Seule la version **v1beta1** de l'API pour les objets **KafkaSource** sur OpenShift Serverless est prise en charge. N'utilisez pas la version **v1alpha1** de cette API, car elle est désormais obsolète.

Exemple d'objet KafkaSource

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: kafka-source
spec:
  consumerGroup: knative-group
  bootstrapServers:
  - my-cluster-kafka-bootstrap.kafka:9092
  topics:
  - knative-demo-topic
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

2. Appliquer le fichier YAML **KafkaSource**:

```
$ oc apply -f <filename>
```

Vérification

- Vérifiez que la source d'événements Kafka a été créée en entrant la commande suivante :

```
$ oc get pods
```

Exemple de sortie

```
NAME                                 READY   STATUS    RESTARTS   AGE
kafkasource-kafka-source-5ca0248f-...   1/1     Running   0          13m
```

5.2.5.4. Configuration de l'authentification SASL pour les sources Apache Kafka

Simple Authentication and Security Layer (SASL) est utilisé par Apache Kafka pour l'authentification. Si vous utilisez l'authentification SASL sur votre cluster, les utilisateurs doivent fournir des informations d'identification à Knative pour communiquer avec le cluster Kafka ; sinon, les événements ne peuvent pas être produits ou consommés.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** CR sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous disposez d'un nom d'utilisateur et d'un mot de passe pour un cluster Kafka.
- Vous avez choisi le mécanisme SASL à utiliser, par exemple **PLAIN**, **SCRAM-SHA-256**, ou **SCRAM-SHA-512**.
- Si TLS est activé, vous avez également besoin du fichier de certificat **ca.crt** pour le cluster Kafka.
- Vous avez installé le CLI OpenShift (**oc**).

Procédure

1. Créez les fichiers de certificats en tant que secrets dans l'espace de noms que vous avez choisi :

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \ 1
  --from-literal=user="my-sasl-user"
```

1 Le type SASL peut être **PLAIN**, **SCRAM-SHA-256**, ou **SCRAM-SHA-512**.

2. Créez ou modifiez votre source Kafka de manière à ce qu'elle contienne la configuration suivante : **spec**:

```
apiVersion: sources.knative.dev/v1beta1
kind: KafkaSource
metadata:
  name: example-source
spec:
...
  net:
    sasl:
      enable: true
      user:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: user
      password:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: password
      type:
        secretKeyRef:
          name: <kafka_auth_secret>
          key: saslType
    tls:
      enable: true
      caCert: 1
        secretKeyRef:
          name: <kafka_auth_secret>
          key: ca.crt
...
```

1 La spécification **caCert** n'est pas nécessaire si vous utilisez un service Kafka en nuage public, tel que Red Hat OpenShift Streams for Apache Kafka.

5.2.6. Sources d'événements personnalisées

Si vous avez besoin d'intégrer des événements provenant d'un producteur d'événements qui n'est pas inclus dans Knative, ou d'un producteur qui émet des événements qui ne sont pas au format **CloudEvent**, vous pouvez le faire en créant une source d'événements personnalisée. Vous pouvez créer une source d'événement personnalisée en utilisant l'une des méthodes suivantes :

- Utiliser un objet **PodSpecable** comme source d'événements, en créant une liaison de type "sink".
- Utiliser un conteneur comme source d'événements en créant un conteneur source.

5.2.6.1. Fixation de l'évier

L'objet **SinkBinding** permet de découpler la production d'événements de l'adressage de livraison. La liaison Sink est utilisée pour connecter *event producers* à un consommateur d'événements, ou *sink*. Un

producteur d'événements est une ressource Kubernetes qui intègre un modèle **PodSpec** et produit des événements. Un sink est un objet Kubernetes adressable qui peut recevoir des événements.

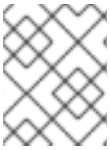
L'objet **SinkBinding** injecte des variables d'environnement dans le site **PodTemplateSpec** du puits, ce qui signifie que le code de l'application n'a pas besoin d'interagir directement avec l'API Kubernetes pour localiser la destination de l'événement. Ces variables d'environnement sont les suivantes :

K_SINK

L'URL de l'évier résolu.

K_CE_OVERRIDES

Un objet JSON qui spécifie les dérogations à l'événement sortant.



NOTE

L'objet **SinkBinding** ne prend actuellement pas en charge les noms de révision personnalisés pour les services.

5.2.6.1.1. Création d'un sink binding à l'aide de YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les sources d'événements de manière déclarative et reproductible. Pour créer une liaison de puits à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **SinkBinding**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Pour vérifier que la liaison des puits est correctement configurée, créez un service d'affichage d'événements Knative, ou puits d'événements, qui déverse les messages entrants dans son journal.
   a. Créer un fichier YAML de service :

Exemple de fichier YAML de service

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

b. Créer le service :

```
$ oc apply -f <filename>
```

2. Créer une instance de liaison d'évier qui dirige les événements vers le service.

a. Créer un fichier YAML de liaison avec le puits :

Exemple de fichier YAML de service

```
apiVersion: sources.knative.dev/v1alpha1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: batch/v1
    kind: Job 1
    selector:
      matchLabels:
        app: heartbeat-cron

  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1 Dans cet exemple, tout Job portant l'étiquette **app: heartbeat-cron** sera lié au puits d'événements.

b. Créer la liaison de l'évier :

```
$ oc apply -f <filename>
```

3. Créer un objet **CronJob**.

a. Créez un fichier YAML de travail cron :

Exemple de fichier YAML d'une tâche cron

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
```

```
    template:
      spec:
        restartPolicy: Never
        containers:
          - name: single-heartbeat
            image: quay.io/openshift-knative/heartbeats:latest
            args:
              - --period=1
            env:
              - name: ONE_SHOT
                value: "true"
              - name: POD_NAME
                valueFrom:
                  fieldRef:
                    fieldPath: metadata.name
              - name: POD_NAMESPACE
                valueFrom:
                  fieldRef:
                    fieldPath: metadata.namespace
```

IMPORTANT

Pour utiliser le sink binding, vous devez ajouter manuellement une étiquette **bindings.knative.dev/include=true** à vos ressources Knative.

Par exemple, pour ajouter cette étiquette à une ressource **CronJob**, ajoutez les lignes suivantes à la définition YAML de la ressource **Job**:

```
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
```

b. Créez la tâche cron :

```
$ oc apply -f <filename>
```

4. Vérifiez que le contrôleur est correctement mappé en entrant la commande suivante et en inspectant la sortie :

```
$ oc get sinkbindings.sources.knative.dev bind-heartbeat -oyaml
```

Exemple de sortie

```
spec:
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
      namespace: default
  subject:
```

```
  apiVersion: batch/v1
  kind: Job
  namespace: default
  selector:
    matchLabels:
      app: heartbeat-cron
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés au puits d'événements Knative en examinant les journaux de la fonction dumper de messages.

1. Entrez la commande :

```
$ oc get pods
```

2. Entrez la commande :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

5.2.6.1.2. Création d'un sink binding à l'aide du CLI Knative

Vous pouvez utiliser la commande **kn source binding create** pour créer une liaison de puits en utilisant l'interface de programmation Knative (**kn**). L'utilisation de l'interface de programmation Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.

- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Installer le CLI Knative (**kn**).
- Installez le CLI OpenShift (**oc**).

NOTE

La procédure suivante nécessite la création de fichiers YAML.

Si vous modifiez les noms des fichiers YAML par rapport à ceux utilisés dans les exemples, vous devez vous assurer que vous mettez également à jour les commandes CLI correspondantes.

Procédure

1. Pour vérifier que la liaison des puits est correctement configurée, créez un service d'affichage d'événements Knative, ou puits d'événements, qui déverse les messages entrants dans son journal :

```
$ kn service create event-display --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

2. Créer une instance de liaison d'évier qui dirige les événements vers le service :

```
$ kn source binding create bind-heartbeat --subject Job:batch/v1:app=heartbeat-cron --sink ksvc:event-display
```

3. Créer un objet **CronJob**.
   a. Créez un fichier YAML de travail cron :

Exemple de fichier YAML d'une tâche cron

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "* * * * *"
  jobTemplate:
    metadata:
      labels:
        app: heartbeat-cron
        bindings.knative.dev/include: "true"
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - name: single-heartbeat
              image: quay.io/openshift-knative/heartbeats:latest
              args:
```

```
        - --period=1
      env:
        - name: ONE_SHOT
          value: "true"
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
```

IMPORTANT

Pour utiliser le sink binding, vous devez ajouter manuellement une étiquette **bindings.knative.dev/include=true** à vos CR Knative.

Par exemple, pour ajouter cette étiquette à un CR **CronJob**, ajoutez les lignes suivantes à la définition YAML du CR **Job**:

```
jobTemplate:
  metadata:
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: "true"
```

b. Créez la tâche cron :

```
$ oc apply -f <filename>
```

4. Vérifiez que le contrôleur est correctement mappé en entrant la commande suivante et en inspectant la sortie :

```
$ kn source binding describe bind-heartbeat
```

Exemple de sortie

```
Name:         bind-heartbeat
Namespace:    demo-2
Annotations:  sources.knative.dev/creator=minikube-user,
sources.knative.dev/lastModifier=minikub ...
Age:          2m
Subject:
  Resource:   job (batch/v1)
  Selector:
    app:      heartbeat-cron
Sink:
  Name:       event-display
  Resource:   Service (serving.knative.dev/v1)
```

```
Conditions:
  OK TYPE    AGE REASON
  ++ Ready   2m
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés au puits d'événements Knative en examinant les journaux de la fonction dumper de messages.

- Affichez les journaux des fonctions de l'expéditeur de messages en entrant les commandes suivantes :

```
$ oc get pods
```

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.eventing.samples.heartbeat
  source: https://knative.dev/eventing-contrib/cmd/heartbeats/#event-test/mypod
  id: 2b72d7bf-c38f-4a98-a433-608fbcdd2596
  time: 2019-10-18T15:23:20.809775386Z
  contenttype: application/json
Extensions,
  beats: true
  heart: yes
  the: 42
Data,
  {
    "id": 1,
    "label": ""
  }
```

5.2.6.1.2.1. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

5.2.6.1.3. Création d'une liaison de puits à l'aide de la console web

Une fois Knative Eventing installé sur votre cluster, vous pouvez créer un sink binding en utilisant la console web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer une source d'événement.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créer un service Knative à utiliser comme puits :
   a. Dans la perspective **Developer**, naviguez vers **Add → YAML**.
   b. Copiez l'exemple YAML :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: event-display
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

   c. Cliquez sur **Create**.
2. Créez une ressource **CronJob** qui est utilisée comme source d'événements et qui envoie un événement toutes les minutes.
   a. Dans la perspective **Developer**, naviguez vers **Add → YAML**.
   b. Copiez l'exemple YAML :

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: heartbeat-cron
spec:
  # Run every minute
  schedule: "*/1 * * * *"
  jobTemplate:
    metadata:
```

```
    labels:
      app: heartbeat-cron
      bindings.knative.dev/include: true (1)
  spec:
    template:
      spec:
        restartPolicy: Never
        containers:
          - name: single-heartbeat
            image: quay.io/openshift-knative/heartbeats
            args:
            - --period=1
            env:
              - name: ONE_SHOT
                value: "true"
              - name: POD_NAME
                valueFrom:
                  fieldRef:
                    fieldPath: metadata.name
              - name: POD_NAMESPACE
                valueFrom:
                  fieldRef:
                    fieldPath: metadata.namespace
```

(1) Assurez-vous d'inclure l'étiquette **bindings.knative.dev/include: true**. Le comportement de sélection de l'espace de noms par défaut d'OpenShift Serverless utilise le mode d'inclusion.

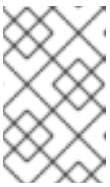
c. Cliquez sur **Create**.

3. Créez un lien sink dans le même espace de noms que le service créé à l'étape précédente, ou tout autre sink vers lequel vous souhaitez envoyer des événements.

   a. Dans la perspective **Developer**, naviguez vers **Add → Event Source**. La page **Event Sources** s'affiche.

   b. Facultatif : si vous avez plusieurs fournisseurs pour vos sources d'événements, sélectionnez le fournisseur requis dans la liste **Providers** pour filtrer les sources d'événements disponibles du fournisseur.

   c. Sélectionnez **Sink Binding** puis cliquez sur **Create Event Source** La page **Create Event Source** s'affiche.

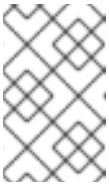


NOTE

Vous pouvez configurer les paramètres de **Sink Binding** en utilisant **Form view** ou **YAML view** et passer d'une vue à l'autre. Les données sont conservées lors du passage d'une vue à l'autre.

   d. Dans le champ **apiVersion**, entrez **batch/v1**.

   e. Dans le champ **Kind**, entrez **Job**.



NOTE

Le type **CronJob** n'est pas pris en charge directement par la liaison OpenShift Serverless sink, de sorte que le champ **Kind** doit cibler les objets **Job** créés par la tâche cron, plutôt que l'objet de la tâche cron elle-même.

f. Sélectionnez un **Sink**. Il peut s'agir d'un **Resource** ou d'un **URI**. Dans cet exemple, le service **event-display** créé à l'étape précédente est utilisé comme puits **Resource**.

g. Dans la section **Match labels**:

i. Saisissez **app** dans le champ **Name**.

ii. Saisissez **heartbeat-cron** dans le champ **Value**.

NOTE

Le sélecteur d'étiquette est nécessaire lors de l'utilisation de tâches cron avec le sink binding, plutôt que le nom de la ressource. En effet, les travaux créés par un travail cron n'ont pas de nom prévisible et contiennent une chaîne générée de manière aléatoire dans leur nom. Par exemple, **hearthbeat-cron-1cc23f**.

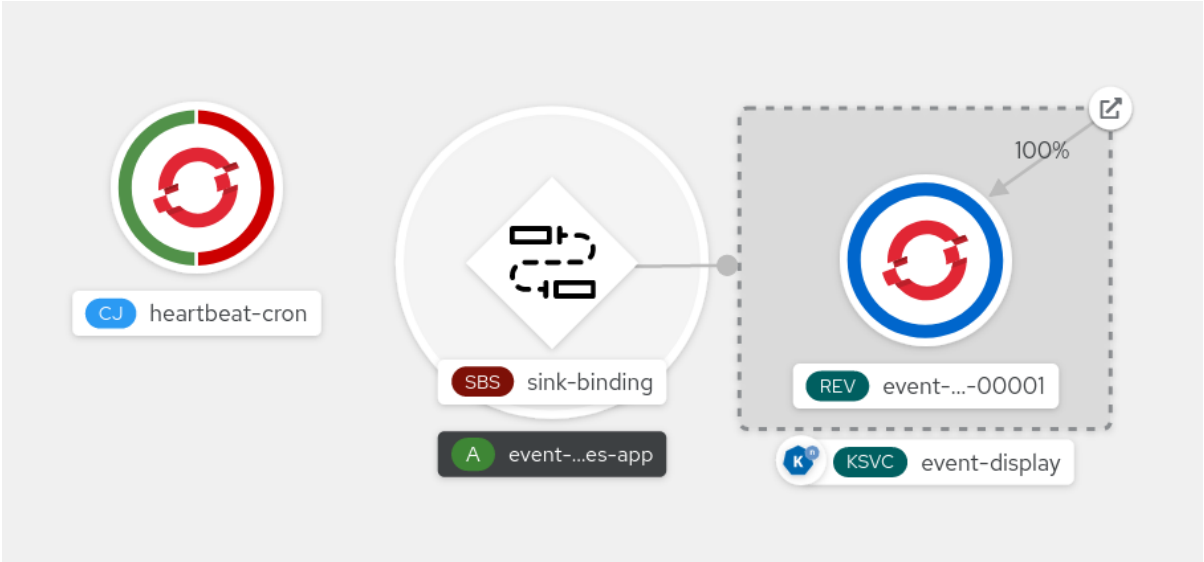
h. Cliquez sur **Create**.

Vérification

Vous pouvez vérifier que la liaison et l'évier, ainsi que la tâche cron ont été créés et fonctionnent correctement en consultant la page **Topology** et les journaux de pods.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.

2. Affichez le travail cron de liaison, d'évitement et de battements de cœur de l'évier.



3. Observez que des travaux réussis sont enregistrés par la tâche cron une fois que la liaison sink a été ajoutée. Cela signifie que la liaison sink reconfigure avec succès les travaux créés par la tâche cron.

4. Parcourez les journaux du pod de service **event-display** pour voir les événements produits par le job cron heartbeats.

5.2.6.1.4. Référence pour la fixation de l'évier

Vous pouvez utiliser un objet **PodSpecable** comme source d'événement en créant une liaison de puits. Vous pouvez configurer plusieurs paramètres lors de la création d'un objet **SinkBinding**.

SinkBinding prennent en charge les paramètres suivants :

Field	Description	Obligatoire ou facultatif
apiVersion	Spécifie la version de l'API, par exemple **sources.knative.dev/v1**.	Exigée
kind	Identifie cet objet de ressource en tant qu'objet **SinkBinding**.	Exigée
metadata	Spécifie les métadonnées qui identifient de manière unique l'objet **SinkBinding**. Par exemple, une adresse **name**.	Exigée
spec	Spécifie les informations de configuration pour cet objet **SinkBinding**.	Exigée
spec.sink	Une référence à un objet qui se résout en un URI à utiliser comme puits.	Exigée
spec.subject	Fait référence aux ressources pour lesquelles le contrat d'exécution est complété par des implémentations contraignantes.	Exigée
spec.ceOverrides	Définit les surcharges permettant de contrôler le format de sortie et les modifications apportées à l'événement envoyé à l'émetteur.	En option

5.2.6.1.4.1. Paramètre du sujet

Le paramètre **Subject** fait référence aux ressources pour lesquelles le contrat d'exécution est complété par des implémentations de liaison. Vous pouvez configurer plusieurs champs pour une définition **Subject**.

La définition de **Subject** prend en charge les champs suivants :

Field	Description	Obligatoire ou facultatif
apiVersion	Version API du référent.	Exigée

Field	Description	Obligatoire ou facultatif
kind	C'est en quelque sorte le référent.	Exigée
namespace	Espace de noms du référent. S'il est omis, il s'agit par défaut de l'espace de noms de l'objet.	En option
name	Nom du référent.	Ne pas utiliser si vous configurez **selector**.
selector	Sélecteur de référents.	Ne pas utiliser si vous configurez **name**.
selector.matchExpressions	Une liste des exigences en matière de sélecteur d'étiquettes.	N'utilisez qu'une seule des deux options suivantes : **matchExpressions** ou **matchLabels**.
selector.matchExpressions.key	Clé de l'étiquette à laquelle s'applique le sélecteur.	Nécessaire si l'on utilise **matchExpressions**.
selector.matchExpressions.operator	Représente la relation d'une clé avec un ensemble de valeurs. Les opérateurs valides sont **In**, **NotIn**, **Exists** et **DoesNotExist**.	Nécessaire si l'on utilise **matchExpressions**.
selector.matchExpressions.values	Un tableau de valeurs de chaînes de caractères. Si la valeur du paramètre **operator** est **In** ou **NotIn**, le tableau de valeurs doit être non vide. Si la valeur du paramètre **operator** est **Exists** ou **DoesNotExist**, le tableau de valeurs doit être vide. Ce tableau est remplacé lors d'un patch de fusion stratégique.	Nécessaire si l'on utilise **matchExpressions**.
selector.matchLabels	Une carte de paires clé-valeur. Chaque paire clé-valeur de la carte **matchLabels** est équivalente à un élément de **matchExpressions**, où le champ clé est **matchLabels.<key>**, le champ **operator** est **In**, et le tableau **values** ne contient que **matchLabels.<value>**.	N'utilisez qu'une seule des deux options suivantes : **matchExpressions** ou **matchLabels**.

Exemples de paramètres du sujet

Étant donné le YAML suivant, l'objet **Deployment** nommé **mysubject** dans l'espace de noms **default** est sélectionné :

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: apps/v1
    kind: Deployment
    namespace: default
    name: mysubject
  ...
```

Étant donné le YAML suivant, tout objet **Job** portant l'étiquette **working=example** dans l'espace de noms **default** est sélectionné :

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: batch/v1
    kind: Job
    namespace: default
    selector:
      matchLabels:
        working: example
  ...
```

Étant donné le YAML suivant, tout objet **Pod** portant l'étiquette **working=example** ou **working=sample** dans l'espace de noms **default** est sélectionné :

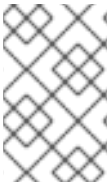
```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  subject:
    apiVersion: v1
    kind: Pod
    namespace: default
    selector:
      - matchExpression:
        key: working
        operator: In
        values:
          - example
          - sample
  ...
```

5.2.6.1.4.2. Surcharges de CloudEvent

Une définition de **ceOverrides** fournit des paramètres qui contrôlent le format de sortie du CloudEvent et les modifications envoyées au récepteur. Vous pouvez configurer plusieurs champs pour la définition **ceOverrides**.

Une définition de **ceOverrides** prend en charge les champs suivants :

Field	Description	Obligatoire ou facultatif
extensions	Spécifie quels attributs sont ajoutés ou remplacés sur l'événement sortant. Chaque paire clé-valeur **extensions** est définie indépendamment sur l'événement en tant qu'extension d'attribut.	En option



NOTE

Seuls les noms d'attributs **CloudEvent** valides sont autorisés en tant qu'extensions. Vous ne pouvez pas définir les attributs définis par la spécification à partir de la configuration des extensions. Par exemple, vous ne pouvez pas modifier l'attribut **type**.

Exemple de surcharges de CloudEvent

```
apiVersion: sources.knative.dev/v1
kind: SinkBinding
metadata:
  name: bind-heartbeat
spec:
  ...
  ceOverrides:
    extensions:
      extra: this is an extra attribute
      additional: 42
```

Cette opération définit la variable d'environnement **K_CE_OVERRIDES** sur le serveur **subject**:

Exemple de sortie

```
{ "extensions": { "extra": "this is an extra attribute", "additional": "42" } }
```

5.2.6.1.4.3. L'étiquette d'inclusion

Pour utiliser un sink binding, vous devez attribuer l'étiquette **bindings.knative.dev/include: "true"** à la ressource ou à l'espace de noms dans lequel la ressource est incluse. Si la définition de la ressource n'inclut pas l'étiquette, un administrateur de cluster peut l'attacher à l'espace de noms en exécutant la commande suivante :

```
$ oc label namespace <namespace> bindings.knative.dev/include=true
```

5.2.6.2. Source du conteneur

Les sources de conteneurs créent une image de conteneur qui génère des événements et les envoie à un récepteur. Vous pouvez utiliser une source de conteneur pour créer une source d'événements personnalisée, en créant une image de conteneur et un objet **ContainerSource** qui utilise l'URI de votre image.

5.2.6.2.1. Lignes directrices pour la création d'une image de conteneur

Deux variables d'environnement sont injectées par le contrôleur de source du conteneur : **K_SINK** et **K_CE_OVERRIDES**. Ces variables sont résolues à partir des spécifications **sink** et **ceOverrides**, respectivement. Les événements sont envoyés à l'URI de destination spécifié dans la variable d'environnement **K_SINK**. Le message doit être envoyé en tant que **POST** en utilisant le format **CloudEvent** En utilisant le format HTTP.

Exemples d'images de conteneurs

Voici un exemple d'image de conteneur de battements de cœur :

```
package main

import (
 "context"
 "encoding/json"
 "flag"
 "fmt"
 "log"
 "os"
 "strconv"
 "time"

 duckv1 "knative.dev/pkg/apis/duck/v1"

 cloudevents "github.com/cloudevents/sdk-go/v2"
 "github.com/kelseyhightower/envconfig"
)

type Heartbeat struct {
 Sequence int    `json:"id"`
 Label    string `json:"label"`
}

var (
 eventSource string
 eventType   string
 sink        string
 label       string
 periodStr   string
)

func init() {
 flag.StringVar(&eventSource, "eventSource", "", "the event-source (CloudEvents)")
 flag.StringVar(&eventType, "eventType", "dev.knative.eventing.samples.heartbeat", "the event-type
(CloudEvents)")
 flag.StringVar(&sink, "sink", "", "the host url to heartbeat to")
 flag.StringVar(&label, "label", "", "a special label")
 flag.StringVar(&periodStr, "period", "5", "the number of seconds between heartbeats")
}
```

```
type envConfig struct {
 // Sink URL where to send heartbeat cloud events
 Sink string `envconfig:"K_SINK"`

 // CEOverrides are the CloudEvents overrides to be applied to the outbound event.
 CEOverrides string `envconfig:"K_CE_OVERRIDES"`

 // Name of this pod.
 Name string `envconfig:"POD_NAME" required:"true"`

 // Namespace this pod exists in.
 Namespace string `envconfig:"POD_NAMESPACE" required:"true"`

 // Whether to run continuously or exit.
 OneShot bool `envconfig:"ONE_SHOT" default:"false"`
}

func main() {
 flag.Parse()

 var env envConfig
 if err := envconfig.Process("", &env); err != nil {
  log.Printf("[ERROR] Failed to process env var: %s", err)
  os.Exit(1)
 }

 if env.Sink != "" {
  sink = env.Sink
 }

 var ceOverrides *duckv1.CloudEventOverrides
 if len(env.CEOverrides) > 0 {
  overrides := duckv1.CloudEventOverrides{}
  err := json.Unmarshal([]byte(env.CEOverrides), &overrides)
  if err != nil {
   log.Printf("[ERROR] Unparseable CloudEvents overrides %s: %v", env.CEOverrides, err)
   os.Exit(1)
  }
  ceOverrides = &overrides
 }

 p, err := cloudevents.NewHTTP(cloudevents.WithTarget(sink))
 if err != nil {
  log.Fatalf("failed to create http protocol: %s", err.Error())
 }

 c, err := cloudevents.NewClient(p, cloudevents.WithUUIDs(), cloudevents.WithTimeNow())
 if err != nil {
  log.Fatalf("failed to create client: %s", err.Error())
 }

 var period time.Duration
 if p, err := strconv.Atoi(periodStr); err != nil {
  period = time.Duration(5) * time.Second
 } else {
```

```
  period = time.Duration(p) * time.Second
 }

 if eventSource == "" {
  eventSource = fmt.Sprintf("https://knative.dev/eventing-contrib/cmd/heartbeats/#%s/%s",
env.Namespace, env.Name)
  log.Printf("Heartbeats Source: %s", eventSource)
 }

 if len(label) > 0 && label[0] == '"' {
  label, _ = strconv.Unquote(label)
 }
 hb := &Heartbeat{
  Sequence: 0,
  Label:    label,
 }
 ticker := time.NewTicker(period)
 for {
  hb.Sequence++

  event := cloudevents.NewEvent("1.0")
  event.SetType(eventType)
  event.SetSource(eventSource)
  event.SetExtension("the", 42)
  event.SetExtension("heart", "yes")
  event.SetExtension("beats", true)

  if ceOverrides != nil && ceOverrides.Extensions != nil {
   for n, v := range ceOverrides.Extensions {
    event.SetExtension(n, v)
   }
  }

  if err := event.SetData(cloudevents.ApplicationJSON, hb); err != nil {
   log.Printf("failed to set cloudevents data: %s", err.Error())
  }

  log.Printf("sending cloudevent to %s", sink)
  if res := c.Send(context.Background(), event); !cloudevents.IsACK(res) {
   log.Printf("failed to send cloudevent: %v", res)
  }

  if env.OneShot {
   return
  }

  // Wait for next tick
  <-ticker.C
 }
}
```

Voici un exemple de source de conteneur qui fait référence à l'image de conteneur Heartbeats précédente :

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
```

```
metadata:
  name: test-heartbeats
spec:
  template:
    spec:
      containers:
        # This corresponds to a heartbeats image URI that you have built and published
        - image: gcr.io/knative-releases/knative.dev/eventing/cmd/heartbeats
          name: heartbeats
          args:
            - --period=1
          env:
            - name: POD_NAME
              value: "example-pod"
            - name: POD_NAMESPACE
              value: "event-test"
  sink:
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: example-service
...
```

5.2.6.2.2. Créer et gérer des sources de conteneurs en utilisant le CLI Knative

Vous pouvez utiliser les commandes **kn source container** pour créer et gérer des sources de conteneurs en utilisant le CLI Knative (**kn**). L'utilisation de l'interface de programmation Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Créer un conteneur source

```
$ kn source container create <container_source_name> --image <image_uri> --sink <sink>
```

Supprimer un conteneur source

```
kn source container delete <container_source_name>
```

Décrire la source d'un conteneur

```
kn source container describe <container_source_name>
```

Liste des sources de conteneurs existantes

```
$ kn source container list
```

Liste des sources de conteneurs existantes au format YAML

```
$ kn source container list -o yaml
```

Mise à jour d'un conteneur source

Cette commande met à jour l'URI de l'image d'un conteneur existant :

```
$ kn source container update <container_source_name> --image <image_uri>
```

5.2.6.2.3. Création d'un conteneur source à l'aide de la console web

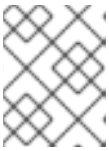
Une fois Knative Eventing installé sur votre cluster, vous pouvez créer une source de conteneur en utilisant la console web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer une source d'événements.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Dans la perspective **Developer**, naviguez vers **Add → Event Source**. La page **Event Sources** s'affiche.
2. Sélectionnez **Container Source** puis cliquez sur **Create Event Source**. La page **Create Event Source** s'affiche.
3. Configurez les paramètres de **Container Source** en utilisant les boutons **Form view** ou **YAML view**:



NOTE

Vous pouvez passer de la vue **Form view** à la vue **YAML view**. Les données sont conservées lors du passage d'une vue à l'autre.

   a. Dans le champ **Image**, saisissez l'URI de l'image que vous souhaitez exécuter dans le conteneur créé par la source du conteneur.
   b. Dans le champ **Name**, saisissez le nom de l'image.
   c. Facultatif : dans le champ **Arguments**, saisissez les arguments à transmettre au conteneur.
   d. Facultatif : dans le champ **Environment variables**, ajoutez les variables d'environnement à définir dans le conteneur.
   e. Dans la section **Sink**, ajoutez un puits vers lequel les événements de la source du conteneur sont acheminés. Si vous utilisez la vue **Form**, vous pouvez choisir parmi les options suivantes :
      i. Sélectionnez **Resource** pour utiliser un canal, un courtier ou un service comme puits pour la source d'événement.
      ii. Sélectionnez **URI** pour spécifier où les événements de la source du conteneur sont acheminés.
4. Une fois la configuration de la source du conteneur terminée, cliquez sur **Create**.

5.2.6.2.4. Référence de la source du conteneur

Vous pouvez utiliser un conteneur comme source d'événements en créant un objet **ContainerSource**. Vous pouvez configurer plusieurs paramètres lors de la création d'un objet **ContainerSource**.

ContainerSource prennent en charge les champs suivants :

Field	Description	Obligatoire ou facultatif
apiVersion	Spécifie la version de l'API, par exemple **sources.knative.dev/v1**.	Exigée
kind	Identifie cet objet de ressource en tant qu'objet **ContainerSource**.	Exigée
metadata	Spécifie les métadonnées qui identifient de manière unique l'objet **ContainerSource**. Par exemple, une adresse **name**.	Exigée
spec	Spécifie les informations de configuration pour cet objet **ContainerSource**.	Exigée
spec.sink	Une référence à un objet qui se résout en un URI à utiliser comme puits.	Exigée
spec.template	Une spécification **template** pour l'objet **ContainerSource**.	Exigée
spec.ceOverrides	Définit les surcharges permettant de contrôler le format de sortie et les modifications apportées à l'événement envoyé à l'émetteur.	En option

Exemple de paramètre de modèle

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  template:
    spec:
      containers:
        - image: quay.io/openshift-knative/heartbeats:latest
          name: heartbeats
          args:
            - --period=1
```

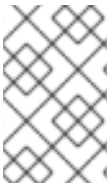
```
        env:
          - name: POD_NAME
            value: "mypod"
          - name: POD_NAMESPACE
            value: "event-test"
    ...
```

5.2.6.2.4.1. Surcharges de CloudEvent

Une définition de **ceOverrides** fournit des paramètres qui contrôlent le format de sortie du CloudEvent et les modifications envoyées au récepteur. Vous pouvez configurer plusieurs champs pour la définition **ceOverrides**.

Une définition de **ceOverrides** prend en charge les champs suivants :

Field	Description	Obligatoire ou facultatif
extensions	Spécifie quels attributs sont ajoutés ou remplacés sur l'événement sortant. Chaque paire clé-valeur **extensions** est définie indépendamment sur l'événement en tant qu'extension d'attribut.	En option



NOTE

Seuls les noms d'attributs **CloudEvent** valides sont autorisés en tant qu'extensions. Vous ne pouvez pas définir les attributs définis par la spécification à partir de la configuration des extensions. Par exemple, vous ne pouvez pas modifier l'attribut **type**.

Exemple de surcharges de CloudEvent

```
apiVersion: sources.knative.dev/v1
kind: ContainerSource
metadata:
  name: test-heartbeats
spec:
  ...
  ceOverrides:
    extensions:
      extra: this is an extra attribute
      additional: 42
```

Cette opération définit la variable d'environnement **K_CE_OVERRIDES** sur le serveur **subject**:

Exemple de sortie

```
{ "extensions": { "extra": "this is an extra attribute", "additional": "42" } }
```

5.2.7. Connexion d'une source d'événements à un puits à l'aide de la perspective du développeur

Lorsque vous créez une source d'événement en utilisant la console web d'OpenShift Container Platform, vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette source. Le puits peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

5.2.7.1. Connecter une source d'événements à un puits à l'aide de la perspective du développeur

Conditions préalables

- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Developer**.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez créé un puits, tel qu'un service, un canal ou un courtier Knative.

Procédure

1. Créez une source d'événement de n'importe quel type, en naviguant vers **Add → Event Source** et en sélectionnant le type de source d'événement que vous souhaitez créer.
2. Dans la section **Sink** de la vue du formulaire **Create Event Source**, sélectionnez votre évier dans la liste **Resource**.
3. Cliquez sur **Create**.

Vérification

Vous pouvez vérifier que la source d'événements a été créée et qu'elle est connectée au puits en consultant la page **Topology**.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.
2. Affichez la source de l'événement et cliquez sur le puits connecté pour afficher les détails du puits dans le panneau de droite.

5.3. LES PUITS D'ÉVÉNEMENTS

5.3.1. Les puits d'événements

Lorsque vous créez une source d'événements, vous pouvez spécifier un puits d'événements vers lequel les événements sont envoyés à partir de la source. Un puits d'événements est une ressource adressable ou appelable qui peut recevoir des événements d'autres ressources. Les services Knative, les canaux et les courtiers sont des exemples de puits d'événements. Il existe également un type de puits Apache Kafka spécifique.

Les objets adressables reçoivent et accusent réception d'un événement transmis par HTTP à une adresse définie dans leur champ **status.address.url**. Dans un cas particulier, l'objet principal Kubernetes **Service** remplit également l'interface adressable.

Les objets appelables sont capables de recevoir un événement transmis par HTTP et de le transformer, en renvoyant **0** ou **1** nouveaux événements dans la réponse HTTP. Ces événements renvoyés peuvent être traités de la même manière que les événements provenant d'une source externe.

5.3.1.1. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

ASTUCE

Vous pouvez configurer les CR qui peuvent être utilisés avec l'indicateur **--sink** pour les commandes CLI Knative (**kn**) en personnalisant **kn**.

5.3.2. Création de puits d'événements

Lorsque vous créez une source d'événements, vous pouvez spécifier un puits d'événements vers lequel les événements sont envoyés à partir de la source. Un puits d'événements est une ressource adressable ou appelable qui peut recevoir des événements d'autres ressources. Les services Knative, les canaux et les courtiers sont des exemples de puits d'événements. Il existe également un type de puits Apache Kafka spécifique.

Pour plus d'informations sur la création de ressources pouvant être utilisées comme puits d'événements, voir la documentation suivante :

- Applications sans serveur
- Création de courtiers
- Création de canaux
- Puits Kafka

5.3.3. Sink pour Apache Kafka

Les puits Apache Kafka sont un type de puits d'événements disponibles si un administrateur de cluster a activé Apache Kafka sur votre cluster. Vous pouvez envoyer des événements directement d'une source d'événements à un sujet Kafka en utilisant un puits Kafka.

5.3.3.1. Créer un puits Apache Kafka en utilisant YAML

Vous pouvez créer un puits Kafka qui envoie des événements à un sujet Kafka. Par défaut, un sink Kafka utilise le mode de contenu binaire, qui est plus efficace que le mode structuré. Pour créer un puits Kafka à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **KafkaSink**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Eventing et la ressource personnalisée (CR) **KnativeKafka** sont installés sur votre cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez accès à un cluster Red Hat AMQ Streams (Kafka) qui produit les messages Kafka que vous souhaitez importer.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créer une définition d'objet **KafkaSink** sous forme de fichier YAML :

   Evier Kafka YAML

   ```
   apiVersion: eventing.knative.dev/v1alpha1
   kind: KafkaSink
   metadata:
     name: <sink-name>
     namespace: <namespace>
   spec:
     topic: <topic-name>
     bootstrapServers:
      - <bootstrap-server>
   ```

2. Pour créer le puits Kafka, appliquez le fichier YAML **KafkaSink**:

   ```
   $ oc apply -f <filename>
   ```

3. Configurer une source d'événements de manière à ce que le puits soit spécifié dans sa spécification :

   Exemple d'un puits Kafka connecté à une source de serveur API

   ```
   apiVersion: sources.knative.dev/v1alpha2
   kind: ApiServerSource
   metadata:
     name: <source-name> 1
     namespace: <namespace> 2
   ```

```
spec:
  serviceAccountName: <service-account-name> 3
  mode: Resource
  resources:
  - apiVersion: v1
    kind: Event
  sink:
    ref:
      apiVersion: eventing.knative.dev/v1alpha1
      kind: KafkaSink
      name: <sink-name> 4
```

1. Le nom de la source de l'événement.
2. L'espace de noms de la source de l'événement.
3. Le compte de service pour la source de l'événement.
4. Le nom du puits Kafka.

5.3.3.2. Créer un puits d'événements pour Apache Kafka en utilisant la console web d'OpenShift Container Platform

Vous pouvez créer un puits Kafka qui envoie des événements à un sujet Kafka en utilisant la perspective **Developer** dans la console web d'OpenShift Container Platform. Par défaut, un sink Kafka utilise le mode de contenu binaire, qui est plus efficace que le mode structuré.

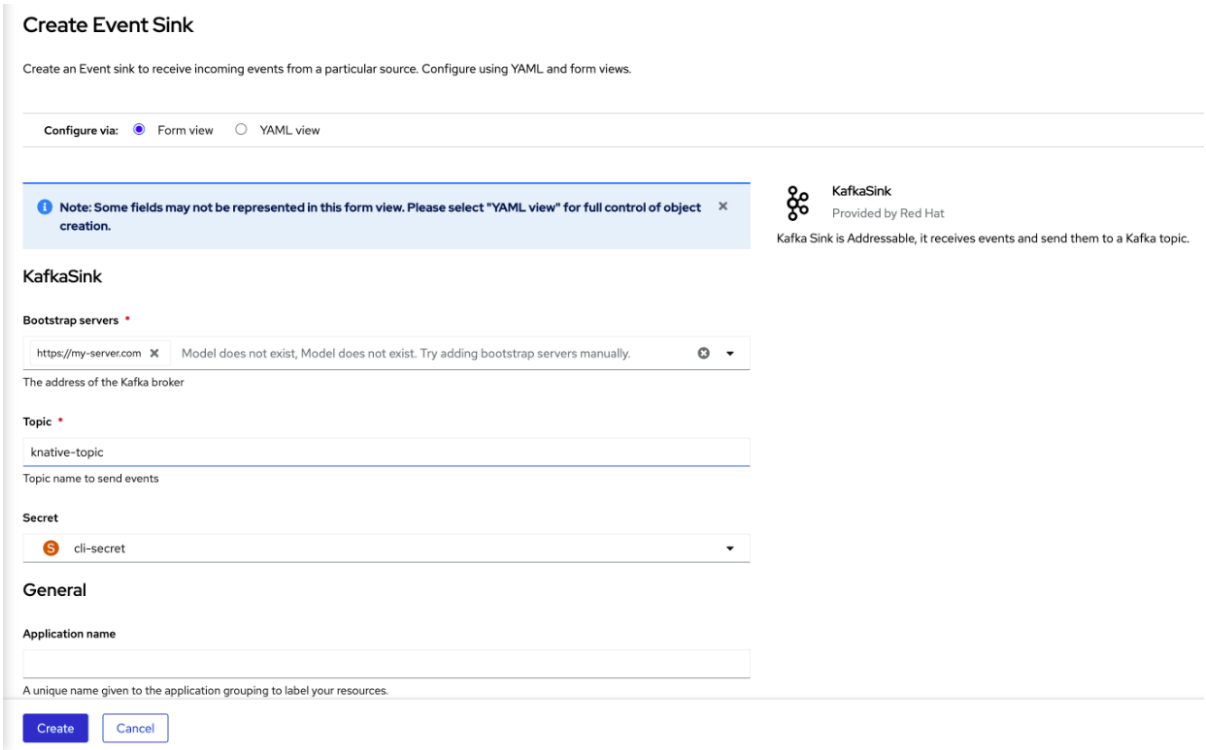
En tant que développeur, vous pouvez créer un puits d'événements pour recevoir des événements d'une source particulière et les envoyer à un sujet Kafka.

Conditions préalables

- Vous avez installé l'OpenShift Serverless Operator, avec Knative Serving, Knative Eventing, et Knative broker for Apache Kafka APIs, depuis l'OperatorHub.
- Vous avez créé un sujet Kafka dans votre environnement Kafka.

Procédure

1. Dans la perspective **Developer**, accédez à la vue **Add**.
2. Cliquez sur **Event Sink** dans la page **Eventing catalog**.
3. Recherchez **KafkaSink** dans les éléments du catalogue et cliquez dessus.
4. Cliquez sur **Create Event Sink**
5. Dans la vue du formulaire, saisissez l'URL du serveur d'amorçage, qui est une combinaison du nom d'hôte et du port.



6. Saisissez le nom de la rubrique à laquelle envoyer les données d'événement.
7. Saisissez le nom du puits d'événement.
8. Cliquez sur **Create**.

Vérification

1. Dans la perspective **Developer**, accédez à la vue **Topology**.
2. Cliquez sur le puits d'événement créé pour afficher ses détails dans le panneau de droite.

5.3.3.3. Configuration de la sécurité pour les puits Apache Kafka

Transport Layer Security (TLS) est utilisé par les clients et les serveurs Apache Kafka pour chiffrer le trafic entre Knative et Kafka, ainsi que pour l'authentification. TLS est la seule méthode de cryptage du trafic prise en charge par l'implémentation du courtier Knative pour Apache Kafka.

Simple Authentication and Security Layer (SASL) est utilisé par Apache Kafka pour l'authentification. Si vous utilisez l'authentification SASL sur votre cluster, les utilisateurs doivent fournir des informations d'identification à Knative pour communiquer avec le cluster Kafka ; sinon, les événements ne peuvent pas être produits ou consommés.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et les ressources personnalisées (CR) **KnativeKafka** sont installés sur votre cluster OpenShift Container Platform.
- Le puits Kafka est activé dans le CR **KnativeKafka**.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous disposez d'un certificat d'autorité de certification pour le cluster Kafka stocké dans un fichier **.pem**.
- Vous disposez d'un certificat client pour le cluster Kafka et d'une clé stockée sous forme de fichiers **.pem**.
- Vous avez installé le CLI OpenShift (**oc**).
- Vous avez choisi le mécanisme SASL à utiliser, par exemple **PLAIN**, **SCRAM-SHA-256**, ou **SCRAM-SHA-512**.

Procédure

1. Créez les fichiers de certificat en tant que secret dans le même espace de noms que votre objet **KafkaSink**:

IMPORTANT

Les certificats et les clés doivent être au format PEM.

- Pour l'authentification par SASL sans cryptage :

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SASL_PLAINTEXT \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-literal=user=<username> \
  --from-literal=password=<password>
```

- Pour l'authentification à l'aide de SASL et le cryptage à l'aide de TLS :

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SASL_SSL \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-file=ca.crt=<my_caroot.pem_file_path> \ 1
  --from-literal=user=<username> \
  --from-literal=password=<password>
```

1 L'adresse **ca.crt** peut être omise pour utiliser le jeu de CA racine du système si vous utilisez un service Kafka géré dans un nuage public, tel que Red Hat OpenShift Streams pour Apache Kafka.

- Pour l'authentification et le cryptage à l'aide de TLS :

```
$ oc create secret -n <namespace> generic <secret_name> \
  --from-literal=protocol=SSL \
  --from-file=ca.crt=<my_caroot.pem_file_path> \ 1
  --from-file=user.crt=<my_cert.pem_file_path> \
  --from-file=user.key=<my_key.pem_file_path>
```

1 L'adresse **ca.crt** peut être omise pour utiliser le jeu de CA racine du système si vous utilisez un service Kafka géré dans un nuage public, tel que Red Hat OpenShift Streams pour Apache Kafka.

2. Créez ou modifiez un objet **KafkaSink** et ajoutez une référence à votre secret dans la spécification **auth**:

```
apiVersion: eventing.knative.dev/v1alpha1
kind: KafkaSink
metadata:
  name: <sink_name>
  namespace: <namespace>
spec:
...
  auth:
    secret:
      ref:
        name: <secret_name>
...
```

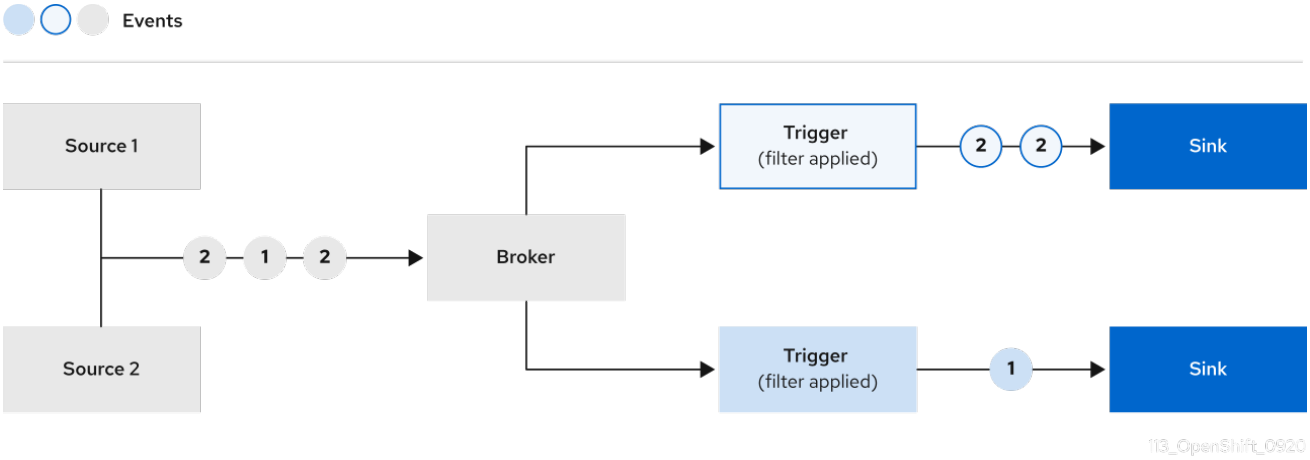
3. Appliquer l'objet **KafkaSink**:

```
$ oc apply -f <filename>
```

5.4. COURTIERS

5.4.1. Courtiers

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Les événements sont envoyés d'une source d'événements à un broker sous la forme d'une requête HTTP **POST**. Une fois que les événements sont entrés dans le courtier, ils peuvent être filtrés par les attributs CloudEvent à l'aide de déclencheurs, puis envoyés sous la forme d'une requête HTTP **POST** à un puits d'événements.



5.4.2. Types de courtiers

Les administrateurs de clusters peuvent définir l'implémentation par défaut des courtiers pour un cluster. Lorsque vous créez un courtier, l'implémentation du courtier par défaut est utilisée, à moins que vous ne fournissiez des configurations définies dans l'objet **Broker**.

5.4.2.1. Mise en œuvre par défaut du courtier à des fins de développement

Knative fournit une implémentation par défaut d'un courtier basé sur un canal. Ce courtier peut être

utilisé à des fins de développement et de test, mais il ne fournit pas de garanties suffisantes en matière de livraison d'événements pour les environnements de production. Le courtier par défaut est soutenu par l'implémentation du canal **InMemoryChannel** par défaut.

Si vous souhaitez utiliser Apache Kafka pour réduire les sauts de réseau, utilisez l'implémentation du courtier Knative pour Apache Kafka. Ne configurez pas le courtier basé sur les canaux pour qu'il soit soutenu par l'implémentation du canal **KafkaChannel**.

5.4.2.2. Mise en œuvre d'un courtier Knative pour Apache Kafka, prêt pour la production

Pour les déploiements Knative Eventing prêts pour la production, Red Hat recommande d'utiliser l'implémentation du courtier Knative pour Apache Kafka. Le courtier est une implémentation native d'Apache Kafka du courtier Knative, qui envoie les CloudEvents directement à l'instance Kafka.

Le courtier Knative dispose d'une intégration native avec Kafka pour le stockage et l'acheminement des événements. Cela permet une meilleure intégration avec Kafka pour le modèle de courtier et de déclencheur par rapport à d'autres types de courtiers, et réduit les sauts de réseau. Les autres avantages de la mise en œuvre du courtier Knative sont les suivants :

- Garanties de livraison au moins une fois
- Livraison ordonnée des événements, basée sur l'extension de partitionnement CloudEvents
- Haute disponibilité du plan de contrôle
- Un plan de données extensible horizontalement

L'implémentation du courtier Knative pour Apache Kafka stocke les CloudEvents entrants en tant qu'enregistrements Kafka, en utilisant le mode de contenu binaire. Cela signifie que tous les attributs et extensions de CloudEvent sont mappés en tant qu'en-têtes sur l'enregistrement Kafka, tandis que la spécification **data** du CloudEvent correspond à la valeur de l'enregistrement Kafka.

5.4.3. Création de courtiers

Knative fournit une implémentation par défaut d'un courtier basé sur un canal. Ce courtier peut être utilisé à des fins de développement et de test, mais il n'offre pas de garanties suffisantes en matière de livraison d'événements pour les environnements de production.

Si un administrateur de cluster a configuré votre déploiement OpenShift Serverless pour utiliser Apache Kafka comme type de courtier par défaut, la création d'un courtier en utilisant les paramètres par défaut crée un courtier Knative pour Apache Kafka.

Si votre déploiement OpenShift Serverless n'est pas configuré pour utiliser le courtier Knative pour Apache Kafka comme type de courtier par défaut, le courtier basé sur les canaux est créé lorsque vous utilisez les paramètres par défaut dans les procédures suivantes.

5.4.3.1. Créer un courtier en utilisant le CLI Knative

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. L'utilisation de la CLI Knative (**kn**) pour créer des courtiers offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn broker create** pour créer un courtier.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un courtier :

```
$ kn broker create <nom_du_courtier>
```

Vérification

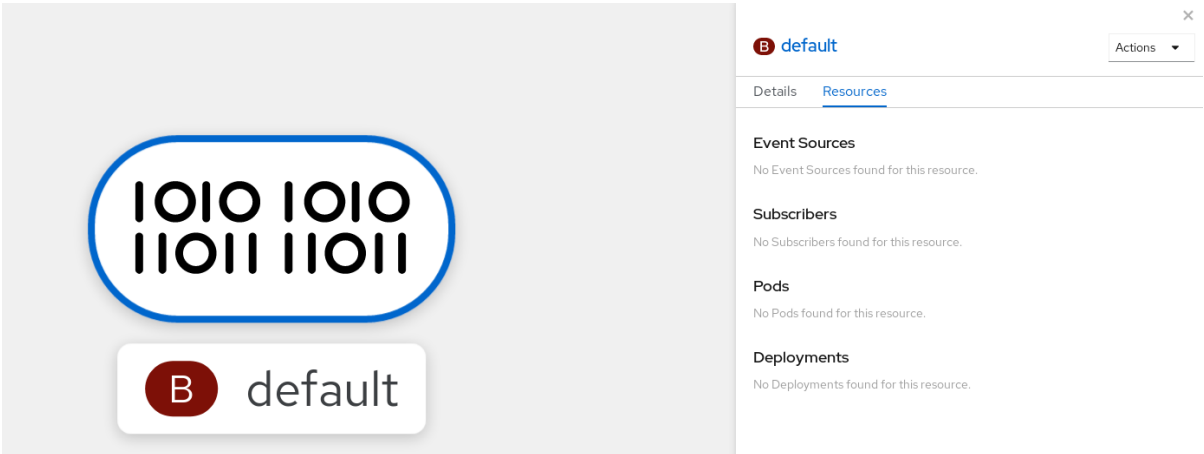
1. La commande **kn** permet d'obtenir la liste de tous les courtiers existants :

```
$ kn broker list
```

Exemple de sortie

```
NAME      URL                                                    AGE   CONDITIONS   READY
REASON
default   http://broker-ingress.knative-eventing.svc.cluster.local/test/default   45s   5 OK / 5
True
```

2. Facultatif : Si vous utilisez la console web de OpenShift Container Platform, vous pouvez naviguer vers la vue **Topology** dans la perspective **Developer**, et observer que le courtier existe :



5.4.3.2. Création d'un courtier par l'annotation d'un déclencheur

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Vous pouvez créer un broker en ajoutant l'annotation **eventing.knative.dev/injection: enabled** à un objet **Trigger**.

IMPORTANT

Si vous créez un courtier en utilisant l'annotation **eventing.knative.dev/injection: enabled**, vous ne pouvez pas supprimer ce courtier sans les autorisations de l'administrateur de la grappe. Si vous supprimez le courtier sans qu'un administrateur de cluster ait d'abord supprimé cette annotation, le courtier est à nouveau créé après la suppression.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créez un objet **Trigger** sous la forme d'un fichier YAML comportant l'annotation **eventing.knative.dev/injection: enabled**:

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  annotations:
    eventing.knative.dev/injection: enabled
  name: <trigger_name>
spec:
  broker: default
  subscriber: 1
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: <service_name>
```

1 Spécifiez les détails du puits d'événements, ou *subscriber*, vers lequel le déclencheur envoie des événements.

2. Appliquer le fichier YAML **Trigger**:

```
$ oc apply -f <filename>
```

Vérification

Vous pouvez vérifier que le courtier a été créé avec succès en utilisant le CLI **oc**, ou en l'observant dans la vue **Topology** de la console web.

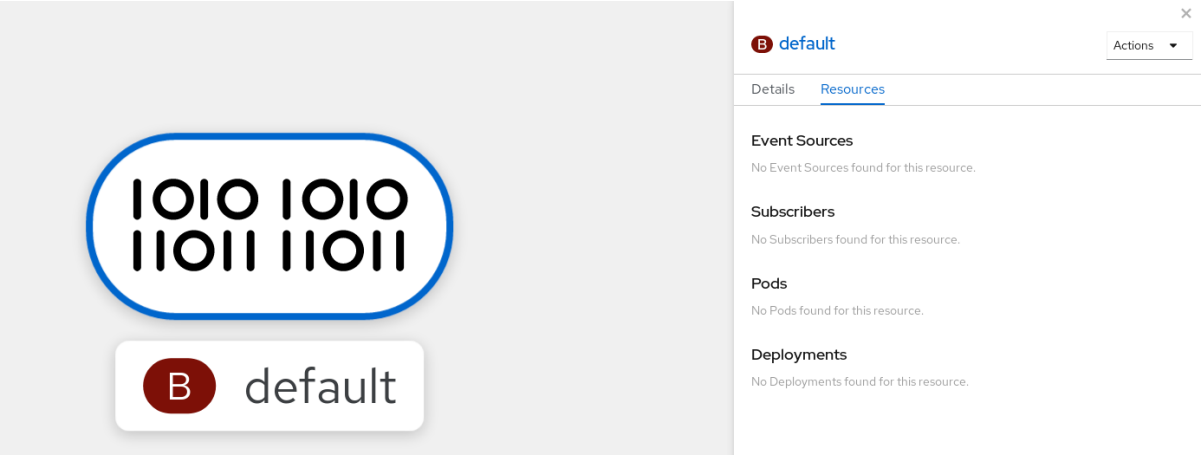
1. Entrez la commande suivante **oc** pour obtenir le courtier :

```
oc -n <namespace> get broker default
```

Exemple de sortie

```
NAME      READY    REASON    URL                                                                AGE
default   True               http://broker-ingress.knative-eventing.svc.cluster.local/test/default   3m56s
```

2. Facultatif : Si vous utilisez la console web de OpenShift Container Platform, vous pouvez naviguer vers la vue **Topology** dans la perspective **Developer**, et observer que le courtier existe :



5.4.3.3. Créer un courtier en étiquetant un espace de noms

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Vous pouvez créer automatiquement le courtier **default** en étiquetant un espace de noms dont vous êtes le propriétaire ou pour lequel vous disposez de droits d'écriture.

NOTE

Les courtiers créés à l'aide de cette méthode ne sont pas supprimés si vous retirez l'étiquette. Vous devez les supprimer manuellement.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Étiqueter un espace de noms avec **eventing.knative.dev/injection=enabled**:

```
oc label namespace <namespace> eventing.knative.dev/injection=enabled
```

Vérification

Vous pouvez vérifier que le courtier a été créé avec succès en utilisant le CLI **oc**, ou en l'observant dans la vue **Topology** de la console web.

1. Utilisez la commande **oc** pour obtenir le courtier :

```
oc -n <namespace> get broker <broker_name>
```

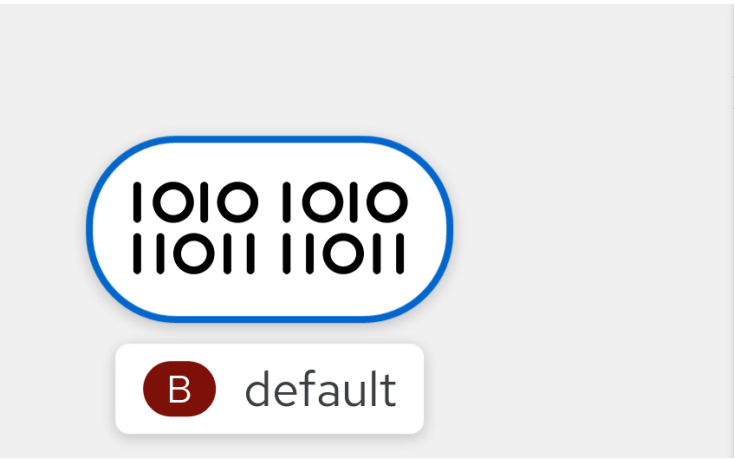
Example command

```
$ oc -n default get broker default
```

Exemple de sortie

```
NAME      READY     REASON    URL                                                     AGE
default   True                http://broker-ingress.knative-eventing.svc.cluster.local/test/default   3m56s
```

2. Facultatif : Si vous utilisez la console web de OpenShift Container Platform, vous pouvez naviguer vers la vue **Topology** dans la perspective **Developer**, et observer que le courtier existe :



5.4.3.4. Suppression d'un courtier créé par injection

Si vous créez un courtier par injection et que vous souhaitez ensuite le supprimer, vous devez le faire manuellement. Les courtiers créés à l'aide d'une étiquette d'espace de noms ou d'une annotation de déclenchement ne sont pas supprimés définitivement si vous supprimez l'étiquette ou l'annotation.

Conditions préalables

- Installez le CLI OpenShift (**oc**).

Procédure

1. Supprimer l'étiquette **eventing.knative.dev/injection=enabled** de l'espace de noms :

```
oc label namespace <namespace> eventing.knative.dev/injection-
```

La suppression de l'annotation empêche Knative de recréer le courtier après l'avoir supprimé.

2. Supprime le courtier de l'espace de noms sélectionné :

```
oc -n <namespace> delete broker <broker_name>
```

Vérification

- Utilisez la commande **oc** pour obtenir le courtier :

```
oc -n <namespace> get broker <broker_name>
```

Example command

```
$ oc -n default get broker default
```

Exemple de sortie

```
No resources found.
Error from server (NotFound): brokers.eventing.knative.dev "default" not found
```

5.4.3.5. Création d'un courtier à l'aide de la console web

Une fois Knative Eventing installé sur votre cluster, vous pouvez créer un broker en utilisant la console web. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un broker.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

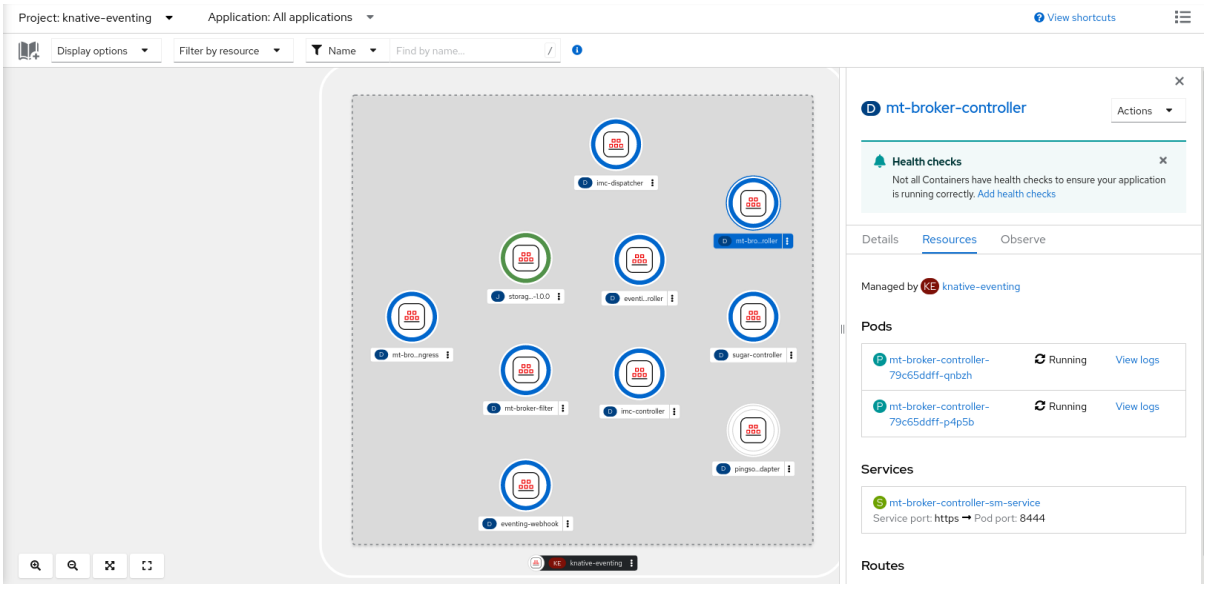
Procédure

1. Dans la perspective **Developer**, naviguez vers **Add → Broker**. La page **Broker** s'affiche.
2. Facultatif. Mettez à jour le site **Name** du courtier. Si vous ne mettez pas à jour le nom, le courtier généré s'appelle **default**.
3. Cliquez sur **Create**.

Vérification

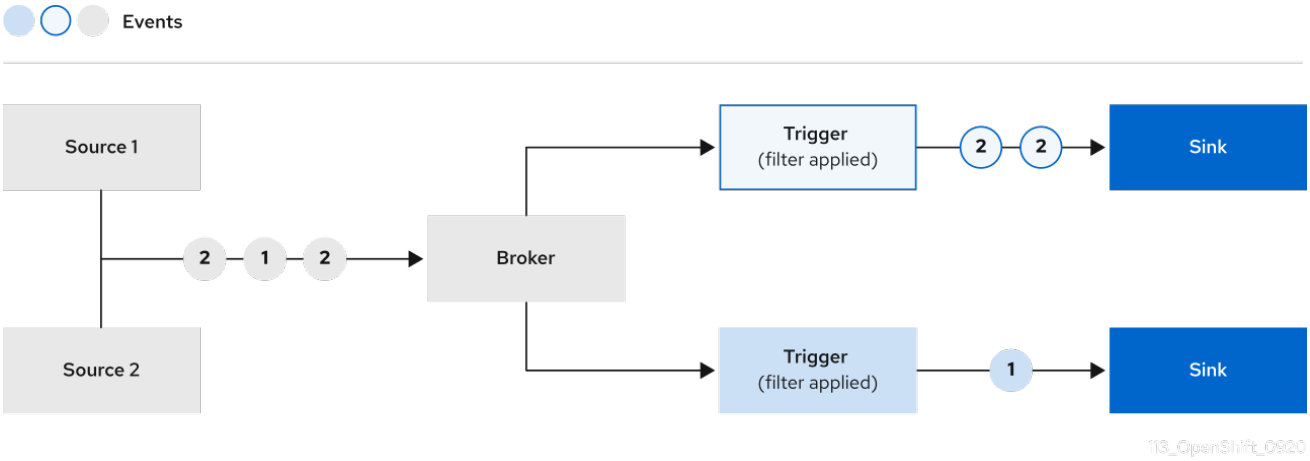
Vous pouvez vérifier que le courtier a été créé en affichant les composants du courtier dans la page **Topology**.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.
2. Consultez les composants **mt-broker-ingress**, **mt-broker-filter**, et **mt-broker-controller**.



5.4.3.6. Création d'un courtier en utilisant la perspective de l'administrateur

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Les événements sont envoyés d'une source d'événements à un broker sous la forme d'une requête HTTP **POST**. Une fois que les événements sont entrés dans le courtier, ils peuvent être filtrés par les attributs CloudEvent à l'aide de déclencheurs, puis envoyés sous la forme d'une requête HTTP **POST** à un puits d'événements.



Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans la liste **Create**, sélectionnez **Broker**. Vous serez dirigé vers la page **Create Broker**.

3. En option : Modifier la configuration YAML du courtier.
4. Cliquez sur **Create**.

5.4.3.7. Prochaines étapes

- Configurez les paramètres de livraison d'événements qui sont appliqués dans les cas où un événement n'est pas livré à un récepteur d'événements. Voir Exemples de configuration des paramètres de livraison d'événements.

5.4.3.8. Ressources supplémentaires

- Configuration de la classe de courtier par défaut
- DéclencheursSources d'événements
- Livraison de l'événement

5.4.4. Configuration du canal d'accompagnement du courtier par défaut

Si vous utilisez un courtier basé sur des canaux, vous pouvez définir le type de canal de soutien par défaut pour le courtier sur **InMemoryChannel** ou **KafkaChannel**.

Conditions préalables

- Vous disposez de droits d'administrateur sur OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Eventing sur votre cluster.
- Vous avez installé le CLI OpenShift (**oc**).
- Si vous souhaitez utiliser les canaux Apache Kafka comme type de canal de sauvegarde par défaut, vous devez également installer le CR **KnativeKafka** sur votre cluster.

Procédure

1. Modifier la ressource personnalisée (CR) **KnativeEventing** pour ajouter des détails de configuration pour la carte de configuration **config-br-default-channel**:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config: (1)
    config-br-default-channel:
      channel-template-spec: |
        apiVersion: messaging.knative.dev/v1beta1
        kind: KafkaChannel (2)
        spec:
          numPartitions: 6 (3)
          replicationFactor: 3 (4)
```

1. Dans **spec.config**, vous pouvez spécifier les cartes de configuration pour lesquelles vous souhaitez ajouter des configurations modifiées.
2. La configuration du type de canal de sauvegarde par défaut. Dans cet exemple, l'implémentation du canal par défaut pour le cluster est **KafkaChannel**.
3. Le nombre de partitions pour le canal Kafka qui soutient le courtier.
4. Le facteur de réplication pour le canal Kafka qui soutient le courtier.

2. Appliquer la mise à jour de **KnativeEventing** CR :

```
$ oc apply -f <filename>
```

5.4.5. Configuration de la classe de courtier par défaut

Vous pouvez utiliser la carte de configuration **config-br-defaults** pour spécifier les paramètres de la classe de courtier par défaut pour Knative Eventing. Vous pouvez spécifier la classe de courtier par défaut pour l'ensemble du cluster ou pour un ou plusieurs espaces de noms. Actuellement, les types de courtiers **MTChannelBasedBroker** et **Kafka** sont pris en charge.

Conditions préalables

- Vous disposez de droits d'administrateur sur OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Eventing sur votre cluster.
- Si vous souhaitez utiliser le courtier Knative pour Apache Kafka comme courtier par défaut, vous devez également installer le CR **KnativeKafka** sur votre cluster.

Procédure

- Modifier la ressource personnalisée **KnativeEventing** pour ajouter des détails de configuration pour la carte de configuration **config-br-defaults**:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  defaultBrokerClass: Kafka 1
  config: 2
    config-br-defaults: 3
      default-br-config: |
        clusterDefault: 4
          brokerClass: Kafka
          apiVersion: v1
          kind: ConfigMap
          name: kafka-broker-config 5
          namespace: knative-eventing 6
        namespaceDefaults: 7
          my-namespace:
```

```
        brokerClass: MTChannelBasedBroker
        apiVersion: v1
        kind: ConfigMap
        name: config-br-default-channel (8)
        namespace: knative-eventing (9)
...
```

1. La classe de courtier par défaut pour Knative Eventing.
2. Dans **spec.config**, vous pouvez spécifier les cartes de configuration pour lesquelles vous souhaitez ajouter des configurations modifiées.
3. La carte de configuration **config-br-defaults** spécifie les paramètres par défaut pour tout courtier qui ne spécifie pas de paramètres **spec.config** ou de classe de courtier.
4. La configuration de la classe de courtage par défaut à l'échelle du cluster. Dans cet exemple, l'implémentation de la classe de courtage par défaut pour le cluster est **Kafka**.
5. La carte de configuration **kafka-broker-config** spécifie les paramètres par défaut du courtier Kafka. Voir "Configurer le courtier Knative pour les paramètres Apache Kafka" dans la section "Ressources supplémentaires".
6. L'espace de noms dans lequel la carte de configuration **kafka-broker-config** existe.
7. La configuration de la classe de courtage par défaut de l'espace de noms. Dans cet exemple, l'implémentation de la classe de courtage par défaut pour l'espace de noms **my-namespace** est **MTChannelBasedBroker**. Vous pouvez spécifier des implémentations de classe de courtage par défaut pour plusieurs espaces de noms.
8. La carte de configuration **config-br-default-channel** spécifie le canal d'appui par défaut du courtier. Voir "Configurer le canal d'appui par défaut du courtier" dans la section "Ressources supplémentaires".
9. L'espace de noms dans lequel la carte de configuration **config-br-default-channel** existe.

IMPORTANT

La configuration d'une valeur par défaut spécifique à l'espace de nommage est prioritaire sur les paramètres applicables à l'ensemble du cluster.

5.4.6. Mise en œuvre d'un courtier Knative pour Apache Kafka

Pour les déploiements Knative Eventing prêts pour la production, Red Hat recommande d'utiliser l'implémentation du courtier Knative pour Apache Kafka. Le courtier est une implémentation native d'Apache Kafka du courtier Knative, qui envoie les CloudEvents directement à l'instance Kafka.

Le courtier Knative dispose d'une intégration native avec Kafka pour le stockage et l'acheminement des événements. Cela permet une meilleure intégration avec Kafka pour le modèle de courtier et de déclencheur par rapport à d'autres types de courtiers, et réduit les sauts de réseau. Les autres avantages de la mise en œuvre du courtier Knative sont les suivants :

- Garanties de livraison au moins une fois
- Livraison ordonnée des événements, basée sur l'extension de partitionnement CloudEvents

- Haute disponibilité du plan de contrôle
- Un plan de données extensible horizontalement

L'implémentation du courtier Knative pour Apache Kafka stocke les CloudEvents entrants en tant qu'enregistrements Kafka, en utilisant le mode de contenu binaire. Cela signifie que tous les attributs et extensions de CloudEvent sont mappés en tant qu'en-têtes sur l'enregistrement Kafka, tandis que la spécification **data** du CloudEvent correspond à la valeur de l'enregistrement Kafka.

5.4.6.1. Création d'un courtier Apache Kafka lorsqu'il n'est pas configuré comme type de courtier par défaut

Si votre déploiement OpenShift Serverless n'est pas configuré pour utiliser le courtier Kafka comme type de courtier par défaut, vous pouvez utiliser l'une des procédures suivantes pour créer un courtier basé sur Kafka.

5.4.6.1.1. Créer un courtier Apache Kafka en utilisant YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui permet de décrire des applications de manière déclarative et reproductible. Pour créer un courtier Kafka à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **Broker**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).

Procédure

1. Créer un courtier basé sur Kafka sous la forme d'un fichier YAML :

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: Kafka 1
  name: example-kafka-broker
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: kafka-broker-config 2
    namespace: knative-eventing
```

1. La classe du courtier. Si elle n'est pas spécifiée, les courtiers utilisent la classe par défaut, telle qu'elle est configurée par les administrateurs du cluster. Pour utiliser le courtier Kafka, cette valeur doit être **Kafka**.
2. La carte de configuration par défaut des courtiers Knative pour Apache Kafka. Cette carte

2. Appliquer le fichier YAML du courtier basé sur Kafka :

```
$ oc apply -f <filename>
```

5.4.6.1.2. Création d'un courtier Apache Kafka qui utilise un sujet Kafka géré en externe

Si vous souhaitez utiliser un courtier Kafka sans lui permettre de créer son propre sujet interne, vous pouvez utiliser un sujet Kafka géré de manière externe. Pour ce faire, vous devez créer un objet Kafka **Broker** qui utilise l'annotation **kafka.eventing.knative.dev/external.topic**.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster OpenShift Container Platform.
- Vous avez accès à une instance Kafka telle que Red Hat AMQ Streams , et avez créé un sujet Kafka.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).

Procédure

1. Créer un courtier basé sur Kafka sous la forme d'un fichier YAML :

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: Kafka (1)
    kafka.eventing.knative.dev/external.topic: <topic_name> (2)
...
```

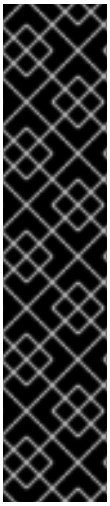
(1) La classe du courtier. Si elle n'est pas spécifiée, les courtiers utilisent la classe par défaut, telle qu'elle est configurée par les administrateurs du cluster. Pour utiliser le courtier Kafka, cette valeur doit être **Kafka**.

(2) Le nom du sujet Kafka que vous souhaitez utiliser.

2. Appliquer le fichier YAML du courtier basé sur Kafka :

```
$ oc apply -f <filename>
```

5.4.6.1.3. Mise en œuvre d'un courtier Knative pour Apache Kafka avec un plan de données isolé



IMPORTANT

L'implémentation de Knative Broker pour Apache Kafka avec un plan de données isolé est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas leur utilisation en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

L'implémentation de Knative Broker pour Apache Kafka comporte 2 plans :

Control plane

Il s'agit de contrôleurs qui communiquent avec l'API Kubernetes, surveillent les objets personnalisés et gèrent le plan de données.

Plan de données

L'ensemble des composants qui écoutent les événements entrants, communiquent avec Apache Kafka et envoient des événements aux puits d'événements. L'implémentation Knative Broker pour le plan de données Apache Kafka est l'endroit où les événements circulent. L'implémentation consiste en des déploiements **kafka-broker-receiver** et **kafka-broker-dispatcher**.

Lorsque vous configurez une classe Broker de **Kafka**, l'implémentation Knative Broker pour Apache Kafka utilise un plan de données partagé. Cela signifie que les déploiements **kafka-broker-receiver** et **kafka-broker-dispatcher** dans l'espace de noms **knative-eventing** sont utilisés pour tous les Brokers Apache Kafka dans le cluster.

Cependant, lorsque vous configurez une classe Broker de **KafkaNamespaced**, le contrôleur de courtier Apache Kafka crée un nouveau plan de données pour chaque espace de noms dans lequel un courtier existe. Ce plan de données est utilisé par tous les courtiers **KafkaNamespaced** dans cet espace de noms. Cela permet d'isoler les plans de données, de sorte que les déploiements **kafka-broker-receiver** et **kafka-broker-dispatcher** dans l'espace de noms de l'utilisateur ne sont utilisés que pour le courtier de cet espace de noms.

IMPORTANT

En raison de la séparation des plans de données, cette fonction de sécurité crée plus de déploiements et utilise plus de ressources. À moins que vous n'ayez de telles exigences en matière d'isolation, utilisez un **regular** Broker avec une classe de **Kafka**.

5.4.6.1.4. Créer un courtier Knative pour Apache Kafka qui utilise un plan de données isolé

IMPORTANT

L'implémentation de Knative Broker pour Apache Kafka avec un plan de données isolé est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas leur utilisation en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Pour créer un courtier **KafkaNamespaced**, vous devez définir l'annotation **eventing.knative.dev/broker.class** en **KafkaNamespaced**.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster OpenShift Container Platform.
- Vous avez accès à une instance Apache Kafka, telle que Red Hat AMQ Streams , et avez créé un sujet Kafka.
- Vous avez créé un projet, ou avez accès à un projet, avec les rôles et permissions appropriés pour créer des applications et autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).

Procédure

1. Créer un courtier basé sur Apache Kafka en utilisant un fichier YAML :

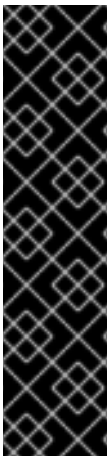
```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  annotations:
    eventing.knative.dev/broker.class: KafkaNamespaced 1
  name: default
  namespace: my-namespace 2
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: my-config 3
...
```

1 Pour utiliser le courtier Apache Kafka avec des plans de données isolés, la valeur de la classe de courtier doit être **KafkaNamespaced**.

2 3 L'objet **ConfigMap** référencé **my-config** doit se trouver dans le même espace de noms que l'objet **Broker**, en l'occurrence **my-namespace**.

2. Appliquer le fichier YAML du courtier basé sur Apache Kafka :

```
$ oc apply -f <filename>
```



IMPORTANT

L'objet **ConfigMap** dans **spec.config** doit être dans le même espace de noms que l'objet **Broker**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: my-namespace
data:
  ...
```

Après la création du premier objet **Broker** avec la classe **KafkaNamespaced**, les déploiements **kafka-broker-receiver** et **kafka-broker-dispatcher** sont créés dans l'espace de noms. Par la suite, tous les courtiers de la classe **KafkaNamespaced** dans le même espace de noms utiliseront le même plan de données. Si aucun courtier de la classe **KafkaNamespaced** n'existe dans l'espace de noms, le plan de données de l'espace de noms est supprimé.

5.4.6.2. Configuration des paramètres du courtier Apache Kafka

Vous pouvez configurer le facteur de réplication, les serveurs d'amorçage et le nombre de partitions de sujets pour un courtier Kafka, en créant une carte de configuration et en y faisant référence dans l'objet Kafka **Broker**.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** custom resource (CR) sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet qui a les rôles et les autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).

Procédure

1. Modifiez la carte de configuration **kafka-broker-config** ou créez votre propre carte de configuration qui contient la configuration suivante :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: <config_map_name> 1
  namespace: <namespace> 2
data:
```

```
default.topic.partitions: <integer> 3
default.topic.replication.factor: <integer> 4
bootstrap.servers: <list_of_servers> 5
```

1. Le nom de la carte de configuration.
2. L'espace de noms dans lequel la carte de configuration existe.
3. Le nombre de partitions de sujets pour le courtier Kafka. Cela permet de contrôler la vitesse à laquelle les événements peuvent être envoyés au courtier. Un nombre plus élevé de partitions nécessite davantage de ressources informatiques.
4. Le facteur de réplication des messages thématiques. Cela permet d'éviter les pertes de données. Un facteur de réplication plus élevé nécessite davantage de ressources informatiques et de stockage.
5. Une liste de serveurs d'amorçage séparés par des virgules. Cela peut être à l'intérieur ou à l'extérieur du cluster OpenShift Container Platform, et c'est une liste de clusters Kafka que le courtier reçoit des événements de et envoie des événements à.

IMPORTANT

La valeur **default.topic.replication.factor** doit être inférieure ou égale au nombre d'instances de courtiers Kafka dans votre cluster. Par exemple, si vous n'avez qu'un seul courtier Kafka, la valeur **default.topic.replication.factor** ne doit pas être supérieure à **"1"**.

Exemple de carte de configuration du courtier Kafka

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kafka-broker-config
  namespace: knative-eventing
data:
  default.topic.partitions: "10"
  default.topic.replication.factor: "3"
  bootstrap.servers: "my-cluster-kafka-bootstrap.kafka:9092"
```

2. Appliquer la carte de configuration :

```
oc apply -f <config_map_filename> $ oc apply -f <config_map_filename>
```

3. Spécifiez la carte de configuration pour l'objet Kafka **Broker**:

Exemple d'objet Broker

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
  name: <broker_name> 1
  namespace: <namespace> 2
  annotations:
```

```
    eventing.knative.dev/broker.class: Kafka 3
spec:
  config:
    apiVersion: v1
    kind: ConfigMap
    name: <config_map_name> 4
    namespace: <namespace> 5
...
```

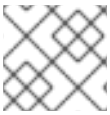
1. Le nom du courtier.
2. L'espace de noms dans lequel le courtier existe.
3. L'annotation de la classe du courtier. Dans cet exemple, le courtier est un courtier Kafka qui utilise la valeur de classe **Kafka**.
4. Le nom de la carte de configuration.
5. L'espace de noms dans lequel la carte de configuration existe.

4. Appliquer le courtier :

```
oc apply -f <broker_filename>
```

5.4.6.3. Configuration de la sécurité pour l'implémentation du courtier Knative pour Apache Kafka

Les clusters Kafka sont généralement sécurisés en utilisant les méthodes d'authentification TLS ou SASL. Vous pouvez configurer un courtier ou un canal Kafka pour travailler avec un cluster Red Hat AMQ Streams protégé en utilisant TLS ou SASL.



NOTE

Red Hat recommande d'activer à la fois SASL et TLS.

5.4.6.3.1. Configuration de l'authentification TLS pour les brokers Apache Kafka

Transport Layer Security (TLS) est utilisé par les clients et les serveurs Apache Kafka pour chiffrer le trafic entre Knative et Kafka, ainsi que pour l'authentification. TLS est la seule méthode de cryptage du trafic prise en charge par l'implémentation du courtier Knative pour Apache Kafka.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** CR sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous disposez d'un certificat d'autorité de certification pour le cluster Kafka stocké dans un fichier **.pem**.
- Vous disposez d'un certificat client pour le cluster Kafka et d'une clé stockée sous forme de fichiers **.pem**.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez les fichiers de certificat en tant que secret dans l'espace de noms **knative-eventing**:

```
$ oc create secret -n knative-eventing generic <secret_name> \
  --from-literal=protocol=SSL \
  --from-file=ca.crt=caroot.pem \
  --from-file=user.crt=certificate.pem \
  --from-file=user.key=key.pem
```

IMPORTANT

Utilisez les noms de clés **ca.crt**, **user.crt**, et **user.key**. Ne les modifiez pas.

2. Modifiez le CR **KnativeKafka** et ajoutez une référence à votre secret dans la spécification **broker**:

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  broker:
    enabled: true
    defaultConfig:
      authSecretName: <secret_name>
...
```

5.4.6.3.2. Configuration de l'authentification SASL pour les brokers Apache Kafka

Simple Authentication and Security Layer (SASL) est utilisé par Apache Kafka pour l'authentification. Si vous utilisez l'authentification SASL sur votre cluster, les utilisateurs doivent fournir des informations d'identification à Knative pour communiquer avec le cluster Kafka ; sinon, les événements ne peuvent pas être produits ou consommés.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** CR sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Vous disposez d'un nom d'utilisateur et d'un mot de passe pour un cluster Kafka.
- Vous avez choisi le mécanisme SASL à utiliser, par exemple **PLAIN**, **SCRAM-SHA-256**, ou **SCRAM-SHA-512**.
- Si TLS est activé, vous avez également besoin du fichier de certificat **ca.crt** pour le cluster Kafka.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez les fichiers de certificat en tant que secret dans l'espace de noms **knative-eventing**:

```
$ oc create secret -n knative-eventing generic <secret_name> \
  --from-literal=protocol=SASL_SSL \
  --from-literal=sasl.mechanism=<sasl_mechanism> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=user="my-sasl-user"
```

   - Utilisez les noms de clés **ca.crt**, **password**, et **sasl.mechanism**. Ne les modifiez pas.
   - Si vous souhaitez utiliser SASL avec des certificats d'autorité de certification publique, vous devez utiliser l'indicateur **tls.enabled=true**, plutôt que l'argument **ca.crt**, lors de la création du secret. Par exemple :

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. Modifiez le CR **KnativeKafka** et ajoutez une référence à votre secret dans la spécification **broker**:

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  broker:
    enabled: true
    defaultConfig:
      authSecretName: <secret_name>
...
```

5.4.6.4. Ressources supplémentaires

- Documentation de Red Hat AMQ Streams
- TLS et SASL sur Kafka

5.4.7. Gestion des courtiers

Le CLI de Knative (**kn**) fournit des commandes qui peuvent être utilisées pour décrire et répertorier les courtiers existants.

5.4.7.1. Lister les courtiers existants en utilisant le CLI Knative

L'utilisation de la CLI Knative (**kn**) pour lister les courtiers offre une interface utilisateur simplifiée et intuitive. Vous pouvez utiliser la commande **kn broker list** pour lister les courtiers existants dans votre cluster en utilisant la CLI Knative.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Dressez la liste de tous les courtiers existants :

```
$ kn broker list
```

Exemple de sortie

```
NAME      URL                                                                AGE   CONDITIONS   READY
REASON
default   http://broker-ingress.knative-eventing.svc.cluster.local/test/default   45s   5 OK / 5
True
```

5.4.7.2. Décrire un courtier existant en utilisant le CLI Knative

L'utilisation de la CLI Knative (**kn**) pour décrire les courtiers offre une interface utilisateur simplifiée et intuitive. Vous pouvez utiliser la commande **kn broker describe** pour imprimer des informations sur les courtiers existants dans votre cluster en utilisant la CLI Knative.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Décrire un courtier existant :

```
$ kn broker describe <nom_du_courtier>
```

Exemple de commande utilisant le courtier par défaut

```
$ kn broker describe default
```

Exemple de sortie

```
Name:         default
Namespace:    default
Annotations:  eventing.knative.dev/broker.class=MTChannelBasedBroker,
eventing.knative.dev/creato ...
Age:          22s

Address:
  URL:    http://broker-ingress.knative-eventing.svc.cluster.local/default/default

Conditions:
  OK TYPE                  AGE REASON
  ++ Ready                 22s
  ++ Addressable           22s
  ++ FilterReady           22s
  ++ IngressReady          22s
  ++ TriggerChannelReady   22s
```

5.4.8. Livraison de l'événement

Vous pouvez configurer les paramètres de distribution des événements qui sont appliqués dans les cas où un événement ne parvient pas à être distribué à un récepteur d'événements. La configuration des paramètres d'acheminement des événements, y compris d'un puits de lettres mortes, garantit que les événements qui ne sont pas acheminés vers un puits d'événements sont réessayés. Dans le cas contraire, les événements non livrés sont abandonnés.

5.4.8.1. Modèles de comportement de livraison d'événements pour les canaux et les courtiers

Les différents types de canaux et de courtiers ont leurs propres modèles de comportement qui sont suivis pour la livraison des événements.

5.4.8.1.1. Canaux et courtiers Knative pour Apache Kafka

Si un événement est transmis avec succès à un canal Kafka ou à un récepteur de courtier, le récepteur répond par un code d'état **202**, ce qui signifie que l'événement a été stocké en toute sécurité dans un sujet Kafka et qu'il n'est pas perdu.

Si le récepteur répond par un autre code d'état, l'événement n'est pas stocké en toute sécurité et l'utilisateur doit prendre des mesures pour résoudre le problème.

5.4.8.2. Paramètres de livraison d'événements configurables

Les paramètres suivants peuvent être configurés pour l'envoi d'événements :

Lettre morte pour l'évier

Vous pouvez configurer le paramètre de livraison **deadLetterSink** de sorte que si un événement n'est pas livré, il soit stocké dans le puits d'événements spécifié. Les événements non livrés qui ne sont pas stockés dans un puits de lettres mortes sont abandonnés. Le puits d'événements peut être n'importe quel objet adressable conforme au contrat de puits d'événements Knative, tel qu'un service Knative, un service Kubernetes ou un URI.

Tentatives

Vous pouvez définir un nombre minimum de tentatives de livraison avant que l'événement ne soit envoyé au puits de lettres mortes, en configurant le paramètre de livraison **retry** avec une valeur entière.

Délai de recul

Vous pouvez définir le paramètre de livraison **backoffDelay** pour spécifier le délai avant qu'une nouvelle tentative de livraison d'un événement ne soit effectuée après un échec. La durée du paramètre **backoffDelay** est spécifiée au format ISO 8601. Par exemple, **PT1S** indique un délai de 1 seconde.

Reculer dans la politique

The **backoffPolicy** delivery parameter can be used to specify the retry back off policy. The policy can be specified as either **linear** or **exponential**. When using the **linear** back off policy, the back off delay is equal to **backoffDelay * <numberOfRetries>**. When using the **exponential** backoff policy, the back off delay is equal to **backoffDelay*2^<numberOfRetries>**.

5.4.8.3. Exemples de configuration des paramètres de distribution des événements

Vous pouvez configurer les paramètres de livraison d'événements pour les objets **Broker**, **Trigger**, **Channel** et **Subscription**. Si vous configurez des paramètres de distribution d'événements pour un courtier ou un canal, ces paramètres sont propagés aux déclencheurs ou aux abonnements créés pour ces objets. Vous pouvez également définir des paramètres de livraison d'événements pour les déclencheurs ou les abonnements afin de remplacer les paramètres du courtier ou du canal.

Exemple d'objet Broker

```
apiVersion: eventing.knative.dev/v1
kind: Broker
metadata:
...
spec:
  delivery:
    deadLetterSink:
      ref:
        apiVersion: eventing.knative.dev/v1alpha1
        kind: KafkaSink
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
...
```

Exemple d'objet Trigger

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
...
spec:
  broker: <broker_name>
  delivery:
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
```

```
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
...
```

Exemple d'objet Channel

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
...
spec:
  delivery:
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
...
```

Exemple d'objet Subscription

```
apiVersion: messaging.knative.dev/v1
kind: Subscription
metadata:
...
spec:
  channel:
    apiVersion: messaging.knative.dev/v1
    kind: Channel
    name: <channel_name>
  delivery:
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: <sink_name>
    backoffDelay: <duration>
    backoffPolicy: <policy_type>
    retry: <integer>
...
```

5.4.8.4. Configuration de l'ordre de livraison des événements pour les déclencheurs

Si vous utilisez un courtier Kafka, vous pouvez configurer l'ordre de livraison des événements des déclencheurs aux puits d'événements.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et Knative broker implementation for Apache Kafka sont installés sur votre cluster OpenShift Container Platform.
- L'utilisation du courtier Kafka est activée sur votre cluster et vous avez créé un courtier Kafka.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé le CLI OpenShift (**oc**).

Procédure

1. Créer ou modifier un objet **Trigger** et définir l'annotation **kafka.eventing.knative.dev/delivery.order**:

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name>
  annotations:
    kafka.eventing.knative.dev/delivery.order: ordered
...
```

Les garanties de livraison aux consommateurs prises en charge sont les suivantes :

unordered

Un consommateur non ordonné est un consommateur non bloquant qui délivre des messages non ordonnés, tout en préservant une bonne gestion des décalages.

ordered

Un consommateur ordonné est un consommateur bloquant par partition qui attend une réponse positive de l'abonné CloudEvent avant de délivrer le message suivant de la partition. La garantie de commande par défaut est **unordered**.

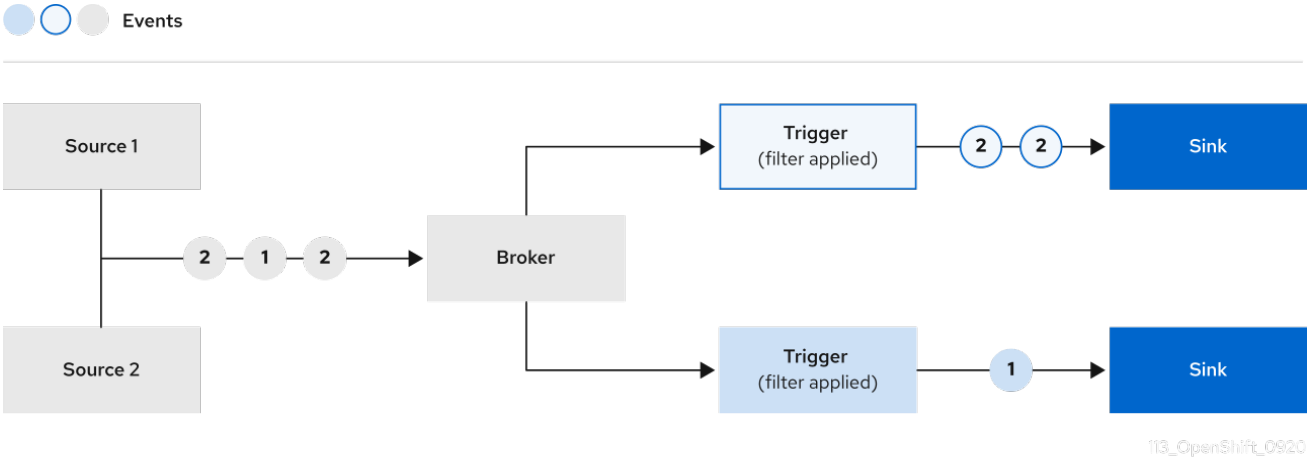
2. Appliquer l'objet **Trigger**:

```
$ oc apply -f <filename>
```

5.5. DÉCLENCHEURS

5.5.1. Vue d'ensemble des déclencheurs

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Les événements sont envoyés d'une source d'événements à un broker sous la forme d'une requête HTTP **POST**. Une fois que les événements sont entrés dans le courtier, ils peuvent être filtrés par les attributs CloudEvent à l'aide de déclencheurs, puis envoyés sous la forme d'une requête HTTP **POST** à un puits d'événements.



Si vous utilisez un courtier Knative pour Apache Kafka, vous pouvez configurer l'ordre de livraison des événements des déclencheurs aux puits d'événements. Voir Configuration de l'ordre de livraison des événements pour les déclencheurs.

5.5.1.1. Configuration de l'ordre de livraison des événements pour les déclencheurs

Si vous utilisez un courtier Kafka, vous pouvez configurer l'ordre de livraison des événements des déclencheurs aux puits d'événements.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing et Knative broker implementation for Apache Kafka sont installés sur votre cluster OpenShift Container Platform.
- L'utilisation du courtier Kafka est activée sur votre cluster et vous avez créé un courtier Kafka.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé le CLI OpenShift (**oc**).

Procédure

1. Créer ou modifier un objet **Trigger** et définir l'annotation **kafka.eventing.knative.dev/delivery.order**:

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name>
  annotations:
    kafka.eventing.knative.dev/delivery.order: ordered
...
```

Les garanties de livraison aux consommateurs prises en charge sont les suivantes :

unordered

Un consommateur non ordonné est un consommateur non bloquant qui délivre des messages non ordonnés, tout en préservant une bonne gestion des décalages.

ordered

Un consommateur ordonné est un consommateur bloquant par partition qui attend une réponse positive de l'abonné CloudEvent avant de délivrer le message suivant de la partition. La garantie de commande par défaut est **unordered**.

2. Appliquer l'objet **Trigger**:

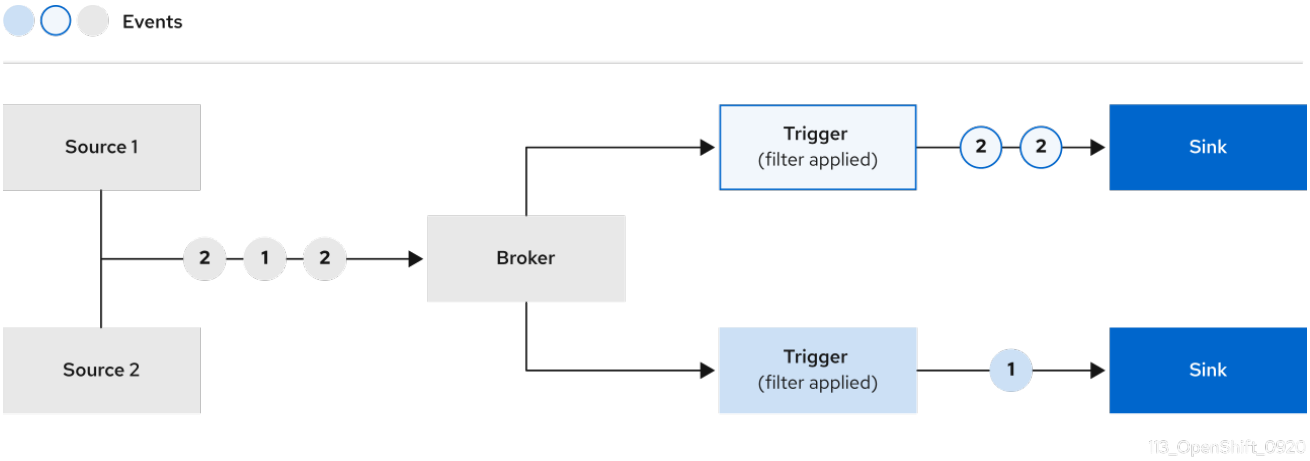
```
$ oc apply -f <filename>
```

5.5.1.2. Prochaines étapes

- Configurez les paramètres de livraison d'événements qui sont appliqués dans les cas où un événement n'est pas livré à un récepteur d'événements. Voir Exemples de configuration des paramètres de livraison d'événements.

5.5.2. Création de déclencheurs

Les courtiers peuvent être utilisés en combinaison avec des déclencheurs pour transmettre des événements d'une source d'événements à un puits d'événements. Les événements sont envoyés d'une source d'événements à un broker sous la forme d'une requête HTTP **POST**. Une fois que les événements sont entrés dans le courtier, ils peuvent être filtrés par les attributs CloudEvent à l'aide de déclencheurs, puis envoyés sous la forme d'une requête HTTP **POST** à un puits d'événements.



5.5.2.1. Création d'un déclencheur à l'aide de la perspective de l'administrateur

L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un déclencheur. Une fois que Knative Eventing est installé sur votre cluster et que vous avez créé un broker, vous pouvez créer un trigger en utilisant la console web.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.
- Vous avez créé un courtier Knative.

- Vous avez créé un service Knative à utiliser en tant qu'abonné.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans l'onglet **Broker**, sélectionnez le menu Options  pour le courtier auquel vous voulez ajouter un déclencheur.
3. Cliquez sur **Add Trigger** dans la liste.
4. Dans la boîte de dialogue **Add Trigger**, sélectionnez un **Subscriber** pour le déclencheur. L'abonné est le service Knative qui recevra les événements du courtier.
5. Cliquez sur **Add**.

5.5.2.2. Création d'un déclencheur à l'aide de la perspective du développeur

L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un déclencheur. Une fois que Knative Eventing est installé sur votre cluster et que vous avez créé un broker, vous pouvez créer un trigger en utilisant la console web.

Conditions préalables

- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez créé un courtier et un service Knative ou un autre puits d'événements pour vous connecter au déclencheur.

Procédure

1. Dans la perspective **Developer**, naviguez jusqu'à la page **Topology**.
2. Survolez le courtier pour lequel vous souhaitez créer un déclencheur et faites glisser la flèche. L'option **Add Trigger** s'affiche.
3. Cliquez sur **Add Trigger**.
4. Sélectionnez votre évier dans la liste **Subscriber**.
5. Cliquez sur **Add**.

Vérification

- Une fois l'abonnement créé, vous pouvez le visualiser sur la page **Topology**, où il est représenté par une ligne qui relie le courtier au puits d'événements.

Suppression d'un déclencheur

1. Dans la perspective **Developer**, naviguez jusqu'à la page **Topology**.
2. Cliquez sur le déclencheur que vous souhaitez supprimer.
3. Dans le menu contextuel de **Actions**, sélectionnez **Delete Trigger**.

5.5.2.3. Création d'un déclencheur à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn trigger create** pour créer un déclencheur.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un déclencheur :

```
$ kn trigger create <trigger_name> --broker <broker_name> --filter <key=value> --sink
<sink_name>
```

Vous pouvez également créer un déclencheur et créer simultanément le courtier **default** à l'aide de l'injection de courtier :

```
kn trigger create <trigger_name> --inject-broker --filter <key=value> --sink <sink_name>
```

Par défaut, les déclencheurs transmettent tous les événements envoyés à un courtier aux puits qui sont abonnés à ce courtier. L'utilisation de l'attribut **--filter** pour les déclencheurs vous permet de filtrer les événements d'un courtier, de sorte que les abonnés ne reçoivent qu'un sous-ensemble d'événements en fonction des critères que vous avez définis.

5.5.3. Liste des déclencheurs à partir de la ligne de commande

L'utilisation du CLI Knative (**kn**) pour lister les déclencheurs offre une interface utilisateur rationalisée et intuitive.

5.5.3.1. Lister les déclencheurs en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn trigger list** pour dresser la liste des déclencheurs existants dans votre cluster.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Imprimer une liste des déclencheurs disponibles :

```
$ kn trigger list
```

Exemple de sortie

```
NAME    BROKER    SINK          AGE   CONDITIONS   READY   REASON
email   default   ksvc:edisplay   4s    5 OK / 5     True
ping    default   ksvc:edisplay   32s   5 OK / 5     True
```

2. Facultatif : Imprimer une liste de déclencheurs au format JSON :

```
$ kn trigger list -o json
```

5.5.4. Décrire les déclencheurs à partir de la ligne de commande

L'utilisation du CLI Knative (**kn**) pour décrire les déclencheurs offre une interface utilisateur rationalisée et intuitive.

5.5.4.1. Décrire un déclencheur à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn trigger describe** pour imprimer des informations sur les déclencheurs existants dans votre cluster en utilisant le CLI Knative.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un déclencheur.

Procédure

- Entrez la commande :

```
$ kn trigger describe <trigger_name>
```

Exemple de sortie

```
Name:         ping
Namespace:    default
Labels:       eventing.knative.dev/broker=default
Annotations:  eventing.knative.dev/creator=kube:admin,
eventing.knative.dev/lastModifier=kube:admin
Age:          2m
Broker:       default
Filter:
  type:       dev.knative.event
```

```
Sink:
  Name:       edisplay
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                2m
  ++ BrokerReady          2m
  ++ DependencyReady      2m
  ++ Subscribed           2m
  ++ SubscriberResolved   2m
```

5.5.5. Connexion d'un déclencheur à un puits

Vous pouvez connecter un déclencheur à un puits, de sorte que les événements provenant d'un courtier soient filtrés avant d'être envoyés au puits. Un puits connecté à un déclencheur est configuré en tant que **subscriber** dans la spécification des ressources de l'objet **Trigger**.

Exemple d'un objet Trigger connecté à un puits Apache Kafka

```
apiVersion: eventing.knative.dev/v1
kind: Trigger
metadata:
  name: <trigger_name> 1
spec:
...
  subscriber:
    ref:
      apiVersion: eventing.knative.dev/v1alpha1
      kind: KafkaSink
      name: <kafka_sink_name> 2
```

1. Le nom du déclencheur connecté à l'évier.
2. Le nom d'un objet **KafkaSink**.

5.5.6. Filtrer les déclencheurs à partir de la ligne de commande

L'utilisation de la CLI Knative (**kn**) pour filtrer les événements à l'aide de déclencheurs offre une interface utilisateur simplifiée et intuitive. Vous pouvez utiliser la commande **kn trigger create**, ainsi que les drapeaux appropriés, pour filtrer les événements à l'aide de déclencheurs.

5.5.6.1. Filtrer les événements avec des déclencheurs en utilisant le CLI Knative

Dans l'exemple de déclenchement suivant, seuls les événements ayant l'attribut **type: dev.knative.samples.helloworld** sont envoyés au récepteur d'événements :

```
$ kn trigger create <trigger_name> --broker <broker_name> --filter
type=dev.knative.samples.helloworld --sink ksvc:<service_name>
```

Vous pouvez également filtrer les événements en utilisant plusieurs attributs. L'exemple suivant montre comment filtrer les événements à l'aide des attributs type, source et extension :

```
$ kn trigger create <trigger_name> --broker <broker_name> --sink ksvc:<service_name> \
--filter type=dev.knative.samples.helloworld \
--filter source=dev.knative.samples/helloworldsource \
--filter myextension=my-extension-value
```

5.5.7. Mise à jour des déclencheurs à partir de la ligne de commande

L'utilisation du CLI Knative (**kn**) pour mettre à jour les déclencheurs offre une interface utilisateur rationalisée et intuitive.

5.5.7.1. Mise à jour d'un déclencheur à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn trigger update** avec certains drapeaux pour mettre à jour les attributs d'un déclencheur.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Mettre à jour un déclencheur :

  ```
  kn trigger update <trigger_name> --filter <key=value> --sink <sink_name> [flags] (en anglais)
  ```

  - Vous pouvez mettre à jour un déclencheur pour filtrer les attributs d'événement exacts qui correspondent aux événements entrants. Par exemple, en utilisant l'attribut **type**:

    ```
    $ kn trigger update <trigger_name> --filter type=knative.dev.event
    ```

  - Vous pouvez supprimer un attribut de filtre d'un déclencheur. Par exemple, vous pouvez supprimer l'attribut de filtre avec la clé **type**:

    ```
    kn trigger update <trigger_name> --filter type-
    ```

  - Vous pouvez utiliser le paramètre **--sink** pour modifier le puits d'événement d'un déclencheur :

    ```
    $ kn trigger update <trigger_name> --sink ksvc:my-event-sink
    ```

5.5.8. Suppression de déclencheurs à partir de la ligne de commande

L'utilisation de la CLI Knative (**kn**) pour supprimer un déclencheur offre une interface utilisateur rationalisée et intuitive.

5.5.8.1. Suppression d'un déclencheur à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn trigger delete** pour supprimer un déclencheur.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Supprimer un déclencheur :

```
kn trigger delete <trigger_name>
```

Vérification

1. Dresser la liste des déclencheurs existants :

```
$ kn trigger list
```

2. Vérifiez que le déclencheur n'existe plus :

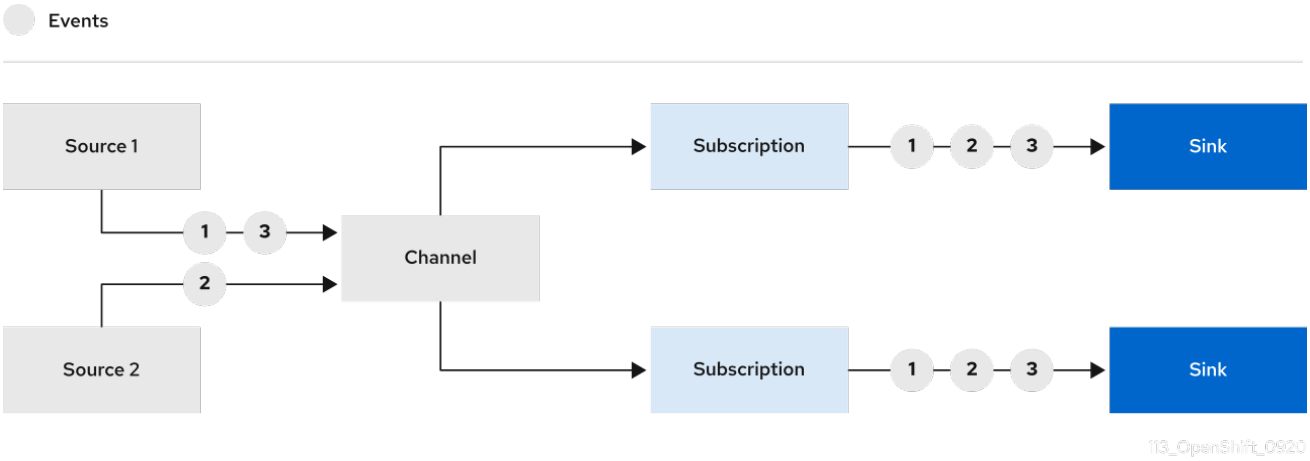
Exemple de sortie

```
No triggers found.
```

5.6. CANAUX

5.6.1. Chaînes et abonnements

Les canaux sont des ressources personnalisées qui définissent une couche unique de transmission et de persistance des événements. Une fois que des événements ont été envoyés à un canal à partir d'une source ou d'un producteur d'événements, ces événements peuvent être envoyés à plusieurs services Knative ou à d'autres puits à l'aide d'un abonnement.

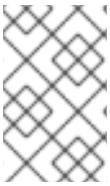


Vous pouvez créer des canaux en instanciant un objet **Channel** pris en charge et configurer les tentatives de re-livraison en modifiant la spécification **delivery** dans un objet **Subscription**.

Après avoir créé un objet **Channel**, un webhook d'admission à la mutation ajoute un ensemble de propriétés **spec.channelTemplate** pour l'objet **Channel** en fonction de l'implémentation du canal par défaut. Par exemple, pour une implémentation par défaut de **InMemoryChannel**, l'objet **Channel** se présente comme suit :

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
  namespace: default
spec:
  channelTemplate:
    apiVersion: messaging.knative.dev/v1
    kind: InMemoryChannel
```

Le contrôleur de canal crée ensuite l'instance de canal d'appui sur la base de la configuration **spec.channelTemplate**.



NOTE

Les propriétés de **spec.channelTemplate** ne peuvent pas être modifiées après la création, car elles sont définies par le mécanisme de canal par défaut et non par l'utilisateur.

Lorsque ce mécanisme est utilisé dans l'exemple précédent, deux objets sont créés : un canal générique de soutien et un canal **InMemoryChannel**. Si vous utilisez une implémentation de canal par défaut différente, le canal **InMemoryChannel** est remplacé par un canal spécifique à votre implémentation. Par exemple, avec le courtier Knative pour Apache Kafka, le canal **KafkaChannel** est créé.

Le canal d'appui agit comme un proxy qui copie ses abonnements sur l'objet canal créé par l'utilisateur et définit le statut de l'objet canal créé par l'utilisateur pour refléter le statut du canal d'appui.

5.6.1.1. Types de mise en œuvre des canaux

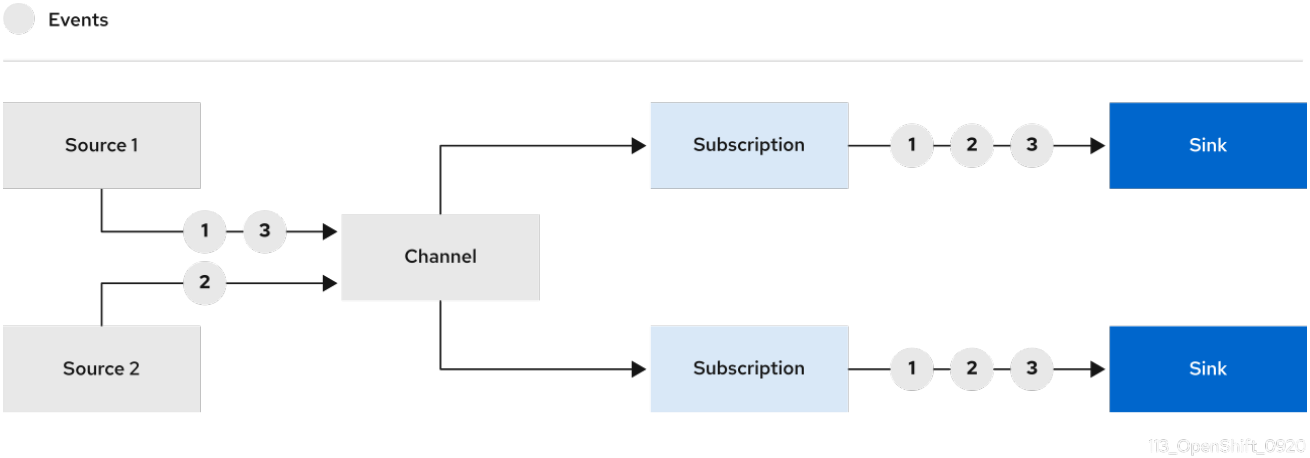
InMemoryChannel et **KafkaChannel** peuvent être utilisés avec OpenShift Serverless pour le développement.

Les limitations des canaux de type **InMemoryChannel** sont les suivantes :

- Aucune persistance des événements n'est disponible. Si un pod tombe en panne, les événements relatifs à ce pod sont perdus.
- **InMemoryChannel** les canaux n'implémentent pas l'ordre des événements, de sorte que deux événements reçus simultanément dans le canal peuvent être transmis à un abonné dans n'importe quel ordre.
- Si un abonné rejette un événement, il n'y a pas de tentative de re-livraison par défaut. Vous pouvez configurer les tentatives de re-livraison en modifiant la spécification **delivery** dans l'objet **Subscription**.

5.6.2. Création de canaux

Les canaux sont des ressources personnalisées qui définissent une couche unique de transmission et de persistance des événements. Une fois que des événements ont été envoyés à un canal à partir d'une source ou d'un producteur d'événements, ces événements peuvent être envoyés à plusieurs services Knative ou à d'autres puits à l'aide d'un abonnement.



Vous pouvez créer des canaux en instanciant un objet **Channel** pris en charge et configurer les tentatives de re-livraison en modifiant la spécification **delivery** dans un objet **Subscription**.

5.6.2.1. Créer un canal en utilisant la perspective de l'administrateur

Une fois Knative Eventing installé sur votre cluster, vous pouvez créer un canal en utilisant la perspective Administrateur.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans la liste **Create**, sélectionnez **Channel**. Vous serez dirigé vers la page **Channel**.
3. Sélectionnez le type d'objet **Channel** que vous souhaitez créer dans la liste **Type**.

NOTE

Actuellement, seuls les objets de canal **InMemoryChannel** sont pris en charge par défaut. Les canaux Knative pour Apache Kafka sont disponibles si vous avez installé l'implémentation du courtier Knative pour Apache Kafka sur OpenShift Serverless.

4. Cliquez sur **Create**.

5.6.2.2. Créer un canal en utilisant la perspective du développeur

L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un canal. Une fois Knative Eventing installé sur votre cluster, vous pouvez créer un canal en utilisant la console web.

Conditions préalables

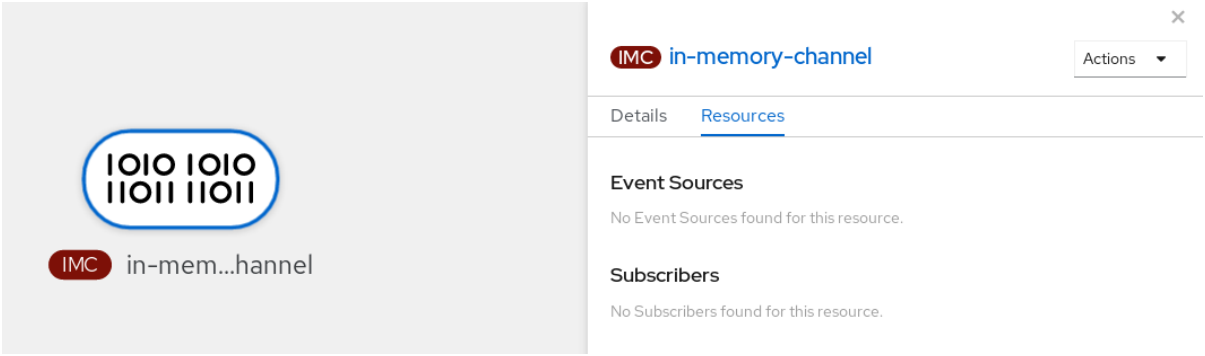
- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Dans la perspective **Developer**, naviguez vers **Add → Channel**.
2. Sélectionnez le type d'objet **Channel** que vous souhaitez créer dans la liste **Type**.
3. Cliquez sur **Create**.

Vérification

- Confirmez l'existence du canal en vous rendant sur la page **Topology**.



5.6.2.3. Créer un canal en utilisant le CLI Knative

L'utilisation de la CLI Knative (**kn**) pour créer des canaux offre une interface utilisateur plus rationnelle et intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn channel create** pour créer un canal.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un canal :

```
kn channel create <channel_name> --type <channel_type> $ kn channel create
<channel_name>
```

Le type de canal est facultatif, mais lorsqu'il est spécifié, il doit être donné dans le format **Group:Version:Kind**. Par exemple, vous pouvez créer un objet **InMemoryChannel**:

```
$ kn channel create mychannel --type messaging.knative.dev:v1:InMemoryChannel
```

Exemple de sortie

```
Channel 'mychannel' created in namespace 'default'.
```

Vérification

- Pour confirmer que le canal existe désormais, dressez la liste des canaux existants et examinez les résultats :

```
$ kn channel list
```

Exemple de sortie

```
kn channel list
NAME TYPE URL AGE READY REASON
mychannel InMemoryChannel http://mychannel-kn-channel.default.svc.cluster.local 93s
True
```

Suppression d'un canal

- Supprimer un canal :

```
kn channel delete <nom_du_canal>
```

5.6.2.4. Création d'un canal de mise en œuvre par défaut à l'aide de YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les canaux de manière déclarative et reproductible. Pour créer un canal sans serveur à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **Channel**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créer un objet **Channel** sous la forme d'un fichier YAML :

```
apiVersion: messaging.knative.dev/v1
kind: Channel
metadata:
  name: example-channel
  namespace: default
```

2. Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

5.6.2.5. Créer un canal pour Apache Kafka en utilisant YAML

La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les canaux de manière déclarative et reproductible. Vous pouvez créer un canal Knative Eventing soutenu par des sujets Kafka en créant un canal Kafka. Pour créer un canal Kafka à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **KafkaChannel**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

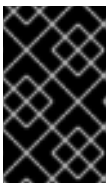
- OpenShift Serverless Operator, Knative Eventing et la ressource personnalisée **KnativeKafka** sont installés sur votre cluster OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créer un objet **KafkaChannel** sous la forme d'un fichier YAML :

```
apiVersion: messaging.knative.dev/v1beta1
kind: KafkaChannel
metadata:
  name: example-channel
```

```
  namespace: default
spec:
  numPartitions: 3
  replicationFactor: 1
```



IMPORTANT

Seule la version **v1beta1** de l'API pour les objets **KafkaChannel** sur OpenShift Serverless est prise en charge. N'utilisez pas la version **v1alpha1** de cette API, car elle est désormais obsolète.

2. Appliquer le fichier YAML **KafkaChannel**:

```
$ oc apply -f <filename>
```

5.6.2.6. Prochaines étapes

- Après avoir créé un canal, vous pouvez le connecter à un récepteur afin que celui-ci puisse recevoir des événements.
- Configurez les paramètres de livraison d'événements qui sont appliqués dans les cas où un événement n'est pas livré à un récepteur d'événements. Voir Exemples de configuration des paramètres de livraison d'événements.

5.6.3. Raccordement des canaux aux éviers

Les événements envoyés à un canal par une source ou un producteur d'événements peuvent être transmis à un ou plusieurs puits à l'aide de *subscriptions*. Vous pouvez créer des abonnements en configurant un objet **Subscription**, qui spécifie le canal et le puits (également connu sous le nom de *subscriber*) qui consomme les événements envoyés à ce canal.

5.6.3.1. Créer un abonnement à l'aide de la perspective du développeur

Une fois que vous avez créé un canal et un puits d'événements, vous pouvez créer un abonnement pour activer la livraison d'événements. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un abonnement.

Conditions préalables

- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web.
- Vous avez créé un puits d'événements, tel qu'un service Knative, et un canal.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Dans la perspective **Developer**, naviguez jusqu'à la page **Topology**.
2. Créez un abonnement en utilisant l'une des méthodes suivantes :

a. Survolez le canal pour lequel vous souhaitez créer un abonnement et faites glisser la flèche. L'option **Add Subscription** s'affiche.

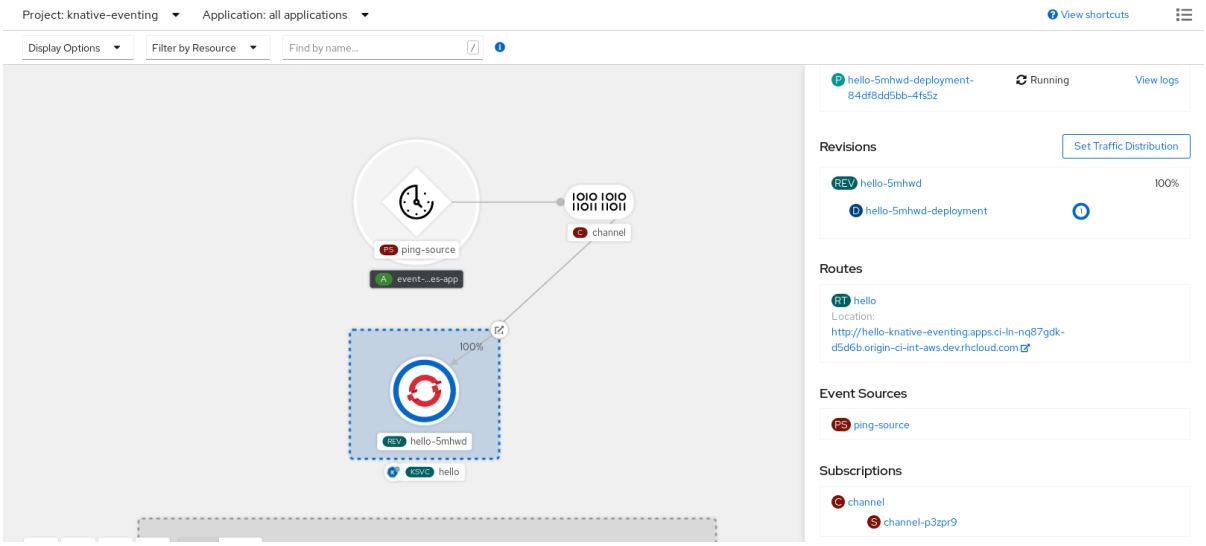
i. Sélectionnez votre évier dans la liste **Subscriber**.

ii. Cliquez sur **Add**.

b. Si le service est disponible dans la vue **Topology** sous le même espace de noms ou projet que le canal, cliquez sur le canal pour lequel vous souhaitez créer un abonnement et faites glisser la flèche directement sur un service pour créer immédiatement un abonnement du canal à ce service.

Vérification

- Une fois l'abonnement créé, il est représenté par une ligne reliant le canal au service dans la vue **Topology**:



5.6.3.2. Création d'un abonnement à l'aide de YAML

Après avoir créé un canal et un puits d'événements, vous pouvez créer un abonnement pour permettre la livraison d'événements. La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les abonnements de manière déclarative et reproductible. Pour créer un abonnement à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **Subscription**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un objet **Subscription**:
  - Créez un fichier YAML et copiez-y l'exemple de code suivant :

```
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
metadata:
  name: my-subscription 1
  namespace: default
spec:
  channel: 2
    apiVersion: messaging.knative.dev/v1beta1
    kind: Channel
    name: example-channel
  delivery: 3
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: error-handler
  subscriber: 4
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1. Nom de l'abonnement.
2. Paramètres de configuration pour le canal auquel l'abonnement se connecte.
3. Paramètres de configuration pour la livraison d'événements. Ce paramètre indique à l'abonnement ce qu'il advient des événements qui ne peuvent pas être livrés à l'abonné. Lorsque cette option est configurée, les événements qui n'ont pas pu être consommés sont envoyés à **deadLetterSink**. L'événement est abandonné, aucune nouvelle livraison de l'événement n'est tentée et une erreur est consignée dans le système. La valeur **deadLetterSink** doit être une destination.
4. Paramètres de configuration de l'abonné. Il s'agit du puits d'événements vers lequel les événements sont acheminés à partir du canal.

  - Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

5.6.3.3. Créer un abonnement en utilisant le CLI Knative

Après avoir créé un canal et un puits d'événements, vous pouvez créer un abonnement pour permettre la livraison d'événements. L'utilisation de la CLI Knative (**kn**) pour créer des abonnements offre une interface utilisateur plus rationnelle et intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn subscription create** avec les drapeaux appropriés pour créer un abonnement.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un abonnement pour connecter un puits à un canal :

```
$ kn subscription create <subscription_name> \
  --channel <group:version:kind>:<channel_name> \ 1
  --sink <sink_prefix>:<sink_name> \ 2
  --sink-dead-letter <sink_prefix>:<sink_name> 3
```

1. **--channel** spécifie la source des événements du nuage à traiter. Vous devez fournir le nom du canal. Si vous n'utilisez pas le canal par défaut **InMemoryChannel** qui est soutenu par la ressource personnalisée **Channel**, vous devez préfixer le nom du canal par **<group:version:kind>** pour le type de canal spécifié. Par exemple, il s'agira de **messaging.knative.dev:v1beta1:KafkaChannel** pour un canal soutenu par Apache Kafka.

2. **--sink** spécifie la destination cible à laquelle l'événement doit être livré. Par défaut, le site **<sink_name>** est interprété comme un service Knative de ce nom, dans le même espace de noms que l'abonnement. Vous pouvez spécifier le type de l'évier en utilisant l'un des préfixes suivants :

   ksvc
   : Un service Knative.

   channel
   : Un canal qui doit être utilisé comme destination. Seuls les types de canaux par défaut peuvent être référencés ici.

   broker
   : Un courtier en concours complet.

3. Facultatif : **--sink-dead-letter** est un drapeau facultatif qui peut être utilisé pour spécifier un puits vers lequel les événements doivent être envoyés dans les cas où les événements ne sont pas livrés. Pour plus d'informations, voir la documentation OpenShift Serverless *Event delivery*.

Example command

```
$ kn subscription create mysubscription --channel mychannel --sink ksvc:event-display
```

Exemple de sortie

```
```

```
Subscription 'mysubscription' created in namespace 'default'.
```

Vérification

- Pour confirmer que le canal est connecté au puits d'événements, ou *subscriber*, par un abonnement, dressez la liste des abonnements existants et inspectez la sortie :

```
$ kn subscription list
```

Exemple de sortie

```
NAME           CHANNEL           SUBSCRIBER          REPLY   DEAD LETTER SINK
READY   REASON
mysubscription   Channel:mychannel   ksvc:event-display                      True
```

Suppression d'un abonnement

- Supprimer un abonnement :

```
kn subscription delete <nom_de_l'abonnement>
```

5.6.3.4. Création d'un abonnement à l'aide de la perspective de l'administrateur

Après avoir créé un canal et un puits d'événements, également appelé *subscriber*, vous pouvez créer un abonnement pour permettre la livraison d'événements. Les abonnements sont créés en configurant un objet **Subscription**, qui spécifie le canal et l'abonné à qui livrer les événements. Vous pouvez également spécifier certaines options propres à l'abonné, telles que la manière de gérer les échecs.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.
- Vous avez créé un canal Knative.
- Vous avez créé un service Knative à utiliser en tant qu'abonné.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans l'onglet **Channel**, sélectionnez le menu Options pour la chaîne à laquelle vous souhaitez ajouter un abonnement.
3. Cliquez sur **Add Subscription** dans la liste.

4. Dans la boîte de dialogue **Add Subscription**, sélectionnez un **Subscriber** pour l'abonnement. L'abonné est le service Knative qui reçoit les événements du canal.

5. Cliquez sur **Add**.

5.6.3.5. Prochaines étapes

- Configurez les paramètres de livraison d'événements qui sont appliqués dans les cas où un événement n'est pas livré à un récepteur d'événements. Voir Exemples de configuration des paramètres de livraison d'événements.

5.6.4. Mise en œuvre du canal par défaut

Vous pouvez utiliser la carte de configuration **default-ch-webhook** pour spécifier l'implémentation du canal par défaut de Knative Eventing. Vous pouvez spécifier l'implémentation du canal par défaut pour l'ensemble du cluster ou pour un ou plusieurs espaces de noms. Actuellement, les types de canaux **InMemoryChannel** et **KafkaChannel** sont pris en charge.

5.6.4.1. Configuration de la mise en œuvre du canal par défaut

Conditions préalables

- Vous disposez de droits d'administrateur sur OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Eventing sur votre cluster.
- Si vous souhaitez utiliser les canaux Knative pour Apache Kafka en tant qu'implémentation de canal par défaut, vous devez également installer le CR **KnativeKafka** sur votre cluster.

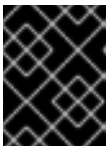
Procédure

- Modifier la ressource personnalisée **KnativeEventing** pour ajouter des détails de configuration pour la carte de configuration **default-ch-webhook**:

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config: (1)
    default-ch-webhook: (2)
      default-ch-config: |
        clusterDefault: (3)
          apiVersion: messaging.knative.dev/v1
          kind: InMemoryChannel
          spec:
            delivery:
              backoffDelay: PT0.5S
              backoffPolicy: exponential
              retry: 5
        namespaceDefaults: (4)
          my-namespace:
            apiVersion: messaging.knative.dev/v1beta1
```

```
    kind: KafkaChannel
    spec:
      numPartitions: 1
      replicationFactor: 1
```

1. Dans **spec.config**, vous pouvez spécifier les cartes de configuration pour lesquelles vous souhaitez ajouter des configurations modifiées.
2. La carte de configuration **default-ch-webhook** peut être utilisée pour spécifier l'implémentation du canal par défaut pour le cluster ou pour un ou plusieurs espaces de noms.
3. La configuration du type de canal par défaut à l'échelle du cluster. Dans cet exemple, l'implémentation du canal par défaut pour le cluster est **InMemoryChannel**.
4. La configuration du type de canal par défaut de l'espace de noms. Dans cet exemple, l'implémentation du canal par défaut pour l'espace de noms **my-namespace** est **KafkaChannel**.



IMPORTANT

La configuration d'une valeur par défaut spécifique à l'espace de nommage est prioritaire sur les paramètres applicables à l'ensemble du cluster.

5.6.5. Configuration de la sécurité pour les canaux

5.6.5.1. Configuration de l'authentification TLS pour les canaux Knative pour Apache Kafka

Transport Layer Security (TLS) est utilisé par les clients et les serveurs Apache Kafka pour chiffrer le trafic entre Knative et Kafka, ainsi que pour l'authentification. TLS est la seule méthode de cryptage du trafic prise en charge par l'implémentation du courtier Knative pour Apache Kafka.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** CR sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous disposez d'un certificat d'autorité de certification pour le cluster Kafka stocké dans un fichier **.pem**.
- Vous disposez d'un certificat client pour le cluster Kafka et d'une clé stockée sous forme de fichiers **.pem**.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez les fichiers de certificats en tant que secrets dans l'espace de noms que vous avez choisi :

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-file=user.crt=certificate.pem \
  --from-file=user.key=key.pem
```

IMPORTANT

Utilisez les noms de clés **ca.crt**, **user.crt**, et **user.key**. Ne les modifiez pas.

2. Commencez à modifier la ressource personnalisée **KnativeKafka**:

```
$ oc edit knativekafka
```

3. Faites référence à votre secret et à l'espace de noms du secret :

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: <kafka_auth_secret>
    authSecretNamespace: <kafka_auth_secret_namespace>
    bootstrapServers: <bootstrap_servers>
    enabled: true
  source:
    enabled: true
```

NOTE

Veillez à spécifier le port correspondant dans le serveur d'amorçage.

Par exemple :

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: tls-user
    authSecretNamespace: kafka
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9094
    enabled: true
  source:
    enabled: true
```

5.6.5.2. Configuration de l'authentification SASL pour les canaux Knative pour Apache Kafka

Simple Authentication and Security Layer (SASL) est utilisé par Apache Kafka pour l'authentification. Si vous utilisez l'authentification SASL sur votre cluster, les utilisateurs doivent fournir des informations d'identification à Knative pour communiquer avec le cluster Kafka ; sinon, les événements ne peuvent pas être produits ou consommés.

Conditions préalables

- Vous avez des permissions d'administrateur de cluster ou dédié sur OpenShift Container Platform.
- OpenShift Serverless Operator, Knative Eventing et **KnativeKafka** CR sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous disposez d'un nom d'utilisateur et d'un mot de passe pour un cluster Kafka.
- Vous avez choisi le mécanisme SASL à utiliser, par exemple **PLAIN**, **SCRAM-SHA-256**, ou **SCRAM-SHA-512**.
- Si TLS est activé, vous avez également besoin du fichier de certificat **ca.crt** pour le cluster Kafka.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Créez les fichiers de certificats en tant que secrets dans l'espace de noms que vous avez choisi :

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-file=ca.crt=caroot.pem \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

   - Utilisez les noms de clés **ca.crt**, **password**, et **sasl.mechanism**. Ne les modifiez pas.
   - Si vous souhaitez utiliser SASL avec des certificats d'autorité de certification publique, vous devez utiliser l'indicateur **tls.enabled=true**, plutôt que l'argument **ca.crt**, lors de la création du secret. Par exemple :

```
$ oc create secret -n <namespace> generic <kafka_auth_secret> \
  --from-literal=tls.enabled=true \
  --from-literal=password="SecretPassword" \
  --from-literal=saslType="SCRAM-SHA-512" \
  --from-literal=user="my-sasl-user"
```

2. Commencez à modifier la ressource personnalisée **KnativeKafka**:

```
$ oc edit knativekafka
```

3. Faites référence à votre secret et à l'espace de noms du secret :

```
apiVersion: operator.serverless.openshift.io/v1alpha1
```

```
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: <kafka_auth_secret>
    authSecretNamespace: <kafka_auth_secret_namespace>
    bootstrapServers: <bootstrap_servers>
    enabled: true
  source:
    enabled: true
```

NOTE

Veillez à spécifier le port correspondant dans le serveur d'amorçage.

Par exemple :

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  namespace: knative-eventing
  name: knative-kafka
spec:
  channel:
    authSecretName: scram-user
    authSecretNamespace: kafka
    bootstrapServers: eventing-kafka-bootstrap.kafka.svc:9093
    enabled: true
  source:
    enabled: true
```

5.7. ABONNEMENTS

5.7.1. Création d'abonnements

Après avoir créé un canal et un puits d'événements, vous pouvez créer un abonnement pour permettre la livraison d'événements. Les abonnements sont créés en configurant un objet **Subscription**, qui spécifie le canal et le puits (également connu sous le nom de *subscriber*) vers lesquels les événements doivent être acheminés.

5.7.1.1. Création d'un abonnement à l'aide de la perspective de l'administrateur

Après avoir créé un canal et un puits d'événements, également appelé *subscriber*, vous pouvez créer un abonnement pour permettre la livraison d'événements. Les abonnements sont créés en configurant un objet **Subscription**, qui spécifie le canal et l'abonné à qui livrer les événements. Vous pouvez également spécifier certaines options propres à l'abonné, telles que la manière de gérer les échecs.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.

- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Administrator**.
- Vous disposez des droits d'administrateur de cluster pour OpenShift Container Platform.
- Vous avez créé un canal Knative.
- Vous avez créé un service Knative à utiliser en tant qu'abonné.

Procédure

1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, naviguez vers **Serverless → Eventing**.
2. Dans l'onglet **Channel**, sélectionnez le menu Options  pour la chaîne à laquelle vous souhaitez ajouter un abonnement.
3. Cliquez sur **Add Subscription** dans la liste.
4. Dans la boîte de dialogue **Add Subscription**, sélectionnez un **Subscriber** pour l'abonnement. L'abonné est le service Knative qui reçoit les événements du canal.
5. Cliquez sur **Add**.

5.7.1.2. Créer un abonnement à l'aide de la perspective du développeur

Une fois que vous avez créé un canal et un puits d'événements, vous pouvez créer un abonnement pour activer la livraison d'événements. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour créer un abonnement.

Conditions préalables

- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web.
- Vous avez créé un puits d'événements, tel qu'un service Knative, et un canal.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Dans la perspective **Developer**, naviguez jusqu'à la page **Topology**.
2. Créez un abonnement en utilisant l'une des méthodes suivantes :
   a. Survolez le canal pour lequel vous souhaitez créer un abonnement et faites glisser la flèche. L'option **Add Subscription** s'affiche.

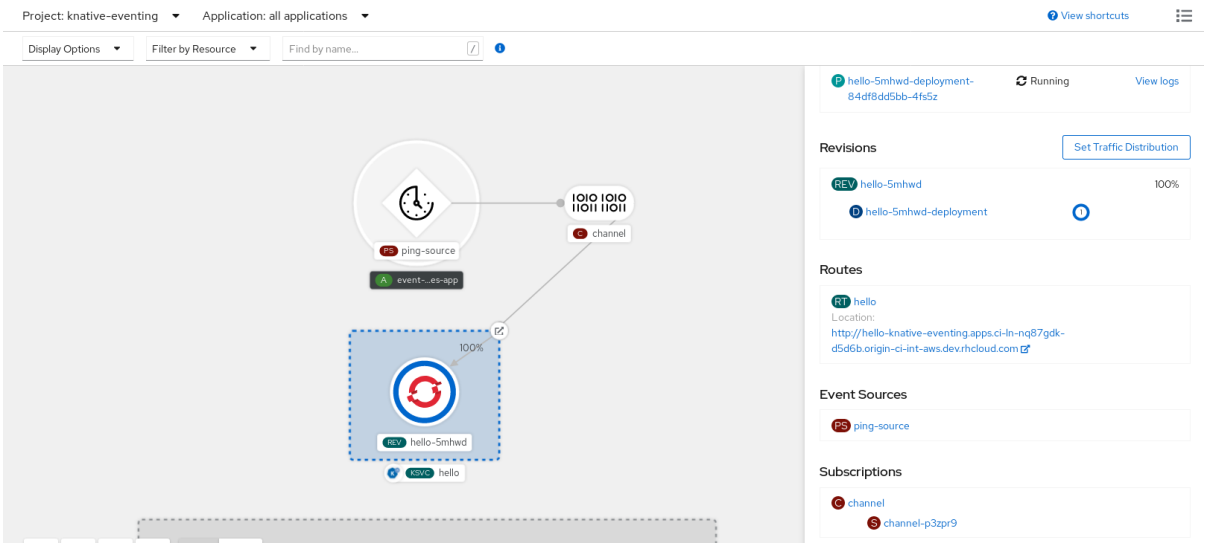
i. Sélectionnez votre évier dans la liste **Subscriber**.

ii. Cliquez sur **Add**.

b. Si le service est disponible dans la vue **Topology** sous le même espace de noms ou projet que le canal, cliquez sur le canal pour lequel vous souhaitez créer un abonnement et faites glisser la flèche directement sur un service pour créer immédiatement un abonnement du canal à ce service.

Vérification

- Une fois l'abonnement créé, il est représenté par une ligne reliant le canal au service dans la vue **Topology**:



5.7.1.3. Création d'un abonnement à l'aide de YAML

Après avoir créé un canal et un puits d'événements, vous pouvez créer un abonnement pour permettre la livraison d'événements. La création de ressources Knative à l'aide de fichiers YAML utilise une API déclarative, qui vous permet de décrire les abonnements de manière déclarative et reproductible. Pour créer un abonnement à l'aide de YAML, vous devez créer un fichier YAML qui définit un objet **Subscription**, puis l'appliquer à l'aide de la commande **oc apply**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.

- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un objet **Subscription**:
  - Créez un fichier YAML et copiez-y l'exemple de code suivant :

```
apiVersion: messaging.knative.dev/v1beta1
kind: Subscription
metadata:
  name: my-subscription 1
  namespace: default
spec:
  channel: 2
    apiVersion: messaging.knative.dev/v1beta1
    kind: Channel
    name: example-channel
  delivery: 3
    deadLetterSink:
      ref:
        apiVersion: serving.knative.dev/v1
        kind: Service
        name: error-handler
  subscriber: 4
    ref:
      apiVersion: serving.knative.dev/v1
      kind: Service
      name: event-display
```

1. Nom de l'abonnement.
2. Paramètres de configuration pour le canal auquel l'abonnement se connecte.
3. Paramètres de configuration pour la livraison d'événements. Ce paramètre indique à l'abonnement ce qu'il advient des événements qui ne peuvent pas être livrés à l'abonné. Lorsque cette option est configurée, les événements qui n'ont pas pu être consommés sont envoyés à **deadLetterSink**. L'événement est abandonné, aucune nouvelle livraison de l'événement n'est tentée et une erreur est consignée dans le système. La valeur **deadLetterSink** doit être une destination.
4. Paramètres de configuration de l'abonné. Il s'agit du puits d'événements vers lequel les événements sont acheminés à partir du canal.

  - Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

5.7.1.4. Créer un abonnement en utilisant le CLI Knative

Après avoir créé un canal et un puits d'événements, vous pouvez créer un abonnement pour permettre

la livraison d'événements. L'utilisation de la CLI Knative (**kn**) pour créer des abonnements offre une interface utilisateur plus rationnelle et intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn subscription create** avec les drapeaux appropriés pour créer un abonnement.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un abonnement pour connecter un puits à un canal :

```
$ kn subscription create <subscription_name> \
  --channel <group:version:kind>:<channel_name> \ 1
  --sink <sink_prefix>:<sink_name> \ 2
  --sink-dead-letter <sink_prefix>:<sink_name> 3
```

1. **--channel** spécifie la source des événements du nuage à traiter. Vous devez fournir le nom du canal. Si vous n'utilisez pas le canal par défaut **InMemoryChannel** qui est soutenu par la ressource personnalisée **Channel**, vous devez préfixer le nom du canal par **<group:version:kind>** pour le type de canal spécifié. Par exemple, il s'agira de **messaging.knative.dev:v1beta1:KafkaChannel** pour un canal soutenu par Apache Kafka.

2. **--sink** spécifie la destination cible à laquelle l'événement doit être livré. Par défaut, le site **<sink_name>** est interprété comme un service Knative de ce nom, dans le même espace de noms que l'abonnement. Vous pouvez spécifier le type de l'évier en utilisant l'un des préfixes suivants :

   ksvc
   : Un service Knative.

   channel
   : Un canal qui doit être utilisé comme destination. Seuls les types de canaux par défaut peuvent être référencés ici.

   broker
   : Un courtier en concours complet.

3. Facultatif : **--sink-dead-letter** est un drapeau facultatif qui peut être utilisé pour spécifier un puits vers lequel les événements doivent être envoyés dans les cas où les événements ne sont pas livrés. Pour plus d'informations, voir la documentation OpenShift Serverless *Event delivery*.

Example command

```
$ kn subscription create mysubscription --channel mychannel --sink ksvc:event-display
```

Exemple de sortie

```
Subscription 'mysubscription' created in namespace 'default'.
```

Vérification

- Pour confirmer que le canal est connecté au puits d'événements, ou *subscriber*, par un abonnement, dressez la liste des abonnements existants et inspectez la sortie :

```
$ kn subscription list
```

Exemple de sortie

```
NAME           CHANNEL           SUBSCRIBER          REPLY   DEAD LETTER SINK
READY   REASON
mysubscription   Channel:mychannel   ksvc:event-display                      True
```

Suppression d'un abonnement

- Supprimer un abonnement :

```
kn subscription delete <nom_de_l'abonnement>
```

5.7.1.5. Prochaines étapes

- Configurez les paramètres de livraison d'événements qui sont appliqués dans les cas où un événement n'est pas livré à un récepteur d'événements. Voir Exemples de configuration des paramètres de livraison d'événements.

5.7.2. Gestion des abonnements

5.7.2.1. Décrire les abonnements en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn subscription describe** pour imprimer des informations sur un abonnement dans le terminal en utilisant la CLI Knative (**kn**). L'utilisation de l'interface de programmation Knative pour décrire les abonnements offre une interface utilisateur plus rationnelle et plus intuitive que l'affichage direct des fichiers YAML.

Conditions préalables

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un abonnement dans votre cluster.

Procédure

- Décrire un abonnement :

```
$ kn subscription describe <nom_de_l'abonnement>
```

Exemple de sortie

```
Name:           my-subscription
Namespace:      default
Annotations:    messaging.knative.dev/creator=openshift-user,
messaging.knative.dev/lastModifier=min ...
Age:            43s
Channel:        Channel:my-channel (messaging.knative.dev/v1)
Subscriber:
  URI:          http://edisplay.default.example.com
Reply:
  Name:         default
  Resource:     Broker (eventing.knative.dev/v1)
DeadLetterSink:
  Name:         my-sink
  Resource:     Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                43s
  ++ AddedToChannel       43s
  ++ ChannelReady         43s
  ++ ReferencesResolved   43s
```

5.7.2.2. Lister les abonnements en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn subscription list** pour lister les abonnements existants sur votre cluster en utilisant la CLI Knative (**kn**). L'utilisation de la CLI Knative pour lister les abonnements offre une interface utilisateur simplifiée et intuitive.

Conditions préalables

- Vous avez installé le CLI Knative (**kn**).

Procédure

- Liste des abonnements sur votre cluster :

```
$ kn subscription list
```

Exemple de sortie

```
NAME            CHANNEL            SUBSCRIBER          REPLY   DEAD LETTER SINK
READY   REASON
mysubscription   Channel:mychannel   ksvc:event-display                      True
```

5.7.2.3. Mise à jour des abonnements à l'aide du CLI Knative

Vous pouvez utiliser la commande **kn subscription update** ainsi que les drapeaux appropriés pour mettre à jour un abonnement à partir du terminal en utilisant le CLI Knative (**kn**). L'utilisation de la CLI Knative pour mettre à jour les abonnements offre une interface utilisateur plus rationnelle et intuitive que la mise à jour directe des fichiers YAML.

Conditions préalables

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un abonnement.

Procédure

- Mettre à jour un abonnement :

```
$ kn subscription update <subscription_name> \
  --sink <sink_prefix>:<sink_name> \ 1
  --sink-dead-letter <sink_prefix>:<sink_name> 2
```

1 **--sink** spécifie la destination cible mise à jour à laquelle l'événement doit être transmis. Vous pouvez spécifier le type de puits en utilisant l'un des préfixes suivants :

ksvc
: Un service Knative.

channel
: Un canal qui doit être utilisé comme destination. Seuls les types de canaux par défaut peuvent être référencés ici.

broker
: Un courtier en concours complet.

2 Facultatif : **--sink-dead-letter** est un drapeau facultatif qui peut être utilisé pour spécifier un puits vers lequel les événements doivent être envoyés dans les cas où les événements ne sont pas livrés. Pour plus d'informations, voir la documentation OpenShift Serverless *Event delivery*.

Example command

```
$ kn subscription update mysubscription --sink ksvc:event-display
```

5.8. DÉCOUVERTE D'ÉVÉNEMENTS

5.8.1. Liste des sources d'événements et des types de sources d'événements

Il est possible d'afficher une liste de toutes les sources d'événements ou de tous les types de sources d'événements qui existent ou qui sont disponibles pour être utilisés sur votre cluster OpenShift Container Platform. Vous pouvez utiliser le CLI Knative (**kn**) ou la perspective **Developer** dans la console web d'OpenShift Container Platform pour répertorier les sources d'événements ou les types de sources d'événements disponibles.

5.8.2. Liste des types de sources d'événements à partir de la ligne de commande

L'utilisation du CLI Knative (**kn**) offre une interface utilisateur simplifiée et intuitive pour visualiser les types de sources d'événements disponibles sur votre cluster.

5.8.2.1. Liste des types de sources d'événements disponibles en utilisant le CLI Knative

Vous pouvez répertorier les types de sources d'événements qui peuvent être créés et utilisés sur votre cluster à l'aide de la commande CLI **kn source list-types**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Liste les types de sources d'événements disponibles dans le terminal :

```
$ kn source list-types
```

Exemple de sortie

```
TYPE              NAME                                DESCRIPTION
ApiServerSource   apiserversources.sources.knative.dev   Watch and send Kubernetes API events to a sink
PingSource        pingsources.sources.knative.dev        Periodically send ping events to a sink
SinkBinding       sinkbindings.sources.knative.dev       Binding for connecting a PodSpecable to a sink
```

2. Facultatif : vous pouvez également dresser la liste des types de sources d'événements disponibles au format YAML :

```
$ kn source list-types -o yaml
```

5.8.3. Liste des types de sources d'événements du point de vue du développeur

Il est possible d'afficher une liste de tous les types de sources d'événements disponibles sur votre cluster. L'utilisation de la console web d'OpenShift Container Platform offre une interface utilisateur rationalisée et intuitive pour visualiser les types de sources d'événements disponibles.

5.8.3.1. Affichage des types de sources d'événements disponibles dans la perspective du développeur

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- OpenShift Serverless Operator et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Accédez à la perspective **Developer**.
2. Cliquez sur **Add**.
3. Cliquez sur **Event Source**.

4. Affichez les types de sources d'événements disponibles.

5.8.4. Liste des sources d'événements à partir de la ligne de commande

L'utilisation du CLI Knative (**kn**) offre une interface utilisateur rationalisée et intuitive pour visualiser les sources d'événements existantes sur votre cluster.

5.8.4.1. Liste des sources d'événements disponibles en utilisant le CLI Knative

Vous pouvez dresser la liste des sources d'événements existantes à l'aide de la commande **kn source list**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Liste les sources d'événements existantes dans le terminal :

```
$ kn source list
```

Exemple de sortie

```
NAME   TYPE              RESOURCE                              SINK         READY
a1     ApiServerSource   apiserversources.sources.knative.dev   ksvc:eshow2   True
b1     SinkBinding       sinkbindings.sources.knative.dev       ksvc:eshow3   False
p1     PingSource        pingsources.sources.knative.dev        ksvc:eshow1   True
```

2. Facultatif : vous pouvez répertorier les sources d'événements d'un type spécifique uniquement, en utilisant l'option **--type**:

```
$ kn source list --type <event_source_type>
```

Example command

```
$ kn source list --type PingSource
```

Exemple de sortie

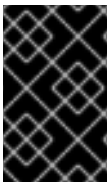
```
NAME   TYPE              RESOURCE                              SINK         READY
p1     PingSource        pingsources.sources.knative.dev        ksvc:eshow1   True
```

5.9. CONFIGURATION DE L'ÉVÉNEMENT TUNING

5.9.1. Remplacer les configurations de déploiement du système Knative Eventing

Vous pouvez remplacer les configurations par défaut pour certains déploiements spécifiques en modifiant la spécification **deployments** dans la ressource personnalisée (CR) **KnativeEventing**.

Actuellement, la modification des paramètres de configuration par défaut est prise en charge pour les champs **eventing-controller**, **eventing-webhook**, et **imc-controller**, ainsi que pour les champs **readiness** et **liveness** pour les sondes.



IMPORTANT

La spécification **replicas** ne peut pas remplacer le nombre de répliques pour les déploiements qui utilisent l'Horizontal Pod Autoscaler (HPA), et ne fonctionne pas pour le déploiement **eventing-webhook**.

NOTE

Vous ne pouvez remplacer que les sondes définies par défaut dans le déploiement.

Tous les déploiements de Knative Serving définissent par défaut une sonde de préparation et une sonde de disponibilité, à ces exceptions près :

- **net-kourier-controller** et **3scale-kourier-gateway** ne définissent qu'une sonde de préparation.
- **net-istio-controller** et **net-istio-webhook** ne définissent pas de sondes.

5.9.1.1. Remplacer les configurations de déploiement

Actuellement, le remplacement des paramètres de configuration par défaut est possible pour les champs **eventing-controller**, **eventing-webhook** et **imc-controller**, ainsi que pour les champs **readiness** et **liveness** pour les sondes.

IMPORTANT

La spécification **replicas** ne peut pas remplacer le nombre de répliques pour les déploiements qui utilisent l'Horizontal Pod Autoscaler (HPA), et ne fonctionne pas pour le déploiement **eventing-webhook**.

Dans l'exemple suivant, une CR **KnativeEventing** remplace le déploiement **eventing-controller** de sorte que :

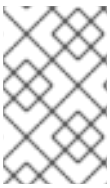
- Le délai d'attente de la sonde **readiness eventing-controller** est fixé à 10 secondes.
- Le déploiement a spécifié des limites de ressources de CPU et de mémoire.
- Le déploiement comporte 3 répliques.
- L'étiquette **example-label: label** est ajoutée.
- L'annotation **example-annotation: annotation** est ajoutée.
- Le champ **nodeSelector** est défini pour sélectionner les nœuds portant l'étiquette **disktype: hdd**.

Exemple de CR KnativeEventing

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
```

```
  name: knative-eventing
  namespace: knative-eventing
spec:
  deployments:
  - name: eventing-controller
    readinessProbes: 1
      - container: controller
        timeoutSeconds: 10
    resources:
    - container: eventing-controller
      requests:
        cpu: 300m
        memory: 100Mi
      limits:
        cpu: 1000m
        memory: 250Mi
    replicas: 3
    labels:
      example-label: label
    annotations:
      example-annotation: annotation
    nodeSelector:
      disktype: hdd
```

1 Vous pouvez utiliser les surcharges de sonde **readiness** et **liveness** pour remplacer tous les champs d'une sonde dans un conteneur d'un déploiement comme spécifié dans l'API Kubernetes, à l'exception des champs liés au gestionnaire de la sonde : **exec**, **grpc**, **httpGet**, et **tcpSocket**.



NOTE

Les paramètres d'étiquetage et d'annotation de **KnativeEventing** CR remplacent les étiquettes et les annotations du déploiement, tant pour le déploiement lui-même que pour les pods qui en résultent.

Ressources supplémentaires

- Section sur la configuration des sondes de la documentation de l'API Kubernetes

5.9.2. Haute disponibilité

La haute disponibilité (HA) est une fonctionnalité standard des API Kubernetes qui permet de garantir que les API restent opérationnelles en cas de perturbation. Dans un déploiement HA, si un contrôleur actif tombe en panne ou est supprimé, un autre contrôleur est immédiatement disponible. Ce contrôleur prend en charge le traitement des API qui étaient gérées par le contrôleur qui n'est plus disponible.

HA dans OpenShift Serverless est disponible via l'élection de leader, qui est activée par défaut après l'installation du plan de contrôle Knative Serving ou Eventing. Lors de l'utilisation d'un modèle HA d'élection de leader, les instances de contrôleurs sont déjà planifiées et en cours d'exécution dans le cluster avant qu'elles ne soient nécessaires. Ces instances de contrôleurs sont en concurrence pour l'utilisation d'une ressource partagée, connue sous le nom de verrou d'élection du leader. L'instance du contrôleur qui a accès à la ressource de verrouillage de l'élection du leader à un moment donné est appelée le leader.

HA dans OpenShift Serverless est disponible via l'élection de leader, qui est activée par défaut après l'installation du plan de contrôle Knative Serving ou Eventing. Lors de l'utilisation d'un modèle HA d'élection de leader, les instances de contrôleurs sont déjà planifiées et en cours d'exécution dans le cluster avant qu'elles ne soient nécessaires. Ces instances de contrôleurs sont en concurrence pour l'utilisation d'une ressource partagée, connue sous le nom de verrou d'élection du leader. L'instance du contrôleur qui a accès à la ressource de verrouillage de l'élection du leader à un moment donné est appelée le leader.

5.9.2.1. Configuration des répliques de haute disponibilité pour Knative Eventing

La haute disponibilité (HA) est disponible par défaut pour les composants Knative Eventing **eventing-controller**, **eventing-webhook**, **imc-controller**, **imc-dispatcher**, et **mt-broker-controller**, qui sont configurés pour avoir deux répliques chacun par défaut. Vous pouvez changer le nombre de répliques pour ces composants en modifiant la valeur **spec.high-availability.replicas** dans la ressource personnalisée (CR) **KnativeEventing**.

NOTE

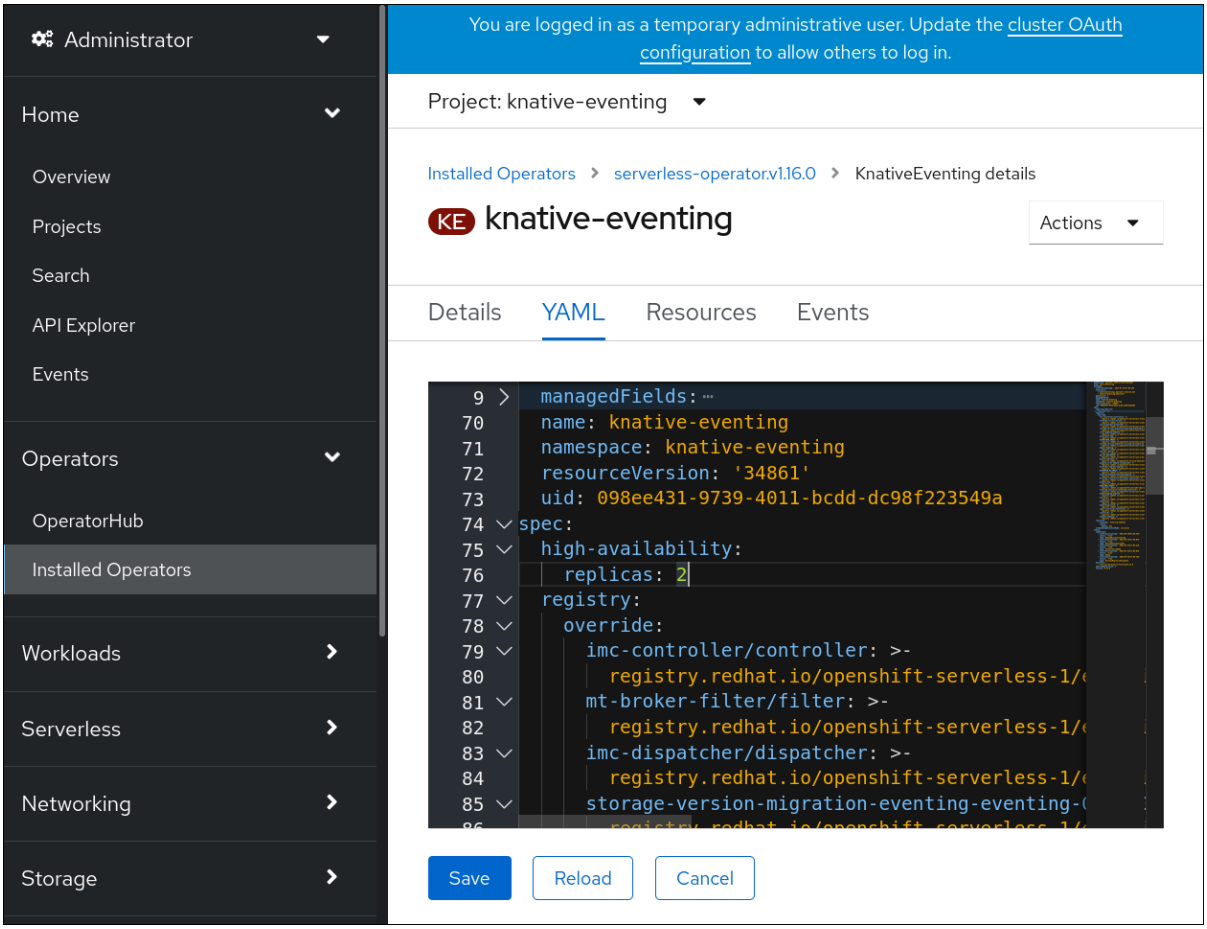
Pour Knative Eventing, les déploiements **mt-broker-filter** et **mt-broker-ingress** ne sont pas mis à l'échelle par HA. Si plusieurs déploiements sont nécessaires, redimensionnez ces composants manuellement.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- L'opérateur OpenShift Serverless et Knative Eventing sont installés sur votre cluster.

Procédure

1. Dans la perspective de la console web de OpenShift Container Platform **Administrator**, naviguez vers **OperatorHub → Installed Operators**.
2. Sélectionnez l'espace de noms **knative-eventing**.
3. Cliquez sur **Knative Eventing** dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator afin d'accéder à l'onglet **Knative Eventing**.
4. Cliquez sur **knative-eventing**, puis sur l'onglet **YAML** dans la page **knative-eventing**.



5. Modifier le nombre de répliques dans le CR **KnativeEventing**:

Exemple YAML

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

5.9.2.2. Configurer les répliques de haute disponibilité pour l'implémentation du courtier Knative pour Apache Kafka

La haute disponibilité (HA) est disponible par défaut pour l'implémentation du courtier Knative pour les composants Apache Kafka **kafka-controller** et **kafka-webhook-eventing**, qui sont configurés pour avoir deux répliques par défaut. Vous pouvez changer le nombre de répliques pour ces composants en modifiant la valeur **spec.high-availability.replicas** dans la ressource personnalisée (CR) **KnativeKafka**.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- L'opérateur OpenShift Serverless et le courtier Knative pour Apache Kafka sont installés sur votre cluster.

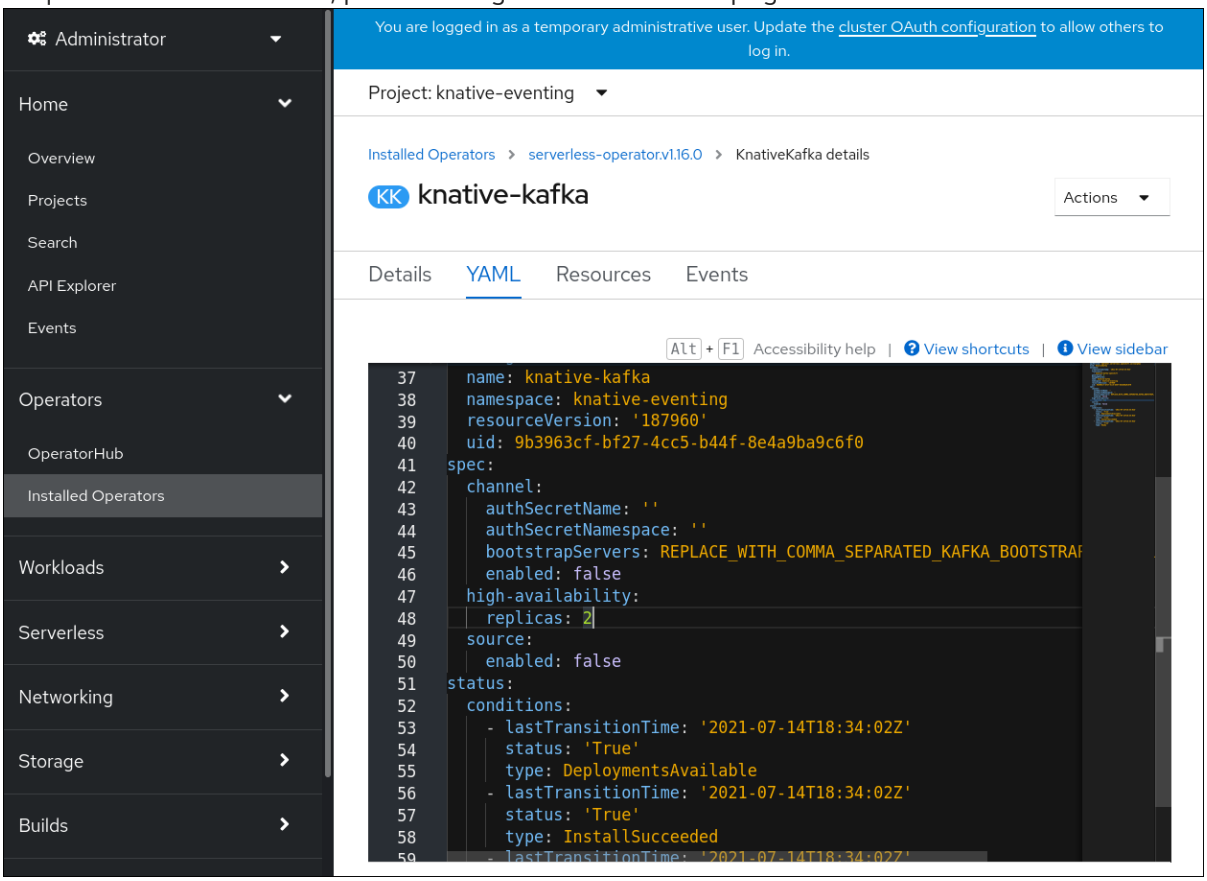
Procédure

1. Dans la perspective de la console web de OpenShift Container Platform **Administrator**, naviguez vers **OperatorHub → Installed Operators**.

2. Sélectionnez l'espace de noms **knative-eventing**.

3. Cliquez sur **Knative Kafka** dans la liste de **Provided APIs** pour l'OpenShift Serverless Operator afin d'accéder à l'onglet **Knative Kafka**.

4. Cliquez sur **knative-kafka**, puis sur l'onglet **YAML** dans la page **knative-kafka**.



5. Modifier le nombre de répliques dans le CR **KnativeKafka**:

Exemple YAML

```
apiVersion: operator.serverless.openshift.io/v1alpha1
kind: KnativeKafka
metadata:
  name: knative-kafka
  namespace: knative-eventing
spec:
  high-availability:
    replicas: 3
```

CHAPITRE 6. FONCTIONS

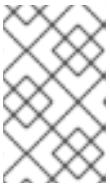
6.1. MISE EN PLACE DES FONCTIONS OPENSHIFT SERVERLESS

Pour améliorer le processus de déploiement du code de votre application, vous pouvez utiliser OpenShift Serverless pour déployer des fonctions sans état et pilotées par les événements en tant que service Knative sur OpenShift Container Platform. Si vous souhaitez développer des fonctions, vous devez effectuer les étapes de configuration.

6.1.1. Conditions préalables

Pour activer l'utilisation d'OpenShift Serverless Functions sur votre cluster, vous devez effectuer les étapes suivantes :

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.



NOTE

Les fonctions sont déployées en tant que service Knative. Si vous souhaitez utiliser une architecture pilotée par les événements avec vos fonctions, vous devez également installer Knative Eventing.

- Vous avez installé le CLI**oc** .
- Vous avez installé le CLI Knative (**kn**). L'installation du CLI Knative permet d'utiliser les commandes **kn func** pour créer et gérer des fonctions.
- Vous avez installé Docker Container Engine ou Podman version 3.4.7 ou supérieure.
- Vous avez accès à un registre d'images disponible, tel que le OpenShift Container Registry.
- Si vous utilisez Quay.io comme registre d'images, vous devez vous assurer que le dépôt n'est pas privé, ou que vous avez suivi la documentation d'OpenShift Container Platform sur l'autorisation des pods à référencer des images à partir d'autres registres sécurisés.
- Si vous utilisez le OpenShift Container Registry, un administrateur de cluster doit exposer le registre.

6.1.2. Mise en place de Podman

Pour utiliser des fonctionnalités avancées de gestion de conteneurs, vous pourriez vouloir utiliser Podman avec OpenShift Serverless Functions. Pour ce faire, vous devez démarrer le service Podman et configurer le CLI Knative (**kn**) pour vous y connecter.

Procédure

1. Démarrez le service Podman qui sert l'API Docker sur un socket UNIX à l'adresse **${XDG_RUNTIME_DIR}/podman/podman.sock**:

```
$ systemctl start --user podman.socket
```

NOTE

Sur la plupart des systèmes, cette prise est située à l'adresse **/run/user/$(id -u)/podman/podman.sock**.

2. Établir la variable d'environnement qui est utilisée pour construire une fonction :

```
$ export DOCKER_HOST="unix://${XDG_RUNTIME_DIR}/podman/podman.sock"
```

3. Exécutez la commande build dans le répertoire de votre projet de fonction avec le drapeau **-v** pour voir la sortie verbose. Vous devriez voir une connexion à votre socket UNIX local :

```
$ kn func build -v
```

6.1.3. Installation de Podman sur macOS

Pour utiliser des fonctionnalités avancées de gestion de conteneurs, vous pourriez vouloir utiliser Podman avec OpenShift Serverless Functions. Pour ce faire sur macOS, vous devez démarrer la machine Podman et configurer le CLI Knative (**kn**) pour vous y connecter.

Procédure

1. Créer la machine Podman :

```
$ podman machine init --memory=8192 --cpus=2 --disk-size=20
```

2. Démarrez la machine Podman, qui sert l'API Docker sur un socket UNIX :

```
$ podman machine start
Starting machine "podman-machine-default"
Waiting for VM ...
Mounting volume... /Users/myuser:/Users/user

[...truncated output...]

You can still connect Docker API clients by setting DOCKER_HOST using the
following command in your terminal session:

 export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-
machine-default/podman.sock'

Machine "podman-machine-default" started successfully
```

NOTE

Sur la plupart des systèmes macOS, cette prise se trouve à l'adresse **/Users/myuser/.local/share/containers/podman/machine/podman-machine-default/podman.sock**.

3. Établir la variable d'environnement qui est utilisée pour construire une fonction :

```
$ export
DOCKER_HOST='unix:///Users/myuser/.local/share/containers/podman/machine/podman-machine-default/podman.sock'
```

4. Exécutez la commande build dans le répertoire de votre projet de fonction avec le drapeau **-v** pour voir la sortie verbose. Vous devriez voir une connexion à votre socket UNIX local :

```
$ kn func build -v
```

6.1.4. Prochaines étapes

- Pour plus d'informations sur Docker Container Engine ou Podman, voir Options d'outils de construction de conteneurs.
- Voir la section Démarrer avec les fonctions.

6.2. COMMENCER AVEC LES FONCTIONS

La gestion du cycle de vie des fonctions comprend la création, l'élaboration et le déploiement d'une fonction. En option, vous pouvez également tester une fonction déployée en l'invoquant. Vous pouvez effectuer toutes ces opérations sur OpenShift Serverless en utilisant l'outil **kn func**.

6.2.1. Conditions préalables

Avant de pouvoir effectuer les procédures suivantes, vous devez vous assurer que vous avez effectué toutes les tâches prérequises dans la section Configuration d'OpenShift Serverless Functions.

6.2.2. Création de fonctions

Avant de pouvoir construire et déployer une fonction, vous devez la créer à l'aide de la CLI Knative (**kn**). Vous pouvez spécifier le chemin d'accès, la durée d'exécution, le modèle et le registre d'images en tant que drapeaux sur la ligne de commande, ou utiliser le drapeau **-c** pour lancer l'expérience interactive dans le terminal.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Créer un projet de fonction :

```
$ kn func create -r <repository> -l <runtime> -t <template> <path>
```

  - Les valeurs d'exécution acceptées sont **quarkus**, **node**, **typescript**, **go**, **python**, **springboot** et **rust**.
  - Les valeurs acceptées sont **http** et **cloudevents**.

    Example command

```
$ kn func create -l typescript -t cloudevents examplefunc
```

Exemple de sortie

```
Created typescript function in /home/user/demo/examplefunc
```

- Vous pouvez également spécifier un référentiel qui contient un modèle personnalisé.

Example command

```
$ kn func create -r https://github.com/boson-project/templates/ -l node -t hello-world
examplefunc
```

Exemple de sortie

```
Created node function in /home/user/demo/examplefunc
```

6.2.3. Exécuter une fonction localement

Vous pouvez utiliser la commande **kn func run** pour exécuter une fonction localement dans le répertoire actuel ou dans le répertoire spécifié par l'indicateur **--path**. Si la fonction que vous exécutez n'a jamais été construite auparavant, ou si les fichiers du projet ont été modifiés depuis la dernière fois qu'elle a été construite, la commande **kn func run** construit la fonction avant de l'exécuter par défaut.

Exemple de commande pour exécuter une fonction dans le répertoire courant

```
$ kn func run
```

Exemple de commande pour exécuter une fonction dans un répertoire spécifié comme chemin d'accès

```
$ kn func run --path=<directory_path>
```

Vous pouvez également forcer la reconstruction d'une image existante avant d'exécuter la fonction, même si les fichiers du projet n'ont pas été modifiés, en utilisant l'option **--build**:

Exemple de commande d'exécution utilisant le drapeau de construction

```
$ kn func run --build
```

Si l'indicateur **build** est défini comme faux, la construction de l'image est désactivée et la fonction est exécutée à l'aide de l'image précédemment construite :

Exemple de commande d'exécution utilisant le drapeau de construction

```
$ kn func run --build=false
```

Vous pouvez utiliser la commande help pour en savoir plus sur les options de la commande **kn func run**:

Commande d'aide à la construction

```
$ kn func help run
```

6.2.4. Fonctions du bâtiment

Avant de pouvoir exécuter une fonction, vous devez construire le projet de la fonction. Si vous utilisez la commande **kn func run**, la fonction est construite automatiquement. Cependant, vous pouvez utiliser la commande **kn func build** pour construire une fonction sans l'exécuter, ce qui peut être utile pour les utilisateurs avancés ou les scénarios de débogage.

La commande **kn func build** crée une image de conteneur OCI qui peut être exécutée localement sur votre ordinateur ou sur un cluster OpenShift Container Platform. Cette commande utilise le nom du projet de la fonction et le nom du registre d'images pour construire un nom d'image entièrement qualifié pour votre fonction.

6.2.4.1. Types de conteneurs d'images

Par défaut, **kn func build** crée une image de conteneur en utilisant la technologie Red Hat Source-to-Image (S2I).

Exemple de commande de compilation utilisant Red Hat Source-to-Image (S2I)

```
$ kn func build
```

6.2.4.2. Types de registres d'images

OpenShift Container Registry est utilisé par défaut comme registre d'images pour le stockage des images de fonctions.

Exemple de commande de construction utilisant OpenShift Container Registry

```
$ kn func build
```

Exemple de sortie

```
Building function image
Function image has been built, image: registry.redhat.io/example/example-function:latest
```

Vous pouvez remplacer l'utilisation d'OpenShift Container Registry comme registre d'images par défaut en utilisant le drapeau **--registry**:

Exemple de commande de construction surchargeant OpenShift Container Registry pour utiliser quay.io

```
$ kn func build --registry quay.io/username
```

Exemple de sortie

```
Building function image
Function image has been built, image: quay.io/username/example-function:latest
```

6.2.4.3. Pousser le drapeau

Vous pouvez ajouter l'option **--push** à une commande **kn func build** pour pousser automatiquement l'image de la fonction une fois qu'elle a été construite avec succès :

Exemple de commande de construction utilisant OpenShift Container Registry

```
$ kn func build --push
```

6.2.4.4. Commande d'aide

Vous pouvez utiliser la commande help pour en savoir plus sur les options de la commande **kn func build**:

Commande d'aide à la construction

```
$ kn func help build
```

6.2.5. Déployer des fonctions

Vous pouvez déployer une fonction sur votre cluster en tant que service Knative à l'aide de la commande **kn func deploy**. Si la fonction ciblée est déjà déployée, elle est mise à jour avec une nouvelle image de conteneur qui est poussée vers un registre d'images de conteneur, et le service Knative est mis à jour.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous devez avoir déjà créé et initialisé la fonction que vous souhaitez déployer.

Procédure

- Déployer une fonction :

  ```
  $ kn func deploy [-n <namespace> -p <path> -i <image>]
  ```

  #### Exemple de sortie

  ```
  Function deployed at: http://func.example.com
  ```

  - Si aucune adresse **namespace** n'est spécifiée, la fonction est déployée dans l'espace de noms actuel.
  - La fonction est déployée à partir du répertoire actuel, à moins qu'une adresse **path** ne soit spécifiée.
  - Le nom du service Knative est dérivé du nom du projet et ne peut pas être modifié à l'aide de cette commande.

6.2.6. Invoquer une fonction déployée avec un événement de test

Vous pouvez utiliser la commande CLI **kn func invoke** pour envoyer une requête de test afin d'invoquer une fonction localement ou sur votre cluster OpenShift Container Platform. Vous pouvez utiliser cette commande pour tester qu'une fonction fonctionne et qu'elle est capable de recevoir des événements correctement. L'invocation d'une fonction localement est utile pour un test rapide pendant le développement de la fonction. L'invocation d'une fonction sur le cluster est utile pour des tests plus proches de l'environnement de production.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous devez avoir déjà déployé la fonction que vous souhaitez invoquer.

Procédure

- Invoquer une fonction :

```
$ kn func invoke
```

  - La commande **kn func invoke** ne fonctionne que lorsqu'une image de conteneur locale est en cours d'exécution ou lorsqu'une fonction est déployée dans le cluster.
  - La commande **kn func invoke** s'exécute par défaut dans le répertoire local et suppose que ce répertoire est un projet de fonction.

6.2.7. Suppression d'une fonction

Vous pouvez supprimer une fonction en utilisant la commande **kn func delete**. Cette opération est utile lorsqu'une fonction n'est plus nécessaire et permet d'économiser des ressources sur votre cluster.

Procédure

- Supprimer une fonction :

```
$ kn func delete [<function_name> -n <namespace> -p <path>]
```

  - Si le nom ou le chemin de la fonction à supprimer n'est pas spécifié, le répertoire actuel est parcouru à la recherche d'un fichier **func.yaml** qui est utilisé pour déterminer la fonction à supprimer.
  - Si l'espace de noms n'est pas spécifié, il prend par défaut la valeur de **namespace** dans le fichier **func.yaml**.

6.2.8. Ressources supplémentaires

- Exposer manuellement un registre par défaut
- Page du marché pour le plugin Intellij Knative

- Page du marché pour le plugin Visual Studio Code Knative

6.2.9. Prochaines étapes

- Voir Utilisation des fonctions avec Knative Eventing

6.3. CRÉATION ET DÉPLOIEMENT DE FONCTIONS SUR LE CLUSTER

Au lieu de construire une fonction localement, vous pouvez construire une fonction directement sur le cluster. Lorsque vous utilisez ce flux de travail sur une machine de développement locale, vous ne devez travailler qu'avec le code source de la fonction. Ceci est utile, par exemple, lorsque vous ne pouvez pas installer d'outils de construction de fonctions sur le cluster, tels que docker ou podman.

6.3.1. Construire et déployer des fonctions sur le cluster

Vous pouvez utiliser le CLI de Knative (**kn**) pour initier la construction d'un projet de fonction et ensuite déployer la fonction directement sur le cluster. Pour construire un projet de fonction de cette manière, le code source de votre projet de fonction doit exister dans une branche du dépôt Git accessible à votre cluster.

Conditions préalables

- Red Hat OpenShift Pipelines doit être installé sur votre cluster.
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Dans chaque espace de noms où vous souhaitez exécuter OpenShift Pipelines et déployer une fonction, vous devez créer les ressources suivantes :
   a. Créez la tâche **s2i** Tekton pour pouvoir utiliser Source-to-Image dans le pipeline :

   ```
   $ oc apply -f https://raw.githubusercontent.com/openshift-knative/kn-plugin-func/serverless-1.28.0/pipelines/resources/tekton/task/func-s2i/0.1/func-s2i.yaml
   ```

   b. Créez la tâche **kn func** deploy Tekton pour pouvoir déployer la fonction dans le pipeline :

   ```
   $ oc apply -f https://raw.githubusercontent.com/openshift-knative/kn-plugin-func/serverless-1.28.0/pipelines/resources/tekton/task/func-deploy/0.1/func-deploy.yaml
   ```

2. Créer une fonction :

   ```
   $ kn func create <function_name> -l <runtime>
   ```

3. Après avoir créé un nouveau projet de fonction, vous devez ajouter le projet à un dépôt Git et vous assurer que le dépôt est disponible pour le cluster. Les informations relatives à ce dépôt Git sont utilisées pour mettre à jour le fichier **func.yaml** dans l'étape suivante.
4. Mettez à jour la configuration dans le fichier **func.yaml** pour votre projet de fonction afin d'activer les constructions sur le cluster pour le dépôt Git :

```
...
git:
  url: <git_repository_url> 1
  revision: main 2
  contextDir: <directory_path> 3
...
```

1. Obligatoire. Indiquez le dépôt Git qui contient le code source de votre fonction.
2. Facultatif. Spécifiez la révision du dépôt Git à utiliser. Il peut s'agir d'une branche, d'une balise ou d'un commit.
3. Facultatif. Indiquez le chemin d'accès au répertoire de la fonction si celle-ci n'est pas située dans le dossier racine du référentiel Git.

5. Mettez en œuvre la logique commerciale de votre fonction. Ensuite, utilisez Git pour livrer et pousser les changements.
6. Déployez votre fonction :

```
$ kn func deploy --remote
```

Si vous n'êtes pas connecté au registre de conteneurs référencé dans la configuration de votre fonction, vous êtes invité à fournir des informations d'identification pour le registre de conteneurs distant qui héberge l'image de la fonction :

Exemple de résultats et d'invites

```
  Creating Pipeline resources
Please provide credentials for image registry used by Pipeline.
? Server: https://index.docker.io/v1/
? Username: my-repo
? Password: ********
  Function deployed at URL: http://test-function.default.svc.cluster.local
```

7. Pour mettre à jour votre fonction, validez les nouvelles modifications à l'aide de Git, puis exécutez à nouveau la commande **kn func deploy --remote**.

6.3.2. Spécification de la révision des fonctions

Lors de la construction et du déploiement d'une fonction sur le cluster, vous devez spécifier l'emplacement du code de la fonction en indiquant le dépôt Git, la branche et le sous-répertoire dans le dépôt. Vous n'avez pas besoin de spécifier la branche si vous utilisez la branche **main**. De même, vous n'avez pas besoin de spécifier le sous-répertoire si votre fonction se trouve à la racine du référentiel. Vous pouvez spécifier ces paramètres dans le fichier de configuration **func.yaml** ou en utilisant des drapeaux avec la commande **kn func deploy**.

Conditions préalables

- Red Hat OpenShift Pipelines doit être installé sur votre cluster.
- Vous avez installé le CLI OpenShift (**oc**).
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Déployez votre fonction :

```
$ kn func deploy --remote \ 1
            --git-url <repo-url> \ 2
            [--git-branch <branch>] \ 3
            [--git-dir <function-dir>] 4
```

1. Avec l'option **--remote**, la construction s'exécute à distance.
2. Remplacez **<repo-url>** par l'URL du dépôt Git.
3. Remplacez **<branch>** par la branche, le tag ou le commit Git. Si vous utilisez le dernier commit sur la branche **main**, vous pouvez ignorer ce drapeau.
4. Remplacez **<function-dir>** par le répertoire contenant la fonction s'il est différent du répertoire racine du référentiel.

Par exemple :

```
$ kn func deploy --remote \
            --git-url https://example.com/alice/myfunc.git \
            --git-branch my-feature \
            --git-dir functions/example-func/
```

6.4. DÉVELOPPER LES FONCTIONS DE QUARKUS

Après avoir créé un projet de fonction Quarkus , vous pouvez modifier les fichiers modèles fournis pour ajouter une logique commerciale à votre fonction. Cela inclut la configuration de l'invocation de la fonction et des en-têtes et codes d'état renvoyés.

6.4.1. Conditions préalables

- Avant de pouvoir développer des fonctions, vous devez effectuer les étapes de configuration dans Configuration des fonctions OpenShift Serverless.

6.4.2. Structure du modèle de fonction Quarkus

Lorsque vous créez une fonction Quarkus en utilisant la CLI Knative (**kn**), le répertoire du projet ressemble à un projet Maven typique. En outre, le projet contient le fichier **func.yaml**, qui est utilisé pour configurer la fonction.

Les fonctions de déclenchement **http** et **event** ont la même structure de modèle :

Structure du modèle

```
.
├── func.yaml 1
├── mvnw
├── mvnw.cmd
├── pom.xml 2
└── README.md
```

```
└── src
    ├── main
    │   ├── java
    │   │   └── functions
    │   │       ├── Function.java 3
    │   │       ├── Input.java
    │   │       └── Output.java
    │   └── resources
    │       └── application.properties
    └── test
        └── java
            └── functions 4
                ├── FunctionTest.java
                └── NativeFunctionIT.java
```

1 Utilisé pour déterminer le nom et le registre de l'image.

2 Le fichier POM (Project Object Model) contient la configuration du projet, notamment des informations sur les dépendances. Vous pouvez ajouter des dépendances supplémentaires en modifiant ce fichier.

Exemple de dépendances supplémentaires

```
...
 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.12</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>org.assertj</groupId>
   <artifactId>assertj-core</artifactId>
   <version>3.8.0</version>
   <scope>test</scope>
  </dependency>
 </dependencies>
...
```

Les dépendances sont téléchargées lors de la première compilation.

3 Le projet de fonction doit contenir une méthode Java annotée avec **@Funq**. Vous pouvez placer cette méthode dans la classe **Function.java**.

4 Contient des cas de test simples qui peuvent être utilisés pour tester votre fonction localement.

6.4.3. A propos de l'invocation des fonctions de Quarkus

Vous pouvez créer un projet Quarkus qui répond aux événements cloud ou qui répond à de simples requêtes HTTP. Dans Knative, les événements cloud sont transportés par HTTP sous la forme d'une requête POST, de sorte que les deux types de fonctions peuvent écouter et répondre aux requêtes HTTP entrantes.

Lorsqu'une demande est reçue, les fonctions Quarkus sont invoquées avec une instance d'un type autorisé.

Tableau 6.1. Options d'invocation des fonctions

Méthode d'invocation	Type de données contenu dans l'instance	Exemple de données
Demande HTTP POST	Objet JSON dans le corps de la demande	**{ "customerId": "0123456", "productId": "6543210" }**
Demande HTTP GET	Données dans la chaîne de requête	**?customerId=0123456&productId=6543210**
CloudEvent	Objet JSON dans la propriété **data**	**{ "customerId": "0123456", "productId": "6543210" }**

L'exemple suivant montre une fonction qui reçoit et traite les données d'achat **customerId** et **productId** énumérées dans le tableau précédent :

Exemple de fonction Quarkus

```
public class Functions {
    @Funq
    public void processPurchase(Purchase purchase) {
        // process the purchase
    }
}
```

La classe JavaBean **Purchase** correspondante qui contient les données d'achat se présente comme suit :

Exemple de classe

```
public class Purchase {
    private long customerId;
    private long productId;
    // getters and setters
}
```

6.4.3.1. Exemples d'invocation

L'exemple de code suivant définit trois fonctions appelées **withBeans**, **withCloudEvent** et **withBinary**;

Exemple :

```
import io.quarkus.funqy.Funq;
import io.quarkus.funqy.knative.events.CloudEvent;

public class Input {
    private String message;
```

```
    // getters and setters
}

public class Output {
    private String message;

    // getters and setters
}

public class Functions {
    @Funq
    public Output withBeans(Input in) {
        // function body
    }

    @Funq
    public CloudEvent<Output> withCloudEvent(CloudEvent<Input> in) {
        // function body
    }

    @Funq
    public void withBinary(byte[] in) {
        // function body
    }
}
```

La fonction **withBeans** de la classe **Functions** peut être invoquée par :

- Une requête HTTP POST avec un corps JSON :

```
$ curl "http://localhost:8080/withBeans" -X POST \
    -H "Content-Type: application/json" \
    -d '{"message": "Hello there."}'
```

- Une requête HTTP GET avec des paramètres de requête :

```
$ curl "http://localhost:8080/withBeans?message=Hello%20there." -X GET
```

- Un objet **CloudEvent** en codage binaire :

```
$ curl "http://localhost:8080/" -X POST \
  -H "Content-Type: application/json" \
  -H "Ce-SpecVersion: 1.0" \
  -H "Ce-Type: withBeans" \
  -H "Ce-Source: cURL" \
  -H "Ce-Id: 42" \
  -d '{"message": "Hello there."}'
```

- Un objet **CloudEvent** dans un encodage structuré :

```
$ curl http://localhost:8080/ \
    -H "Content-Type: application/cloudevents+json" \
    -d '{ "data": {"message":"Hello there."},
          "datacontenttype": "application/json",
```

```
        "id": "42",
        "source": "curl",
        "type": "withBeans",
        "specversion": "1.0"}'
```

La fonction **withCloudEvent** de la classe **Functions** peut être invoquée en utilisant un objet **CloudEvent**, de la même manière que la fonction **withBeans**. Cependant, contrairement à **withBeans**, **withCloudEvent** ne peut pas être invoquée avec une simple requête HTTP.

La fonction **withBinary** de la classe **Functions** peut être invoquée par :

- Un objet **CloudEvent** en codage binaire :

```
$ curl "http://localhost:8080/" -X POST \
  -H "Content-Type: application/octet-stream" \
  -H "Ce-SpecVersion: 1.0"\
  -H "Ce-Type: withBinary" \
  -H "Ce-Source: cURL" \
  -H "Ce-Id: 42" \
  --data-binary '@img.jpg'
```

- Un objet **CloudEvent** dans un encodage structuré :

```
$ curl http://localhost:8080/ \
  -H "Content-Type: application/cloudevents+json" \
  -d "{ \"data_base64\": \"$(base64 --wrap=0 img.jpg)\",
       \"datacontenttype\": \"application/octet-stream\",
       \"id\": \"42\",
       \"source\": \"curl\",
       \"type\": \"withBinary\",
       \"specversion\": \"1.0\"}"
```

6.4.4. Attributs du CloudEvent

Si vous devez lire ou écrire les attributs d'un CloudEvent, tel que **type** ou **subject**, vous pouvez utiliser l'interface générique **CloudEvent<T>** et le constructeur **CloudEventBuilder**. Le paramètre de type **<T>** doit être l'un des types autorisés.

Dans l'exemple suivant, **CloudEventBuilder** est utilisé pour renvoyer le succès ou l'échec du traitement de l'achat :

```
public class Functions {

  private boolean _processPurchase(Purchase purchase) {
    // do stuff
  }

  public CloudEvent<Void> processPurchase(CloudEvent<Purchase> purchaseEvent) {
    System.out.println("subject is: " + purchaseEvent.subject());

    if (!_processPurchase(purchaseEvent.data())) {
      return CloudEventBuilder.create()
          .type("purchase.error")
          .build();
    }
```

```
        return CloudEventBuilder.create()
                .type("purchase.success")
                .build();
    }
}
```

6.4.5. Valeurs de retour des fonctions Quarkus

Les fonctions peuvent renvoyer une instance de n'importe quel type de la liste des types autorisés. Elles peuvent également renvoyer le type **Uni<T>**, où le paramètre de type **<T>** peut être de n'importe quel type de la liste des types autorisés.

Le type **Uni<T>** est utile si une fonction fait appel à des API asynchrones, car l'objet renvoyé est sérialisé dans le même format que l'objet reçu. En effet, l'objet retourné est sérialisé dans le même format que l'objet reçu :

- Si une fonction reçoit une requête HTTP, l'objet renvoyé est envoyé dans le corps de la réponse HTTP.
- Si une fonction reçoit un objet **CloudEvent** en codage binaire, l'objet renvoyé est envoyé dans la propriété data d'un objet **CloudEvent** codé en binaire.

L'exemple suivant montre une fonction qui récupère une liste d'achats :

Example command

```
public class Functions {
    @Funq
    public List<Purchase> getPurchasesByName(String name) {
      // logic to retrieve purchases
    }
}
```

- L'invocation de cette fonction par le biais d'une requête HTTP produit une réponse HTTP qui contient une liste d'achats dans le corps de la réponse.
- L'invocation de cette fonction par le biais d'un objet **CloudEvent** entrant produit une réponse **CloudEvent** avec une liste d'achats dans la propriété **data**.

6.4.5.1. Types autorisés

L'entrée et la sortie d'une fonction peuvent être de l'un des types **void**, **String** ou **byte[]**. En outre, il peut s'agir de types primitifs et de leurs enveloppes, par exemple **int** et **Integer**. Il peut également s'agir des objets complexes suivants : Javabeans, maps, lists, arrays et le type spécial **CloudEvents<T>**.

Les cartes, les listes, les tableaux, le paramètre de type **<T>** du type **CloudEvents<T>** et les attributs des Javabeans ne peuvent être que des types énumérés ici.

Exemple :

```
public class Functions {
    public List<Integer> getIds();
    public Purchase[] getPurchasesByName(String name);
    public String getNameById(int id);
```

```
    public Map<String,Integer> getNameIdMapping();
    public void processImage(byte[] img);
}
```

6.4.6. Test des fonctions de Quarkus

Les fonctions Quarkus peuvent être testées localement sur votre ordinateur. Dans le projet par défaut qui est créé lorsque vous créez une fonction à l'aide de **kn func create**, il y a le répertoire **src/test/**, qui contient les tests Maven de base. Ces tests peuvent être étendus si nécessaire.

Conditions préalables

- Vous avez créé une fonction Quarkus.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Accédez au dossier de projet de votre fonction.
2. Exécuter les tests Maven :

```
$ ./mvnw test
```

6.4.7. Prochaines étapes

- Construire et déployer une fonction.

6.5. DÉVELOPPER DES FONCTIONS NODE.JS

Après avoir créé un projet de fonction Node.js , vous pouvez modifier les fichiers modèles fournis pour ajouter une logique commerciale à votre fonction. Cela inclut la configuration de l'invocation de la fonction et des en-têtes et codes d'état renvoyés.

6.5.1. Conditions préalables

- Avant de pouvoir développer des fonctions, vous devez suivre les étapes de la section Configuration des fonctions OpenShift Serverless.

6.5.2. Structure du modèle de fonction Node.js

Lorsque vous créez une fonction Node.js à l'aide de la CLI Knative (**kn**), le répertoire du projet ressemble à un projet Node.js classique. La seule exception est le fichier supplémentaire **func.yaml**, qui est utilisé pour configurer la fonction.

Les fonctions de déclenchement **http** et **event** ont la même structure de modèle :

Structure du modèle

```
.
├── func.yaml 1
├── index.js 2
```

```
├── package.json 3
├── README.md
└── test 4
    ├── integration.js
    └── unit.js
```

1. Le fichier de configuration **func.yaml** est utilisé pour déterminer le nom et le registre de l'image.
2. Votre projet doit contenir un fichier **index.js** qui exporte une seule fonction.
3. Vous n'êtes pas limité aux dépendances fournies dans le fichier modèle **package.json**. Vous pouvez ajouter des dépendances supplémentaires comme vous le feriez dans n'importe quel autre projet Node.js.

   Exemple d'ajout de dépendances npm

   ```
   npm install --save opossum
   ```

   Lorsque le projet est construit pour être déployé, ces dépendances sont incluses dans l'image du conteneur d'exécution créé.
4. Les scripts de tests d'intégration et d'unité sont fournis dans le cadre du modèle de fonction.

6.5.3. A propos de l'invocation des fonctions Node.js

Lorsque vous utilisez le CLI Knative (**kn**) pour créer un projet de fonction, vous pouvez générer un projet qui répond aux CloudEvents, ou un projet qui répond aux simples requêtes HTTP. Dans Knative, les CloudEvents sont transportés par HTTP sous la forme d'une requête POST, de sorte que les deux types de fonctions écoutent les événements HTTP entrants et y répondent.

Les fonctions Node.js peuvent être invoquées à l'aide d'une simple requête HTTP. Lorsqu'une requête est reçue, les fonctions sont invoquées avec un objet **context** comme premier paramètre.

6.5.3.1. Objets contextuels Node.js

Les fonctions sont invoquées en fournissant un objet **context** comme premier paramètre. Cet objet permet d'accéder aux informations de la requête HTTP entrante.

Exemple d'objet contextuel

```
function handle(context, data)
```

Ces informations comprennent la méthode de requête HTTP, les chaînes de requête ou les en-têtes envoyés avec la requête, la version HTTP et le corps de la requête. Les demandes entrantes qui contiennent un **CloudEvent** attachent l'instance entrante du CloudEvent à l'objet de contexte de sorte qu'il est possible d'y accéder en utilisant **context.cloudevent**.

6.5.3.1.1. Méthodes de l'objet contextuel

L'objet **context** possède une seule méthode, **cloudEventResponse()**, qui accepte une valeur de données et renvoie un CloudEvent.

Dans un système Knative, si une fonction déployée en tant que service est invoquée par un courtier d'événements envoyant un CloudEvent, le courtier examine la réponse. Si la réponse est un CloudEvent, cet événement est traité par le courtier.

Exemple de méthode d'objet contextuel

```
// Expects to receive a CloudEvent with customer data
function handle(context, customer) {
  // process the customer
  const processed = handle(customer);
  return context.cloudEventResponse(customer)
    .source('/handle')
    .type('fn.process.customer')
    .response();
}
```

6.5.3.1.2. Données CloudEvent

Si la demande entrante est un CloudEvent, toutes les données associées au CloudEvent sont extraites de l'événement et fournies en tant que deuxième paramètre. Par exemple, si un CloudEvent est reçu et qu'il contient une chaîne JSON dans sa propriété data qui est similaire à ce qui suit :

```
{
  "customerId": "0123456",
  "productId": "6543210"
}
```

Lorsqu'elle est invoquée, le deuxième paramètre de la fonction, après l'objet **context**, est un objet JavaScript qui possède les propriétés **customerId** et **productId**.

Exemple de signature

```
function handle(context, data)
```

Dans cet exemple, le paramètre **data** est un objet JavaScript qui contient les propriétés **customerId** et **productId**.

6.5.4. Valeurs de retour des fonctions Node.js

Les fonctions peuvent renvoyer n'importe quel type JavaScript valide ou ne pas avoir de valeur de retour. Lorsqu'une fonction n'a pas de valeur de retour spécifiée et qu'aucun échec n'est indiqué, l'appelant reçoit une réponse **204 No Content**.

Les fonctions peuvent également renvoyer un objet CloudEvent ou **Message** afin de pousser les événements dans le système Knative Eventing. Dans ce cas, le développeur n'est pas tenu de comprendre ou de mettre en œuvre la spécification de messagerie CloudEvent. Les en-têtes et autres informations pertinentes des valeurs renvoyées sont extraites et envoyées avec la réponse.

Exemple :

```
function handle(context, customer) {
  // process customer and return a new CloudEvent
  return new CloudEvent({
    source: 'customer.processor',
```

```
    type: 'customer.processed'
  })
}
```

6.5.4.1. Renvoi des en-têtes

Vous pouvez définir un en-tête de réponse en ajoutant une propriété **headers** à l'objet **return**. Ces en-têtes sont extraits et envoyés avec la réponse à l'appelant.

Exemple d'en-tête de réponse

```
function handle(context, customer) {
  // process customer and return custom headers
  // the response will be '204 No content'
  return { headers: { customerid: customer.id } };
}
```

6.5.4.2. Renvoi des codes d'état

Vous pouvez définir un code d'état qui sera renvoyé à l'appelant en ajoutant une propriété **statusCode** à l'objet **return**:

Exemple de code d'état

```
function handle(context, customer) {
  // process customer
  if (customer.restricted) {
    return { statusCode: 451 }
  }
}
```

Des codes d'état peuvent également être définis pour les erreurs créées et déclenchées par la fonction :

Exemple de code d'état d'erreur

```
function handle(context, customer) {
  // process customer
  if (customer.restricted) {
    const err = new Error('Unavailable for legal reasons');
    err.statusCode = 451;
    throw err;
  }
}
```

6.5.5. Tester les fonctions Node.js

Les fonctions Node.js peuvent être testées localement sur votre ordinateur. Dans le projet par défaut qui est créé lorsque vous créez une fonction en utilisant **kn func create**, il y a un dossier **test** qui contient quelques tests unitaires et d'intégration simples.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction en utilisant **kn func create**.

Procédure

1. Naviguez jusqu'au dossier **test** de votre fonction.
2. Exécutez les tests :

```
$ npm test
```

6.5.6. Prochaines étapes

- Voir la documentation de référence de l'objet contextuel Node.js.
- Construire et déployer une fonction.

6.6. DÉVELOPPER DES FONCTIONS TYPESCRIPT

Après avoir créé un projet de fonction TypeScript, vous pouvez modifier les fichiers modèles fournis pour ajouter une logique métier à votre fonction. Cela inclut la configuration de l'invocation de la fonction et des en-têtes et codes d'état renvoyés.

6.6.1. Conditions préalables

- Avant de pouvoir développer des fonctions, vous devez suivre les étapes de la section Configuration des fonctions OpenShift Serverless.

6.6.2. Structure du modèle de fonction TypeScript

Lorsque vous créez une fonction TypeScript à l'aide de l'interface de programmation Knative (**kn**), le répertoire du projet ressemble à un projet TypeScript classique. La seule exception est le fichier supplémentaire **func.yaml**, qui est utilisé pour configurer la fonction.

Les fonctions de déclenchement **http** et **event** ont la même structure de modèle :

Structure du modèle

```
.
├── func.yaml 1
├── package.json 2
├── package-lock.json
├── README.md
├── src
│   └── index.ts 3
├── test 4
│   ├── integration.ts
│   └── unit.ts
└── tsconfig.json
```

1. Le fichier de configuration **func.yaml** est utilisé pour déterminer le nom et le registre de l'image.

2. Vous n'êtes pas limité aux dépendances fournies dans le fichier modèle **package.json**. Vous pouvez ajouter des dépendances supplémentaires comme vous le feriez dans n'importe quel autre

Exemple d'ajout de dépendances npm

```
npm install --save opossum
```

Lorsque le projet est construit pour être déployé, ces dépendances sont incluses dans l'image du conteneur d'exécution créé.

3. Votre projet doit contenir un fichier **src/index.js** qui exporte une fonction nommée **handle**.

4. Les scripts de tests d'intégration et d'unité sont fournis dans le cadre du modèle de fonction.

6.6.3. A propos de l'invocation de fonctions TypeScript

Lorsque vous utilisez le CLI Knative (**kn**) pour créer un projet de fonction, vous pouvez générer un projet qui répond aux CloudEvents ou à de simples requêtes HTTP. Dans Knative, les CloudEvents sont transportés par HTTP sous la forme d'une requête POST, de sorte que les deux types de fonctions écoutent les événements HTTP entrants et y répondent.

Les fonctions TypeScript peuvent être invoquées à l'aide d'une simple requête HTTP. Lorsqu'une requête est reçue, les fonctions sont invoquées avec un objet **context** comme premier paramètre.

6.6.3.1. Objets contextuels TypeScript

Pour invoquer une fonction, vous fournissez un objet **context** comme premier paramètre. L'accès aux propriétés de l'objet **context** peut fournir des informations sur la requête HTTP entrante.

Exemple d'objet contextuel

```
function handle(context:Context): string
```

Ces informations comprennent la méthode de requête HTTP, les chaînes de requête ou les en-têtes envoyés avec la requête, la version HTTP et le corps de la requête. Les demandes entrantes qui contiennent un **CloudEvent** attachent l'instance entrante du CloudEvent à l'objet de contexte de sorte qu'il est possible d'y accéder en utilisant **context.cloudevent**.

6.6.3.1.1. Méthodes de l'objet contextuel

L'objet **context** possède une seule méthode, **cloudEventResponse()**, qui accepte une valeur de données et renvoie un CloudEvent.

Dans un système Knative, si une fonction déployée en tant que service est invoquée par un courtier d'événements envoyant un CloudEvent, le courtier examine la réponse. Si la réponse est un CloudEvent, cet événement est traité par le courtier.

Exemple de méthode d'objet contextuel

```
// Expects to receive a CloudEvent with customer data
export function handle(context: Context, cloudevent?: CloudEvent): CloudEvent {
  // process the customer
  const customer = cloudevent.data;
```

```
  const processed = processCustomer(customer);
  return context.cloudEventResponse(customer)
    .source('/customer/process')
    .type('customer.processed')
    .response();
}
```

6.6.3.1.2. Types de contexte

Les fichiers de définition des types TypeScript exportent les types suivants pour les utiliser dans vos fonctions.

Définitions de type exportées

```
// Invokable is the expeted Function signature for user functions
export interface Invokable {
    (context: Context, cloudevent?: CloudEvent): any
}

// Logger can be used for structural logging to the console
export interface Logger {
  debug: (msg: any) => void,
  info:  (msg: any) => void,
  warn:  (msg: any) => void,
  error: (msg: any) => void,
  fatal: (msg: any) => void,
  trace: (msg: any) => void,
}

// Context represents the function invocation context, and provides
// access to the event itself as well as raw HTTP objects.
export interface Context {
    log: Logger;
    req: IncomingMessage;
    query?: Record<string, any>;
    body?: Record<string, any>|string;
    method: string;
    headers: IncomingHttpHeaders;
    httpVersion: string;
    httpVersionMajor: number;
    httpVersionMinor: number;
    cloudevent: CloudEvent;
    cloudEventResponse(data: string|object): CloudEventResponse;
}

// CloudEventResponse is a convenience class used to create
// CloudEvents on function returns
export interface CloudEventResponse {
    id(id: string): CloudEventResponse;
    source(source: string): CloudEventResponse;
    type(type: string): CloudEventResponse;
    version(version: string): CloudEventResponse;
    response(): CloudEvent;
}
```

6.6.3.1.3. Données CloudEvent

Si la demande entrante est un CloudEvent, toutes les données associées au CloudEvent sont extraites de l'événement et fournies en tant que deuxième paramètre. Par exemple, si un CloudEvent est reçu et qu'il contient une chaîne JSON dans sa propriété data qui est similaire à ce qui suit :

```
{
  "customerId": "0123456",
  "productId": "6543210"
}
```

Lorsqu'elle est invoquée, le deuxième paramètre de la fonction, après l'objet **context**, est un objet JavaScript qui possède les propriétés **customerId** et **productId**.

Exemple de signature

```
function handle(context: Context, cloudevent?: CloudEvent): CloudEvent
```

Dans cet exemple, le paramètre **cloudevent** est un objet JavaScript qui contient les propriétés **customerId** et **productId**.

6.6.4. Valeurs de retour des fonctions TypeScript

Les fonctions peuvent renvoyer n'importe quel type JavaScript valide ou ne pas avoir de valeur de retour. Lorsqu'une fonction n'a pas de valeur de retour spécifiée et qu'aucun échec n'est indiqué, l'appelant reçoit une réponse **204 No Content**.

Les fonctions peuvent également renvoyer un objet CloudEvent ou **Message** afin de pousser les événements dans le système Knative Eventing. Dans ce cas, le développeur n'est pas tenu de comprendre ou de mettre en œuvre la spécification de messagerie CloudEvent. Les en-têtes et autres informations pertinentes des valeurs renvoyées sont extraites et envoyées avec la réponse.

Exemple :

```
export const handle: Invokable = function (
  context: Context,
  cloudevent?: CloudEvent
): Message {
  // process customer and return a new CloudEvent
  const customer = cloudevent.data;
  return HTTP.binary(
    new CloudEvent({
      source: 'customer.processor',
      type: 'customer.processed'
    })
  );
};
```

6.6.4.1. Renvoi des en-têtes

Vous pouvez définir un en-tête de réponse en ajoutant une propriété **headers** à l'objet **return**. Ces en-têtes sont extraits et envoyés avec la réponse à l'appelant.

Exemple d'en-tête de réponse

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, any> {
  // process customer and return custom headers
  const customer = cloudevent.data as Record<string, any>;
  return { headers: { 'customer-id': customer.id } };
}
```

6.6.4.2. Renvoi des codes d'état

Vous pouvez définir un code d'état qui sera renvoyé à l'appelant en ajoutant une propriété **statusCode** à l'objet **return**:

Exemple de code d'état

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, any> {
  // process customer
  const customer = cloudevent.data as Record<string, any>;
  if (customer.restricted) {
    return {
      statusCode: 451
    }
  }
  // business logic, then
  return {
    statusCode: 240
  }
}
```

Des codes d'état peuvent également être définis pour les erreurs créées et déclenchées par la fonction :

Exemple de code d'état d'erreur

```
export function handle(context: Context, cloudevent?: CloudEvent): Record<string, string> {
  // process customer
  const customer = cloudevent.data as Record<string, any>;
  if (customer.restricted) {
    const err = new Error('Unavailable for legal reasons');
    err.statusCode = 451;
    throw err;
  }
}
```

6.6.5. Tester les fonctions TypeScript

Les fonctions TypeScript peuvent être testées localement sur votre ordinateur. Dans le projet par défaut qui est créé lorsque vous créez une fonction à l'aide de **kn func create**, il y a un dossier **test** qui contient quelques tests unitaires et d'intégration simples.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction en utilisant **kn func create**.

Procédure

1. Si vous n'avez pas encore effectué de tests, installez d'abord les dépendances :

```
$ npm install
```

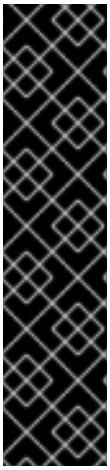
2. Naviguez jusqu'au dossier **test** de votre fonction.
3. Exécutez les tests :

```
$ npm test
```

6.6.6. Prochaines étapes

- Voir la documentation de référence de l'objet contextuel TypeScript.
- Construire et déployer une fonction.
- Voir la documentation de l'API Pino pour plus d'informations sur la journalisation avec des fonctions.

6.7. DÉVELOPPER DES FONCTIONS PYTHON



IMPORTANT

OpenShift Serverless Functions with Python est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Après avoir créé un projet de fonction Python, vous pouvez modifier les fichiers modèles fournis pour ajouter une logique commerciale à votre fonction. Il s'agit notamment de configurer l'invocation de la fonction ainsi que les en-têtes et les codes d'état renvoyés.

6.7.1. Conditions préalables

- Avant de pouvoir développer des fonctions, vous devez suivre les étapes de la section Configuration des fonctions OpenShift Serverless.

6.7.2. Structure du modèle de fonction Python

Lorsque vous créez une fonction Python en utilisant l'interface de programmation Knative (**kn**), le répertoire du projet ressemble à un projet Python classique. Les fonctions Python ont très peu de restrictions. Les seules exigences sont que votre projet contienne un fichier **func.py** qui contient une fonction **main()** et un fichier de configuration **func.yaml**.

Les développeurs ne sont pas limités aux dépendances fournies dans le fichier modèle **requirements.txt**. Des dépendances supplémentaires peuvent être ajoutées comme dans tout autre projet Python. Lorsque le projet est construit pour être déployé, ces dépendances seront incluses dans l'image du conteneur d'exécution créé.

Les fonctions de déclenchement **http** et **event** ont la même structure de modèle :

Structure du modèle

```
fn
├── func.py 1
├── func.yaml 2
├── requirements.txt 3
└── test_func.py 4
```

1. Contient une fonction **main()**.
2. Utilisé pour déterminer le nom et le registre de l'image.
3. Des dépendances supplémentaires peuvent être ajoutées au fichier **requirements.txt** comme dans tout autre projet Python.
4. Contient un test unitaire simple qui peut être utilisé pour tester votre fonction localement.

6.7.3. A propos de l'invocation des fonctions Python

Les fonctions Python peuvent être invoquées à l'aide d'une simple requête HTTP. Lorsqu'une requête est reçue, les fonctions sont invoquées avec un objet **context** comme premier paramètre.

L'objet **context** est une classe Python dotée de deux attributs :

- L'attribut **request** est toujours présent et contient l'objet Flask **request**.
- Le deuxième attribut, **cloud_event**, est renseigné si la demande entrante est un objet **CloudEvent**.

Les développeurs peuvent accéder à toutes les données de **CloudEvent** à partir de l'objet contextuel.

Exemple d'objet contextuel

```
def main(context: Context):
    """
    The context parameter contains the Flask request object and any
    CloudEvent received with the request.
    """
    print(f"Method: {context.request.method}")
    print(f"Event data {context.cloud_event.data}")
    # ... business logic here
```

6.7.4. Valeurs de retour des fonctions Python

Les fonctions peuvent renvoyer n'importe quelle valeur prise en charge par Flask. En effet, le cadre d'invocation transmet ces valeurs directement au serveur Flask.

Exemple :

```
def main(context: Context):
    body = { "message": "Howdy!" }
    headers = { "content-type": "application/json" }
    return body, 200, headers
```

Les fonctions peuvent définir des en-têtes et des codes de réponse en tant que valeurs de réponse secondaires et tertiaires à partir de l'invocation de la fonction.

6.7.4.1. Retourner les CloudEvents

Les développeurs peuvent utiliser le décorateur **@event** pour indiquer à l'invocateur que la valeur de retour de la fonction doit être convertie en CloudEvent avant d'envoyer la réponse.

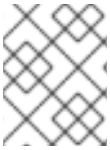
Exemple :

```
@event("event_source"="/my/function", "event_type"="my.type")
def main(context):
    # business logic here
    data = do_something()
    # more data processing
    return data
```

Cet exemple envoie un CloudEvent comme valeur de réponse, avec un type de **"my.type"** et une source de **"/my/function"**. La propriété CloudEvent **data** est définie sur la variable **data** renvoyée. Les attributs décoratifs **event_source** et **event_type** sont tous deux facultatifs.

6.7.5. Test des fonctions Python

Vous pouvez tester les fonctions Python localement sur votre ordinateur. Le projet par défaut contient un fichier **test_func.py**, qui fournit un test unitaire simple pour les fonctions.



NOTE

Le cadre de test par défaut pour les fonctions Python est **unittest**. Vous pouvez utiliser un autre cadre de test si vous le souhaitez.

Conditions préalables

- Pour exécuter localement des tests de fonctions Python, vous devez installer les dépendances nécessaires :

```
$ pip install -r requirements.txt
```

Procédure

1. Naviguez jusqu'au dossier de votre fonction qui contient le fichier **test_func.py**.
2. Exécutez les tests :

```
$ python3 test_func.py
```

6.7.6. Prochaines étapes

- Construire et déployer une fonction.

6.8. UTILISATION DES FONCTIONS AVEC KNATIVE EVENTING

Les fonctions sont déployées en tant que services Knative sur un cluster OpenShift Container Platform. Vous pouvez connecter les fonctions aux composants Knative Eventing afin qu'elles puissent recevoir des événements entrants.

6.8.1. Connecter une source d'événements à une fonction à l'aide de la perspective du développeur

Les fonctions sont déployées en tant que services Knative sur un cluster OpenShift Container Platform. Lorsque vous créez une source d'événement à l'aide de la console web d'OpenShift Container Platform, vous pouvez spécifier une fonction déployée à laquelle les événements sont envoyés à partir de cette source.

Conditions préalables

- OpenShift Serverless Operator, Knative Serving et Knative Eventing sont installés sur votre cluster OpenShift Container Platform.
- Vous vous êtes connecté à la console web et vous vous trouvez dans la perspective **Developer**.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez créé et déployé une fonction.

Procédure

1. Créez une source d'événement de n'importe quel type, en naviguant vers **Add → Event Source** et en sélectionnant le type de source d'événement que vous souhaitez créer.
2. Dans la section **Sink** de la vue du formulaire **Create Event Source**, sélectionnez votre fonction dans la liste **Resource**.
3. Cliquez sur **Create**.

Vérification

Vous pouvez vérifier que la source d'événement a été créée et qu'elle est connectée à la fonction en consultant la page **Topology**.

1. Dans la perspective **Developer**, naviguez jusqu'à **Topology**.
2. Affichez la source de l'événement et cliquez sur la fonction connectée pour afficher les détails de la fonction dans le panneau de droite.

6.9. CONFIGURATION DU PROJET DE FONCTION DANS FUNC.YAML

Le fichier **func.yaml** contient la configuration de votre projet de fonction. Les valeurs spécifiées dans **func.yaml** sont utilisées lorsque vous exécutez une commande **kn func**. Par exemple, lorsque vous exécutez la commande **kn func build**, la valeur du champ **build** est utilisée. Dans certains cas, vous

pouvez remplacer ces valeurs par des drapeaux de ligne de commande ou des variables d'environnement.

6.9.1. Champs configurables dans func.yaml

La plupart des champs de **func.yaml** sont générés automatiquement lorsque vous créez, construisez et déployez votre fonction. Cependant, il existe également des champs que vous modifiez manuellement pour changer certaines choses, comme le nom de la fonction ou le nom de l'image.

6.9.1.1. buildEnvs

Le champ **buildEnvs** vous permet de définir des variables d'environnement qui seront disponibles pour l'environnement qui construit votre fonction. Contrairement aux variables définies à l'aide de **envs**, une variable définie à l'aide de **buildEnv** n'est pas disponible pendant l'exécution de la fonction.

Vous pouvez définir une variable **buildEnv** directement à partir d'une valeur. Dans l'exemple suivant, la variable **buildEnv** nommée **EXAMPLE1** se voit directement attribuer la valeur **one**:

```
buildEnvs:
- name: EXAMPLE1
  value: one
```

Vous pouvez également définir une variable **buildEnv** à partir d'une variable d'environnement locale. Dans l'exemple suivant, la variable **buildEnv** nommée **EXAMPLE2** se voit attribuer la valeur de la variable d'environnement locale **LOCAL_ENV_VAR**:

```
buildEnvs:
- name: EXAMPLE1
  value: '{{ env:LOCAL_ENV_VAR }}'
```

6.9.1.2. envs

Le champ **envs** vous permet de définir des variables d'environnement qui seront disponibles pour votre fonction au moment de l'exécution. Vous pouvez définir une variable d'environnement de différentes manières :

1. Directement à partir d'une valeur.
2. À partir d'une valeur assignée à une variable d'environnement locale. Voir la section "Referencing local environment variables from func.yaml fields" pour plus d'informations.
3. À partir d'une paire clé-valeur stockée dans un secret ou une carte de configuration.
4. Vous pouvez également importer toutes les paires clé-valeur stockées dans un secret ou une carte de configuration, les clés étant utilisées comme noms des variables d'environnement créées.

Cet exemple montre les différentes façons de définir une variable d'environnement :

```
name: test
namespace: ""
runtime: go
...
envs:
```

```
- name: EXAMPLE1 (1)
  value: value
- name: EXAMPLE2 (2)
  value: '{{ env:LOCAL_ENV_VALUE }}'
- name: EXAMPLE3 (3)
  value: '{{ secret:mysecret:key }}'
- name: EXAMPLE4 (4)
  value: '{{ configMap:myconfigmap:key }}'
- value: '{{ secret:mysecret2 }}' (5)
- value: '{{ configMap:myconfigmap2 }}' (6)
```

1. Variable d'environnement définie directement à partir d'une valeur.
2. Variable d'environnement définie à partir d'une valeur attribuée à une variable d'environnement locale.
3. Variable d'environnement attribuée à partir d'une paire clé-valeur stockée dans un secret.
4. Variable d'environnement attribuée à partir d'une paire clé-valeur stockée dans une carte de configuration.
5. Un ensemble de variables d'environnement importées à partir de paires clé-valeur d'un secret.
6. Un ensemble de variables d'environnement importées à partir des paires clé-valeur d'une carte de configuration.

6.9.1.3. constructeur

Le champ **builder** indique la stratégie utilisée par la fonction pour construire l'image. Il accepte les valeurs **pack** ou **s2i**.

6.9.1.4. construire

Le champ **build** indique comment la fonction doit être construite. La valeur **local** indique que la fonction est construite localement sur votre machine. La valeur **git** indique que la fonction est construite sur un cluster en utilisant les valeurs spécifiées dans le champ **git**.

6.9.1.5. volumes

Le champ **volumes** vous permet de monter les secrets et les cartes de configuration en tant que volume accessible à la fonction au chemin spécifié, comme le montre l'exemple suivant :

```
name: test
namespace: ""
runtime: go
...
volumes:
- secret: mysecret (1)
  path: /workspace/secret
- configMap: myconfigmap (2)
  path: /workspace/configmap
```

1. Le secret **mysecret** est monté en tant que volume résidant sur **/workspace/secret**.

2 La carte de configuration **myconfigmap** est montée en tant que volume résidant à l'adresse **/workspace/configmap**.

6.9.1.6. options

Le champ **options** vous permet de modifier les propriétés du service Knative pour la fonction déployée, telles que l'autoscaling. Si ces options ne sont pas définies, les options par défaut sont utilisées.

Les options suivantes sont disponibles :

- **scale**
  - **min**: Le nombre minimum de répliques. Doit être un nombre entier non négatif. La valeur par défaut est 0.
  - **max**: Nombre maximal de répliques. Il doit s'agir d'un nombre entier non négatif. La valeur par défaut est 0, ce qui signifie qu'il n'y a pas de limite.
  - **metric**: Définit quel type de métrique est surveillé par l'Autoscaler. Il peut être défini sur **concurrency**, qui est la valeur par défaut, ou sur **rps**.
  - **target**: Recommandation sur le moment de la mise à l'échelle en fonction du nombre de requêtes entrantes simultanées. L'option **target** peut être une valeur flottante supérieure à 0,01. La valeur par défaut est 100, sauf si l'option **options.resources.limits.concurrency** est définie, auquel cas **target** prend sa valeur par défaut.
  - **utilization**: Pourcentage d'utilisation de requêtes simultanées autorisé avant la mise à l'échelle. Il peut s'agir d'une valeur flottante comprise entre 1 et 100. La valeur par défaut est 70.
- **resources**
  - **requests**
    - **cpu**: Une demande de ressource CPU pour le conteneur avec la fonction déployée.
    - **memory**: Une demande de ressource mémoire pour le conteneur avec la fonction déployée.
  - **limits**
    - **cpu**: Une limite de ressources CPU pour le conteneur avec la fonction déployée.
    - **memory**: Une limite de ressources mémoire pour le conteneur avec la fonction déployée.
    - **concurrency**: Limite stricte du nombre de requêtes simultanées pouvant être traitées par une seule réplique. Il peut s'agir d'une valeur entière supérieure ou égale à 0. La valeur par défaut est 0, ce qui signifie qu'il n'y a pas de limite.

Voici un exemple de configuration des options de **scale**:

```
name: test
namespace: ""
runtime: go
...
```

```
options:
  scale:
    min: 0
    max: 10
    metric: concurrency
    target: 75
    utilization: 75
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 1000m
      memory: 256Mi
      concurrency: 100
```

6.9.1.7. image

Le champ **image** définit le nom de l'image de votre fonction après sa construction. Vous pouvez modifier ce champ. Si vous le faites, la prochaine fois que vous exécuterez **kn func build** ou **kn func deploy**, l'image de la fonction sera créée avec le nouveau nom.

6.9.1.8. imageDigest

Le champ **imageDigest** contient le hachage SHA256 du manifeste de l'image lorsque la fonction est déployée. Ne pas modifier cette valeur.

6.9.1.9. étiquettes

Le champ **labels** vous permet de définir des étiquettes sur une fonction déployée.

Vous pouvez définir une étiquette directement à partir d'une valeur. Dans l'exemple suivant, l'étiquette de la clé **role** se voit directement attribuer la valeur **backend**:

```
labels:
- key: role
  value: backend
```

Vous pouvez également définir une étiquette à partir d'une variable d'environnement locale. Dans l'exemple suivant, l'étiquette de la touche **author** se voit attribuer la valeur de la variable d'environnement locale **USER**:

```
labels:
- key: author
  value: '{{ env:USER }}'
```

6.9.1.10. nom

Le champ **name** définit le nom de votre fonction. Cette valeur est utilisée comme nom de votre service Knative lorsqu'il est déployé. Vous pouvez modifier ce champ pour renommer la fonction lors des déploiements ultérieurs.

6.9.1.11. espace de noms

Le champ **namespace** indique l'espace de noms dans lequel votre fonction est déployée.

6.9.1.12. durée d'exécution

Le champ **runtime** indique la langue d'exécution de votre fonction, par exemple **python**.

6.9.2. Référencement des variables d'environnement locales à partir des champs de func.yaml

Si vous souhaitez éviter de stocker des informations sensibles telles qu'une clé API dans la configuration de la fonction, vous pouvez ajouter une référence à une variable d'environnement disponible dans l'environnement local. Pour ce faire, modifiez le champ **envs** dans le fichier **func.yaml**.

Conditions préalables

- Vous devez avoir créé le projet de fonction.
- L'environnement local doit contenir la variable à laquelle vous souhaitez faire référence.

Procédure

- Pour faire référence à une variable d'environnement locale, utilisez la syntaxe suivante :

  ```
  {{ env:ENV_VAR }}
  ```

  Remplacez **ENV_VAR** par le nom de la variable de l'environnement local que vous souhaitez utiliser.

  Par exemple, vous pouvez disposer de la variable **API_KEY** dans l'environnement local. Vous pouvez assigner sa valeur à la variable **MY_API_KEY**, que vous pouvez ensuite utiliser directement dans votre fonction :

  Exemple de fonction

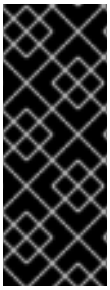
  ```
  name: test
  namespace: ""
  runtime: go
  ...
  envs:
  - name: MY_API_KEY
    value: '{{ env:API_KEY }}'
  ...
  ```

6.9.3. Ressources supplémentaires

- Commencer avec les fonctions
- Accéder aux secrets et aux cartes de configuration à partir de fonctions Serverless
- Documentation Knative sur l'Autoscaling
- Documentation Kubernetes sur la gestion des ressources pour les conteneurs
- Documentation Knative sur la configuration de la concurrence

6.10. ACCÈS AUX SECRETS ET AUX CARTES DE CONFIGURATION À PARTIR DE FONCTIONS

Une fois que vos fonctions ont été déployées dans le cluster, elles peuvent accéder aux données stockées dans les secrets et les cartes de configuration. Ces données peuvent être montées en tant que volumes ou assignées à des variables d'environnement. Vous pouvez configurer cet accès de manière interactive en utilisant le CLI Knative, ou manuellement en éditant le fichier YAML de configuration de la fonction.



IMPORTANT

Pour accéder aux secrets et aux cartes de configuration, la fonction doit être déployée sur le cluster. Cette fonctionnalité n'est pas disponible pour une fonction exécutée localement.

Si une valeur de secret ou de carte de configuration n'est pas accessible, le déploiement échoue avec un message d'erreur spécifiant les valeurs inaccessibles.

6.10.1. Modifier l'accès des fonctions aux secrets et aux cartes de configuration de manière interactive

Vous pouvez gérer les secrets et les cartes de configuration auxquels votre fonction accède en utilisant l'utilitaire interactif **kn func config**. Les opérations disponibles comprennent l'énumération, l'ajout et la suppression des valeurs stockées dans les cartes de configuration et les secrets en tant que variables d'environnement, ainsi que l'énumération, l'ajout et la suppression des volumes. Cette fonctionnalité vous permet de gérer les données stockées sur le cluster qui sont accessibles à votre fonction.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Exécutez la commande suivante dans le répertoire du projet de la fonction :

```
$ kn func config
```

   Vous pouvez également spécifier le répertoire du projet de la fonction à l'aide de l'option **--path** ou **-p**.

2. Utilisez l'interface interactive pour effectuer l'opération nécessaire. Par exemple, l'utilisation de l'utilitaire pour dresser la liste des volumes configurés produit un résultat similaire à celui-ci :

```
$ kn func config
? What do you want to configure? Volumes
? What operation do you want to perform? List
Configured Volumes mounts:
- Secret "mysecret" mounted at path: "/workspace/secret"
- Secret "mysecret2" mounted at path: "/workspace/secret2"
```

Ce schéma montre toutes les opérations disponibles dans l'utilitaire interactif et comment y accéder :

```
kn func config
   ├─> Environment variables
   │           ├─> Add
   │           │   ├─> ConfigMap: Add all key-value pairs from a config map
   │           │   ├─> ConfigMap: Add value from a key in a config map
   │           │   ├─> Secret: Add all key-value pairs from a secret
   │           │   └─> Secret: Add value from a key in a secret
   │           ├─> List: List all configured environment variables
   │           └─> Remove: Remove a configured environment variable
   └─> Volumes
         ├─> Add
         │   ├─> ConfigMap: Mount a config map as a volume
         │   └─> Secret: Mount a secret as a volume
         ├─> List: List all configured volumes
         └─> Remove: Remove a configured volume
```

3. Facultatif. Déployez la fonction pour que les modifications soient prises en compte :

```
$ kn func deploy -p test
```

6.10.2. Modifier l'accès des fonctions aux secrets et aux cartes de configuration de manière interactive en utilisant des commandes spécialisées

Chaque fois que vous lancez l'utilitaire **kn func config**, vous devez parcourir l'ensemble de la boîte de dialogue pour sélectionner l'opération dont vous avez besoin, comme indiqué dans la section précédente. Pour gagner du temps, vous pouvez exécuter directement une opération spécifique en lançant une forme plus spécifique de la commande **kn func config**:

- Pour dresser la liste des variables d'environnement configurées :

```
$ kn func config envs [-p <function-project-path>]
```

- Pour ajouter des variables d'environnement à la configuration de la fonction :

```
$ kn func config envs add [-p <function-project-path>]
```

- Pour supprimer les variables d'environnement de la configuration de la fonction :

```
$ kn func config envs remove [-p <function-project-path>]
```

- Pour dresser la liste des volumes configurés :

```
$ kn func config volumes [-p <function-project-path>]
```

- Pour ajouter un volume à la configuration des fonctions :

```
$ kn func config volumes add [-p <function-project-path>]
```

- Pour supprimer un volume de la configuration des fonctions :

```
$ kn func config volumes remove [-p <function-project-path>]
```

6.10.3. Ajout d'un accès fonctionnel aux secrets et aux cartes de configuration manuellement

Vous pouvez ajouter manuellement la configuration pour accéder aux secrets et aux cartes de configuration à votre fonction. Cela peut être préférable à l'utilisation de l'utilitaire et des commandes interactives **kn func config**, par exemple lorsque vous disposez d'un extrait de configuration existant.

6.10.3.1. Montage d'un secret en tant que volume

Vous pouvez monter un secret comme un volume. Une fois qu'un secret est monté, vous pouvez y accéder à partir de la fonction comme un fichier normal. Cela vous permet de stocker sur le cluster des données nécessaires à la fonction, par exemple une liste d'URI auxquels la fonction doit accéder.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque secret que vous souhaitez monter en tant que volume, ajoutez le fichier YAML suivant à la section **volumes**:

```
name: test
namespace: ""
runtime: go
...
volumes:
- secret: mysecret
  path: /workspace/secret
```

   - Remplacer **mysecret** par le nom du secret cible.
   - Remplacez /**workspace**/**secret** par le chemin où vous souhaitez monter le secret. Par exemple, pour monter le secret **addresses**, utilisez le langage YAML suivant :

```
name: test
namespace: ""
runtime: go
...
volumes:
- configMap: addresses
  path: /workspace/secret-addresses
```

3. Sauvegarder la configuration.

6.10.3.2. Montage d'une carte de configuration en tant que volume

Vous pouvez monter une carte de configuration comme un volume. Une fois qu'une carte de configuration est montée, vous pouvez y accéder à partir de la fonction comme un fichier normal. Cela vous permet de stocker sur le cluster les données nécessaires à la fonction, par exemple une liste d'URI auxquels la fonction doit accéder.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque carte de configuration que vous souhaitez monter en tant que volume, ajoutez le fichier YAML suivant à la section **volumes**:

```
name: test
namespace: ""
runtime: go
...
volumes:
- configMap: myconfigmap
  path: /workspace/configmap
```

   - Remplacez **myconfigmap** par le nom de la carte de configuration cible.
   - Remplacez /**workspace**/**configmap** par le chemin où vous souhaitez monter la carte de configuration.
     Par exemple, pour monter la carte de configuration **addresses**, utilisez le fichier YAML suivant :

```
name: test
namespace: ""
runtime: go
...
volumes:
- configMap: addresses
  path: /workspace/configmap-addresses
```

3. Sauvegarder la configuration.

6.10.3.3. Définition d'une variable d'environnement à partir d'une valeur clé définie dans un secret

Vous pouvez définir une variable d'environnement à partir d'une valeur clé définie comme un secret. Une valeur précédemment stockée dans un secret peut alors être accédée en tant que variable d'environnement par la fonction au moment de l'exécution. Cela peut être utile pour accéder à une valeur stockée dans un secret, comme l'identifiant d'un utilisateur.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque valeur d'une paire clé-valeur secrète que vous souhaitez affecter à une variable d'environnement, ajoutez le fichier YAML suivant à la section **envs**:

```
name: test
namespace: ""
runtime: go
...
envs:
- name: EXAMPLE
  value: '{{ secret:mysecret:key }}'
```

   - Remplacez **EXAMPLE** par le nom de la variable d'environnement.
   - Remplacer **mysecret** par le nom du secret cible.
   - Remplacer **key** par la clé correspondant à la valeur cible.
     Par exemple, pour accéder à l'identifiant de l'utilisateur stocké dans **userdetailssecret**, utilisez le langage YAML suivant :

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailssecret:userid }}'
```

3. Sauvegarder la configuration.

6.10.3.4. Définition d'une variable d'environnement à partir d'une valeur clé définie dans une carte de configuration

Vous pouvez définir une variable d'environnement à partir d'une valeur clé définie dans une carte de configuration. Une valeur précédemment stockée dans une carte de configuration peut alors être accédée en tant que variable d'environnement par la fonction au moment de l'exécution. Cela peut être utile pour accéder à une valeur stockée dans une carte de configuration, telle que l'identifiant d'un utilisateur.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque valeur d'une paire clé-valeur de la carte de configuration que vous souhaitez affecter à une variable d'environnement, ajoutez le fichier YAML suivant à la section **envs**:

```
name: test
namespace: ""
runtime: go
...
envs:
- name: EXAMPLE
  value: '{{ configMap:myconfigmap:key }}'
```

   - Remplacez **EXAMPLE** par le nom de la variable d'environnement.
   - Remplacez **myconfigmap** par le nom de la carte de configuration cible.
   - Remplacer **key** par la clé correspondant à la valeur cible.
     Par exemple, pour accéder à l'identifiant de l'utilisateur stocké dans **userdetailsmap**, utilisez le langage YAML suivant :

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailsmap:userid }}'
```

3. Sauvegarder la configuration.

6.10.3.5. Définition de variables d'environnement à partir de toutes les valeurs définies dans un secret

Vous pouvez définir une variable d'environnement à partir de toutes les valeurs définies dans un secret. Les valeurs précédemment stockées dans un secret peuvent alors être accédées en tant que variables d'environnement par la fonction au moment de l'exécution. Cela peut être utile pour accéder simultanément à une collection de valeurs stockées dans un secret, par exemple, un ensemble de données relatives à un utilisateur.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.

2. Pour chaque secret pour lequel vous souhaitez importer toutes les paires clé-valeur en tant que variables d'environnement, ajoutez le YAML suivant à la section **envs**:

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ secret:mysecret }}' 1
```

1 Remplacer **mysecret** par le nom du secret cible.

Par exemple, pour accéder à toutes les données de l'utilisateur qui sont stockées dans **userdetailssecret**, utilisez le langage YAML suivant :

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailssecret }}'
```

3. Sauvegarder la configuration.

6.10.3.6. Définition de variables d'environnement à partir de toutes les valeurs définies dans une carte de configuration

Vous pouvez définir une variable d'environnement à partir de toutes les valeurs définies dans une carte de configuration. Les valeurs précédemment stockées dans une carte de configuration peuvent alors être accédées en tant que variables d'environnement par la fonction au moment de l'exécution. Cela peut être utile pour accéder simultanément à une collection de valeurs stockées dans une carte de configuration, par exemple, un ensemble de données relatives à un utilisateur.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque carte de configuration pour laquelle vous souhaitez importer toutes les paires clé-valeur en tant que variables d'environnement, ajoutez le YAML suivant à la section **envs**:

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:myconfigmap }}' 1
```

1 Remplacez **myconfigmap** par le nom de la carte de configuration cible.

Par exemple, pour accéder à toutes les données de l'utilisateur qui sont stockées dans **userdetailsmap**, utilisez le langage YAML suivant :

```
name: test
namespace: ""
runtime: go
...
envs:
- value: '{{ configMap:userdetailsmap }}'
```

3. Enregistrer le fichier.

6.11. AJOUTER DES ANNOTATIONS AUX FONCTIONS

Vous pouvez ajouter des annotations Kubernetes à une fonction Serverless déployée. Les annotations vous permettent d'attacher des métadonnées arbitraires à une fonction, par exemple, une note sur l'objectif de la fonction. Les annotations sont ajoutées à la section **annotations** du fichier de configuration **func.yaml**.

La fonction d'annotation des fonctions présente deux limites :

- Une fois qu'une annotation de fonction se propage au service Knative correspondant sur le cluster, elle ne peut pas être supprimée du service en la supprimant du fichier **func.yaml**. Vous devez supprimer l'annotation du service Knative en modifiant directement le fichier YAML du service ou en utilisant la console web de OpenShift Container Platform.
- Vous ne pouvez pas définir des annotations qui sont définies par Knative, par exemple les annotations **autoscaling**.

6.11.1. Ajouter des annotations à une fonction

Vous pouvez ajouter des annotations à une fonction. À l'instar d'une étiquette, une annotation est définie comme une carte clé-valeur. Les annotations sont utiles, par exemple, pour fournir des métadonnées sur une fonction, comme l'auteur de la fonction.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé une fonction.

Procédure

1. Ouvrez le fichier **func.yaml** de votre fonction.
2. Pour chaque annotation que vous souhaitez ajouter, ajoutez le YAML suivant à la section **annotations**:

```
name: test
```

```
namespace: ""
runtime: go
...
annotations:
  <annotation_name>: "<annotation_value>" 1
```

1 Remplacez **<annotation_name>: "<annotation_value>"** par votre annotation.

Par exemple, pour indiquer qu'une fonction a été rédigée par Alice, vous pouvez inclure l'annotation suivante :

```
name: test
namespace: ""
runtime: go
...
annotations:
  author: "alice@example.com"
```

3. Sauvegarder la configuration.

La prochaine fois que vous déployez votre fonction sur le cluster, les annotations sont ajoutées au service Knative correspondant.

6.12. GUIDE DE RÉFÉRENCE POUR LE DÉVELOPPEMENT DES FONCTIONS

OpenShift Serverless Functions fournit des modèles qui peuvent être utilisés pour créer des fonctions de base. Un modèle initie le boilerplate du projet de fonction et le prépare à être utilisé avec l'outil **kn func**. Chaque modèle de fonction est adapté à un runtime spécifique et suit ses conventions. Avec un modèle, vous pouvez lancer votre projet de fonction automatiquement.

Des modèles sont disponibles pour les durées d'exécution suivantes :

- Node.js
- Quarkus
- TypeScript

6.12.1. Référence d'un objet contextuel Node.js

L'objet **context** possède plusieurs propriétés auxquelles le développeur de la fonction peut accéder. L'accès à ces propriétés permet de fournir des informations sur les requêtes HTTP et d'écrire des données dans les journaux de la grappe.

6.12.1.1. journal

Fournit un objet de journalisation qui peut être utilisé pour écrire des données dans les journaux de la grappe. Le journal est conforme à l'API de journalisation Pino.

Exemple de journal

```
function handle(context) {
  context.log.info(“Processing customer”);
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.function.com'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"Pr
ocessing customer"}
```

Vous pouvez modifier le niveau de journalisation en choisissant l'une des valeurs suivantes : **fatal**, **error**, **warn**, **info**, **debug**, **trace** ou **silent**. Pour ce faire, modifiez la valeur de **logLevel** en attribuant l'une de ces valeurs à la variable d'environnement **FUNC_LOG_LEVEL** à l'aide de la commande **config**.

6.12.1.2. interrogation

Renvoie la chaîne de requête de la demande, le cas échéant, sous forme de paires clé-valeur. Ces attributs se trouvent également sur l'objet contextuel lui-même.

Exemple de requête

```
function handle(context) {
  // Log the 'name' query parameter
  context.log.info(context.query.name);
  // Query parameters are also attached to the context
  context.log.info(context.name);
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.com?name=tiger'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"tig
er"}
```

6.12.1.3. corps

Renvoie le corps de la demande s'il y en a un. Si le corps de la demande contient du code JSON, celui-ci sera analysé afin que les attributs soient directement disponibles.

Exemple de corps

```
function handle(context) {
```

```
  // log the incoming request body's 'hello' parameter
  context.log.info(context.body.hello);
}
```

Vous pouvez accéder à la fonction en utilisant la commande **curl** pour l'invoquer :

Example command

```
$ kn func invoke -d '{"Hello": "world"}'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"world"}
```

6.12.1.4. en-têtes

Renvoie les en-têtes de la requête HTTP sous forme d'objet.

Exemple d'en-tête

```
function handle(context) {
  context.log.info(context.headers["custom-header"]);
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.function.com'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"some-value"}
```

6.12.1.5. Requêtes HTTP

méthode
: Renvoie la méthode de requête HTTP sous la forme d'une chaîne de caractères.

httpVersion
: Renvoie la version HTTP sous forme de chaîne de caractères.

httpVersionMajor
: Renvoie le numéro de version majeure de HTTP sous la forme d'une chaîne de caractères.

httpVersionMinor
: Renvoie le numéro de version mineure HTTP sous forme de chaîne de caractères.

6.12.2. Référence d'un objet contextuel TypeScript

L'objet **context** possède plusieurs propriétés auxquelles le développeur de la fonction peut accéder. L'accès à ces propriétés permet de fournir des informations sur les requêtes HTTP entrantes et d'écrire des données dans les journaux de la grappe.

6.12.2.1. journal

Fournit un objet de journalisation qui peut être utilisé pour écrire des données dans les journaux de la grappe. Le journal est conforme à l'API de journalisation Pino.

Exemple de journal

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.body as Record<string, string>).hello);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.function.com'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"Pr
ocessing customer"}
```

Vous pouvez modifier le niveau de journalisation en choisissant l'une des valeurs suivantes : **fatal**, **error**, **warn**, **info**, **debug**, **trace** ou **silent**. Pour ce faire, modifiez la valeur de **logLevel** en attribuant l'une de ces valeurs à la variable d'environnement **FUNC_LOG_LEVEL** à l'aide de la commande **config**.

6.12.2.2. interrogation

Renvoie la chaîne de requête de la demande, le cas échéant, sous forme de paires clé-valeur. Ces attributs se trouvent également sur l'objet contextuel lui-même.

Exemple de requête

```
export function handle(context: Context): string {
    // log the 'name' query parameter
  if (context.query) {
    context.log.info((context.query as Record<string, string>).name);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.function.com' --data '{"name": "tiger"}'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"tiger"}
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"tiger"}
```

6.12.2.3. corps

Renvoie le corps de la demande, le cas échéant. Si le corps de la demande contient du code JSON, celui-ci sera analysé de manière à ce que les attributs soient directement disponibles.

Exemple de corps

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.body as Record<string, string>).hello);
  } else {
    context.log.info('No data received');
  }
  return 'OK';
}
```

Vous pouvez accéder à la fonction en utilisant la commande **kn func invoke**:

Example command

```
$ kn func invoke --target 'http://example.function.com' --data '{"hello": "world"}'
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"world"}
```

6.12.2.4. en-têtes

Renvoie les en-têtes de la requête HTTP sous forme d'objet.

Exemple d'en-tête

```
export function handle(context: Context): string {
  // log the incoming request body's 'hello' parameter
  if (context.body) {
    context.log.info((context.headers as Record<string, string>)['custom-header']);
  } else {
```

```
    context.log.info('No data received');
  }
  return 'OK';
}
```

Vous pouvez accéder à la fonction en utilisant la commande **curl** pour l'invoquer :

Example command

```
$ curl -H'x-custom-header: some-value" http://example.function.com
```

Exemple de sortie

```
{"level":30,"time":1604511655265,"pid":3430203,"hostname":"localhost.localdomain","reqId":1,"msg":"so
me-value"}
```

6.12.2.5. Requêtes HTTP

méthode

Renvoie la méthode de requête HTTP sous la forme d'une chaîne de caractères.

httpVersion

Renvoie la version HTTP sous forme de chaîne de caractères.

httpVersionMajor

Renvoie le numéro de version majeure de HTTP sous la forme d'une chaîne de caractères.

httpVersionMinor

Renvoie le numéro de version mineure HTTP sous forme de chaîne de caractères.

CHAPITRE 7. CLI KNATIVE

7.1. COMMANDES CLI DE KNATIVE SERVING

7.1.1. commandes de service kn

Vous pouvez utiliser les commandes suivantes pour créer et gérer les services Knative.

7.1.1.1. Créer des applications sans serveur en utilisant le CLI Knative

L'utilisation de la CLI Knative (**kn**) pour créer des applications sans serveur offre une interface utilisateur plus rationalisée et plus intuitive que la modification directe des fichiers YAML. Vous pouvez utiliser la commande **kn service create** pour créer une application sans serveur de base.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Créer un service Knative :

```
$ kn service create <service-name> --image <image> --tag <tag-value>
```

Où ?

- **--image** est l'URI de l'image pour l'application.
- **--tag** est un indicateur facultatif qui peut être utilisé pour ajouter une étiquette à la révision initiale créée avec le service.

Example command

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

Exemple de sortie

```
Creating service 'event-display' in namespace 'default':

  0.271s The Route is still working to reflect the latest desired specification.
  0.580s Configuration "event-display" is waiting for a Revision to become ready.
  3.857s ...
  3.861s Ingress has not yet been reconciled.
  4.270s Ready to serve.

Service 'event-display' created with latest revision 'event-display-bxshg-1' and URL:
http://event-display-default.apps-crc.testing
```

-

7.1.1.2. Mettre à jour des applications sans serveur en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn service update** pour des sessions interactives sur la ligne de commande lorsque vous construisez un service de manière incrémentale. Contrairement à la commande **kn service apply**, lorsque vous utilisez la commande **kn service update**, vous ne devez spécifier que les changements que vous souhaitez mettre à jour, plutôt que la configuration complète du service Knative.

Exemple de commandes

- Mettre à jour un service en ajoutant une nouvelle variable d'environnement :

```
$ kn service update <service_name> --env <key>=<value>
```

- Mettre à jour un service en ajoutant un nouveau port :

```
$ kn service update <service_name> --port 80
```

- Mettre à jour un service en ajoutant de nouveaux paramètres de demande et de limite :

```
$ kn service update <service_name> --request cpu=500m --limit memory=1024Mi --limit
cpu=1000m
```

- Attribuer la balise **latest** à une révision :

```
$ kn service update -YRFFGUNA nom_du_service> --tag -YRFFGUNA
nom_de_la_révision>=latest
```

- Mettre à jour une balise de **testing** à **staging** pour la dernière révision **READY** d'un service :

```
$ kn service update <service_name> --untag testing --tag @latest=staging
```

- Ajoutez la balise **test** à une révision qui reçoit 10 % du trafic, et envoyez le reste du trafic à la dernière révision **READY** d'un service :

```
$ kn service update -YRFFGUNA nom_du_service> --tag -YRFFGUNA
nom_de_la_révision>=test --traffic test=10,@latest=90
```

7.1.1.3. Application des déclarations de service

Vous pouvez configurer un service Knative de manière déclarative en utilisant la commande **kn service apply**. Si le service n'existe pas, il est créé, sinon le service existant est mis à jour avec les options qui ont été modifiées.

La commande **kn service apply** est particulièrement utile pour les scripts shell ou dans un pipeline d'intégration continue, où les utilisateurs souhaitent généralement spécifier entièrement l'état du service en une seule commande pour déclarer l'état cible.

Lorsque vous utilisez **kn service apply**, vous devez fournir la configuration complète du service Knative. Cela diffère de la commande **kn service update**, qui vous demande seulement de spécifier dans la commande les options que vous souhaitez mettre à jour.

Exemple de commandes

- Créer un service :

```
kn service apply <service_name> --image <image> $ kn service apply <service_name> --
image <image>
```

- Ajouter une variable d'environnement à un service :

```
$ kn service apply <service_name> --image <image> --env <key>=<value>
```

- Lire la déclaration de service à partir d'un fichier JSON ou YAML :

```
$ kn service apply <service_name> -f <filename>
```

7.1.1.4. Décrire des applications sans serveur en utilisant le CLI Knative

Vous pouvez décrire un service Knative en utilisant la commande **kn service describe**.

Exemple de commandes

- Décrire un service :

```
kn service describe --verbose <service_name>
```

Le drapeau **--verbose** est facultatif mais peut être inclus pour fournir une description plus détaillée. La différence entre une sortie normale et une sortie verbeuse est illustrée dans les exemples suivants :

Exemple de sortie sans l'indicateur --verbose

```
Name:       hello
Namespace:  default
Age:        2m
URL:        http://hello-default.apps.ocp.example.com

Revisions:
  100%  @latest (hello-00001) [1] (2m)
        Image:  docker.io/openshift/hello-openshift (pinned to aaea76)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                1m
  ++ ConfigurationsReady     1m
  ++ RoutesReady             1m
```

Exemple de sortie avec l'indicateur --verbose

```
Name:         hello
Namespace:    default
Annotations:  serving.knative.dev/creator=system:admin
              serving.knative.dev/lastModifier=system:admin
Age:          3m
```

```
URL:         http://hello-default.apps.ocp.example.com
Cluster:     http://hello.default.svc.cluster.local

Revisions:
  100%  @latest (hello-00001) [1] (3m)
        Image:  docker.io/openshift/hello-openshift (pinned to aaea76)
        Env:    RESPONSE=Hello Serverless!

Conditions:
  OK TYPE                   AGE REASON
  ++ Ready                  3m
  ++ ConfigurationsReady     3m
  ++ RoutesReady            3m
```

- Décrire un service au format YAML :

```
$ kn service describe <service_name> -o yaml
```

- Décrire un service au format JSON :

```
$ kn service describe -YRFFGUNA nom_du_service> -o json
```

- Imprimer uniquement l'URL du service :

```
$ kn service describe <service_name> -o url
```

7.1.2. commandes du service kn en mode déconnecté

7.1.2.1. A propos du mode hors ligne de la CLI Knative

Lorsque vous exécutez les commandes **kn service**, les modifications se propagent immédiatement au cluster. Cependant, vous pouvez également exécuter les commandes **kn service** en mode déconnecté. Lorsque vous créez un service en mode déconnecté, aucune modification n'est apportée au cluster, mais le fichier descripteur de service est créé sur votre machine locale.

IMPORTANT

Le mode hors ligne de la CLI Knative est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, permettant aux clients de tester les fonctionnalités et de fournir un retour d'information au cours du processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Une fois le fichier descripteur créé, vous pouvez le modifier manuellement et le suivre dans un système de contrôle de version. Vous pouvez également propager les modifications au cluster en utilisant les commandes **kn service create -f**, **kn service apply -f**, ou **oc apply -f** sur les fichiers descripteurs.

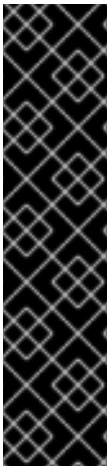
Le mode hors ligne a plusieurs utilisations :

- Vous pouvez modifier manuellement le fichier descripteur avant de l'utiliser pour effectuer des modifications sur le cluster.
- Vous pouvez suivre localement le fichier descripteur d'un service dans un système de contrôle de version. Cela vous permet de réutiliser le fichier descripteur ailleurs que dans le cluster cible, par exemple dans des pipelines d'intégration continue (CI), des environnements de développement ou des démonstrations.
- Vous pouvez examiner les fichiers descripteurs créés pour en savoir plus sur les services Knative. En particulier, vous pouvez voir comment le service résultant est influencé par les différents arguments passés à la commande **kn**.

Le mode hors ligne a ses avantages : il est rapide et ne nécessite pas de connexion au cluster. Cependant, le mode hors ligne ne dispose pas de la validation côté serveur. Par conséquent, vous ne pouvez pas, par exemple, vérifier que le nom du service est unique ou que l'image spécifiée peut être extraite.

7.1.2.2. Création d'un service en mode hors ligne

Vous pouvez exécuter les commandes **kn service** en mode déconnecté, de sorte qu'aucune modification n'est apportée au cluster et que le fichier de descripteur de service est créé sur votre machine locale. Une fois le fichier descripteur créé, vous pouvez le modifier avant de propager les changements au cluster.



IMPORTANT

Le mode hors ligne de la CLI Knative est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas de les utiliser en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, permettant aux clients de tester les fonctionnalités et de fournir un retour d'information au cours du processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

Conditions préalables

- OpenShift Serverless Operator et Knative Serving sont installés sur votre cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. En mode déconnecté, créez un fichier de descripteurs de service Knative local :

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest \
    --target ./ \
    --namespace test
```

Exemple de sortie

```
Service 'event-display' created in namespace 'test'.
```

- L'option **--target ./** active le mode hors ligne et spécifie **./** comme répertoire de stockage de la nouvelle arborescence.
  Si vous n'indiquez pas de répertoire existant, mais que vous utilisez un nom de fichier, tel que **--target my-service.yaml**, aucune arborescence n'est créée. Seul le fichier de descripteurs de service **my-service.yaml** est créé dans le répertoire actuel.

  Le nom de fichier peut avoir l'extension **.yaml**, **.yml**, ou **.json**. Le choix de **.json** crée le fichier du descripteur de service au format JSON.

- L'option **--namespace test** place le nouveau service dans l'espace de noms **test**.
  Si vous n'utilisez pas **--namespace**, et que vous êtes connecté à un cluster OpenShift Container Platform, le fichier de descripteurs est créé dans l'espace de noms actuel. Sinon, le fichier de descripteurs est créé dans l'espace de noms **default**.

2. Examinez la structure de répertoire créée :

```
$ tree ./
```

Exemple de sortie

```
./
└── test
    └── ksvc
        └── event-display.yaml

2 directories, 1 file
```

- Le répertoire actuel **./** spécifié avec **--target** contient le nouveau répertoire **test/** qui porte le nom de l'espace de noms spécifié.

- Le répertoire **test/** contient le répertoire **ksvc**, nommé d'après le type de ressource.

- Le répertoire **ksvc** contient le fichier descripteur **event-display.yaml**, nommé d'après le nom du service spécifié.

3. Examinez le fichier de descripteurs de service généré :

```
$ cat test/ksvc/event-display.yaml
```

Exemple de sortie

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  creationTimestamp: null
  name: event-display
  namespace: test
spec:
  template:
    metadata:
```

```
      annotations:
        client.knative.dev/user-image: quay.io/openshift-knative/knative-eventing-sources-event-
display:latest
      creationTimestamp: null
    spec:
      containers:
      - image: quay.io/openshift-knative/knative-eventing-sources-event-display:latest
        name: ""
        resources: {}
status: {}
```

4. Liste des informations sur le nouveau service :

```
$ kn service describe event-display --target ./ --namespace test
```

Exemple de sortie

```
Name:       event-display
Namespace:  test
Age:
URL:

Revisions:

Conditions:
  OK TYPE    AGE REASON
```

- L'option **--target ./** spécifie le répertoire racine de la structure de répertoires contenant les sous-répertoires de l'espace de noms.
  Vous pouvez également spécifier directement un nom de fichier YAML ou JSON à l'aide de l'option **--target**. Les extensions de fichier acceptées sont **.yaml**, **.yml**, et **.json**.

- L'option **--namespace** spécifie l'espace de noms, qui communique à **kn** le sous-répertoire contenant le fichier de descripteur de service nécessaire.
  Si vous n'utilisez pas **--namespace** et que vous êtes connecté à un cluster OpenShift Container Platform, **kn** recherche le service dans le sous-répertoire portant le nom de l'espace de noms actuel. Sinon, **kn** effectue la recherche dans le sous-répertoire **default/**.

5. Utilisez le fichier descripteur de service pour créer le service sur le cluster :

```
$ kn service create -f test/ksvc/event-display.yaml
```

Exemple de sortie

```
Creating service 'event-display' in namespace 'test':

  0.058s The Route is still working to reflect the latest desired specification.
  0.098s ...
  0.168s Configuration "event-display" is waiting for a Revision to become ready.
 23.377s ...
 23.419s Ingress has not yet been reconciled.
 23.534s Waiting for load balancer to be ready
 23.723s Ready to serve.
```

```
Service 'event-display' created to latest revision 'event-display-00001' is available at URL:
http://event-display-test.apps.example.com
```

7.1.3. kn commandes de conteneurs

Vous pouvez utiliser les commandes suivantes pour créer et gérer plusieurs conteneurs dans un service spec Knative.

7.1.3.1. Prise en charge des conteneurs multiples par le client Knative

Vous pouvez utiliser la commande **kn container add** pour imprimer les spécifications des conteneurs YAML sur la sortie standard. Cette commande est utile pour les cas d'utilisation multi-conteneurs car elle peut être utilisée avec d'autres drapeaux standard **kn** pour créer des définitions.

La commande **kn container add** accepte tous les drapeaux relatifs aux conteneurs qui peuvent être utilisés avec la commande **kn service create**. La commande **kn container add** peut également être enchaînée en utilisant les pipes UNIX (|) pour créer plusieurs définitions de conteneurs à la fois.

Exemple de commandes

- Ajouter un conteneur à partir d'une image et l'imprimer sur la sortie standard :

```
kn container add <container_name> --image <image_uri>
```

Example command

```
$ kn container add sidecar --image docker.io/example/sidecar
```

Exemple de sortie

```
containers:
- image: docker.io/example/sidecar
  name: sidecar
  resources: {}
```

- Enchaînez deux commandes **kn container add**, puis passez-les à une commande **kn service create** pour créer un service Knative avec deux conteneurs :

```
$ kn container add <first_container_name> --image <image_uri> | \
kn container add <second_container_name> --image <image_uri> | \
kn service create <service_name> --image <image_uri> --extra-containers -
```

--extra-containers - spécifie un cas particulier où **kn** lit l'entrée du tuyau au lieu d'un fichier YAML.

Example command

```
$ kn container add sidecar --image docker.io/example/sidecar:first | \
kn container add second --image docker.io/example/sidecar:second | \
kn service create my-service --image docker.io/example/my-app:latest --extra-containers -
```

L'option **--extra-containers** peut également accepter un chemin vers un fichier YAML :

```
$ kn service create <service_name> --image <image_uri> --extra-containers <filename>
```

Example command

```
$ kn service create my-service --image docker.io/example/my-app:latest --extra-containers
my-extra-containers.yaml
```

7.1.4. commandes de domaine kn

Vous pouvez utiliser les commandes suivantes pour créer et gérer les mappages de domaines.

7.1.4.1. Créer un mappage de domaine personnalisé en utilisant le CLI Knative

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Vous avez créé un service ou une route Knative, et vous contrôlez un domaine personnalisé que vous souhaitez associer à ce CR.

NOTE

Votre domaine personnalisé doit pointer vers le DNS du cluster OpenShift Container Platform.

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Associe un domaine à un CR dans l'espace de noms actuel :

```
kn domain create <domain_mapping_name> --ref <target_name>
```

Example command

```
$ kn domain create example-domain-map --ref example-service
```

L'indicateur **--ref** spécifie un CR cible adressable pour le mappage de domaine.

Si aucun préfixe n'est fourni lors de l'utilisation de l'indicateur **--ref**, il est supposé que la cible est un service Knative dans l'espace de noms actuel.

- Associer un domaine à un service Knative dans un espace de noms spécifié :

```
kn domain create <domain_mapping_name> --ref <ksvc:service_name:service_namespace>
```

Example command

```
$ kn domain create example-domain-map --ref ksvc:example-service:example-namespace
```

- Associer un domaine à une route Knative :

```
$ kn domain create <domain_mapping_name> --ref <kroute:route_name>
```

Example command

```
$ kn domain create example-domain-map --ref kroute:example-route
```

7.1.4.2. Gérer les mappages de domaines personnalisés en utilisant le CLI Knative

Après avoir créé une ressource personnalisée (CR) sur **DomainMapping**, vous pouvez dresser la liste des CR existantes, consulter les informations relatives à une CR existante, mettre à jour les CR ou les supprimer à l'aide de l'interface de programmation Knative (**kn**).

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur votre cluster.
- Vous avez créé au moins un CR **DomainMapping**.
- Vous avez installé l'outil CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

- Liste des CR existants sur **DomainMapping**:

```
$ kn domain list -n <domain_mapping_namespace>
```

- Voir les détails d'un **DomainMapping** CR existant :

```
$ kn domain describe <domain_mapping_name>
```

- Mettre à jour un CR **DomainMapping** pour qu'il pointe vers une nouvelle cible :

```
$ kn domain update --ref <target>
```

- Supprimer un CR **DomainMapping**:

```
kn domain delete <domain_mapping_name>
```

7.2. CONFIGURATION DE LA CLI KNATIVE

Vous pouvez personnaliser la configuration de votre CLI Knative (**kn**) en créant un fichier de configuration **config.yaml**. Vous pouvez fournir cette configuration en utilisant le drapeau **--config**, sinon la configuration est récupérée à partir d'un emplacement par défaut. L'emplacement de la configuration par défaut est conforme à la spécification du répertoire de base XDG et est différent pour les systèmes UNIX et les systèmes Windows.

Pour les systèmes UNIX :

- Si la variable d'environnement **XDG_CONFIG_HOME** est définie, l'emplacement de configuration par défaut recherché par le CLI Knative (**kn**) est **$XDG_CONFIG_HOME/kn**.
- Si la variable d'environnement **XDG_CONFIG_HOME** n'est pas définie, le CLI de Knative (**kn**) recherche la configuration dans le répertoire personnel de l'utilisateur à l'adresse **$HOME/.config/kn/config.yaml**.

Pour les systèmes Windows, l'emplacement par défaut de la configuration du CLI de Knative (**kn**) est **PDATA%\kn**.

Exemple de fichier de configuration

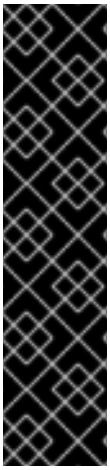
```
plugins:
  path-lookup: true (1)
  directory: ~/.config/kn/plugins (2)
eventing:
  sink-mappings: (3)
  - prefix: svc (4)
    group: core (5)
    version: v1 (6)
    resource: services (7)
```

1. Spécifie si le CLI Knative (**kn**) doit rechercher des plugins dans la variable d'environnement **PATH**. Il s'agit d'une option de configuration booléenne. La valeur par défaut est **false**.
2. Spécifie le répertoire dans lequel le CLI Knative (**kn**) recherche des plugins. Le chemin par défaut dépend du système d'exploitation, comme décrit précédemment. Il peut s'agir de n'importe quel répertoire visible par l'utilisateur.
3. La spécification **sink-mappings** définit la ressource adressable de Kubernetes qui est utilisée lorsque vous utilisez l'indicateur **--sink** avec une commande CLI Knative (**kn**).
4. Le préfixe que vous souhaitez utiliser pour décrire votre puits. **svc** pour un service, **channel** et **broker** sont des préfixes prédéfinis pour le CLI Knative (**kn**).
5. Le groupe API de la ressource Kubernetes.
6. La version de la ressource Kubernetes.
7. Le nom pluriel du type de ressource Kubernetes. Par exemple, **services** ou **brokers**.

7.3. PLUGINS CLI KNATIVE

Le CLI Knative (**kn**) supporte l'utilisation de plugins, qui vous permettent d'étendre les fonctionnalités de votre installation **kn** en ajoutant des commandes personnalisées et d'autres commandes partagées qui ne font pas partie de la distribution principale. Les plugins de l'interface de programmation Knative (**kn**) sont utilisés de la même manière que la fonctionnalité principale **kn**.

Actuellement, Red Hat prend en charge le plugin **kn-source-kafka** et le plugin **kn-event**.



IMPORTANT

Le plugin **kn-event** est une fonctionnalité d'aperçu technologique uniquement. Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas leur utilisation en production. Ces fonctionnalités offrent un accès anticipé aux fonctionnalités des produits à venir, ce qui permet aux clients de tester les fonctionnalités et de fournir un retour d'information pendant le processus de développement.

Pour plus d'informations sur la portée de l'assistance des fonctionnalités de l'aperçu technologique de Red Hat, voir Portée de l'assistance des fonctionnalités de l'aperçu technologique.

7.3.1. Construire des événements en utilisant le plugin kn-event

Vous pouvez utiliser l'interface de type constructeur de la commande **kn event build** pour créer un événement. Vous pouvez ensuite envoyer cet événement ultérieurement ou l'utiliser dans un autre contexte.

Conditions préalables

- Vous avez installé le CLI Knative (**kn**).

Procédure

- Créer un événement :

```
$ kn event build --field <field-name>=<value> --type <type-name> --id <id> --output <format>
```

où :

  - L'option **--field** ajoute des données à l'événement sous la forme d'une paire champ-valeur. Vous pouvez l'utiliser plusieurs fois.
  - L'option **--type** vous permet de spécifier une chaîne de caractères désignant le type d'événement.
  - L'indicateur **--id** précise l'identifiant de l'événement.
  - Vous pouvez utiliser les arguments **json** ou **yaml** avec l'option **--output** pour modifier le format de sortie de l'événement.
    Tous ces indicateurs sont facultatifs.

Construire un événement simple

```
$ kn event build -o yaml
```

Événement résultant au format YAML

```
data: {}
datacontenttype: application/json
id: 81a402a2-9c29-4c27-b8ed-246a253c9e58
source: kn-event/v0.4.0
```

```
specversion: "1.0"
time: "2021-10-15T10:42:57.713226203Z"
type: dev.knative.cli.plugin.event.generic
```

Création d'un exemple d'événement de transaction

```
$ kn event build \
    --field operation.type=local-wire-transfer \
    --field operation.amount=2345.40 \
    --field operation.from=87656231 \
    --field operation.to=2344121 \
    --field automated=true \
    --field signature='FGzCPLvYWdEgsdpb3qXkaVp7Da0=' \
    --type org.example.bank.bar \
    --id $(head -c 10 < /dev/urandom | base64 -w 0) \
    --output json
```

Événement résultant au format JSON

```
{
  "specversion": "1.0",
  "id": "RjtL8UH66X+UJg==",
  "source": "kn-event/v0.4.0",
  "type": "org.example.bank.bar",
  "datacontenttype": "application/json",
  "time": "2021-10-15T10:43:23.113187943Z",
  "data": {
    "automated": true,
    "operation": {
      "amount": "2345.40",
      "from": 87656231,
      "to": 2344121,
      "type": "local-wire-transfer"
    },
    "signature": "FGzCPLvYWdEgsdpb3qXkaVp7Da0="
  }
}
```

7.3.2. Envoi d'événements à l'aide du plugin kn-event

Vous pouvez utiliser la commande **kn event send** pour envoyer un événement. Les événements peuvent être envoyés soit à des adresses publiques, soit à des ressources adressables à l'intérieur d'un cluster, comme les services Kubernetes, ainsi que les services Knative, les brokers et les canaux. La commande utilise la même interface de type constructeur que la commande **kn event build**.

Conditions préalables

- Vous avez installé le CLI Knative (**kn**).

Procédure

- Envoyer un événement :

```
$ kn event send --field <field-name>=<value> --type <type-name> --id <id> --to-url <url> --to
<cluster-resource> --namespace <namespace>
```

où :

- L'option **--field** ajoute des données à l'événement sous la forme d'une paire champ-valeur. Vous pouvez l'utiliser plusieurs fois.
- L'option **--type** vous permet de spécifier une chaîne de caractères désignant le type d'événement.
- L'indicateur **--id** précise l'identifiant de l'événement.
- Si vous envoyez l'événement à une destination accessible au public, indiquez l'URL à l'aide de l'indicateur **--to-url**.
- Si vous envoyez l'événement à une ressource Kubernetes en cluster, spécifiez la destination à l'aide de l'indicateur **--to**.
  - Spécifiez la ressource Kubernetes en utilisant le format **<Kind>:<ApiVersion>:<name>**.
- L'option **--namespace** spécifie l'espace de noms. S'il est omis, l'espace de noms est pris dans le contexte actuel.
  Tous ces indicateurs sont facultatifs, à l'exception de la spécification de la destination, pour laquelle vous devez utiliser **--to-url** ou **--to**.

  L'exemple suivant montre l'envoi d'un événement à une URL :

  Example command

```
$ kn event send \
  --field player.id=6354aa60-ddb1-452e-8c13-24893667de20 \
  --field player.game=2345 \
  --field points=456 \
  --type org.example.gaming.foo \
  --to-url http://ce-api.foo.example.com/
```

  L'exemple suivant montre l'envoi d'un événement à une ressource du cluster :

  Example command

```
$ kn event send \
  --type org.example.kn.ping \
  --id $(uuidgen) \
  --field event.type=test \
  --field event.data=98765 \
  --to Service:serving.knative.dev/v1:event-display
```

7.4. COMMANDES CLI DE KNATIVE EVENTING

7.4.1. commandes kn source

Vous pouvez utiliser les commandes suivantes pour répertorier, créer et gérer les sources d'événements Knative.

7.4.1.1. Liste des types de sources d'événements disponibles en utilisant le CLI Knative

Vous pouvez répertorier les types de sources d'événements qui peuvent être créés et utilisés sur votre cluster à l'aide de la commande CLI **kn source list-types**.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

1. Liste les types de sources d'événements disponibles dans le terminal :

```
$ kn source list-types
```

Exemple de sortie

```
TYPE              NAME                                DESCRIPTION
ApiServerSource   apiserversources.sources.knative.dev   Watch and send Kubernetes API events to a sink
PingSource        pingsources.sources.knative.dev        Periodically send ping events to a sink
SinkBinding       sinkbindings.sources.knative.dev       Binding for connecting a PodSpecable to a sink
```

2. Facultatif : vous pouvez également dresser la liste des types de sources d'événements disponibles au format YAML :

```
$ kn source list-types -o yaml
```

7.4.1.2. Drapeau d'évitement de l'CLI Knative

Lorsque vous créez une source d'événements à l'aide de l'interface de programmation Knative (**kn**), vous pouvez spécifier un puits vers lequel les événements sont envoyés à partir de cette ressource à l'aide de l'indicateur **--sink**. Le récepteur peut être n'importe quelle ressource adressable ou appelable qui peut recevoir des événements entrants d'autres ressources.

L'exemple suivant crée une liaison de puits qui utilise un service, **http://event-display.svc.cluster.local**, comme puits :

Exemple de commande utilisant l'indicateur d'évitement

```
$ kn source binding create bind-heartbeat \
  --namespace sinkbinding-example \
  --subject "Job:batch/v1:app=heartbeat-cron" \
  --sink http://event-display.svc.cluster.local \ 1
  --ce-override "sink=bound"
```

1 **svc** dans **http://event-display.svc.cluster.local** détermine que le puits est un service Knative. D'autres préfixes de puits par défaut incluent **channel**, et **broker**.

7.4.1.3. Créer et gérer des sources de conteneurs en utilisant le CLI Knative

Vous pouvez utiliser les commandes **kn source container** pour créer et gérer des sources de conteneurs en utilisant le CLI Knative (**kn**). L'utilisation de l'interface de programmation Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Créer un conteneur source

```
$ kn source container create <container_source_name> --image <image_uri> --sink <sink>
```

Supprimer un conteneur source

```
kn source container delete <container_source_name>
```

Décrire la source d'un conteneur

```
kn source container describe <container_source_name>
```

Liste des sources de conteneurs existantes

```
$ kn source container list
```

Liste des sources de conteneurs existantes au format YAML

```
$ kn source container list -o yaml
```

Mise à jour d'un conteneur source

Cette commande met à jour l'URI de l'image d'un conteneur existant :

```
$ kn source container update <container_source_name> --image <image_uri>
```

7.4.1.4. Création d'une source de serveur API à l'aide du CLI Knative

Vous pouvez utiliser la commande **kn source apiserver create** pour créer une source de serveur API à l'aide de l'interface de programmation **kn**. L'utilisation du CLI **kn** pour créer une source de serveur API offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Conditions préalables

- OpenShift Serverless Operator et Knative Eventing sont installés sur le cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez installé l'OpenShift CLI (**oc**).

- Vous avez installé le CLI Knative (**kn**).

PROCÉDURE

Si vous souhaitez réutiliser un compte de service existant, vous pouvez modifier votre ressource **ServiceAccount** existante pour y inclure les autorisations requises au lieu de créer une nouvelle ressource.

1. Créez un compte de service, un rôle et une liaison de rôle pour la source d'événement sous la forme d'un fichier YAML :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: events-sa
  namespace: default (1)

---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: event-watcher
  namespace: default (2)
rules:
  - apiGroups:
      - ""
    resources:
      - events
    verbs:
      - get
      - list
      - watch

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: k8s-ra-event-watcher
  namespace: default (3)
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: event-watcher
subjects:
  - kind: ServiceAccount
    name: events-sa
    namespace: default (4)
```

(1) (2) (3) (4) Remplacez cet espace de noms par l'espace de noms que vous avez sélectionné pour l'installation de la source d'événements.

2. Appliquer le fichier YAML :

```
$ oc apply -f <filename>
```

3. Créez une source de serveur API dotée d'un puits d'événements. Dans l'exemple suivant, le puits est un courtier :

```
$ kn source apiserver create <event_source_name> --sink broker:<broker_name> --
resource "event:v1" --service-account <service_account_name> --mode Resource
```

4. Pour vérifier que la source du serveur API est correctement configurée, créez un service Knative qui déverse les messages entrants dans son journal :

```
$ kn service create <service_name> --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

5. Si vous avez utilisé un courtier comme puits d'événements, créez un déclencheur pour filtrer les événements du courtier **default** vers le service :

```
kn trigger create <trigger_name> --sink ksvc:<service_name> $ kn trigger create
<trigger_name>
```

6. Créer des événements en lançant un pod dans l'espace de noms par défaut :

```
$ oc create deployment hello-node --image quay.io/openshift-knative/knative-eventing-
sources-event-display:latest
```

7. Vérifiez que le contrôleur est correctement mappé en inspectant la sortie générée par la commande suivante :

```
$ kn source apiserver describe <nom_de_la_source>
```

Exemple de sortie

```
Name:               mysource
Namespace:          default
Annotations:        sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:                3m
ServiceAccountName:  events-sa
Mode:               Resource
Sink:
  Name:       default
  Namespace:  default
  Kind:       Broker (eventing.knative.dev/v1)
Resources:
  Kind:       event (v1)
  Controller:  false
Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                3m
  ++ Deployed              3m
  ++ SinkProvided          3m
  ++ SufficientPermissions     3m
  ++ EventTypesProvided        3m
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés à Knative en consultant les journaux de la fonction dumper de messages.

1. Obtenez les cosses :

```
$ oc get pods
```

2. Affichez les journaux des fonctions de l'expéditeur de messages pour les modules :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.apiserver.resource.update
  datacontenttype: application/json
  ...
Data,
  {
    "apiVersion": "v1",
    "involvedObject": {
      "apiVersion": "v1",
      "fieldPath": "spec.containers{hello-node}",
      "kind": "Pod",
      "name": "hello-node",
      "namespace": "default",
       .....
    },
    "kind": "Event",
    "message": "Started container",
    "metadata": {
      "name": "hello-node.159d7608e3a3572c",
      "namespace": "default",
      ....
    },
    "reason": "Started",
    ...
  }
```

Suppression de la source du serveur API

1. Supprimer le déclencheur :

```
kn trigger delete <trigger_name>
```

2. Supprimer la source de l'événement :

```
$ kn source apiserver delete <nom_de_la_source>
```

3. Supprimez le compte de service, le rôle de cluster et le lien de cluster :

```
$ oc delete -f authentication.yaml
```

7.4.1.5. Création d'une source de ping à l'aide de la CLI Knative

Vous pouvez utiliser la commande **kn source ping create** pour créer une source de ping en utilisant le CLI Knative (**kn**). L'utilisation de l'interface de programmation Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et plus intuitive que la modification directe des fichiers YAML.

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Facultatif : Si vous souhaitez utiliser les étapes de vérification pour cette procédure, installez l'OpenShift CLI (**oc**).

Procédure

1. Pour vérifier que la source de ping fonctionne, créez un service Knative simple qui déverse les messages entrants dans les journaux du service :

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display:latest
```

2. Pour chaque ensemble d'événements ping que vous souhaitez demander, créez une source ping dans le même espace de noms que le consommateur d'événements :

```
$ kn source ping create test-ping-source \
    --schedule "*/2 * * * *" \
    --data '{"message": "Hello world!"}' \
    --sink ksvc:event-display
```

3. Vérifiez que le contrôleur est correctement mappé en entrant la commande suivante et en inspectant la sortie :

```
$ kn source ping describe test-ping-source
```

Exemple de sortie

```
Name:         test-ping-source
Namespace:    default
Annotations:  sources.knative.dev/creator=developer,
sources.knative.dev/lastModifier=developer
Age:          15s
Schedule:     */2 * * * *
```

```
Data:       {"message": "Hello world!"}

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE                 AGE REASON
  ++ Ready                8s
  ++ Deployed             8s
  ++ SinkProvided         15s
  ++ ValidSchedule        15s
  ++ EventTypeProvided    15s
  ++ ResourcesCorrect     15s
```

Vérification

Vous pouvez vérifier que les événements Kubernetes ont été envoyés au puits d'événements Knative en examinant les journaux du pod de puits.

Par défaut, les services Knative terminent leurs pods si aucun trafic n'est reçu pendant une période de 60 secondes. L'exemple présenté dans ce guide crée une source ping qui envoie un message toutes les 2 minutes, chaque message doit donc être observé dans un pod nouvellement créé.

1. Surveillez la création de nouvelles gousses :

```
$ watch oc get pods
```

2. Annulez l'observation des pods en utilisant Ctrl C, puis regardez les journaux du pod créé :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.sources.ping
  source: /apis/v1/namespaces/default/pingsources/test-ping-source
  id: 99e4f4f6-08ff-4bff-acf1-47f61ded68c9
  time: 2020-04-07T16:16:00.000601161Z
  datacontenttype: application/json
Data,
  {
    "message": "Hello world!"
  }
```

Suppression de la source de ping

- Supprimer la source de ping :

```
$ kn delete pingsources.sources.knative.dev <nom_de_la_source_de_ping>
```

7.4.1.6. Créer une source d'événement Apache Kafka en utilisant le CLI Knative

Vous pouvez utiliser la commande **kn source kafka create** pour créer une source Kafka en utilisant le CLI Knative (**kn**). L'utilisation du CLI Knative pour créer des sources d'événements offre une interface utilisateur plus rationnelle et intuitive que la modification directe des fichiers YAML.

Conditions préalables

- OpenShift Serverless Operator, Knative Eventing, Knative Serving et la ressource personnalisée (CR) **KnativeKafka** sont installés sur votre cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous avez accès à un cluster Red Hat AMQ Streams (Kafka) qui produit les messages Kafka que vous souhaitez importer.
- Vous avez installé le CLI Knative (**kn**).
- Facultatif : Vous avez installé l'OpenShift CLI (**oc**) si vous voulez utiliser les étapes de vérification dans cette procédure.

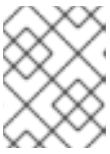
Procédure

1. Pour vérifier que la source d'événements Kafka fonctionne, créez un service Knative qui déverse les événements entrants dans les journaux du service :

```
$ kn service create event-display \
    --image quay.io/openshift-knative/knative-eventing-sources-event-display
```

2. Créer un CR **KafkaSource**:

```
$ kn source kafka create <kafka_source_name> \
    --servers <cluster_kafka_bootstrap>.kafka.svc:9092 \
    --topics <topic_name> --consumergroup my-consumer-group \
    --sink event-display
```



NOTE

Remplacez les valeurs de cette commande par les valeurs de votre nom de source, de vos serveurs d'amorçage et de vos rubriques.

Les options **--servers**, **--topics**, et **--consumergroup** spécifient les paramètres de connexion au cluster Kafka. L'option **--consumergroup** est facultative.

3. Facultatif : Affichez les détails de la CR **KafkaSource** que vous avez créée :

```
$ kn source kafka describe <kafka_source_name>
```

Exemple de sortie

```
Name:              example-kafka-source
Namespace:         kafka
Age:               1h
```

```
BootstrapServers:  example-cluster-kafka-bootstrap.kafka.svc:9092
Topics:            example-topic
ConsumerGroup:     example-consumer-group

Sink:
  Name:       event-display
  Namespace:  default
  Resource:   Service (serving.knative.dev/v1)

Conditions:
  OK TYPE            AGE REASON
  ++ Ready           1h
  ++ Deployed        1h
  ++ SinkProvided    1h
```

Verification steps

1. Déclencher l'envoi par l'instance Kafka d'un message au sujet :

```
$ oc -n kafka run kafka-producer \
    -ti --image=quay.io/strimzi/kafka:latest-kafka-2.7.0 --rm=true \
    --restart=Never -- bin/kafka-console-producer.sh \
    --broker-list <cluster_kafka_bootstrap>:9092 --topic my-topic
```

Saisissez le message dans l'invite. Cette commande suppose que :

- Le cluster Kafka est installé dans l'espace de noms **kafka**.
- L'objet **KafkaSource** a été configuré pour utiliser le sujet **my-topic**.

2. Vérifiez que le message est arrivé en consultant les journaux :

```
$ oc logs $(oc get pod -o name | grep event-display) -c user-container
```

Exemple de sortie

```
☁  cloudevents.Event
Validation: valid
Context Attributes,
  specversion: 1.0
  type: dev.knative.kafka.event
  source: /apis/v1/namespaces/default/kafkasources/example-kafka-source#example-topic
  subject: partition:46#0
  id: partition:46/offset:0
  time: 2021-03-10T11:21:49.4Z
Extensions,
  traceparent: 00-161ff3815727d8755848ec01c866d1cd-7ff3916c44334678-00
Data,
  Hello!
```

7.5. COMMANDES CLI POUR LES FONCTIONS KNATIVE

7.5.1. kn fonctions commandes

7.5.1.1. Création de fonctions

Avant de pouvoir construire et déployer une fonction, vous devez la créer à l'aide de la CLI Knative (**kn**). Vous pouvez spécifier le chemin d'accès, la durée d'exécution, le modèle et le registre d'images en tant que drapeaux sur la ligne de commande, ou utiliser le drapeau **-c** pour lancer l'expérience interactive dans le terminal.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).

Procédure

- Créer un projet de fonction :

```
$ kn func create -r <repository> -l <runtime> -t <template> <path>
```

  - Les valeurs d'exécution acceptées sont **quarkus**, **node**, **typescript**, **go**, **python**, **springboot** et **rust**.
  - Les valeurs acceptées sont **http** et **cloudevents**.

    Example command

```
$ kn func create -l typescript -t cloudevents examplefunc
```

    Exemple de sortie

```
Created typescript function in /home/user/demo/examplefunc
```

  - Vous pouvez également spécifier un référentiel qui contient un modèle personnalisé.

    Example command

```
$ kn func create -r https://github.com/boson-project/templates/ -l node -t hello-world
examplefunc
```

    Exemple de sortie

```
Created node function in /home/user/demo/examplefunc
```

7.5.1.2. Exécuter une fonction localement

Vous pouvez utiliser la commande **kn func run** pour exécuter une fonction localement dans le répertoire actuel ou dans le répertoire spécifié par l'indicateur **--path**. Si la fonction que vous exécutez n'a jamais été construite auparavant, ou si les fichiers du projet ont été modifiés depuis la dernière fois qu'elle a été construite, la commande **kn func run** construit la fonction avant de l'exécuter par défaut.

Exemple de commande pour exécuter une fonction dans le répertoire courant

```
$ kn func run
```

-

Exemple de commande pour exécuter une fonction dans un répertoire spécifié comme chemin d'accès

```
$ kn func run --path=<directory_path>
```

Vous pouvez également forcer la reconstruction d'une image existante avant d'exécuter la fonction, même si les fichiers du projet n'ont pas été modifiés, en utilisant l'option **--build**:

Exemple de commande d'exécution utilisant le drapeau de construction

```
$ kn func run --build
```

Si l'indicateur **build** est défini comme faux, la construction de l'image est désactivée et la fonction est exécutée à l'aide de l'image précédemment construite :

Exemple de commande d'exécution utilisant le drapeau de construction

```
$ kn func run --build=false
```

Vous pouvez utiliser la commande help pour en savoir plus sur les options de la commande **kn func run**:

Commande d'aide à la construction

```
$ kn func help run
```

7.5.1.3. Fonctions du bâtiment

Avant de pouvoir exécuter une fonction, vous devez construire le projet de la fonction. Si vous utilisez la commande **kn func run**, la fonction est construite automatiquement. Cependant, vous pouvez utiliser la commande **kn func build** pour construire une fonction sans l'exécuter, ce qui peut être utile pour les utilisateurs avancés ou les scénarios de débogage.

La commande **kn func build** crée une image de conteneur OCI qui peut être exécutée localement sur votre ordinateur ou sur un cluster OpenShift Container Platform. Cette commande utilise le nom du projet de la fonction et le nom du registre d'images pour construire un nom d'image entièrement qualifié pour votre fonction.

7.5.1.3.1. Types de conteneurs d'images

Par défaut, **kn func build** crée une image de conteneur en utilisant la technologie Red Hat Source-to-Image (S2I).

Exemple de commande de compilation utilisant Red Hat Source-to-Image (S2I)

```
$ kn func build
```

7.5.1.3.2. Types de registres d'images

OpenShift Container Registry est utilisé par défaut comme registre d'images pour le stockage des images de fonctions.

Exemple de commande de construction utilisant OpenShift Container Registry

```
$ kn func build
```

Exemple de sortie

```
Building function image
Function image has been built, image: registry.redhat.io/example/example-function:latest
```

Vous pouvez remplacer l'utilisation d'OpenShift Container Registry comme registre d'images par défaut en utilisant le drapeau **--registry**:

Exemple de commande de construction surchargeant OpenShift Container Registry pour utiliser quay.io

```
$ kn func build --registry quay.io/username
```

Exemple de sortie

```
Building function image
Function image has been built, image: quay.io/username/example-function:latest
```

7.5.1.3.3. Pousser le drapeau

Vous pouvez ajouter l'option **--push** à une commande **kn func build** pour pousser automatiquement l'image de la fonction une fois qu'elle a été construite avec succès :

Exemple de commande de construction utilisant OpenShift Container Registry

```
$ kn func build --push
```

7.5.1.3.4. Commande d'aide

Vous pouvez utiliser la commande help pour en savoir plus sur les options de la commande **kn func build**:

Commande d'aide à la construction

```
$ kn func help build
```

7.5.1.4. Déployer des fonctions

Vous pouvez déployer une fonction sur votre cluster en tant que service Knative à l'aide de la commande **kn func deploy**. Si la fonction ciblée est déjà déployée, elle est mise à jour avec une nouvelle image de conteneur qui est poussée vers un registre d'images de conteneur, et le service Knative est mis à jour.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.

- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous devez avoir déjà créé et initialisé la fonction que vous souhaitez déployer.

Procédure

- Déployer une fonction :

```
$ kn func deploy [-n <namespace> -p <path> -i <image>]
```

Exemple de sortie

```
Function deployed at: http://func.example.com
```

  - Si aucune adresse **namespace** n'est spécifiée, la fonction est déployée dans l'espace de noms actuel.
  - La fonction est déployée à partir du répertoire actuel, à moins qu'une adresse **path** ne soit spécifiée.
  - Le nom du service Knative est dérivé du nom du projet et ne peut pas être modifié à l'aide de cette commande.

7.5.1.5. Liste des fonctions existantes

Vous pouvez dresser la liste des fonctions existantes en utilisant **kn func list**. Si vous souhaitez lister les fonctions qui ont été déployées en tant que services Knative, vous pouvez également utiliser **kn service list**.

Procédure

- Dresser la liste des fonctions existantes :

```
$ kn func list [-n <namespace> -p <path>]
```

Exemple de sortie

```
NAME            NAMESPACE  RUNTIME  URL
READY
example-function  default    node     http://example-function.default.apps.ci-ln-g9f36hb-
d5d6b.origin-ci-int-aws.dev.rhcloud.com  True
```

- Liste des fonctions déployées en tant que services Knative :

```
$ kn service list -n <namespace>
```

Exemple de sortie

```
NAME          URL                                                    LATEST
AGE   CONDITIONS   READY   REASON
```

```
example-function   http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-aws.dev.rhcloud.com   example-function-gzl4c   16m   3 OK / 3     True
```

7.5.1.6. Description d'une fonction

La commande **kn func info** imprime des informations sur une fonction déployée, telles que le nom de la fonction, l'image, l'espace de noms, les informations sur le service Knative, les informations sur les itinéraires et les abonnements aux événements.

Procédure

- Décrire une fonction :

```
$ kn func info [-f <format> -n <namespace> -p <path>]
```

Example command

```
$ kn func info -p function/example-function
```

Exemple de sortie

```
Function name:
  example-function
Function is built in image:
  docker.io/user/example-function:latest
Function is deployed as Knative Service:
  example-function
Function is deployed in namespace:
  default
Routes:
  http://example-function.default.apps.ci-ln-g9f36hb-d5d6b.origin-ci-int-aws.dev.rhcloud.com
```

7.5.1.7. Invoquer une fonction déployée avec un événement de test

Vous pouvez utiliser la commande CLI **kn func invoke** pour envoyer une requête de test afin d'invoquer une fonction localement ou sur votre cluster OpenShift Container Platform. Vous pouvez utiliser cette commande pour tester qu'une fonction fonctionne et qu'elle est capable de recevoir des événements correctement. L'invocation d'une fonction localement est utile pour un test rapide pendant le développement de la fonction. L'invocation d'une fonction sur le cluster est utile pour des tests plus proches de l'environnement de production.

Conditions préalables

- L'opérateur OpenShift Serverless et Knative Serving sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Vous devez avoir déjà déployé la fonction que vous souhaitez invoquer.

Procédure

- Invoquer une fonction :

```
$ kn func invoke
```

  - La commande **kn func invoke** ne fonctionne que lorsqu'une image de conteneur locale est en cours d'exécution ou lorsqu'une fonction est déployée dans le cluster.
  - La commande **kn func invoke** s'exécute par défaut dans le répertoire local et suppose que ce répertoire est un projet de fonction.

7.5.1.7.1. kn func invoke optional parameters

Vous pouvez spécifier des paramètres facultatifs pour la demande en utilisant les drapeaux de commande CLI suivants : **kn func invoke**.

Drapeaux	Description
-t, **--target**	Spécifie l'instance cible de la fonction invoquée, par exemple, **local** ou **remote** ou **https://staging.example.com/**. La cible par défaut est **local**.
-f, **--format**	Spécifie le format du message, par exemple **cloudevent** ou **http**.
--id	Spécifie une chaîne d'identification unique pour la demande.
-n, **--namespace**	Spécifie l'espace de noms sur le cluster.
--source	Spécifie le nom de l'expéditeur de la demande. Cela correspond à l'attribut CloudEvent **source**.
--type	Spécifie le type de demande, par exemple, **boson.fn**. Cela correspond à l'attribut CloudEvent **type**.
--data	Spécifie le contenu de la demande. Pour les demandes CloudEvent, il s'agit de l'attribut CloudEvent **data**.
--file	Spécifie le chemin d'accès à un fichier local contenant les données à envoyer.
--content-type	Spécifie le type de contenu MIME pour la demande.
-p, **--path**	Spécifie le chemin d'accès au répertoire du projet.
-c, **--confirm**	Permet de demander la confirmation interactive de toutes les options.
-v, **--verbose**	Active l'impression de la sortie verbeuse.
-h, **--help**	Imprime des informations sur l'utilisation de **kn func invoke**.

7.5.1.7.1.1. Principaux paramètres

Les paramètres suivants définissent les principales propriétés de la commande **kn func invoke**:

Cible de l'événement (-t, --target)

L'instance cible de la fonction invoquée. Accepte la valeur **local** pour une fonction déployée localement, la valeur **remote** pour une fonction déployée à distance, ou une URL pour une fonction déployée à un point de terminaison arbitraire. Si aucune cible n'est spécifiée, la valeur par défaut est **local**.

Format du message d'événement (-f, --format)

Le format du message pour l'événement, par exemple **http** ou **cloudevent**. Le format par défaut est celui du modèle utilisé lors de la création de la fonction.

Type d'événement (--type)

Le type d'événement envoyé. Vous trouverez des informations sur le paramètre **type** défini dans la documentation de chaque producteur d'événements. Par exemple, la source du serveur API peut définir le paramètre **type** des événements produits comme **dev.knative.apiserver.resource.update**.

Source de l'événement (--source)

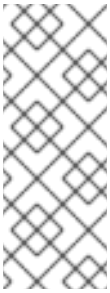
La source d'événement unique qui a produit l'événement. Il peut s'agir d'un URI pour la source de l'événement, par exemple **https://10.96.0.1/**ou du nom de la source de l'événement.

ID de l'événement (--id)

Un identifiant unique et aléatoire créé par le producteur de l'événement.

Données d'événement (--data)

Permet de spécifier une valeur **data** pour l'événement envoyé par la commande **kn func invoke**. Par exemple, vous pouvez spécifier une valeur **--data** telle que **"Hello World"** pour que l'événement contienne cette chaîne de données. Par défaut, aucune donnée n'est incluse dans les événements créés par **kn func invoke**.



NOTE

Les fonctions qui ont été déployées dans un cluster peuvent répondre à des événements provenant d'une source d'événements existante qui fournit des valeurs pour des propriétés telles que **source** et **type**. Ces événements ont souvent une valeur **data** au format JSON, qui capture le contexte spécifique au domaine de l'événement. En utilisant les drapeaux CLI mentionnés dans ce document, les développeurs peuvent simuler ces événements pour des tests locaux.

Vous pouvez également envoyer des données d'événement en utilisant l'indicateur **--file** pour fournir un fichier local contenant les données de l'événement. Dans ce cas, spécifiez le type de contenu en utilisant **--content-type**.

Type de contenu des données (--content-type)

Si vous utilisez l'indicateur **--data** pour ajouter des données à des événements, vous pouvez utiliser l'indicateur **--content-type** pour spécifier le type de données transportées par l'événement. Dans l'exemple précédent, les données sont du texte brut, vous pouvez donc spécifier **kn func invoke --data "Hello world!" --content-type "text/plain"**.

7.5.1.7.1.2. Exemple de commandes

Il s'agit de l'invocation générale de la commande **kn func invoke**:

```
$ kn func invoke --type <event_type> --source <event_source> --data <event_data> --content-type
<content_type> --id <event_ID> --format <format> --namespace <namespace>
```

Par exemple, pour envoyer un événement \N "Hello world!\N", vous pouvez exécuter :

```
$ kn func invoke --type ping --source example-ping --data "Hello world!" --content-type "text/plain" --
id example-ID --format http --namespace my-ns
```

7.5.1.7.1.2.1. Spécification du fichier contenant les données

Pour spécifier le fichier sur disque qui contient les données de l'événement, utilisez les drapeaux **--file** et **--content-type**:

```
$ kn func invoke --file <path> --content-type <content-type>
```

Par exemple, pour envoyer des données JSON stockées dans le fichier **test.json**, utilisez la commande suivante :

```
$ kn func invoke --file ./test.json --content-type application/json
```

7.5.1.7.1.2.2. Spécifier le projet de fonction

Vous pouvez spécifier un chemin d'accès au projet de la fonction en utilisant l'option **--path**:

```
kn func invoke --path <path_to_function> $ kn func invoke --path <path_to_function>
```

Par exemple, pour utiliser le projet de fonction situé dans le répertoire **./example/example-function**, utilisez la commande suivante :

```
$ kn func invoke --path ./example/example-function
```

7.5.1.7.1.2.3. Spécifier où la fonction cible est déployée

Par défaut, **kn func invoke** cible le déploiement local de la fonction :

```
$ kn func invoke
```

Pour utiliser un déploiement différent, utilisez l'option **--target**:

```
kn func invoke --target <target> $ kn func invoke --target <target>
```

Par exemple, pour utiliser la fonction déployée sur le cluster, utilisez le drapeau **--target remote**:

```
$ kn func invoke --target remote
```

Pour utiliser la fonction déployée à une URL arbitraire, utilisez l'option **--target <URL>**:

```
$ kn func invoke --target "https://my-event-broker.example.com"
```

Vous pouvez explicitement cibler le déploiement local. Dans ce cas, si la fonction n'est pas exécutée localement, la commande échoue :

```
$ kn func invoke --target local
```

7.5.1.8. Suppression d'une fonction

Vous pouvez supprimer une fonction en utilisant la commande **kn func delete**. Cette opération est utile lorsqu'une fonction n'est plus nécessaire et permet d'économiser des ressources sur votre cluster.

Procédure

- Supprimer une fonction :

```
$ kn func delete [<function_name> -n <namespace> -p <path>]
```

  - Si le nom ou le chemin de la fonction à supprimer n'est pas spécifié, le répertoire actuel est parcouru à la recherche d'un fichier **func.yaml** qui est utilisé pour déterminer la fonction à supprimer.
  - Si l'espace de noms n'est pas spécifié, il prend par défaut la valeur de **namespace** dans le fichier **func.yaml**.

CHAPITRE 8. OBSERVABILITÉ

8.1. MESURES DE L'ADMINISTRATEUR

8.1.1. Mesures de l'administrateur sans serveur

Les métriques permettent aux administrateurs de cluster de surveiller les performances des composants du cluster OpenShift Serverless et des charges de travail.

Vous pouvez visualiser différentes métriques pour OpenShift Serverless en naviguant vers **Dashboards** dans la console web de OpenShift Container Platform **Administrator** perspective.

8.1.1.1. Conditions préalables

- Voir la documentation OpenShift Container Platform sur la gestion des métriques pour plus d'informations sur l'activation des métriques pour votre cluster.
- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez accès à la perspective **Administrator** dans la console web de OpenShift Container Platform.

AVERTISSEMENT

Si Service Mesh est activé avec mTLS, les métriques pour Knative Serving sont désactivées par défaut car Service Mesh empêche Prometheus de récupérer les métriques.

Pour plus d'informations sur la résolution de ce problème, voir Activation des métriques Knative Serving lors de l'utilisation de Service Mesh avec mTLS.

Le scraping des métriques n'affecte pas l'autoscaling d'un service Knative, car les requêtes de scraping ne passent pas par l'activateur. Par conséquent, aucun scraping n'a lieu si aucun pod n'est en cours d'exécution.

8.1.2. Métriques des contrôleurs sans serveur

Les mesures suivantes sont émises par tout composant qui met en œuvre une logique de contrôleur. Ces mesures donnent des détails sur les opérations de rapprochement et sur le comportement de la file d'attente en fonction duquel les demandes de rapprochement sont ajoutées à la file d'attente.

Nom de la métrique	Description	Type	Tags	Unité
work_queue_depth	La profondeur de la file d'attente.	Jauge	**reconciler**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
reconcile_count	Le nombre d'opérations de rapprochement.	Compteur	**reconciler**, **success**	Entier (pas d'unité)
reconcile_latency	Le temps de latence des opérations de rapprochement.	Histogramme	**reconciler**, **success**	Millisecondes
workqueue_adds_total	Nombre total d'actions d'ajout traitées par la file d'attente.	Compteur	**name**	Entier (pas d'unité)
workqueue_queue_latency_seconds	Durée pendant laquelle un élément reste dans la file d'attente avant d'être demandé.	Histogramme	**name**	Secondes
workqueue_retries_total	Le nombre total de tentatives qui ont été traitées par la file d'attente.	Compteur	**name**	Entier (pas d'unité)
workqueue_work_duration_seconds	Le temps nécessaire pour traiter un élément de la file d'attente.	Histogramme	**name**	Secondes
workqueue_unfinished_work_seconds	Durée pendant laquelle les éléments de la file d'attente en suspens sont en cours de traitement.	Histogramme	**name**	Secondes
workqueue_longest_running_processor_seconds	Durée pendant laquelle les éléments de la file d'attente les plus anciens sont en cours de traitement.	Histogramme	**name**	Secondes

8.1.3. Métriques des webhooks

Les métriques des webhooks fournissent des informations utiles sur les opérations. Par exemple, si un grand nombre d'opérations échouent, cela peut indiquer un problème avec une ressource créée par l'utilisateur.

Nom de la métrique	Description	Type	Tags	Unité
request_count	Le nombre de requêtes acheminées vers le webhook.	Compteur	**admission_allowed**, **kind_group**, **kind_kind**, **kind_version**, **request_operation**, **resource_group**, **resource_namespace**, **resource_resource**, **resource_version**	Entier (pas d'unité)
request_latencies	Le temps de réponse pour une demande de webhook.	Histogramme	**admission_allowed**, **kind_group**, **kind_kind**, **kind_version**, **request_operation**, **resource_group**, **resource_namespace**, **resource_resource**, **resource_version**	Millisecondes

8.1.4. Mesures du concours complet Knative

Les administrateurs de clusters peuvent afficher les mesures suivantes pour les composants Knative Eventing.

En agrégeant les mesures du code HTTP, les événements peuvent être séparés en deux catégories : les événements réussis (2xx) et les événements échoués (5xx).

8.1.4.1. Métriques d'entrée du courtier

Vous pouvez utiliser les mesures suivantes pour déboguer l'entrée du courtier, voir comment elle fonctionne et quels événements sont envoyés par le composant d'entrée.

Nom de la métrique	Description	Type	Tags	Unité
event_count	Nombre d'événements reçus par un courtier.	Compteur	**broker_name**, **event_type**, **namespace_name**, **response_code**, **response_code_class**, **unique_name**	Entier (pas d'unité)
event_dispatch_latencies	Temps nécessaire pour envoyer un événement à un canal.	Histogramme	**broker_name**, **event_type**, **namespace_name**, **response_code**, **response_code_class**, **unique_name**	Millisecondes

8.1.4.2. Métriques de filtrage du courtier

Vous pouvez utiliser les mesures suivantes pour déboguer les filtres du courtier, voir comment ils fonctionnent et quels événements sont distribués par les filtres. Vous pouvez également mesurer la latence de l'action de filtrage sur un événement.

Nom de la métrique	Description	Type	Tags	Unité
event_count	Nombre d'événements reçus par un courtier.	Compteur	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	Entier (pas d'unité)
event_dispatch_latencies	Temps nécessaire pour envoyer un événement à un canal.	Histogramme	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	Millisecondes

Nom de la métrique	Description	Type	Tags	Unité
event_processing_latencies	Temps nécessaire au traitement d'un événement avant qu'il ne soit envoyé à un abonné au déclencheur.	Histogramme	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **trigger_name**, **unique_name**	Millisecondes

8.1.4.3. Métriques du répartiteur InMemoryChannel

Vous pouvez utiliser les mesures suivantes pour déboguer les canaux **InMemoryChannel**, voir comment ils fonctionnent et quels événements sont envoyés par les canaux.

Nom de la métrique	Description	Type	Tags	Unité
event_count	Nombre d'événements envoyés par les canaux **InMemoryChannel**.	Compteur	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	Entier (pas d'unité)
event_dispatch_latencies	Temps nécessaire pour envoyer un événement à partir d'un canal **InMemoryChannel**.	Histogramme	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	Millisecondes

8.1.4.4. Mesures des sources d'événements

Vous pouvez utiliser les mesures suivantes pour vérifier que les événements ont été transmis de la source d'événements au puits d'événements connecté.

Nom de la métrique	Description	Type	Tags	Unité
event_count	Nombre d'événements envoyés par la source d'événements.	Compteur	**broker_name**, **container_name**, **filter_type**, **namespace_name**, **response_code**, **response_code_class**, **trigger_name**, **unique_name**	Entier (pas d'unité)
retry_event_count	Nombre d'événements renvoyés par la source d'événements après un premier échec.	Compteur	**event_source**, **event_type**, **name**, **namespace_name**, **resource_group**, **response_code**, **response_code_class**, **response_error**, **response_timeout**	Entier (pas d'unité)

8.1.5. Métriques de service Knative

Les administrateurs de clusters peuvent afficher les mesures suivantes pour les composants Knative Serving.

8.1.5.1. Mesures de l'activateur

Vous pouvez utiliser les mesures suivantes pour comprendre comment les applications réagissent lorsque le trafic passe par l'activateur.

Nom de la métrique	Description	Type	Tags	Unité

Nom de la métrique	Description	Type	Tags	Unité
request_concurrency	Nombre de demandes simultanées acheminées vers l'activateur, ou nombre moyen de demandes simultanées sur une période donnée.	Jauge	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **revision_name**, **service_name**	Entier (pas d'unité)
request_count	Nombre de demandes acheminées vers l'activateur. Il s'agit des demandes qui ont été satisfaites par le gestionnaire de l'activateur.	Compteur	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**,	Entier (pas d'unité)
request_latencies	Le temps de réponse en millisecondes pour une requête acheminée et satisfaite.	Histogramme	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	Millisecondes

8.1.5.2. Mesures de l'autoscaler

Le composant autoscaler expose un certain nombre de métriques liées au comportement de l'autoscaler pour chaque révision. Par exemple, à tout moment, vous pouvez surveiller le nombre ciblé de pods que l'autoscaler tente d'allouer pour un service, le nombre moyen de requêtes par seconde pendant la fenêtre stable, ou si l'autoscaler est en mode panique si vous utilisez le Knative pod autoscaler (KPA).

Nom de la métrique	Description	Type	Tags	Unité

Nom de la métrique	Description	Type	Tags	Unité
desired_pods	Le nombre de pods que l'autoscaler tente d'allouer pour un service.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
excess_burst_capacity	La capacité d'éclatement excédentaire servie sur la fenêtre stable.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
stable_request_concurrency	Le nombre moyen de requêtes pour chaque pod observé au cours de la fenêtre stable.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
panic_request_concurrency	Le nombre moyen de demandes pour chaque pod observé au cours de la fenêtre de panique.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
target_concurrency_per_pod	Nombre de demandes simultanées que l'autoscaler tente d'envoyer à chaque pod.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
stable_requests_per_second	Nombre moyen de requêtes par seconde pour chaque module observé au cours de la fenêtre stable.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
panic_requests_per_second	Le nombre moyen de requêtes par seconde pour chaque pod observé au cours de la fenêtre de panique.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
target_requests_per_second	Le nombre de requêtes par seconde que l'autoscaler cible pour chaque pod.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
panic_mode	Cette valeur est **1** si l'autoscaler est en mode panique, ou **0** si l'autoscaler n'est pas en mode panique.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
requested_pods	Le nombre de pods que l'autoscaler a demandé au cluster Kubernetes.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
actual_pods	Le nombre de pods qui sont alloués et qui sont actuellement prêts.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
not_ready_pods	Nombre de pods dont l'état n'est pas prêt.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)
pending_pods	Le nombre de pods qui sont actuellement en attente.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
terminating_pods	Le nombre de pods qui se terminent actuellement.	Jauge	**configuration_name**, **namespace_name**, **revision_name**, **service_name**	Entier (pas d'unité)

8.1.5.3. Mesures de la durée d'exécution de Go

Chaque processus du plan de contrôle de Knative Serving émet un certain nombre de statistiques de mémoire d'exécution Go(MemStats).

NOTE

La balise **name** pour chaque métrique est une balise vide.

Nom de la métrique	Description	Type	Tags	Unité
go_alloc	Nombre d'octets d'objets du tas alloués. Cette mesure est identique à **heap_alloc**.	Jauge	**name**	Entier (pas d'unité)
go_total_alloc	Le nombre cumulé d'octets alloués aux objets du tas.	Jauge	**name**	Entier (pas d'unité)
go_sys	Le nombre total d'octets de mémoire obtenu du système d'exploitation.	Jauge	**name**	Entier (pas d'unité)
go_lookups	Le nombre de recherches de pointeurs effectuées par le moteur d'exécution.	Jauge	**name**	Entier (pas d'unité)
go_mallocs	Le nombre cumulé d'objets du tas alloués.	Jauge	**name**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
go_frees	Le nombre cumulé d'objets du tas qui ont été libérés.	Jauge	**name**	Entier (pas d'unité)
go_heap_alloc	Nombre d'octets d'objets du tas alloués.	Jauge	**name**	Entier (pas d'unité)
go_heap_sys	Le nombre d'octets de la mémoire vive obtenu du système d'exploitation.	Jauge	**name**	Entier (pas d'unité)
go_heap_idle	Le nombre d'octets dans les travées inutilisées.	Jauge	**name**	Entier (pas d'unité)
go_heap_in_use	Le nombre d'octets dans les travées qui sont actuellement en cours d'utilisation.	Jauge	**name**	Entier (pas d'unité)
go_heap_released	Nombre d'octets de mémoire physique restitués au système d'exploitation.	Jauge	**name**	Entier (pas d'unité)
go_heap_objects	Nombre d'objets du tas alloués.	Jauge	**name**	Entier (pas d'unité)
go_stack_in_use	Nombre d'octets de la pile en cours d'utilisation.	Jauge	**name**	Entier (pas d'unité)
go_stack_sys	Le nombre d'octets de la mémoire de pile obtenu du système d'exploitation.	Jauge	**name**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
go_mspan_in_use	Le nombre d'octets des structures **mspan** allouées.	Jauge	**name**	Entier (pas d'unité)
go_mspan_sys	Le nombre d'octets de mémoire obtenus du système d'exploitation pour les structures **mspan**.	Jauge	**name**	Entier (pas d'unité)
go_mcache_in_use	Le nombre d'octets des structures **mcache** allouées.	Jauge	**name**	Entier (pas d'unité)
go_mcache_sys	Le nombre d'octets de mémoire obtenus du système d'exploitation pour les structures **mcache**.	Jauge	**name**	Entier (pas d'unité)
go_bucket_hash_sys	Le nombre d'octets de mémoire dans les tables de hachage des seaux de profilage.	Jauge	**name**	Entier (pas d'unité)
go_gc_sys	Nombre d'octets de mémoire dans les métadonnées du ramassage des ordures.	Jauge	**name**	Entier (pas d'unité)
go_other_sys	Le nombre d'octets de mémoire dans les diverses allocations de durée d'exécution en dehors du tas.	Jauge	**name**	Entier (pas d'unité)

Nom de la métrique	Description	Type	Tags	Unité
go_next_gc	La taille du tas cible du prochain cycle de ramassage des ordures.	Jauge	**name**	Entier (pas d'unité)
go_last_gc	Heure à laquelle le dernier ramassage des ordures s'est achevé (*Epoch* ou heure Unix).	Jauge	**name**	Nanosecondes
go_total_gc_pause_ns	Le temps cumulé des pauses du ramasse-miettes *stop-the-world* depuis le début du programme.	Jauge	**name**	Nanosecondes
go_num_gc	Nombre de cycles de ramassage d'ordures terminés.	Jauge	**name**	Entier (pas d'unité)
go_num_forced_gc	Nombre de cycles de ramassage d'ordures forcés en raison de l'appel d'une application à la fonction de ramassage d'ordures.	Jauge	**name**	Entier (pas d'unité)
go_gc_cpu_fraction	La fraction du temps CPU disponible du programme qui a été utilisée par le ramasse-miettes depuis le début du programme.	Jauge	**name**	Entier (pas d'unité)

8.2. MESURES POUR LES DÉVELOPPEURS

8.2.1. Vue d'ensemble des métriques pour les développeurs Serverless

Les métriques permettent aux développeurs de surveiller les performances des services Knative. Vous pouvez utiliser la pile de surveillance OpenShift Container Platform pour enregistrer et visualiser les contrôles de santé et les métriques de vos services Knative.

Vous pouvez visualiser différentes métriques pour OpenShift Serverless en naviguant vers **Dashboards** dans la console web de OpenShift Container Platform **Developer** perspective.

AVERTISSEMENT

Si Service Mesh est activé avec mTLS, les métriques pour Knative Serving sont désactivées par défaut car Service Mesh empêche Prometheus de récupérer les métriques.

Pour plus d'informations sur la résolution de ce problème, voir Activation des métriques Knative Serving lors de l'utilisation de Service Mesh avec mTLS.

Le scraping des métriques n'affecte pas l'autoscaling d'un service Knative, car les requêtes de scraping ne passent pas par l'activateur. Par conséquent, aucun scraping n'a lieu si aucun pod n'est en cours d'exécution.

8.2.1.1. Ressources supplémentaires

- Aperçu de la surveillance
- Permettre le suivi de projets définis par l'utilisateur
- Spécifier comment un service est contrôlé

8.2.2. Métriques de service Knative exposées par défaut

Tableau 8.1. Métriques exposées par défaut pour chaque service Knative sur le port 9090

Nom, unité et type de métrique	Description	Étiquettes métriques
queue_requests_per_second Unité métrique : sans dimension Type métrique : jauge	Nombre de requêtes par seconde qui atteignent le proxy de file d'attente. Formule : **stats.RequestCount / r.reportingPeriodSeconds** **stats.RequestCount** est calculé directement à partir des statistiques du réseau **pkg** pour la durée du rapport.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

Nom, unité et type de métrique	Description	Étiquettes métriques
queue_proxied_operations_per_second Unité métrique : sans dimension Type métrique : jauge	Nombre de requêtes proxy par seconde. Formule : **stats.ProxiedRequestCount / r.reportingPeriodSeconds** **stats.ProxiedRequestCount** est calculé directement à partir des statistiques du réseau **pkg** pour la durée du rapport.	

Nom, unité et type de métrique	Description	Étiquettes métriques
queue_average_concurrent_requests Unité métrique : sans dimension Type métrique : jauge	Nombre de demandes actuellement traitées par ce module. La concurrence moyenne est calculée comme suit du côté de la mise en réseau **pkg**: • Lorsqu'un changement se produit sur **req**, le delta de temps entre les changements est calculé. Sur la base du résultat, le nombre actuel de concurrents sur le delta est calculé et ajouté au nombre actuel de concurrents calculé. En outre, la somme des deltas est conservée. La concurrence actuelle sur le delta est calculée comme suit : **global_concurrency** × delta • Chaque fois qu'un rapport est effectué, la somme et la concurrence calculée actuelle sont réinitialisées. • Lors de l'établissement du rapport sur la concurrence moyenne, la concurrence calculée actuelle est divisée par la somme des deltas. • Lorsqu'une nouvelle demande arrive, le compteur de simultanéité global est augmenté. Lorsqu'une demande est terminée, le compteur est diminué.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

Nom, unité et type de métrique	Description	Étiquettes métriques
queue_average_proxied_concurrent_requests Unité métrique : sans dimension Type métrique : jauge	Nombre de requêtes par procuration actuellement traitées par ce module : **stats.AverageProxiedConcurrency**	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"
process_uptime Unité métrique : secondes Type métrique : jauge	Le nombre de secondes depuis lesquelles le processus est en cours.	destination_configuration="event-display", destination_namespace="pingsource1", destination_pod="event-display-00001-deployment-6b455479cb-75p6w", destination_revision="event-display-00001"

Tableau 8.2. Métriques exposées par défaut pour chaque service Knative sur le port 9091

Nom, unité et type de métrique	Description	Étiquettes métriques
request_count Unité métrique : sans dimension Type métrique : compteur	Le nombre de demandes qui sont acheminées vers **queue-proxy**.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
request_latencies Unité métrique : millisecondes Type de mesure : histogramme	Le temps de réponse en millisecondes.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"

Nom, unité et type de métrique	Description	Étiquettes métriques
app_request_count Unité métrique : sans dimension Type métrique : compteur	Le nombre de demandes qui sont acheminées vers **user-container**.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
app_request_latencies Unité métrique : millisecondes Type de mesure : histogramme	Le temps de réponse en millisecondes.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"
queue_depth Unité métrique : sans dimension Type métrique : jauge	Le nombre actuel d'éléments dans la file d'attente de service et d'attente, ou non signalé si la concurrence est illimitée. **breaker.inFlight** est utilisé.	configuration_name="event-display", container_name="queue-proxy", namespace_name="apiserversource1", pod_name="event-display-00001-deployment-658fd4f9cf-qcnr5", response_code="200", response_code_class="2xx", revision_name="event-display-00001", service_name="event-display"

8.2.3. Service Knative avec métriques d'application personnalisées

Vous pouvez étendre l'ensemble des métriques exportées par un service Knative. L'implémentation exacte dépend de votre application et du langage utilisé.

La liste suivante met en œuvre un exemple d'application Go qui exporte la métrique personnalisée du nombre d'événements traités.

```
package main

import (
  "fmt"
  "log"
  "net/http"
  "os"
```

```
  "github.com/prometheus/client_golang/prometheus" 1
  "github.com/prometheus/client_golang/prometheus/promauto"
  "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
  opsProcessed = promauto.NewCounter(prometheus.CounterOpts{ 2
    Name: "myapp_processed_ops_total",
    Help: "The total number of processed events",
  })
)


func handler(w http.ResponseWriter, r *http.Request) {
  log.Print("helloworld: received a request")
  target := os.Getenv("TARGET")
  if target == "" {
    target = "World"
  }
  fmt.Fprintf(w, "Hello %s!\n", target)
  opsProcessed.Inc() 3
}

func main() {
  log.Print("helloworld: starting server...")

  port := os.Getenv("PORT")
  if port == "" {
    port = "8080"
  }

  http.HandleFunc("/", handler)

  // Separate server for metrics requests
  go func() { 4
    mux := http.NewServeMux()
    server := &http.Server{
      Addr: fmt.Sprintf(":%s", "9095"),
      Handler: mux,
    }
    mux.Handle("/metrics", promhttp.Handler())
    log.Printf("prometheus: listening on port %s", 9095)
    log.Fatal(server.ListenAndServe())
  }()

   // Use same port as normal requests for metrics
  //http.Handle("/metrics", promhttp.Handler()) 5
  log.Printf("helloworld: listening on port %s", port)
  log.Fatal(http.ListenAndServe(fmt.Sprintf(":%s", port), nil))
}
```

1 Y compris les paquets Prometheus.

2 Définition de la métrique **opsProcessed**.

3 Incrémentation de la métrique **opsProcessed**.

4 Configurer l'utilisation d'un serveur séparé pour les demandes de métriques.

5 Configuration pour utiliser le même port que les demandes normales de métriques et le sous-chemin **metrics**.

8.2.4. Configuration pour l'extraction de métriques personnalisées

La collecte de métriques personnalisées est effectuée par une instance de Prometheus destinée à la surveillance de la charge de travail de l'utilisateur. Une fois que vous avez activé la surveillance de la charge de travail de l'utilisateur et créé l'application, vous avez besoin d'une configuration qui définit comment la pile de surveillance récupérera les métriques.

L'exemple de configuration suivant définit le site **ksvc** pour votre application et configure le moniteur de service. La configuration exacte dépend de votre application et de la manière dont elle exporte les métriques.

```
apiVersion: serving.knative.dev/v1 1
kind: Service
metadata:
  name: helloworld-go
spec:
  template:
    metadata:
      labels:
        app: helloworld-go
      annotations:
    spec:
      containers:
      - image: docker.io/skonto/helloworld-go:metrics
        resources:
          requests:
            cpu: "200m"
        env:
        - name: TARGET
          value: "Go Sample v1"
---
apiVersion: monitoring.coreos.com/v1 2
kind: ServiceMonitor
metadata:
  labels:
  name: helloworld-go-sm
spec:
  endpoints:
  - port: queue-proxy-metrics
    scheme: http
  - port: app-metrics
    scheme: http
  namespaceSelector: {}
  selector:
    matchLabels:
      name:  helloworld-go-sm
---
```

```
apiVersion: v1 3
kind: Service
metadata:
  labels:
    name:  helloworld-go-sm
  name:  helloworld-go-sm
spec:
  ports:
  - name: queue-proxy-metrics
    port: 9091
    protocol: TCP
    targetPort: 9091
  - name: app-metrics
    port: 9095
    protocol: TCP
    targetPort: 9095
  selector:
    serving.knative.dev/service: helloworld-go
  type: ClusterIP
```

1. Spécification de l'application.
2. Configuration des métriques de l'application qui sont récupérées.
3. Configuration de la manière dont les métriques sont récupérées.

8.2.5. Examiner les paramètres d'un service

Après avoir configuré l'application pour qu'elle exporte les métriques et la pile de surveillance pour qu'elle les récupère, vous pouvez examiner les métriques dans la console web.

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Serving.

Procédure

1. Facultatif : Exécutez des requêtes sur votre application que vous pourrez voir dans les métriques :

```
$ hello_route=$(oc get ksvc helloworld-go -n ns1 -o jsonpath='{.status.url}') && \
    curl $hello_route
```

Exemple de sortie

```
Hello Go Sample v1!
```

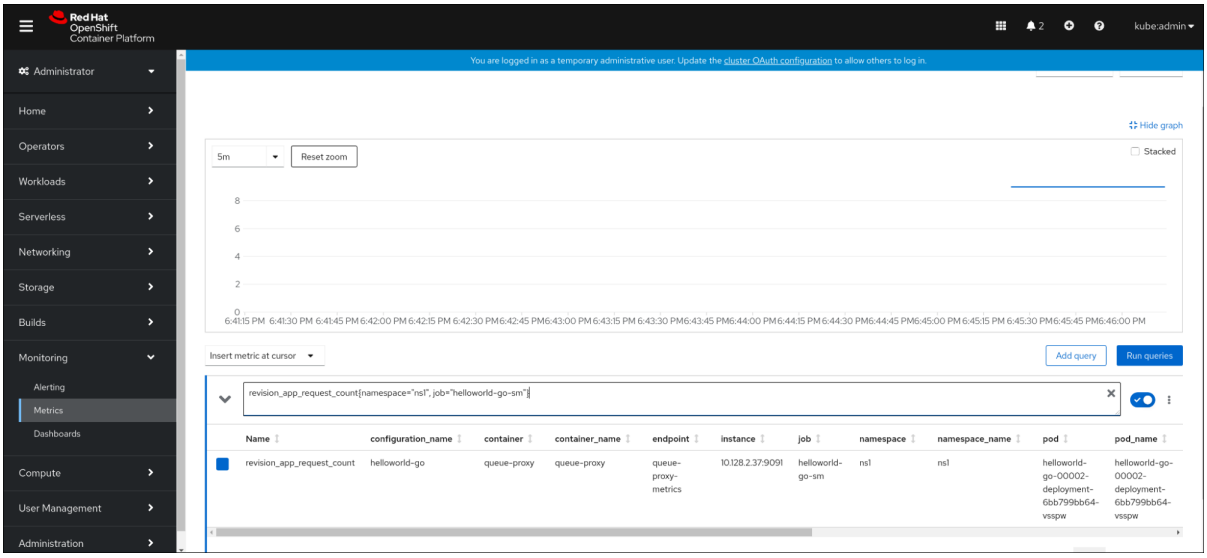
2. Dans la console web, naviguez jusqu'à l'interface **Observe → Metrics**.
3. Dans le champ de saisie, entrez la requête pour la mesure que vous voulez observer, par exemple :

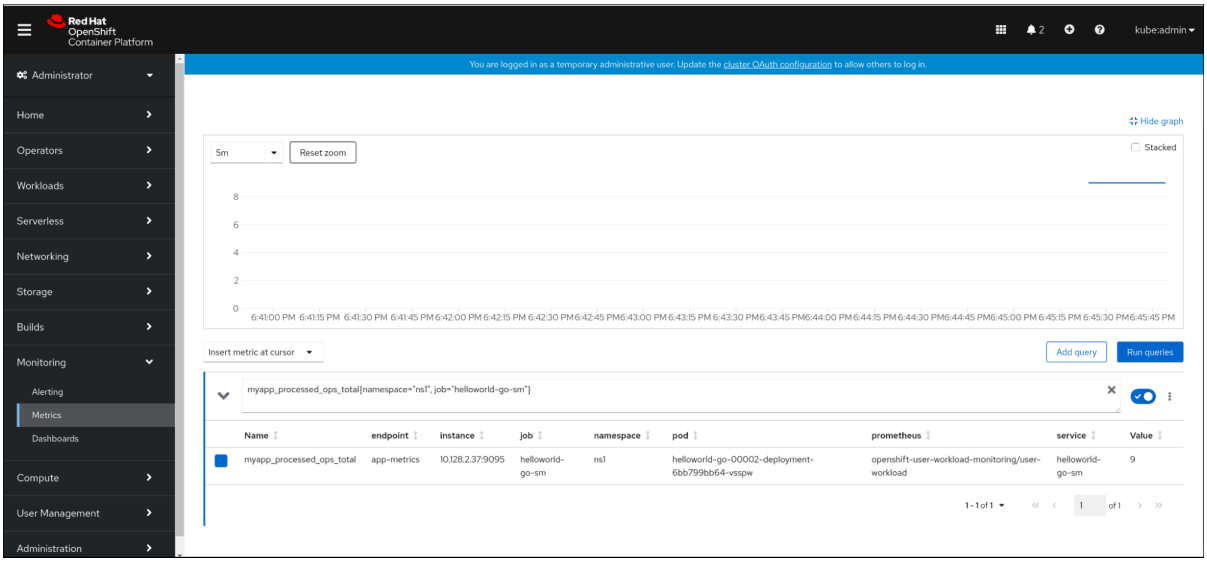
```
revision_app_request_count{namespace="ns1", job="helloworld-go-sm"}
```

Autre exemple :

```
myapp_processed_ops_total{namespace="ns1", job="helloworld-go-sm"}
```

4. Observez les mesures visualisées :





8.2.5.1. Métriques de proxy de file d'attente

Chaque service Knative dispose d'un conteneur proxy qui transmet les connexions au conteneur d'application. Un certain nombre de mesures sont rapportées pour la performance du proxy de file d'attente.

Vous pouvez utiliser les mesures suivantes pour déterminer si les demandes sont mises en file d'attente du côté du proxy et le délai réel de traitement des demandes du côté de l'application.

Nom de la métrique	Description	Type	Tags	Unité
revision_request_count	Le nombre de demandes qui sont acheminées vers le pod **queue-proxy**.	Compteur	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	Entier (pas d'unité)
revision_request_latencies	Le temps de réponse des demandes de révision.	Histogramme	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	Millisecondes
revision_app_request_count	Nombre de requêtes acheminées vers le pod **user-container**.	Compteur	**configuration_name**, **container_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	Entier (pas d'unité)
revision_app_request_latencies	Le temps de réponse des demandes d'application de révision.	Histogramme	**configuration_name**, **namespace_name**, **pod_name**, **response_code**, **response_code_class**, **revision_name**, **service_name**	Millisecondes

Nom de la métrique	Description	Type	Tags	Unité
revision_queue_depth	Nombre actuel d'éléments dans les files d'attente **serving** et **waiting**. Cette mesure n'est pas rapportée si une concurrence illimitée est configurée.	Jauge	**configuration_name**, **event-display**, **container_name**, **namespace_name**, **pod_name**, **response_code_class**, **revision_name**, **service_name**	Entier (pas d'unité)

8.2.6. Tableau de bord pour les indicateurs de service

Vous pouvez examiner les mesures à l'aide d'un tableau de bord dédié qui regroupe les mesures de proxy de file d'attente par espace de noms.

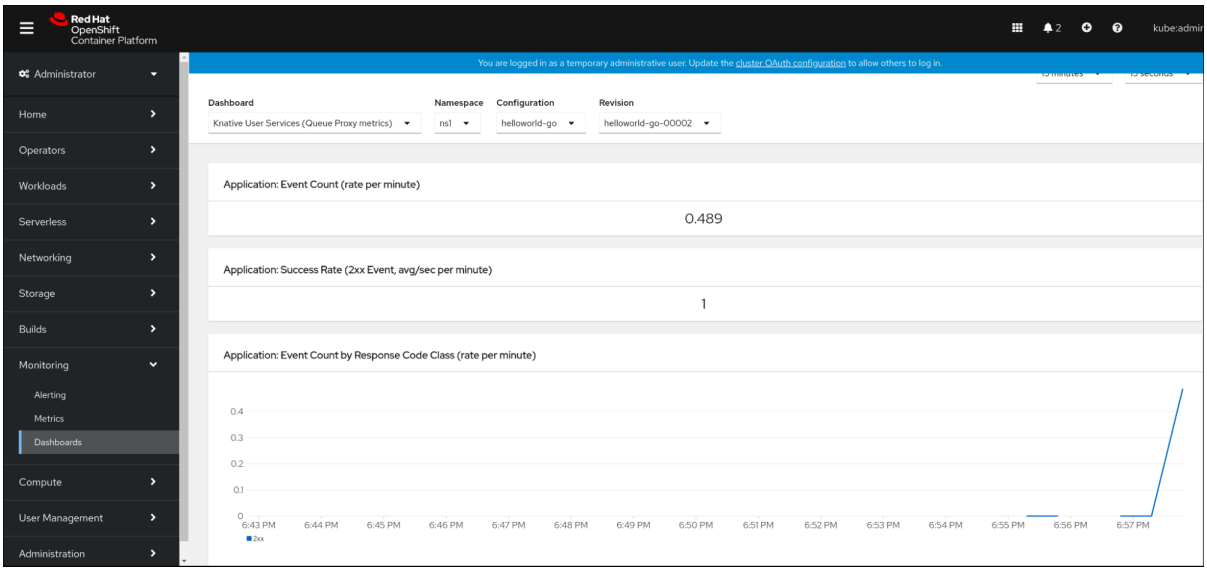
8.2.6.1. Examiner les mesures d'un service dans le tableau de bord

Conditions préalables

- Vous vous êtes connecté à la console web de OpenShift Container Platform.
- Vous avez installé OpenShift Serverless Operator et Knative Serving.

Procédure

1. Dans la console web, naviguez jusqu'à l'interface **Observe → Metrics**.
2. Sélectionnez le tableau de bord **Knative User Services (Queue Proxy metrics)**.
3. Sélectionnez les **Namespace**, **Configuration**, et **Revision** qui correspondent à votre application.
4. Observez les mesures visualisées :



8.3. JOURNALISATION DES CLUSTERS

8.3.1. Utiliser OpenShift Logging avec OpenShift Serverless

8.3.1.1. À propos du déploiement du sous-système de journalisation pour Red Hat OpenShift

Les administrateurs de cluster d'OpenShift Container Platform peuvent déployer le sous-système de journalisation en utilisant la console web d'OpenShift Container Platform ou le CLI pour installer l'OpenShift Elasticsearch Operator et le Red Hat OpenShift Logging Operator. Lorsque les opérateurs sont installés, vous créez une ressource personnalisée (CR) **ClusterLogging** pour planifier les pods du sous-système de journalisation et les autres ressources nécessaires à la prise en charge du sous-système de journalisation. Les opérateurs sont responsables du déploiement, de la mise à niveau et de la maintenance du sous-système de journalisation.

Le CR **ClusterLogging** définit un environnement de sous-système de journalisation complet qui inclut tous les composants de la pile de journalisation pour collecter, stocker et visualiser les journaux. L'opérateur de journalisation de Red Hat OpenShift surveille le CR du sous-système de journalisation et ajuste le déploiement de la journalisation en conséquence.

Les administrateurs et les développeurs d'applications peuvent consulter les journaux des projets pour lesquels ils ont un accès de visualisation.

8.3.1.2. À propos du déploiement et de la configuration du sous-système de journalisation pour Red Hat OpenShift

Le sous-système de journalisation est conçu pour être utilisé avec la configuration par défaut, qui est adaptée aux clusters OpenShift Container Platform de petite et moyenne taille.

Les instructions d'installation qui suivent comprennent un exemple de ressource personnalisée (CR) **ClusterLogging**, que vous pouvez utiliser pour créer une instance de sous-système de journalisation et configurer votre environnement de sous-système de journalisation.

Si vous souhaitez utiliser l'installation par défaut du sous-système de journalisation, vous pouvez utiliser directement l'exemple de CR.

Si vous souhaitez personnaliser votre déploiement, apportez des modifications à l'exemple de CR selon vos besoins. Ce qui suit décrit les configurations que vous pouvez faire lors de l'installation de votre instance OpenShift Logging ou modifier après l'installation. Consultez les sections Configuration pour plus d'informations sur l'utilisation de chaque composant, y compris les modifications que vous pouvez apporter en dehors de la ressource personnalisée **ClusterLogging**.

8.3.1.2.1. Configuration et réglage du sous-système de journalisation

Vous pouvez configurer votre sous-système de journalisation en modifiant la ressource personnalisée **ClusterLogging** déployée dans le projet **openshift-logging**.

Vous pouvez modifier n'importe lequel des composants suivants lors de l'installation ou après l'installation :

Mémoire et CPU

Vous pouvez ajuster les limites de CPU et de mémoire pour chaque composant en modifiant le bloc **resources** avec des valeurs de mémoire et de CPU valides :

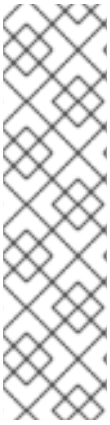
```
spec:
  logStore:
    elasticsearch:
      resources:
        limits:
          cpu:
          memory: 16Gi
        requests:
          cpu: 500m
          memory: 16Gi
      type: "elasticsearch"
  collection:
    logs:
      fluentd:
        resources:
          limits:
            cpu:
            memory:
          requests:
            cpu:
            memory:
        type: "fluentd"
  visualization:
    kibana:
      resources:
        limits:
          cpu:
          memory:
        requests:
          cpu:
          memory:
      type: kibana
```

Stockage Elasticsearch

Vous pouvez configurer une classe et une taille de stockage persistant pour le cluster Elasticsearch à l'aide des paramètres **storageClass name** et **size**. L'opérateur de journalisation de Red Hat OpenShift crée une réclamation de volume persistant (PVC) pour chaque nœud de données dans le cluster Elasticsearch en fonction de ces paramètres.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage:
        storageClassName: "gp2"
        size: "200G"
```

Cet exemple spécifie que chaque nœud de données du cluster sera lié à un PVC qui demande 200 Go de stockage. Chaque groupe de données primaire sera soutenu par une seule réplique.



NOTE

L'omission du bloc **storage** se traduit par un déploiement qui ne comprend que le stockage éphémère.

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage: {}
```

Politique de réplication Elasticsearch

Vous pouvez définir la politique qui définit comment les shards Elasticsearch sont répliqués entre les nœuds de données dans le cluster :

- **FullRedundancy**. Les fichiers de chaque index sont entièrement répliqués sur chaque nœud de données.
- **MultipleRedundancy**. Pour chaque index, les fragments sont répartis sur la moitié des nœuds de données.
- **SingleRedundancy**. Une copie unique de chaque morceau de données (shard). Les journaux sont toujours disponibles et récupérables tant qu'il existe au moins deux nœuds de données.
- **ZeroRedundancy**. Il n'y a pas de copies des morceaux. Les journaux peuvent être indisponibles (ou perdus) en cas de panne ou de défaillance d'un nœud.

8.3.1.2.2. Exemple de ressource personnalisée ClusterLogging modifiée

Voici un exemple de ressource personnalisée **ClusterLogging** modifiée à l'aide des options décrites précédemment.

Exemple de ressource personnalisée ClusterLogging modifiée

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    retentionPolicy:
      application:
        maxAge: 1d
      infra:
        maxAge: 7d
      audit:
        maxAge: 7d
    elasticsearch:
      nodeCount: 3
```

```
    resources:
      limits:
        memory: 32Gi
      requests:
        cpu: 3
        memory: 32Gi
      storage:
        storageClassName: "gp2"
        size: "200G"
    redundancyPolicy: "SingleRedundancy"
visualization:
  type: "kibana"
  kibana:
    resources:
      limits:
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 1Gi
    replicas: 1
collection:
  logs:
    type: "fluentd"
    fluentd:
      resources:
        limits:
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 1Gi
```

8.3.2. Recherche de logs pour les composants Knative Serving

Vous pouvez trouver les journaux des composants Knative Serving en suivant la procédure suivante.

8.3.2.1. Utiliser OpenShift Logging pour trouver les logs des composants Knative Serving

Conditions préalables

- Installez le CLI OpenShift (**oc**).

Procédure

1. Obtenir l'itinéraire Kibana :

```
$ oc -n openshift-logging get route kibana
```

2. Utilisez l'URL de l'itinéraire pour naviguer vers le tableau de bord Kibana et vous connecter.
3. Vérifiez que l'index est défini sur **.all**. Si l'index n'est pas défini sur **.all**, seuls les journaux du système OpenShift Container Platform seront listés.

4. Filtrez les journaux en utilisant l'espace de noms **knative-serving**. Saisissez **kubernetes.namespace_name:knative-serving** dans le champ de recherche pour filtrer les résultats.

NOTE

Knative Serving utilise par défaut une journalisation structurée. Vous pouvez activer l'analyse de ces logs en personnalisant les paramètres OpenShift Logging Fluentd. Cela rend les logs plus consultables et permet de filtrer au niveau des logs pour identifier rapidement les problèmes.

8.3.3. Trouver des logs pour les services Knative Serving

Vous pouvez trouver les journaux des services Knative Serving en suivant la procédure suivante.

8.3.3.1. Utiliser OpenShift Logging pour trouver les logs des services déployés avec Knative Serving

Avec OpenShift Logging, les logs que vos applications écrivent à la console sont collectés dans Elasticsearch. La procédure suivante explique comment appliquer ces fonctionnalités aux applications déployées à l'aide de Knative Serving.

Conditions préalables

- Installez le CLI OpenShift (**oc**).

Procédure

1. Obtenir l'itinéraire Kibana :

```
$ oc -n openshift-logging get route kibana
```

2. Utilisez l'URL de l'itinéraire pour naviguer vers le tableau de bord Kibana et vous connecter.

3. Vérifiez que l'index est défini sur **.all**. Si l'index n'est pas défini sur **.all**, seuls les journaux du système OpenShift seront listés.

4. Filtrez les journaux en utilisant l'espace de noms **knative-serving**. Entrez un filtre pour le service dans la boîte de recherche pour filtrer les résultats.

   #### Exemple de filtre

```
kubernetes.namespace_name:default AND kubernetes.labels.serving_knative_dev\/service:{service_name}
```

   Vous pouvez également filtrer en utilisant /**configuration** ou /**revision**.

5. Affinez votre recherche en utilisant **kubernetes.container_name:<user_container>** pour n'afficher que les journaux générés par votre application. Sinon, vous verrez les journaux du proxy de file d'attente.

NOTE

Utilisez la journalisation structurée basée sur JSON dans votre application pour permettre un filtrage rapide de ces journaux dans les environnements de production.

8.4. TRAÇAGE

8.4.1. Traçage des demandes

Le traçage distribué enregistre le cheminement d'une requête à travers les différents services qui composent une application. Il est utilisé pour relier les informations relatives à différentes unités de travail, afin de comprendre l'ensemble de la chaîne d'événements d'une transaction distribuée. Les unités de travail peuvent être exécutées dans différents processus ou hôtes.

8.4.1.1. Aperçu du traçage distribué

En tant que propriétaire de service, vous pouvez utiliser le traçage distribué pour instrumenter vos services afin de recueillir des informations sur votre architecture de service. Vous pouvez utiliser le traçage distribué pour la surveillance, le profilage du réseau et le dépannage de l'interaction entre les composants dans les applications modernes, cloud-natives et basées sur les microservices.

Le traçage distribué permet d'exécuter les fonctions suivantes :

- Contrôler les transactions distribuées
- Optimiser les performances et la latence
- Effectuer une analyse des causes profondes

Le traçage distribué de Red Hat OpenShift se compose de deux éléments principaux :

- **Red Hat OpenShift distributed tracing platform**- Ce composant est basé sur le projet open source Jaeger.
- **Red Hat OpenShift distributed tracing data collection**- Ce composant est basé sur le projet open source OpenTelemetry.

Ces deux composants sont basés sur les API et l'instrumentation OpenTracing, neutres vis-à-vis des fournisseurs.

8.4.1.2. Ressources supplémentaires

- Architecture de traçage distribuée Red Hat OpenShift
- Installation du traçage distribué

8.4.2. Utiliser le traçage distribué de Red Hat OpenShift

Vous pouvez utiliser le traçage distribué Red Hat OpenShift avec OpenShift Serverless pour surveiller et dépanner les applications sans serveur.

8.4.2.1. Utiliser Red Hat OpenShift distributed tracing pour activer le traçage distribué

Le traçage distribué de Red Hat OpenShift est constitué de plusieurs composants qui fonctionnent ensemble pour collecter, stocker et afficher les données de traçage.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous n'avez pas encore installé OpenShift Serverless Operator, Knative Serving et Knative Eventing. Ceux-ci doivent être installés après l'installation de Red Hat OpenShift distributed tracing.
- Vous avez installé le traçage distribué de Red Hat OpenShift en suivant la documentation OpenShift Container Platform "Installing distributed tracing".
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créer une ressource personnalisée (CR) à l'adresse **OpenTelemetryCollector**:

   #### Exemple d'OpenTelemetryCollector CR

   ```
   apiVersion: opentelemetry.io/v1alpha1
   kind: OpenTelemetryCollector
   metadata:
     name: cluster-collector
     namespace: <namespace>
   spec:
     mode: deployment
     config: |
       receivers:
         zipkin:
       processors:
       exporters:
         jaeger:
           endpoint: jaeger-all-in-one-inmemory-collector-headless.tracing-system.svc:14250
           tls:
             ca_file: "/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt"
         logging:
       service:
         pipelines:
           traces:
             receivers: [zipkin]
             processors: []
             exporters: [jaeger, logging]
   ```

2. Vérifiez que vous avez deux pods en cours d'exécution dans l'espace de noms où le traçage distribué de Red Hat OpenShift est installé :

   ```
   $ oc get pods -n <namespace>
   ```

   #### Exemple de sortie

```
NAME                                        READY   STATUS    RESTARTS   AGE
cluster-collector-collector-85c766b5c-b5g99   1/1     Running   0          5m56s
jaeger-all-in-one-inmemory-ccbc9df4b-ndkl5    2/2     Running   0          15m
```

3. Vérifiez que les services sans tête suivants ont été créés :

```
oc get svc -n <namespace> | grep headless
```

Exemple de sortie

```
cluster-collector-collector-headless          ClusterIP   None            <none>        9411/TCP
7m28s
jaeger-all-in-one-inmemory-collector-headless   ClusterIP   None            <none>
9411/TCP,14250/TCP,14267/TCP,14268/TCP   16m
```

Ces services sont utilisés pour configurer Jaeger, Knative Serving et Knative Eventing. Le nom du service Jaeger peut varier.

4. Installez l'OpenShift Serverless Operator en suivant la documentation "Installing the OpenShift Serverless Operator".

5. Installez Knative Serving en créant le CR **KnativeServing** suivant :

Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
    name: knative-serving
    namespace: knative-serving
spec:
  config:
    tracing:
      backend: "zipkin"
      zipkin-endpoint: "http://cluster-collector-collector-headless.tracing-
system.svc:9411/api/v2/spans"
      debug: "false"
      sample-rate: "0.1" 1
```

1 Le site **sample-rate** définit la probabilité d'échantillonnage. L'utilisation de **sample-rate: "0.1"** signifie qu'une trace sur dix est échantillonnée.

6. Installez Knative Eventing en créant le CR **KnativeEventing** suivant :

Exemple KnativeEventing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
    name: knative-eventing
    namespace: knative-eventing
spec:
  config:
```

```
    tracing:
      backend: "zipkin"
      zipkin-endpoint: "http://cluster-collector-collector-headless.tracing-
system.svc:9411/api/v2/spans"
      debug: "false"
      sample-rate: "0.1" 1
```

1 Le site **sample-rate** définit la probabilité d'échantillonnage. L'utilisation de **sample-rate: "0.1"** signifie qu'une trace sur dix est échantillonnée.

7. Créer un service Knative :

Exemple de service

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: helloworld-go
spec:
  template:
    metadata:
      labels:
        app: helloworld-go
      annotations:
        autoscaling.knative.dev/minScale: "1"
        autoscaling.knative.dev/target: "1"
    spec:
      containers:
      - image: quay.io/openshift-knative/helloworld:v1.2
        imagePullPolicy: Always
        resources:
          requests:
            cpu: "200m"
        env:
        - name: TARGET
          value: "Go Sample v1"
```

8. Formulez des demandes auprès du service :

Exemple de demande HTTPS

```
$ curl https://helloworld-go.example.com
```

9. Obtenir l'URL de la console web Jaeger :

Example command

```
$ oc get route jaeger-all-in-one-inmemory  -o jsonpath='{.spec.host}' -n <namespace>
```

Vous pouvez maintenant examiner les traces en utilisant la console Jaeger.

8.4.3. Utilisation du traçage distribué de Jaeger

Si vous ne souhaitez pas installer tous les composants de Red Hat OpenShift distributed tracing, vous pouvez toujours utiliser le traçage distribué sur OpenShift Container Platform avec OpenShift Serverless.

8.4.3.1. Configurer Jaeger pour activer le traçage distribué

Pour activer le traçage distribué à l'aide de Jaeger, vous devez installer et configurer Jaeger en tant qu'intégration autonome.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez installé OpenShift Serverless Operator, Knative Serving et Knative Eventing.
- Vous avez installé la plateforme de traçage distribuée Red Hat OpenShift Operator.
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créez et appliquez une ressource personnalisée (CR) **Jaeger** qui contient les éléments suivants :

   Jaeger CR

   ```
   apiVersion: jaegertracing.io/v1
   kind: Jaeger
   metadata:
     name: jaeger
     namespace: default
   ```

2. Activer le traçage pour Knative Serving, en éditant le CR **KnativeServing** et en ajoutant une configuration YAML pour le traçage :

   Exemple de traçage YAML pour le service

   ```
   apiVersion: operator.knative.dev/v1beta1
   kind: KnativeServing
   metadata:
     name: knative-serving
     namespace: knative-serving
   spec:
     config:
       tracing:
         sample-rate: "0.1" 1
         backend: zipkin 2
         zipkin-endpoint: "http://jaeger-collector.default.svc.cluster.local:9411/api/v2/spans" 3
         debug: "false" 4
   ```

1. Le site **sample-rate** définit la probabilité d'échantillonnage. L'utilisation de **sample-rate: "0.1"** signifie qu'une trace sur dix est échantillonnée.
2. **backend** doit être réglé sur **zipkin**.
3. L'adresse **zipkin-endpoint** doit pointer vers le point de terminaison de votre service **jaeger-collector**. Pour obtenir ce point de terminaison, remplacez l'espace de noms dans lequel la CR Jaeger est appliquée.
4. Le mode débogage doit être défini sur **false**. L'activation du mode débogage en définissant **debug: "true"** permet d'envoyer toutes les travées au serveur, en contournant l'échantillonnage.

3. Activez le traçage pour Knative Eventing en modifiant le CR **KnativeEventing**:

Exemple de traçage YAML pour l'événementiel

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeEventing
metadata:
  name: knative-eventing
  namespace: knative-eventing
spec:
  config:
    tracing:
      sample-rate: "0.1" 1
      backend: zipkin 2
      zipkin-endpoint: "http://jaeger-collector.default.svc.cluster.local:9411/api/v2/spans" 3
      debug: "false" 4
```

1. Le site **sample-rate** définit la probabilité d'échantillonnage. L'utilisation de **sample-rate: "0.1"** signifie qu'une trace sur dix est échantillonnée.
2. Définir **backend** à **zipkin**.
3. Dirigez le site **zipkin-endpoint** vers le point d'accès à votre service **jaeger-collector**. Pour obtenir ce point de terminaison, remplacez l'espace de noms dans lequel la CR Jaeger est appliquée.
4. Le mode débogage doit être défini sur **false**. L'activation du mode débogage en définissant **debug: "true"** permet d'envoyer toutes les travées au serveur, en contournant l'échantillonnage.

Vérification

Vous pouvez accéder à la console web Jaeger pour voir les données de traçage, en utilisant la route **jaeger**.

1. Obtenez le nom d'hôte de la route **jaeger** en entrant la commande suivante :

```
$ oc get route jaeger -n default
```

Exemple de sortie

```
NAME     HOST/PORT                         PATH   SERVICES      PORT    TERMINATION
WILDCARD
jaeger   jaeger-default.apps.example.com          jaeger-query  <all>   reencrypt     None
```

2. Ouvrez l'adresse du point de terminaison dans votre navigateur pour afficher la console.

CHAPITRE 9. INTÉGRATIONS

9.1. INTÉGRATION DE SERVICE MESH AVEC OPENSHIFT SERVERLESS

L'opérateur OpenShift Serverless fournit Kourier comme ingress par défaut pour Knative. Cependant, vous pouvez utiliser Service Mesh avec OpenShift Serverless, que Kourier soit activé ou non. L'intégration avec Kourier désactivé vous permet de configurer des options de réseau et de routage supplémentaires que l'ingress de Kourier ne prend pas en charge, telles que la fonctionnalité mTLS.

IMPORTANT

OpenShift Serverless ne prend en charge que l'utilisation des fonctionnalités de Red Hat OpenShift Service Mesh qui sont explicitement documentées dans ce guide, et ne prend pas en charge d'autres fonctionnalités non documentées.

9.1.1. Conditions préalables

- Les exemples des procédures suivantes utilisent le domaine **example.com**. Le certificat d'exemple pour ce domaine est utilisé comme autorité de certification (CA) qui signe le certificat du sous-domaine.
  Pour effectuer et vérifier ces procédures dans le cadre de votre déploiement, vous devez disposer d'un certificat signé par une autorité de certification publique largement reconnue ou par une autorité de certification fournie par votre organisation. Les exemples de commandes doivent être adaptés en fonction du domaine, du sous-domaine et de l'autorité de certification.

- Vous devez configurer le certificat wildcard pour qu'il corresponde au domaine de votre cluster OpenShift Container Platform. Par exemple, si l'adresse de votre console OpenShift Container Platform est **https://console-openshift-console.apps.openshift.example.com**vous devez configurer le certificat Wildcard pour que le domaine soit ***.apps.openshift.example.com**. Pour plus d'informations sur la configuration des certificats génériques, consultez la rubrique suivante à propos de *Creating a certificate to encrypt incoming external traffic* .

- Si vous souhaitez utiliser n'importe quel nom de domaine, y compris ceux qui ne sont pas des sous-domaines du domaine de cluster OpenShift Container Platform par défaut, vous devez configurer le mappage de domaine pour ces domaines. Pour plus d'informations, voir la documentation OpenShift Serverless sur la création d'un mappage de domaine personnalisé .

9.1.2. Création d'un certificat pour crypter le trafic externe entrant

Par défaut, la fonctionnalité Service Mesh mTLS ne sécurise que le trafic à l'intérieur du Service Mesh lui-même, entre la passerelle d'entrée et les pods individuels qui ont des sidecars. Pour chiffrer le trafic lorsqu'il circule dans le cluster OpenShift Container Platform, vous devez générer un certificat avant d'activer l'intégration OpenShift Serverless et Service Mesh.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.

- Vous avez installé OpenShift Serverless Operator et Knative Serving.

- Installez le CLI OpenShift (**oc**).

- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Créez un certificat racine et une clé privée qui signent les certificats de vos services Knative :

```
$ openssl req -x509 -sha256 -nodes -days 365 -newkey rsa:2048 \
    -subj '/O=Example Inc./CN=example.com' \
    -keyout root.key \
    -out root.crt
```

2. Créer un certificat de type "wildcard" :

```
$ openssl req -nodes -newkey rsa:2048 \
    -subj "/CN=*.apps.openshift.example.com/O=Example Inc." \
    -keyout wildcard.key \
    -out wildcard.csr
```

3. Signer le certificat Wildcard :

```
$ openssl x509 -req -days 365 -set_serial 0 \
    -CA root.crt \
    -CAkey root.key \
    -in wildcard.csr \
    -out wildcard.crt
```

4. Créez un secret en utilisant le certificat générique :

```
$ oc create -n istio-system secret tls wildcard-certs \
    --key=wildcard.key \
    --cert=wildcard.crt
```

   Ce certificat est récupéré par les passerelles créées lors de l'intégration d'OpenShift Serverless avec Service Mesh, afin que la passerelle d'entrée serve le trafic avec ce certificat.

9.1.3. Intégration de Service Mesh avec OpenShift Serverless

Vous pouvez intégrer Service Mesh avec OpenShift Serverless sans utiliser Kourier comme ingress par défaut. Pour ce faire, n'installez pas le composant Knative Serving avant d'avoir effectué la procédure suivante. Il y a des étapes supplémentaires requises lors de la création de la définition de ressource personnalisée (CRD) **KnativeServing** pour intégrer Knative Serving avec Service Mesh, qui ne sont pas couvertes dans la procédure générale d'installation de Knative Serving. Cette procédure peut être utile si vous souhaitez intégrer Service Mesh comme ingress par défaut et unique pour votre installation OpenShift Serverless.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

- Installez Red Hat OpenShift Service Mesh Operator et créez une ressource **ServiceMeshControlPlane** dans l'espace de noms **istio-system**. Si vous souhaitez utiliser la fonctionnalité mTLS, vous devez également définir le champ **spec.security.dataPlane.mtls** de la ressource **ServiceMeshControlPlane** sur **true**.

IMPORTANT

L'utilisation d'OpenShift Serverless avec Service Mesh n'est prise en charge qu'avec Red Hat OpenShift Service Mesh version 2.0.5 ou ultérieure.

- Installer l'opérateur OpenShift Serverless.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Ajoutez à l'objet **ServiceMeshMemberRoll**, en tant que membres, les espaces de noms que vous souhaitez intégrer à Service Mesh :

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members: 1
    - knative-serving
    - <namespace>
```

1 Une liste d'espaces de noms à intégrer dans Service Mesh.

IMPORTANT

Cette liste d'espaces de noms doit inclure l'espace de noms **knative-serving**.

2. Appliquer la ressource **ServiceMeshMemberRoll**:

```
$ oc apply -f <filename>
```

3. Créez les passerelles nécessaires pour que le Service Mesh puisse accepter le trafic :

Exemple knative-local-gateway objet en utilisant HTTP

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: knative-ingress-gateway
  namespace: knative-serving
spec:
  selector:
    istio: ingressgateway
  servers:
```

```
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - "*"
      tls:
        mode: SIMPLE
        credentialName: <wildcard_certs> 1
---
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
 name: knative-local-gateway
 namespace: knative-serving
spec:
 selector:
   istio: ingressgateway
 servers:
   - port:
       number: 8081
       name: http
       protocol: HTTP 2
     hosts:
       - "*"
---
apiVersion: v1
kind: Service
metadata:
 name: knative-local-gateway
 namespace: istio-system
 labels:
   experimental.istio.io/disable-gateway-port-translation: "true"
spec:
 type: ClusterIP
 selector:
   istio: ingressgateway
 ports:
   - name: http2
     port: 80
     targetPort: 8081
```

1 Ajoutez le nom du secret qui contient le certificat générique.

2 Le site **knative-local-gateway** sert le trafic HTTP. L'utilisation de HTTP signifie que le trafic provenant de l'extérieur de Service Mesh, mais utilisant un nom d'hôte interne, tel que **example.default.svc.cluster.local**, n'est pas crypté. Vous pouvez configurer le cryptage pour ce chemin en créant un autre certificat générique et une passerelle supplémentaire qui utilise une spécification **protocol** différente.

Exemple knative-local-gateway objet utilisant HTTPS

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
```

```
  metadata:
    name: knative-local-gateway
    namespace: knative-serving
  spec:
    selector:
      istio: ingressgateway
    servers:
      - port:
          number: 443
          name: https
          protocol: HTTPS
        hosts:
          - "*"
        tls:
          mode: SIMPLE
          credentialName: <wildcard_certs>
```

4. Appliquer les ressources du site **Gateway**:

```
$ oc apply -f <filename>
```

5. Installez Knative Serving en créant la définition de ressource personnalisée (CRD) suivante **KnativeServing**, qui permet également l'intégration d'Istio :

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
spec:
  ingress:
    istio:
      enabled: true (1)
  deployments: (2)
  - name: activator
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
  - name: autoscaler
    annotations:
      "sidecar.istio.io/inject": "true"
      "sidecar.istio.io/rewriteAppHTTPProbers": "true"
```

(1) Active l'intégration d'Istio.

(2) Active l'injection de sidecar pour les pods du plan de données Knative Serving.

6. Appliquer la ressource **KnativeServing**:

```
$ oc apply -f <filename>
```

7. Créez un service Knative pour lequel l'injection de sidecar est activée et qui utilise une route pass-through :

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: <service_name>
  namespace: <namespace> 1
  annotations:
    serving.knative.openshift.io/enablePassthrough: "true" 2
spec:
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: "true" 3
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
    spec:
      containers:
      - image: <image_url>
```

1. Un espace de noms qui fait partie du rouleau de membres du Service Mesh.
2. Indique à Knative Serving de générer une route OpenShift Container Platform pass-through enabled, de sorte que les certificats que vous avez générés sont servis par la passerelle d'entrée directement.
3. Injecte des sidecars Service Mesh dans les pods de service Knative.

8. Appliquer la ressource **Service**:

```
$ oc apply -f <filename>
```

Vérification

- Accédez à votre application sans serveur en utilisant une connexion sécurisée qui est maintenant approuvée par l'autorité de certification :

```
curl --cacert root.crt <service_url>
```

Example command

```
$ curl --cacert root.crt https://hello-default.apps.openshift.example.com
```

Exemple de sortie

```
Hello Openshift!
```

9.1.4. Activation des métriques Knative Serving lors de l'utilisation de Service Mesh avec mTLS

Si Service Mesh est activé avec mTLS, les métriques pour Knative Serving sont désactivées par défaut, car Service Mesh empêche Prometheus de récupérer les métriques. Cette section explique comment activer les métriques Knative Serving lors de l'utilisation de Service Mesh et de mTLS.

Conditions préalables

- Vous avez installé OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Vous avez installé Red Hat OpenShift Service Mesh avec la fonctionnalité mTLS activée.
- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Installez le CLI OpenShift (**oc**).
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

Procédure

1. Spécifier **prometheus** comme **metrics.backend-destination** dans la spécification **observability** de la ressource personnalisée (CR) Knative Serving :

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
spec:
  config:
    observability:
      metrics.backend-destination: "prometheus"
...
```

   Cette étape permet d'éviter que les métriques soient désactivées par défaut.

2. Appliquez la stratégie réseau suivante pour autoriser le trafic en provenance de l'espace de noms Prometheus :

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring-ns
  namespace: knative-serving
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: "openshift-monitoring"
  podSelector: {}
...
```

3. Modifier et réappliquer le plan de contrôle Service Mesh par défaut dans l'espace de noms **istio-system**, de manière à ce qu'il inclue la spécification suivante :

```
...
spec:
  proxy:
    networking:
      trafficControl:
```

```
      inbound:
        excludedPorts:
        - 8444
...
```

9.1.5. Intégration de Service Mesh avec OpenShift Serverless lorsque Kourier est activé

Vous pouvez utiliser Service Mesh avec OpenShift Serverless même si Kourier est déjà activé. Cette procédure peut être utile si vous avez déjà installé Knative Serving avec Kourier activé, mais que vous décidez d'ajouter une intégration Service Mesh plus tard.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Installez le CLI OpenShift (**oc**).
- Installez OpenShift Serverless Operator et Knative Serving sur votre cluster.
- Installer Red Hat OpenShift Service Mesh. OpenShift Serverless avec Service Mesh et Kourier est pris en charge pour une utilisation avec les versions 1.x et 2.x de Red Hat OpenShift Service Mesh.

Procédure

1. Ajoutez à l'objet **ServiceMeshMemberRoll**, en tant que membres, les espaces de noms que vous souhaitez intégrer à Service Mesh :

```
apiVersion: maistra.io/v1
kind: ServiceMeshMemberRoll
metadata:
  name: default
  namespace: istio-system
spec:
  members:
    - <namespace> 1
...
```

1 Une liste d'espaces de noms à intégrer dans Service Mesh.

2. Appliquer la ressource **ServiceMeshMemberRoll**:

```
$ oc apply -f <filename>
```

3. Créer une stratégie de réseau qui autorise le flux de trafic des pods du système Knative vers les services Knative :

   a. Pour chaque espace de noms que vous souhaitez intégrer à Service Mesh, créez une ressource **NetworkPolicy**:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-serving-system-namespace
  namespace: <namespace> 1
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          knative.openshift.io/part-of: "openshift-serverless"
  podSelector: {}
  policyTypes:
  - Ingress
...
```

1 Ajoutez l'espace de noms que vous souhaitez intégrer à Service Mesh.

NOTE

Le label **knative.openshift.io/part-of: "openshift-serverless"** a été ajouté dans OpenShift Serverless 1.22.0. Si vous utilisez OpenShift Serverless 1.21.1 ou une version antérieure, ajoutez le label **knative.openshift.io/part-of** aux espaces de noms **knative-serving** et **knative-serving-ingress**.

Ajouter l'étiquette à l'espace de noms **knative-serving**:

```
$ oc label namespace knative-serving knative.openshift.io/part-of=openshift-serverless
```

Ajouter l'étiquette à l'espace de noms **knative-serving-ingress**:

```
$ oc label namespace knative-serving-ingress knative.openshift.io/part-of=openshift-serverless
```

b. Appliquer la ressource **NetworkPolicy**:

```
$ oc apply -f <filename>
```

9.1.6. Améliorer l'utilisation de la mémoire de net-istio en utilisant le filtrage secret pour Service Mesh

Par défaut, l'implémentation des informateurs pour la bibliothèque Kubernetes **client-go** récupère toutes les ressources d'un type particulier. Cela peut entraîner une surcharge substantielle lorsque de nombreuses ressources sont disponibles, ce qui peut faire échouer le contrôleur d'entrée Knative **net-istio** sur les grands clusters en raison de fuites de mémoire. Cependant, un mécanisme de filtrage est disponible pour le contrôleur d'entrée Knative **net-istio**, ce qui permet au contrôleur de ne récupérer que les secrets liés à Knative. Vous pouvez activer ce mécanisme en ajoutant une annotation à la ressource personnalisée (CR) **KnativeServing**.

IMPORTANT

Si vous activez le filtrage des secrets, tous vos secrets doivent être étiquetés avec **networking.internal.knative.dev/certificate-uid: "<id>"**. Sinon, Knative Serving ne les détecte pas, ce qui entraîne des échecs. Vous devez étiqueter à la fois les nouveaux secrets et les secrets existants.

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.
- Installez Red Hat OpenShift Service Mesh. OpenShift Serverless with Service Mesh only est pris en charge pour une utilisation avec Red Hat OpenShift Service Mesh version 2.0.5 ou ultérieure.
- Installez OpenShift Serverless Operator et Knative Serving.
- Installez le CLI OpenShift (**oc**).

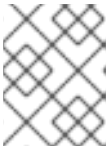
Procédure

- Ajouter l'annotation **serverless.openshift.io/enable-secret-informer-filtering** à la CR **KnativeServing**:

  Exemple KnativeServing CR

```
apiVersion: operator.knative.dev/v1beta1
kind: KnativeServing
metadata:
  name: knative-serving
  namespace: knative-serving
  annotations:
    serverless.openshift.io/enable-secret-informer-filtering: "true" (1)
spec:
  ingress:
    istio:
      enabled: true
  deployments:
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: activator
    - annotations:
        sidecar.istio.io/inject: "true"
        sidecar.istio.io/rewriteAppHTTPProbers: "true"
      name: autoscaler
```

(1) L'ajout de cette annotation injecte une variable d'environnement, **ENABLE_SECRET_INFORMER_FILTERING_BY_CERT_UID=true**, dans le pod contrôleur **net-istio**.



NOTE

Cette annotation est ignorée si vous définissez une valeur différente en remplaçant les déploiements.

9.2. INTÉGRER SERVERLESS AU SERVICE DE GESTION DES COÛTS

La gestion des coûts est un service d'OpenShift Container Platform qui vous permet de mieux comprendre et suivre les coûts pour les clouds et les conteneurs. Il est basé sur le projet open source Koku.

9.2.1. Conditions préalables

- Vous avez des droits d'administrateur de cluster.
- Vous avez configuré la gestion des coûts et ajouté une source OpenShift Container Platform.

9.2.2. Utilisation d'étiquettes pour la gestion des coûts

Les étiquettes, également connues sous le nom de *tags* dans la gestion des coûts, peuvent être appliquées aux nœuds, aux espaces de noms ou aux pods. Chaque étiquette est une paire clé/valeur. Vous pouvez utiliser une combinaison de plusieurs étiquettes pour générer des rapports. Vous pouvez accéder aux rapports sur les coûts en utilisant la console hybride de Red Hat .

Les étiquettes sont héritées des nœuds vers les espaces de noms, et des espaces de noms vers les pods. Cependant, les étiquettes ne sont pas remplacées si elles existent déjà sur une ressource. Par exemple, les services Knative ont un label par défaut **app=<revision_name>**:

Exemple d'étiquette par défaut du service Knative

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: example-service
spec:
...
      labels:
        app: <revision_name>
...
```

Si vous définissez un label pour un espace de noms, tel que **app=my-domain**, le service de gestion des coûts ne prend pas en compte les coûts provenant d'un service Knative avec le tag **app=<revision_name>** lors de l'interrogation de l'application avec le tag **app=my-domain**. Les coûts des services Knative qui ont ce tag doivent être interrogés sous le tag **app=<revision_name>**.

9.2.3. Ressources supplémentaires

- Configurer le balisage de vos sources
- Utilisez l'explorateur de coûts pour visualiser vos coûts

9.3. UTILISER LES RESSOURCES GPU NVIDIA AVEC DES APPLICATIONS SANS SERVEUR

NVIDIA prend en charge l'utilisation des ressources GPU sur OpenShift Container Platform. Voir GPU Operator on OpenShift pour plus d'informations sur la configuration des ressources GPU sur OpenShift Container Platform.

9.3.1. Spécifier les exigences du GPU pour un service

Une fois les ressources GPU activées pour votre cluster OpenShift Container Platform, vous pouvez spécifier les exigences GPU pour un service Knative à l'aide de la CLI Knative (**kn**).

Conditions préalables

- L'opérateur OpenShift Serverless, Knative Serving et Knative Eventing sont installés sur le cluster.
- Vous avez installé le CLI Knative (**kn**).
- Les ressources GPU sont activées pour votre cluster OpenShift Container Platform.
- Vous avez créé un projet ou avez accès à un projet avec les rôles et autorisations appropriés pour créer des applications et d'autres charges de travail dans OpenShift Container Platform.

NOTE

L'utilisation des ressources GPU NVIDIA n'est pas prise en charge par IBM zSystems et IBM Power.

Procédure

1. Créez un service Knative et définissez la limite des ressources GPU requises à **1** en utilisant le drapeau **--limit nvidia.com/gpu=1**:

```
$ kn service create hello --image <service-image> --limit nvidia.com/gpu=1
```

   Une limite de ressources GPU de **1** signifie que le service dispose d'une ressource GPU dédiée. Les services ne partagent pas les ressources GPU. Tout autre service nécessitant des ressources GPU doit attendre que la ressource GPU ne soit plus utilisée.

   Une limite de 1 GPU signifie également que les applications qui dépassent l'utilisation d'une ressource GPU sont restreintes. Si un service demande plus d'une ressource GPU, il est déployé sur un nœud où les besoins en ressources GPU peuvent être satisfaits.

2. Facultatif. Pour un service existant, vous pouvez modifier la limite des besoins en ressources GPU à **3** en utilisant l'indicateur **--limit nvidia.com/gpu=3**:

```
$ kn service update hello --limit nvidia.com/gpu=3
```

9.3.2. Ressources supplémentaires

- Définition de quotas de ressources pour les ressources étendues

CHAPITRE 10. SUPPRESSION DE SERVERLESS

10.1. SUPPRIMER L'APERÇU D'OPENSHIFT SERVERLESS

Si vous devez supprimer OpenShift Serverless de votre cluster, vous pouvez le faire en supprimant manuellement l'OpenShift Serverless Operator et d'autres composants OpenShift Serverless. Avant de pouvoir supprimer OpenShift Serverless Operator, vous devez supprimer Knative Serving et Knative Eventing.

Après avoir désinstallé OpenShift Serverless, vous pouvez supprimer les définitions de ressources personnalisées (CRD) de l'opérateur et de l'API qui restent sur le cluster.

Les étapes pour supprimer complètement OpenShift Serverless sont détaillées dans les procédures suivantes :

- Désinstallation de Knative Eventing .
- Désinstallation de Knative Serving .
- Suppression de l'opérateur OpenShift Serverless .
- Suppression des définitions de ressources personnalisées OpenShift Serverless .

10.2. DÉSINSTALLATION D'OPENSHIFT SERVERLESS KNATIVE EVENTING

Avant de pouvoir supprimer l'OpenShift Serverless Operator, vous devez supprimer Knative Eventing. Pour désinstaller Knative Eventing, vous devez supprimer la ressource personnalisée (CR) **KnativeEventing** et supprimer l'espace de noms **knative-eventing**.

10.2.1. Désinstallation de Knative Eventing

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Supprimer le CR **KnativeEventing**:

```
$ oc delete knativeeventings.operator.knative.dev knative-eventing -n knative-eventing
```

2. Une fois que la commande est terminée et que tous les pods ont été supprimés de l'espace de noms **knative-eventing**, supprimez l'espace de noms :

```
$ oc delete namespace knative-eventing
```

10.3. DÉSINSTALLATION D'OPENSHIFT SERVERLESS KNATIVE SERVING

Avant de pouvoir supprimer l'OpenShift Serverless Operator, vous devez supprimer Knative Serving. Pour désinstaller Knative Serving, vous devez supprimer la ressource personnalisée (CR) **KnativeServing** et supprimer l'espace de noms **knative-serving**.

10.3.1. Désinstallation de Knative Serving

Conditions préalables

- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Installez le CLI OpenShift (**oc**).

Procédure

1. Supprimer le CR **KnativeServing**:

```
$ oc delete knativeservings.operator.knative.dev knative-serving -n knative-serving
```

2. Une fois que la commande est terminée et que tous les pods ont été supprimés de l'espace de noms **knative-serving**, supprimez l'espace de noms :

```
$ oc delete namespace knative-serving
```

10.4. SUPPRESSION DE L'OPÉRATEUR OPENSHIFT SERVERLESS

Après avoir supprimé Knative Serving et Knative Eventing, vous pouvez supprimer l'OpenShift Serverless Operator. Vous pouvez le faire en utilisant la console web d'OpenShift Container Platform ou le CLI **oc**.

10.4.1. Suppression d'opérateurs d'une grappe à l'aide de la console web

Les administrateurs de cluster peuvent supprimer les opérateurs installés dans un espace de noms sélectionné à l'aide de la console web.

Conditions préalables

- Vous avez accès à la console web d'un cluster OpenShift Container Platform en utilisant un compte avec les permissions **cluster-admin**.

Procédure

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Faites défiler ou saisissez un mot-clé dans le champ **Filter by name** pour trouver l'opérateur que vous souhaitez supprimer. Cliquez ensuite dessus.
3. Sur le côté droit de la page **Operator Details**, sélectionnez **Uninstall Operator** dans la liste **Actions**.
   Une boîte de dialogue **Uninstall Operator?** s'affiche.

4. Sélectionnez **Uninstall** pour supprimer l'opérateur, les déploiements de l'opérateur et les pods. Suite à cette action, l'opérateur cesse de fonctionner et ne reçoit plus de mises à jour.

NOTE

Cette action ne supprime pas les ressources gérées par l'opérateur, y compris les définitions de ressources personnalisées (CRD) et les ressources personnalisées (CR). Les tableaux de bord et les éléments de navigation activés par la console Web et les ressources hors cluster qui continuent de fonctionner peuvent nécessiter un nettoyage manuel. Pour les supprimer après la désinstallation de l'opérateur, vous devrez peut-être supprimer manuellement les CRD de l'opérateur.

10.4.2. Suppression d'opérateurs d'une grappe à l'aide de la CLI

Les administrateurs de clusters peuvent supprimer les opérateurs installés dans un espace de noms sélectionné à l'aide de l'interface de ligne de commande.

Conditions préalables

- Accès à un cluster OpenShift Container Platform à l'aide d'un compte disposant des autorisations **cluster-admin**.
- **oc** installée sur le poste de travail.

Procédure

1. Vérifiez la version actuelle de l'opérateur souscrit (par exemple, **jaeger**) dans le champ **currentCSV**:

```
$ oc get subscription jaeger -n openshift-operators -o yaml | grep currentCSV
```

Exemple de sortie

```
  currentCSV: jaeger-operator.v1.8.2
```

2. Supprimer l'abonnement (par exemple, **jaeger**) :

```
$ oc delete subscription jaeger -n openshift-operators
```

Exemple de sortie

```
subscription.operators.coreos.com "jaeger" deleted
```

3. Supprimez le CSV de l'opérateur dans l'espace de noms cible en utilisant la valeur **currentCSV** de l'étape précédente :

```
$ oc delete clusterserviceversion jaeger-operator.v1.8.2 -n openshift-operators
```

Exemple de sortie

```
clusterserviceversion.operators.coreos.com "jaeger-operator.v1.8.2" deleted
```

10.4.3. Actualisation des abonnements défaillants

Dans Operator Lifecycle Manager (OLM), si vous vous abonnez à un opérateur qui fait référence à des images qui ne sont pas accessibles sur votre réseau, vous pouvez trouver des travaux dans l'espace de noms **openshift-marketplace** qui échouent avec les erreurs suivantes :

Exemple de sortie

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-
bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

Exemple de sortie

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get
"https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

En conséquence, l'abonnement est bloqué dans cet état d'échec et l'opérateur est incapable d'installer ou de mettre à niveau.

Vous pouvez actualiser un abonnement défaillant en supprimant l'abonnement, la version du service de cluster (CSV) et d'autres objets connexes. Après avoir recréé l'abonnement, OLM réinstalle la version correcte de l'opérateur.

Conditions préalables

- Vous avez un abonnement défaillant qui ne parvient pas à extraire une image de paquet inaccessible.
- Vous avez confirmé que l'image correcte de la liasse est accessible.

Procédure

1. Obtenir les noms des objets **Subscription** et **ClusterServiceVersion** de l'espace de noms dans lequel l'opérateur est installé :

   ```
   $ oc get sub,csv -n <namespace>
   ```

   #### Exemple de sortie

   ```
   NAME                                         PACKAGE                  SOURCE             CHANNEL
   subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-
   operators   5.0

   NAME                                                  DISPLAY                  VERSION
   REPLACES   PHASE
   clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift
   Elasticsearch Operator   5.0.0-65              Succeeded
   ```

2. Supprimer l'abonnement :

   ```
   oc delete subscription <subscription_name> -n <namespace> $ oc delete subscription
   <subscription_name> -n <namespace>
   ```

3. Supprimer la version du service de cluster :

```
$ oc delete csv <csv_name> -n <namespace>
```

4. Récupère les noms de tous les travaux défaillants et des cartes de configuration correspondantes dans l'espace de noms **openshift-marketplace**:

```
$ oc get job,configmap -n openshift-marketplace
```

Exemple de sortie

```
NAME                                                                COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1
26s        9m30s

NAME                                                                DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3
9m30s
```

5. Supprimer le travail :

```
oc delete job <job_name> -n openshift-marketplace
```

Cela permet de s'assurer que les pods qui tentent d'extraire l'image inaccessible ne sont pas recréés.

6. Supprimer la carte de configuration :

```
oc delete configmap <configmap_name> -n openshift-marketplace
```

7. Réinstallez l'opérateur en utilisant OperatorHub dans la console web.

Vérification

- Vérifiez que l'opérateur a été réinstallé avec succès :

```
$ oc get sub,csv,installplan -n <namespace>
```

10.5. SUPPRESSION DES DÉFINITIONS DE RESSOURCES PERSONNALISÉES OPENSHIFT SERVERLESS

Après la désinstallation d'OpenShift Serverless, les définitions de ressources personnalisées (CRD) Operator et API restent sur le cluster. Vous pouvez utiliser la procédure suivante pour supprimer les CRD restantes.

IMPORTANT

La suppression des CRD Operator et API supprime également toutes les ressources qui ont été définies en les utilisant, y compris les services Knative.

10.5.1. Suppression des CRDs OpenShift Serverless Operator et API

Supprimez les CRD de l'opérateur et de l'API à l'aide de la procédure suivante.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- Vous avez accès à un compte OpenShift Container Platform avec un accès administrateur de cluster.
- Vous avez désinstallé Knative Serving et supprimé l'opérateur OpenShift Serverless.

Procédure

- Pour supprimer les CRD OpenShift Serverless restants, entrez la commande suivante :

```
$ oc get crd -oname | grep 'knative.dev' | xargs oc delete
```

CHAPITRE 11. PRISE EN CHARGE D'OPENSHIFT SERVERLESS

Si vous rencontrez des difficultés avec une procédure décrite dans cette documentation, visitez le portail client de Red Hat à l'adresse http://access.redhat.com. Vous pouvez utiliser le portail client de Red Hat pour rechercher ou parcourir la base de connaissances de Red Hat qui contient des articles d'assistance technique sur les produits Red Hat. Vous pouvez également soumettre un cas d'assistance à Red Hat Global Support Services (GSS), ou accéder à d'autres documentations de produits.

Si vous avez une suggestion pour améliorer ce guide ou si vous avez trouvé une erreur, vous pouvez soumettre un problème Jira pour le composant de documentation le plus pertinent. Fournissez des détails spécifiques, tels que le numéro de section, le nom du guide et la version d'OpenShift Serverless afin que nous puissions facilement localiser le contenu.

11.1. À PROPOS DE LA BASE DE CONNAISSANCES DE RED HAT

La base de connaissances de Red Hat fournit un contenu riche destiné à vous aider à tirer le meilleur parti des produits et des technologies de Red Hat. La base de connaissances de Red Hat comprend des articles, de la documentation sur les produits et des vidéos décrivant les meilleures pratiques en matière d'installation, de configuration et d'utilisation des produits Red Hat. En outre, vous pouvez rechercher des solutions à des problèmes connus, chacune d'entre elles fournissant des descriptions concises de la cause première et des mesures correctives.

11.2. RECHERCHE DANS LA BASE DE CONNAISSANCES DE RED HAT

En cas de problème lié à OpenShift Container Platform, vous pouvez effectuer une recherche initiale pour déterminer si une solution existe déjà dans la base de connaissances de Red Hat.

Conditions préalables

- Vous disposez d'un compte Red Hat Customer Portal.

Procédure

1. Connectez-vous au portail client de Red Hat .
2. Dans le champ de recherche principal du portail client de Red Hat, saisissez des mots-clés et des chaînes de caractères relatifs au problème, y compris :
   - Composants de la plateforme OpenShift Container (tels que **etcd**)
   - Procédure connexe (telle que **installation**)
   - Avertissements, messages d'erreur et autres résultats liés à des défaillances explicites
3. Cliquez sur **Search**.
4. Sélectionnez le filtre de produit **OpenShift Container Platform**.
5. Sélectionnez le filtre de type de contenu **Knowledgebase**.

11.3. SOUMETTRE UN DOSSIER D'ASSISTANCE

Conditions préalables

- Vous avez accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- Vous avez installé l'OpenShift CLI (**oc**).
- Vous disposez d'un compte Red Hat Customer Portal.
- Vous avez un abonnement standard ou premium Red Hat.

Procédure

1. Connectez-vous au portail client de Red Hat et sélectionnez **SUPPORT CASES → Open a case**.
2. Sélectionnez la catégorie appropriée pour votre problème (telle que **Defect / Bug**), le produit (**OpenShift Container Platform**) et la version du produit (**4.12**, si elle n'est pas déjà remplie automatiquement).
3. Examinez la liste des solutions suggérées dans la base de connaissances de Red Hat afin de trouver une correspondance potentielle avec le problème signalé. Si les articles suggérés ne répondent pas au problème, cliquez sur **Continue**.
4. Saisissez un résumé concis mais descriptif du problème et des détails supplémentaires sur les symptômes ressentis, ainsi que vos attentes.
5. Examinez la liste mise à jour des solutions suggérées dans la base de connaissances de Red Hat afin de trouver une correspondance potentielle avec le problème signalé. La liste est affinée au fur et à mesure que vous fournissez des informations supplémentaires au cours du processus de création du cas. Si les articles suggérés ne répondent pas au problème, cliquez sur **Continue**.
6. S'assurer que les informations sur le compte présentées sont conformes aux attentes et, si ce n'est pas le cas, les modifier en conséquence.
7. Vérifiez que l'identifiant de cluster OpenShift Container Platform rempli automatiquement est correct. Si ce n'est pas le cas, obtenez manuellement votre ID de cluster.
   - Pour obtenir manuellement votre ID de cluster à l'aide de la console web d'OpenShift Container Platform :
     a. Naviguez vers **Home → Dashboards → Overview**.
     b. Trouvez la valeur dans le champ **Cluster ID** de la section **Details**.
   - Il est également possible d'ouvrir un nouveau dossier de support via la console web d'OpenShift Container Platform et de faire en sorte que l'identifiant de votre cluster soit rempli automatiquement.
     a. Dans la barre d'outils, naviguez vers **(?) Help → Open Support Case**.
     b. La valeur **Cluster ID** est remplie automatiquement.
   - Pour obtenir l'ID de votre cluster à l'aide de l'OpenShift CLI (**oc**), exécutez la commande suivante :

```
$ oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
```

8. Répondez aux questions suivantes lorsque vous y êtes invité, puis cliquez sur **Continue**:
   - Où expérimentez-vous ce comportement ? Dans quel environnement ?

- Quand le comportement se produit-il ? A quelle fréquence ? De manière répétée ? A certains moments ?
- Quelles informations pouvez-vous fournir sur les délais et l'impact commercial ?

9. Téléchargez les fichiers de données de diagnostic pertinents et cliquez sur **Continue**. Il est recommandé d'inclure les données recueillies à l'aide de la commande **oc adm must-gather** comme point de départ, ainsi que toutes les données spécifiques au problème qui ne sont pas recueillies par cette commande.
10. Saisissez les informations relatives à la gestion du dossier et cliquez sur **Continue**.
11. Prévisualisez les détails du cas et cliquez sur **Submit**.

11.4. COLLECTE D'INFORMATIONS DIAGNOSTIQUES POUR LE SOUTIEN

Lorsque vous ouvrez un dossier d'assistance, il est utile de fournir des informations de débogage sur votre cluster à l'assistance Red Hat. L'outil **must-gather** vous permet de collecter des informations de diagnostic sur votre cluster OpenShift Container Platform, y compris des données relatives à OpenShift Serverless. Pour une assistance rapide, fournissez des informations de diagnostic pour OpenShift Container Platform et OpenShift Serverless.

11.4.1. À propos de l'outil de collecte obligatoire

La commande CLI **oc adm must-gather** recueille les informations de votre cluster les plus susceptibles d'être nécessaires au débogage des problèmes, notamment

- Définitions des ressources
- Journaux de service

Par défaut, la commande **oc adm must-gather** utilise l'image du plugin par défaut et écrit dans **./must-gather.local**.

Vous pouvez également recueillir des informations spécifiques en exécutant la commande avec les arguments appropriés, comme décrit dans les sections suivantes :

- Pour collecter des données relatives à une ou plusieurs caractéristiques spécifiques, utilisez l'argument **--image** avec une image, comme indiqué dans la section suivante.
  Par exemple :

```
$ oc adm must-gather  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.12.0
```

- Pour collecter les journaux d'audit, utilisez l'argument **-- /usr/bin/gather_audit_logs**, comme décrit dans la section suivante.
  Par exemple :

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

NOTE

Les journaux d'audit ne sont pas collectés dans le cadre de l'ensemble d'informations par défaut afin de réduire la taille des fichiers.

Lorsque vous exécutez **oc adm must-gather**, un nouveau module portant un nom aléatoire est créé dans un nouveau projet sur le cluster. Les données sont collectées sur ce module et enregistrées dans un nouveau répertoire commençant par **must-gather.local**. Ce répertoire est créé dans le répertoire de travail actuel.

Par exemple :

```
NAMESPACE                  NAME              READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj    must-gather-bklx4    2/2     Running     0          72s
openshift-must-gather-5drcj    must-gather-s8sdh    2/2     Running     0          72s
...
```

11.4.2. À propos de la collecte de données OpenShift Serverless

Vous pouvez utiliser la commande CLI **oc adm must-gather** pour collecter des informations sur votre cluster, y compris les fonctionnalités et les objets associés à OpenShift Serverless. Pour collecter des données OpenShift Serverless avec **must-gather**, vous devez spécifier l'image OpenShift Serverless et le tag d'image pour votre version installée d'OpenShift Serverless.

Conditions préalables

- Installez le CLI OpenShift (**oc**).

Procédure

- Recueillez des données en utilisant la commande **oc adm must-gather**:

```
$ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:<image_version_tag>
```

Example command

```
$ oc adm must-gather --image=registry.redhat.io/openshift-serverless-1/svls-must-gather-rhel8:1.14.0
```