Plate-forme de conteneurs OpenShift 4.12

Images

Créer et gérer des images et des flux d'images dans OpenShift Container Platform

Last Updated: 2023-04-26

Plate-forme de conteneurs OpenShift 4.12 Images

Créer et gérer des images et des flux d'images dans OpenShift Container Platform

Notice légale

Résumé

Ce document fournit des instructions pour créer et gérer des images et des flux d'images dans OpenShift Container Platform. Il fournit également des instructions sur l'utilisation des modèles.

Table des matières

CHAPITRE 1. APERÇU DES IMAGES

1.1. COMPRENDRE LES CONTENEURS, LES IMAGES ET LES FLUX D'IMAGES

Les conteneurs, les images et les flux d'images sont des concepts importants à comprendre lorsque vous souhaitez créer et gérer des logiciels conteneurisés. Une image contient un ensemble de logiciels prêts à être exécutés, tandis qu'un conteneur est une instance en cours d'exécution d'une image de conteneur. Un flux d'images permet de stocker différentes versions d'une même image de base. Ces différentes versions sont représentées par des balises différentes sur le même nom d'image.

1.2. IMAGES

Les conteneurs dans OpenShift Container Platform sont basés sur des conteneurs formatés OCI ou Docker *images*. Une image est un binaire qui comprend toutes les exigences nécessaires à l'exécution d'un conteneur unique, ainsi que des métadonnées décrivant ses besoins et ses capacités.

On peut considérer qu'il s'agit d'une technologie de conditionnement. Les conteneurs n'ont accès qu'aux ressources définies dans l'image, sauf si vous leur donnez un accès supplémentaire lors de leur création. En déployant la même image dans plusieurs conteneurs sur plusieurs hôtes et en équilibrant la charge entre eux, OpenShift Container Platform peut fournir une redondance et une mise à l'échelle horizontale pour un service packagé dans une image.

Vous pouvez utiliser le CLI podman ou **docker** directement pour construire des images, mais OpenShift Container Platform fournit également des images de construction qui aident à créer de nouvelles images en ajoutant votre code ou votre configuration aux images existantes.

Comme les applications se développent au fil du temps, un seul nom d'image peut en fait renvoyer à plusieurs versions différentes de la même image. Chaque image différente est désignée de manière unique par son hachage, un long nombre hexadécimal tel que **fd44297e2ddb050ec4f…**, qui est généralement raccourci à 12 caractères, tel que **fd44297e2ddb**.

Vous pouvez créer, gérer et utiliser des images de conteneurs.

1.3. REGISTRE DES IMAGES

Un registre d'images est un serveur de contenu qui peut stocker et servir des images de conteneurs. Par exemple, un registre d'images est un serveur de contenu qui peut stocker et servir des images de conteneurs :

```
registry.redhat.io
```

Un registre contient une collection d'un ou plusieurs dépôts d'images, qui contiennent une ou plusieurs images marquées. Red Hat fournit un registre à **registry.redhat.io** pour les abonnés. OpenShift Container Platform peut également fournir son propre registre d'images OpenShift pour gérer des images de conteneurs personnalisées.

1.4. DÉPÔT D'IMAGES

Un référentiel d'images est une collection d'images de conteneurs apparentées et de balises les identifiant. Par exemple, les images Jenkins de OpenShift Container Platform se trouvent dans le référentiel :

-

```
docker.io/openshift/jenkins-2-centos7
```

1.5. BALISES D'IMAGE

Une balise d'image est une étiquette appliquée à une image conteneur dans un référentiel qui distingue une image spécifique des autres images dans un flux d'images. Généralement, la balise représente une sorte de numéro de version. Par exemple, ici **:v3.11.59-2** est la balise :

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

Vous pouvez ajouter des balises supplémentaires à une image. Par exemple, une image peut se voir attribuer les balises **:v3.11.59-2** et **:latest**.

OpenShift Container Platform propose la commande **oc tag**, qui est similaire à la commande **docker tag**, mais qui opère sur les flux d'images au lieu d'agir directement sur les images.

1.6. ID DES IMAGES

Un ID d'image est un code SHA (Secure Hash Algorithm) qui peut être utilisé pour extraire une image. Un ID d'image SHA ne peut pas changer. Un identifiant SHA spécifique fait toujours référence au même contenu d'image de conteneur. Par exemple, un identifiant SHA spécifique fait toujours référence au même contenu d'image de conteneur :

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

1.7. CONTENEURS

Les unités de base des applications de la plateforme OpenShift Container sont appelées conteneurs. Les technologies de conteneurs Linux sont des mécanismes légers permettant d'isoler les processus en cours d'exécution afin qu'ils n'interagissent qu'avec les ressources qui leur sont attribuées. Le mot conteneur est défini comme une instance spécifique en cours d'exécution ou en pause d'une image de conteneur.

De nombreuses instances d'application peuvent être exécutées dans des conteneurs sur un seul hôte sans visibilité sur les processus, les fichiers, le réseau, etc. des autres. En général, chaque conteneur fournit un seul service, souvent appelé micro-service, tel qu'un serveur web ou une base de données, bien que les conteneurs puissent être utilisés pour des charges de travail arbitraires.

Le noyau Linux intègre depuis des années des fonctionnalités pour les technologies de conteneurs. Le projet Docker a développé une interface de gestion pratique pour les conteneurs Linux sur un hôte. Plus récemment, l'Open Container Initiative a développé des normes ouvertes pour les formats de conteneurs et les temps d'exécution des conteneurs. OpenShift Container Platform et Kubernetes ajoutent la possibilité d'orchestrer des conteneurs au format OCI et Docker sur des installations multi-hôtes.

Bien que vous n'interagissiez pas directement avec les runtimes de conteneurs lorsque vous utilisez OpenShift Container Platform, il est important de comprendre leurs capacités et leur terminologie pour comprendre leur rôle dans OpenShift Container Platform et comment vos applications fonctionnent à l'intérieur des conteneurs.

Des outils tels que podman peuvent être utilisés pour remplacer les outils de ligne de commande **docker** afin d'exécuter et de gérer directement les conteneurs. En utilisant **podman**, vous pouvez expérimenter les conteneurs séparément d'OpenShift Container Platform.

1.8. POURQUOI UTILISER LES IMAGESTREAMS ?

Un flux d'images et ses balises associées fournissent une abstraction pour référencer les images de conteneurs à partir d'OpenShift Container Platform. Le flux d'images et ses balises vous permettent de voir quelles images sont disponibles et de vous assurer que vous utilisez l'image spécifique dont vous avez besoin, même si l'image dans le référentiel change.

Les flux d'images ne contiennent pas de données d'images réelles, mais présentent une vue virtuelle unique d'images apparentées, à l'instar d'un référentiel d'images.

Vous pouvez configurer les constructions et les déploiements de manière à ce qu'ils surveillent un flux d'images pour recevoir des notifications lorsque de nouvelles images sont ajoutées et qu'ils réagissent en effectuant une construction ou un déploiement, respectivement.

Par exemple, si un déploiement utilise une certaine image et qu'une nouvelle version de cette image est créée, un déploiement pourrait être automatiquement effectué pour récupérer la nouvelle version de l'image.

Toutefois, si la balise de flux d'images utilisée par le déploiement ou la construction n'est pas mise à jour, même si l'image du conteneur dans le registre d'images du conteneur est mise à jour, la construction ou le déploiement continue d'utiliser l'image précédente, vraisemblablement connue et bonne.

Les images sources peuvent être stockées dans l'un des endroits suivants :

- Le registre intégré d'OpenShift Container Platform.
- Un registre externe, par exemple registry.redhat.io ou quay.io.
- Autres flux d'images dans le cluster OpenShift Container Platform.

Lorsque vous définissez un objet qui fait référence à une balise de flux d'images, comme une configuration de construction ou de déploiement, vous pointez vers une balise de flux d'images et non vers le référentiel. Lorsque vous construisez ou déployez votre application, OpenShift Container Platform interroge le référentiel à l'aide de la balise de flux d'images pour localiser l'ID associé à l'image et utilise cette image exacte.

Les métadonnées du flux d'images sont stockées dans l'instance etcd avec d'autres informations sur le cluster.

L'utilisation de flux d'images présente plusieurs avantages importants :

- Vous pouvez baliser, revenir en arrière et traiter rapidement les images, sans avoir à les repousser à l'aide de la ligne de commande.
- Vous pouvez déclencher des constructions et des déploiements lorsqu'une nouvelle image est poussée vers le registre. OpenShift Container Platform dispose également de déclencheurs génériques pour d'autres ressources, telles que les objets Kubernetes.
- Vous pouvez marquer une balise pour qu'elle soit réimportée périodiquement. Si l'image source a changé, ce changement est pris en compte et reflété dans le flux d'images, ce qui déclenche le flux de construction ou de déploiement, en fonction de la configuration de la construction ou du déploiement.
- Vous pouvez partager des images à l'aide d'un contrôle d'accès précis et les distribuer rapidement à vos équipes.

- Si l'image source change, la balise du flux d'images pointe toujours vers une version connue de l'image, ce qui garantit que votre application n'est pas interrompue de manière inattendue.
- Vous pouvez configurer la sécurité concernant la visualisation et l'utilisation des images au moyen d'autorisations sur les objets du flux d'images.
- Les utilisateurs qui n'ont pas l'autorisation de lire ou de lister des images au niveau du cluster peuvent toujours récupérer les images marquées dans un projet à l'aide des flux d'images.

Vous pouvez gérer les flux d'images, utiliser les flux d'images avec les ressources Kubernetes et déclencher des mises à jour sur les flux d'images .

1.9. BALISES DE FLUX D'IMAGES

Une balise de flux d'images est un pointeur nommé vers une image dans un flux d'images. Une balise de flux d'images est similaire à une balise d'image de conteneur.

1.10. FLUX D'IMAGES IMAGES

Une image de flux d'images vous permet de récupérer une image de conteneur spécifique à partir d'un flux d'images particulier où elle est étiquetée. Une image de flux d'images est un objet ressource de l'API qui rassemble des métadonnées sur un identifiant SHA d'image particulier.

1.11. DÉCLENCHEURS DE FLUX D'IMAGES

Un déclencheur de flux d'images provoque une action spécifique lorsqu'une balise de flux d'images change. Par exemple, l'importation peut entraîner une modification de la valeur de la balise, ce qui déclenche un déclencheur lorsque des déploiements, des constructions ou d'autres ressources sont à l'écoute.

1.12. COMMENT UTILISER L'OPÉRATEUR D'ÉCHANTILLONNAGE EN GRAPPES

Lors du démarrage initial, l'opérateur crée la ressource d'échantillons par défaut pour lancer la création des flux d'images et des modèles. Vous pouvez utiliser l'opérateur Cluster Samples pour gérer les flux d'images et les modèles d'échantillons stockés dans l'espace de noms **openshift**.

En tant qu'administrateur de cluster, vous pouvez utiliser le Cluster Samples Operator pour :

- Configurer l'opérateur.
- Utiliser l'opérateur avec un autre registre .

1.13. À PROPOS DES MODÈLES

Un modèle est une définition d'un objet à répliquer. Vous pouvez utiliser des modèles pour créer et déployer des configurations.

1.14. COMMENT UTILISER RUBY ON RAILS

En tant que développeur, vous pouvez utiliser Ruby on Rails pour :

- Rédigez votre demande :

    - Mettre en place une base de données.
    - Créez une page de bienvenue.
    - Configurez votre application pour OpenShift Container Platform.
    - Stockez votre application dans Git.
- Déployez votre application dans OpenShift Container Platform :
    - Créer le service de base de données.
    - Créer le service frontal.
    - Créez une route pour votre application.

CHAPITRE 2. CONFIGURATION DE L'OPÉRATEUR CLUSTER SAMPLES

Le Cluster Samples Operator, qui opère dans l'espace de noms **openshift**, installe et met à jour les flux d'images OpenShift Container Platform basés sur Red Hat Enterprise Linux (RHEL) et les modèles OpenShift Container Platform.

2.1. COMPRENDRE L'OPÉRATEUR CLUSTER SAMPLES

Lors de l'installation, l'opérateur crée l'objet de configuration par défaut pour lui-même, puis crée les flux d'images et les modèles d'échantillons, y compris les modèles de démarrage rapide.

NOTE

Pour faciliter les importations de flux d'images à partir d'autres registres nécessitant des informations d'identification, un administrateur de cluster peut créer des secrets supplémentaires contenant le contenu d'un fichier Docker **config.json** dans l'espace de noms **openshift** nécessaire à l'importation d'images.

La configuration du Cluster Samples Operator est une ressource à l'échelle du cluster, et le déploiement est contenu dans l'espace de noms **openshift-cluster-samples-operator**.

L'image de l'opérateur d'échantillons de clusters contient des définitions de flux d'images et de modèles pour la version associée d'OpenShift Container Platform. Lorsque chaque échantillon est créé ou mis à jour, le Cluster Samples Operator inclut une annotation qui indique la version d'OpenShift Container Platform. L'opérateur utilise cette annotation pour s'assurer que chaque échantillon correspond à la version. Les échantillons qui ne font pas partie de son inventaire sont ignorés, tout comme les échantillons ignorés. Les modifications apportées aux échantillons gérés par l'opérateur, lorsque l'annotation de la version est modifiée ou supprimée, sont automatiquement annulées.

NOTE

Les images Jenkins font partie de la charge utile de l'installation et sont directement intégrées dans les flux d'images.

La ressource de configuration Cluster Samples Operator comprend un finalisateur qui nettoie les éléments suivants lors de la suppression :

- Flux d'images gérés par l'opérateur.
- Modèles gérés par l'opérateur.
- Ressources de configuration générées par l'opérateur.
- Ressources sur l'état des clusters.

Lors de la suppression de la ressource d'échantillons, l'opérateur de cluster d'échantillons recrée la ressource en utilisant la configuration par défaut.

2.1.1. Échantillons en grappes Utilisation de l'état de gestion par l'opérateur

Le Cluster Samples Operator est démarré en tant que **Managed** par défaut ou si un proxy global est configuré. Dans l'état **Managed**, le Cluster Samples Operator gère activement ses ressources et

maintient le composant actif afin d'extraire les flux d'images d'échantillons et les images du registre et de s'assurer que les modèles d'échantillons requis sont installés.

Dans certaines circonstances, l'opérateur d'échantillonnage en grappes s'amorce lui-même comme **Removed**, y compris :

- Si le Cluster Samples Operator ne peut pas atteindre registry.redhat.io après trois minutes lors du démarrage initial après une installation propre.
- Si le Cluster Samples Operator détecte qu'il se trouve sur un réseau IPv6.
- Si les paramètres de configuration du contrôleur d'images empêchent la création de flux d'images en utilisant le registre d'images par défaut ou le registre d'images spécifié par le paramètre**samplesRegistry** .

NOTE

Pour OpenShift Container Platform, le registre d'images par défaut est **registry.redhat.io**.

Cependant, si le Cluster Samples Operator détecte qu'il est sur un réseau IPv6 et qu'un proxy global OpenShift Container Platform est configuré, alors la vérification IPv6 remplace toutes les vérifications. Par conséquent, le Cluster Samples Operator s'amorce lui-même en tant que **Removed**.

IMPORTANT

Les installations IPv6 ne sont actuellement pas prises en charge par registry.redhat.io. L'opérateur d'échantillons de clusters extrait la plupart des flux d'images d'échantillons et des images de registry.redhat.io.

2.1.1.1. Installation restreinte du réseau

L'amorçage en tant que **Removed** lorsqu'il est impossible d'accéder à **registry.redhat.io** facilite les installations réseau restreintes lorsque la restriction réseau est déjà en place. L'amorçage en tant que **Removed** lorsque l'accès au réseau est restreint donne à l'administrateur de la grappe plus de temps pour décider si des échantillons sont souhaités, car l'opérateur d'échantillons de grappe ne soumet pas d'alertes indiquant que les importations de flux d'images d'échantillons échouent lorsque l'état de gestion est défini sur **Removed**. Lorsque l'opérateur d'échantillons de grappe se présente en tant que **Managed** et tente d'installer des flux d'images d'échantillons, il commence à émettre des alertes deux heures après l'installation initiale en cas d'échec des importations.

2.1.1.2. Installation d'un réseau restreint avec accès initial au réseau

Inversement, si un cluster destiné à être un réseau restreint ou un cluster déconnecté est installé pour la première fois alors que l'accès au réseau existe, le Cluster Samples Operator installe le contenu à partir de **registry.redhat.io** puisqu'il peut y accéder. Si vous souhaitez que le Cluster Samples Operator démarre toujours en tant que **Removed** afin de différer l'installation des échantillons jusqu'à ce que vous ayez décidé quels échantillons sont souhaités, mis en place des miroirs d'image, etc., suivez alors les instructions pour utiliser le Samples Operator avec un registre alternatif et des nœuds de personnalisation, tous deux liés dans la section des ressources supplémentaires, pour remplacer la configuration par défaut du Cluster Samples Operator et s'afficher initialement en tant que **Removed**.

Vous devez placer le fichier YAML supplémentaire suivant dans le répertoire **openshift** créé par **openshift-install create manifest**:

Exemple d'échantillons de grappes Opérateur Fichier YAML avec managementState: Removed

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
metadata:
  name: cluster
spec:
  architectures:
  - x86_64
  managementState: Removed
```

2.1.2. Échantillons en grappes Suivi de l'opérateur et récupération des erreurs d'importation de flux d'images

Après la création ou la mise à jour d'un flux d'images d'échantillons, l'opérateur d'échantillons de grappes surveille la progression de l'importation d'images de chaque balise de flux d'images.

En cas d'échec de l'importation, l'opérateur d'échantillons en grappe retente l'importation par le biais de l'API d'importation d'images de flux d'images, qui est la même API que celle utilisée par la commande **oc import-image**, environ toutes les 15 minutes jusqu'à ce que l'importation réussisse ou que la configuration de l'opérateur d'échantillons en grappe soit modifiée de telle sorte que le flux d'images soit ajouté à la liste **skippedImagestreams** ou que l'état de gestion soit modifié en **Removed**.

Ressources complémentaires

- Si l'Opérateur d'échantillons en grappe est supprimé lors de l'installation, vous pouvez utiliser l'Opérateur d'échantillons en grappe avec un registre alternatif afin que le contenu puisse être importé, puis définir l'Opérateur d'échantillons en grappe sur **Managed** pour obtenir les échantillons.

- Pour s'assurer que Cluster Samples Operator démarre en tant que **Removed** dans une installation réseau restreinte avec un accès réseau initial pour différer l'installation des échantillons jusqu'à ce que vous ayez décidé quels échantillons sont souhaités, suivez les instructions pour personnaliser les nœuds afin de remplacer la configuration par défaut de Cluster Samples Operator et de démarrer initialement en tant que **Removed**.

    - Pour héberger des échantillons dans votre environnement déconnecté, suivez les instructions relatives à l'utilisation de l'Opérateur d'échantillons de cluster avec un registre alternatif.

2.1.3. Échantillons de grappes Assistance de l'opérateur pour la mise en miroir

Lors de l'installation, OpenShift Container Platform crée une carte de configuration nommée **imagestreamtag-to-image** dans l'espace de noms **openshift-cluster-samples-operator**. La carte de configuration **imagestreamtag-to-image** contient une entrée, l'image de remplissage, pour chaque balise de flux d'images.

Le format de la clé pour chaque entrée dans le champ de données de la carte de configuration est **<image_stream_name>_<image_stream_tag_name>**.

Lors d'une installation déconnectée d'OpenShift Container Platform, le statut de l'opérateur Cluster Samples est défini sur **Removed**. Si vous choisissez de le changer en **Managed**, il installe les échantillons.

NOTE

L'utilisation d'échantillons dans un environnement à réseau restreint ou interrompu peut nécessiter l'accès à des services externes à votre réseau. Voici quelques exemples de services : Github, Maven Central, npm, RubyGems, PyPi et autres. Il peut y avoir des étapes supplémentaires à franchir pour permettre aux objets des opérateurs d'échantillons de grappes d'accéder aux services dont ils ont besoin.

Vous pouvez utiliser cette carte de configuration comme référence pour savoir quelles images doivent être mises en miroir pour que vos flux d'images soient importés.

- Lorsque l'opérateur d'échantillonnage de cluster est défini sur **Removed**, vous pouvez créer votre registre en miroir ou déterminer le registre en miroir existant que vous souhaitez utiliser.
- Mettez en miroir les échantillons que vous souhaitez dans le registre mis en miroir en utilisant la nouvelle carte de configuration comme guide.
- Ajoutez tous les flux d'images que vous n'avez pas mis en miroir à la liste **skippedImagestreams** de l'objet de configuration Cluster Samples Operator.
- Définir **samplesRegistry** de l'objet de configuration Cluster Samples Operator sur le registre en miroir.
- Définissez ensuite l'opérateur d'échantillonnage de cluster sur **Managed** pour installer les flux d'images que vous avez mis en miroir.

Pour une procédure détaillée, voir Utilisation des flux d'images de l'opérateur Cluster Samples avec des registres alternatifs ou en miroir.

2.2. PARAMÈTRES DE CONFIGURATION DU CLUSTER SAMPLES OPERATOR

La ressource échantillons offre les champs de configuration suivants :

Paramètres	Description
managementState	**Managed**: L'opérateur d'échantillons en grappe met à jour les échantillons en fonction de la configuration. **Unmanaged**: L'opérateur d'échantillonnage en grappe ignore les mises à jour de son objet de ressource de configuration et de tout flux d'images ou modèle dans l'espace de noms **openshift**. **Removed**: L'opérateur Cluster Samples supprime l'ensemble des flux d'images et des modèles **Managed** dans l'espace de noms**openshift**. Il ignore les nouveaux échantillons créés par l'administrateur de la grappe ou tout échantillon figurant dans les listes ignorées. Une fois les suppressions terminées, le Cluster Samples Operator fonctionne comme s'il était dans l'état **Unmanaged** et ignore tout événement de surveillance sur les ressources d'échantillons, les flux d'images ou les modèles.

Paramètres	Description
samplesRegistry	Permet de spécifier le registre auquel les flux d'images accèdent pour leur contenu. **samplesRegistry** est par défaut **registry.redhat.io** pour OpenShift Container Platform.  NOTE La création ou la mise à jour du contenu RHEL ne commence pas si le secret pour l'accès pull n'est pas en place lorsque **Samples Registry** n'est pas explicitement défini, laissant une chaîne vide, ou lorsqu'il est défini sur registry.redhat.io. Dans les deux cas, les importations d'images fonctionnent à partir de registry.redhat.io, qui nécessite des informations d'identification. La création ou la mise à jour du contenu RHEL n'est pas limitée par l'existence du secret d'extraction si l'adresse **Samples Registry** est remplacée par une valeur autre que la chaîne vide ou le registre.redhat.io.
architectures	Espace réservé pour choisir un type d'architecture.
skippedImagestreams	Flux d'images qui se trouvent dans l'inventaire de l'opérateur d'échantillonnage de cluster mais que l'administrateur de cluster souhaite que l'opérateur ignore ou ne gère pas. Vous pouvez ajouter une liste de noms de flux d'images à ce paramètre. Par exemple, **["httpd","perl"]**.
skippedTemplates	Modèles qui se trouvent dans l'inventaire de l'opérateur d'échantillonnage de cluster, mais que l'administrateur de cluster souhaite que l'opérateur ignore ou ne gère pas.

Les événements secrets, de flux d'images et de surveillance des modèles peuvent arriver avant la création de l'objet de ressource d'échantillons initial ; l'opérateur d'échantillons en grappe détecte l'événement et le remet en file d'attente.

2.2.1. Restrictions de configuration

Lorsque le Cluster Samples Operator commence à prendre en charge plusieurs architectures, la liste des architectures n'est pas autorisée à être modifiée lorsqu'elle se trouve à l'état **Managed**.

Pour modifier les valeurs des architectures, un administrateur de cluster doit :

- Marquez le site **Management State** comme **Removed** et enregistrez la modification.
- Lors d'une modification ultérieure, modifiez l'architecture et remplacez **Management State** par **Managed**.

L'opérateur d'échantillonnage en grappe continue de traiter les secrets lorsqu'il se trouve dans l'état **Removed**. Vous pouvez créer le secret avant de passer à **Removed**, pendant que vous êtes à **Removed**, avant de passer à **Managed**, ou après avoir passé à **Managed**. La création des échantillons est retardée jusqu'à ce que l'événement secret soit traité si vous créez le secret après avoir basculé sur **Managed**.

Cela facilite la modification du registre, lorsque vous choisissez de supprimer tous les échantillons avant de basculer pour faire table rase. Il n'est pas nécessaire de supprimer tous les échantillons avant de basculer.

2.2.2. Conditions

La ressource échantillons conserve les conditions suivantes dans son statut :

Condition	Description
SamplesExists	Indique que les échantillons sont créés dans l'espace de noms **openshift**.
ImageChangesInProgress	**True** lorsque des flux d'images sont créés ou mis à jour, mais que les générations de spécifications de balises et les générations d'états de balises ne correspondent pas toutes. **False** lorsque toutes les générations correspondent ou que des erreurs irrécupérables se sont produites pendant l'importation, la dernière erreur constatée figure dans le champ message. La liste des flux d'images en attente figure dans le champ motif. Cette condition est obsolète dans OpenShift Container Platform.
ConfigurationValid	**True** ou **False** selon que l'une ou l'autre des modifications restreintes mentionnées précédemment est soumise ou non.
RemovePending	Indicateur qu'un réglage de **Management State: Removed** est en attente, mais que l'opérateur de Cluster Samples attend que les suppressions soient terminées.
ImportImageErrorsExist	Indicateur des flux d'images pour lesquels des erreurs se sont produites lors de la phase d'importation de l'image pour l'une de leurs balises. **True** lorsqu'une erreur s'est produite. La liste des flux d'images comportant une erreur figure dans le champ motif. Les détails de chaque erreur signalée se trouvent dans le champ message.
MigrationInProgress	**True** lorsque le Cluster Samples Operator détecte que la version est différente de la version du Cluster Samples Operator avec laquelle le jeu d'échantillons actuel est installé. Cette condition est obsolète dans OpenShift Container Platform.

2.3. ACCÈS À LA CONFIGURATION DE CLUSTER SAMPLES OPERATOR

Vous pouvez configurer le Cluster Samples Operator en éditant le fichier avec les paramètres fournis.

Conditions préalables

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

Procédure

- Accédez à la configuration de l'opérateur d'échantillonnage de cluster :

```
$ oc edit configs.samples.operator.openshift.io/cluster -o yaml
```

La configuration du Cluster Samples Operator ressemble à l'exemple suivant :

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
...
```

2.4. SUPPRESSION DES BALISES DE FLUX D'IMAGES OBSOLÈTES DE L'OPÉRATEUR D'ÉCHANTILLONNAGE EN GRAPPE

L'opérateur d'échantillons de clusters laisse les balises de flux d'images obsolètes dans un flux d'images parce que les utilisateurs peuvent avoir des déploiements qui utilisent les balises de flux d'images obsolètes.

Vous pouvez supprimer les balises de flux d'images obsolètes en modifiant le flux d'images à l'aide de la commande **oc tag**.

NOTE

Les balises de flux d'images obsolètes que les fournisseurs d'échantillons ont supprimées de leurs flux d'images ne sont pas incluses dans les installations initiales.

Conditions préalables

- Vous avez installé le CLI **oc**.

Procédure

- Supprimez les balises de flux d'images obsolètes en éditant le flux d'images à l'aide de la commande **oc tag**.

```
oc tag -d <image_stream_name:tag> $ oc tag -d <image_stream_name:tag>
```

Exemple de sortie

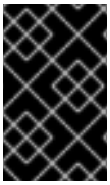
```
Suppression de la balise default/<image_stream_name:tag>.
```

Ressources complémentaires

- Pour plus d'informations sur la configuration des informations d'identification, voir Utilisation des secrets d'extraction d'image.

CHAPITRE 3. UTILISATION DU CLUSTER SAMPLES OPERATOR AVEC UN REGISTRE ALTERNATIF

Vous pouvez utiliser le Cluster Samples Operator avec un autre registre en créant d'abord un registre miroir.



IMPORTANT

Vous devez avoir accès à Internet pour obtenir les images de conteneurs nécessaires. Dans cette procédure, vous placez le registre miroir sur un hôte miroir qui a accès à la fois à votre réseau et à Internet.

3.1. À PROPOS DU REGISTRE MIROIR

Vous pouvez mettre en miroir les images requises pour l'installation d'OpenShift Container Platform et les mises à jour ultérieures du produit dans un registre de miroirs de conteneurs tel que Red Hat Quay, JFrog Artifactory, Sonatype Nexus Repository, ou Harbor. Si vous n'avez pas accès à un registre de conteneurs à grande échelle, vous pouvez utiliser *mirror registry for Red Hat OpenShift* , un registre de conteneurs à petite échelle inclus dans les abonnements à OpenShift Container Platform.

Vous pouvez utiliser n'importe quel registre de conteneurs prenant en charge Docker v2-2, tel que Red Hat Quay, *mirror registry for Red Hat OpenShift* , Artifactory, Sonatype Nexus Repository ou Harbor. Quel que soit le registre choisi, la procédure de mise en miroir du contenu des sites hébergés par Red Hat sur Internet vers un registre d'images isolé est la même. Après avoir mis en miroir le contenu, vous configurez chaque cluster pour qu'il récupère ce contenu à partir de votre registre miroir.

IMPORTANT

Le registre d'images OpenShift ne peut pas être utilisé comme registre cible car il ne prend pas en charge la poussée sans balise, ce qui est nécessaire pendant le processus de mise en miroir.

Si vous choisissez un registre de conteneurs qui n'est pas *mirror registry for Red Hat OpenShift* , il doit être accessible par chaque machine des clusters que vous provisionnez. Si le registre est inaccessible, l'installation, la mise à jour ou les opérations normales telles que la relocalisation de la charge de travail risquent d'échouer. Pour cette raison, vous devez exécuter les registres miroirs de manière hautement disponible, et les registres miroirs doivent au moins correspondre à la disponibilité de production de vos clusters OpenShift Container Platform.

Lorsque vous remplissez votre registre miroir avec des images OpenShift Container Platform, vous pouvez suivre deux scénarios. Si vous avez un hôte qui peut accéder à la fois à Internet et à votre registre miroir, mais pas à vos nœuds de cluster, vous pouvez directement mettre en miroir le contenu à partir de cette machine. Ce processus est appelé *connected mirroring*. Si vous ne disposez pas d'un tel hôte, vous devez mettre en miroir les images sur un système de fichiers, puis amener cet hôte ou ce support amovible dans votre environnement restreint. Ce processus est appelé *disconnected mirroring*.

Dans le cas des registres en miroir, pour connaître la source des images extraites, vous devez consulter l'entrée de journal **Trying to access** dans les journaux CRI-O. Les autres méthodes permettant d'afficher la source d'extraction des images, telles que la commande **crictl images** sur un nœud, affichent le nom de l'image non miroitée, même si l'image est extraite de l'emplacement miroitant.

NOTE

Red Hat ne teste pas les registres tiers avec OpenShift Container Platform.

Informations complémentaires

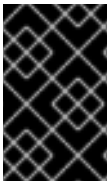
Pour plus d'informations sur la visualisation des journaux CRI-O pour visualiser la source de l'image, voir Visualisation de la source de tirage de l'image .

3.1.1. Préparation de l'hôte miroir

Avant de créer le registre miroir, vous devez préparer l'hôte miroir.

3.1.2. Installer le CLI OpenShift en téléchargeant le binaire

Vous pouvez installer l'OpenShift CLI (**oc**) pour interagir avec OpenShift Container Platform à partir d'une interface de ligne de commande. Vous pouvez installer **oc** sur Linux, Windows ou macOS.



IMPORTANT

Si vous avez installé une version antérieure de **oc**, vous ne pouvez pas l'utiliser pour exécuter toutes les commandes dans OpenShift Container Platform 4.12. Téléchargez et installez la nouvelle version de **oc**.

Installation de la CLI OpenShift sur Linux
Vous pouvez installer le binaire OpenShift CLI (**oc**) sur Linux en utilisant la procédure suivante.

Procédure

1. Naviguez jusqu'à la page de téléchargements OpenShift Container Platform sur le portail client Red Hat.
2. Sélectionnez l'architecture dans la liste déroulante **Product Variant**.
3. Sélectionnez la version appropriée dans la liste déroulante **Version**.
4. Cliquez sur **Download Now** à côté de l'entrée **OpenShift v4.12 Linux Client** et enregistrez le fichier.
5. Décompressez l'archive :

```
tar xvf <file>
```

6. Placez le fichier binaire **oc** dans un répertoire situé sur votre site **PATH**.
Pour vérifier votre **PATH**, exécutez la commande suivante :

```
$ echo $PATH
```

Après l'installation de la CLI OpenShift, elle est disponible à l'aide de la commande **oc**:

```
oc <command>
```

Installation de la CLI OpenShift sur Windows

Vous pouvez installer le binaire OpenShift CLI (**oc**) sur Windows en utilisant la procédure suivante.

Procédure

1. Naviguez jusqu'à la page de téléchargements OpenShift Container Platform sur le portail client Red Hat.
2. Sélectionnez la version appropriée dans la liste déroulante **Version**.
3. Cliquez sur **Download Now** à côté de l'entrée **OpenShift v4.12 Windows Client** et enregistrez le fichier.
4. Décompressez l'archive à l'aide d'un programme ZIP.
5. Déplacez le fichier binaire **oc** dans un répertoire situé sur votre site **PATH**.
   Pour vérifier votre **PATH**, ouvrez l'invite de commande et exécutez la commande suivante :

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

Après l'installation de la CLI OpenShift, elle est disponible à l'aide de la commande **oc**:

```
C:\N> oc <command>
```

Installation de la CLI OpenShift sur macOS
Vous pouvez installer le binaire OpenShift CLI (**oc**) sur macOS en utilisant la procédure suivante.

Procédure

1. Naviguez jusqu'à la page de téléchargements OpenShift Container Platform sur le portail client Red Hat.
2. Sélectionnez la version appropriée dans la liste déroulante **Version**.
3. Cliquez sur **Download Now** à côté de l'entrée **OpenShift v4.12 macOS Client** et enregistrez le fichier.

NOTE

Pour macOS arm64, choisissez l'entrée **OpenShift v4.12 macOS arm64 Client**

4. Décompressez l'archive.
5. Déplacez le binaire **oc** dans un répertoire de votre PATH.
   Pour vérifier votre **PATH**, ouvrez un terminal et exécutez la commande suivante :

```
$ echo $PATH
```

Après l'installation de la CLI OpenShift, elle est disponible à l'aide de la commande **oc**:

```
oc <command>
```

3.2. CONFIGURATION DES INFORMATIONS D'IDENTIFICATION PERMETTANT LA MISE EN MIROIR DES IMAGES

Créez un fichier d'informations d'identification pour le registre des images de conteneurs qui permet la mise en miroir des images de Red Hat vers votre miroir.

Conditions préalables

- Vous avez configuré un registre miroir à utiliser dans votre environnement déconnecté.

Procédure

Effectuez les étapes suivantes sur l'hôte d'installation :

1. Téléchargez votre **registry.redhat.io** pull secret depuis le gestionnaire de cluster Red Hat OpenShift.

2. Faites une copie de votre secret d'extraction au format JSON :

```
$ cat ./pull-secret | jq . > <path>/<pull_secret_file_in_json> 1
```

1 Indiquez le chemin d'accès au dossier dans lequel stocker le secret d'extraction et un nom pour le fichier JSON que vous créez.

Le contenu du fichier ressemble à l'exemple suivant :

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

3. Générer le nom d'utilisateur et le mot de passe ou le jeton encodés en base64 pour votre registre miroir :

```
$ echo -n '<user_name>:<password>' | base64 -w0 1
BGVtbYk3ZHAtqXs=
```

1 Pour **<user_name>** et **<password>**, indiquez le nom d'utilisateur et le mot de passe que vous avez configurés pour votre registre.

4. Modifiez le fichier JSON et ajoutez-y une section décrivant votre registre :

```
"auths": {
 "<mirror_registry>": { 1
  "auth": "<credentials>", 2
  "email": "you@example.com"
 }
},
```

1 Pour **<mirror_registry>**, indiquez le nom de domaine du registre, et éventuellement le port, que votre registre miroir utilise pour servir le contenu. Par exemple, **registry.example.com** ou **registry.example.com:8443**

2 Pour **<credentials>**, indiquez le nom d'utilisateur et le mot de passe encodés en base64 pour le registre miroir.

Le fichier ressemble à l'exemple suivant :

```
{
 "auths": {
  "registry.example.com": {
   "auth": "BGVtbYk3ZHAtqXs=",
   "email": "you@example.com"
  },
  "cloud.openshift.com": {
   "auth": "b3BlbnNo...",
   "email": "you@example.com"
  },
  "quay.io": {
   "auth": "b3BlbnNo...",
   "email": "you@example.com"
  },
  "registry.connect.redhat.com": {
   "auth": "NTE3Njg5Nj...",
   "email": "you@example.com"
  },
  "registry.redhat.io": {
   "auth": "NTE3Njg5Nj...",
   "email": "you@example.com"
  }
 }
}
```

3.3. MISE EN MIROIR DU RÉFÉRENTIEL D'IMAGES D'OPENSHIFT CONTAINER PLATFORM

Miroir du référentiel d'images OpenShift Container Platform sur votre registre à utiliser lors de l'installation ou de la mise à jour du cluster.

Conditions préalables

- Votre hôte miroir a accès à l'internet.
- Vous avez configuré un registre miroir à utiliser dans votre réseau restreint et vous pouvez accéder au certificat et aux informations d'identification que vous avez configurés.
- Vous avez téléchargé le secret d'extraction depuis le Red Hat OpenShift Cluster Manager et l'avez modifié pour inclure l'authentification à votre dépôt miroir.
- Si vous utilisez des certificats auto-signés, vous avez spécifié un Subject Alternative Name dans les certificats.

Procédure

Effectuez les étapes suivantes sur l'hôte miroir :

1. Consultez la page de téléchargement d'OpenShift Cont ainer Platform pour déterminer la version d'OpenShift Container Platform que vous souhaitez installer et déterminez la balise correspondante sur la page Repository Tags.
2. Définir les variables d'environnement nécessaires :
   a. Exporter la version :

      ```
      oCP_RELEASE=<release_version>
      ```

      Pour **<release_version>**, spécifiez la balise qui correspond à la version d'OpenShift Container Platform à installer, par exemple **4.5.4**.

   b. Exporter le nom du registre local et le port de l'hôte :

      ```
      $ LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>'
      ```

      Pour **<local_registry_host_name>**, indiquez le nom de domaine de votre dépôt miroir, et pour **<local_registry_host_port>**, indiquez le port sur lequel il sert le contenu.

   c. Exporter le nom du référentiel local :

      ```
      $ LOCAL_REPOSITORY='<local_repository_name>'
      ```

      Pour **<local_repository_name>**, indiquez le nom du référentiel à créer dans votre registre, par exemple **ocp4/openshift4**.

   d. Exportez le nom du référentiel à mettre en miroir :

      ```
      $ PRODUCT_REPO='openshift-release-dev'
      ```

      Pour une version de production, vous devez spécifier **openshift-release-dev**.

   e. Exportez le chemin d'accès à votre registre secret :

      ```
      $ LOCAL_SECRET_JSON='<path_to_pull_secret>'
      ```

      Pour **<path_to_pull_secret>**, indiquez le chemin absolu et le nom de fichier du secret d'extraction pour le registre miroir que vous avez créé.

f. Exporter le miroir de validation :

```
$ RELEASE_NAME="ocp-release"
```

Pour une version de production, vous devez spécifier **ocp-release**.

g. Exportez le type d'architecture de votre serveur, par exemple **x86_64** ou **aarch64**:

```
aRCHITECTURE=<server_architecture>
```

h. Exportez le chemin d'accès au répertoire qui hébergera les images en miroir :

```
rEMOVABLE_MEDIA_PATH=<path> $ REMOVABLE_MEDIA_PATH=<path> (1)
```

(1) Indiquer le chemin d'accès complet, y compris la barre oblique initiale (/).

3. Mettre en miroir les images de la version dans le registre miroir :

- Si votre hôte miroir n'a pas d'accès à Internet, prenez les mesures suivantes :

i. Connectez le support amovible à un système connecté à Internet.

ii. Examinez les images et les manifestes de configuration pour créer un miroir :

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE} --dry-run
```

iii. Enregistrez l'intégralité de la section **imageContentSources** à partir de la sortie de la commande précédente. Les informations sur vos miroirs sont propres à votre dépôt miroir, et vous devez ajouter la section **imageContentSources** au fichier **install-config.yaml** lors de l'installation.

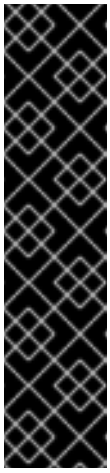
iv. Mettez les images en miroir dans un répertoire du support amovible :

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} --to-
dir=${REMOVABLE_MEDIA_PATH}/mirror
quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE}
```

v. Transférer les supports dans l'environnement réseau restreint et télécharger les images dans le registre local des conteneurs.

```
$ oc image mirror -a ${LOCAL_SECRET_JSON} --from-
dir=${REMOVABLE_MEDIA_PATH}/mirror
"file://openshift/release:${OCP_RELEASE}*"
${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} (1)
```

1 Pour **REMOVABLE_MEDIA_PATH**, vous devez utiliser le même chemin d'accès que celui spécifié lors de la mise en miroir des images.



IMPORTANT

L'exécution de **oc image mirror** peut entraîner l'erreur suivante : **error: unable to retrieve source image**. Cette erreur se produit lorsque les index d'images incluent des références à des images qui n'existent plus dans le registre d'images. Les index d'images peuvent conserver des références plus anciennes afin de permettre aux utilisateurs qui utilisent ces images de bénéficier d'un chemin de mise à niveau vers des points plus récents du graphique de mise à niveau. En guise de solution temporaire, vous pouvez utiliser l'option **--skip-missing** pour contourner l'erreur et continuer à télécharger l'index d'image. Pour plus d'informations, voir Échec de la mise en miroir de l'opérateur du service Mesh.

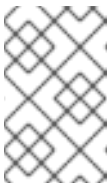
- Si le registre de conteneurs local est connecté à l'hôte miroir, prenez les mesures suivantes :

  i. Poussez directement les images de version dans le registre local à l'aide de la commande suivante :

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON}  \
    --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
${ARCHITECTURE} \
    --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
    --to-release-
image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}
```

  Cette commande extrait les informations de version sous forme de condensé, et sa sortie inclut les données **imageContentSources** dont vous avez besoin lorsque vous installez votre cluster.

  ii. Enregistrez l'intégralité de la section **imageContentSources** à partir de la sortie de la commande précédente. Les informations sur vos miroirs sont propres à votre dépôt miroir, et vous devez ajouter la section **imageContentSources** au fichier **install-config.yaml** lors de l'installation.



NOTE

Le nom de l'image est remplacé par Quay.io pendant le processus de mise en miroir, et les images podman afficheront Quay.io dans le registre de la machine virtuelle de démarrage.

4. Pour créer le programme d'installation basé sur le contenu que vous avez mis en miroir, extrayez-le et épinglez-le à la version :

   - Si votre hôte miroir n'a pas d'accès à Internet, exécutez la commande suivante :

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --icsp-file=<file> \N--
command=openshift-install
\N"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}\N"
```

- Si le registre de conteneurs local est connecté à l'hôte miroir, exécutez la commande suivante :

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}"
```

IMPORTANT

Pour vous assurer que vous utilisez les images correctes pour la version d'OpenShift Container Platform que vous avez sélectionnée, vous devez extraire le programme d'installation du contenu mis en miroir.

Vous devez effectuer cette étape sur une machine disposant d'une connexion internet active.

Si vous êtes dans un environnement déconnecté, utilisez l'indicateur **--image** dans le cadre de la collecte obligatoire et pointez vers l'image de la charge utile.

5. Pour les clusters utilisant une infrastructure fournie par l'installateur, exécutez la commande suivante :

```
$ openshift-install
```

3.4. UTILISATION DES FLUX D'IMAGES DE CLUSTER SAMPLES OPERATOR AVEC DES REGISTRES ALTERNATIFS OU EN MIROIR

La plupart des flux d'images dans l'espace de noms **openshift** géré par le Cluster Samples Operator pointent vers des images situées dans le registre Red Hat à l'adresse registry .redhat.io.

NOTE

Les flux d'images **cli**, **installer**, **must-gather** et **tests**, bien qu'ils fassent partie de la charge utile de l'installation, ne sont pas gérés par l'opérateur d'échantillonnage en grappe. Ils ne sont pas abordés dans cette procédure.

IMPORTANT

Le Cluster Samples Operator doit être défini sur **Managed** dans un environnement déconnecté. Pour installer les flux d'images, vous devez disposer d'un registre miroir.

Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- Créez un secret d'extraction pour votre registre miroir.

Procédure

1. Accéder aux images d'un flux d'images spécifique à mettre en miroir, par exemple :

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep
registry.redhat.io
```

2. Images miroirs de registry.redhat.io associées à tous les flux d'images dont vous avez besoin

```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. Créer l'objet de configuration de l'image du cluster :

```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. Ajoutez les autorités de certification de confiance requises pour le miroir dans l'objet de configuration de l'image du cluster :

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

5. Mettez à jour le champ **samplesRegistry** dans l'objet de configuration Cluster Samples Operator pour qu'il contienne la partie **hostname** de l'emplacement du miroir défini dans la configuration du miroir :

```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```

NOTE

Ceci est nécessaire car le processus d'importation de flux d'images n'utilise pas le mécanisme de miroir ou de recherche pour le moment.

6. Ajoutez tous les flux d'images qui ne sont pas en miroir dans le champ **skippedImagestreams** de l'objet de configuration Cluster Samples Operator. Ou si vous ne souhaitez pas prendre en charge les flux d'images échantillonnés, définissez l'opérateur d'échantillonnage de cluster sur **Removed** dans l'objet de configuration de l'opérateur d'échantillonnage de cluster.

NOTE

Le Cluster Samples Operator émet des alertes si les importations de flux d'images échouent mais que le Cluster Samples Operator les relance périodiquement ou ne semble pas les relancer.

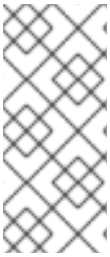
De nombreux modèles de l'espace de noms **openshift** font référence aux flux d'images. L'utilisation de **Removed** pour purger à la fois les flux d'images et les modèles éliminera les tentatives d'utilisation de ces derniers s'ils ne sont pas fonctionnels en raison de l'absence de flux d'images.

3.4.1. Échantillons de grappes Assistance de l'opérateur pour la mise en miroir

Lors de l'installation, OpenShift Container Platform crée une carte de configuration nommée **imagestreamtag-to-image** dans l'espace de noms **openshift-cluster-samples-operator**. La carte de configuration **imagestreamtag-to-image** contient une entrée, l'image de remplissage, pour chaque balise de flux d'images.

Le format de la clé pour chaque entrée dans le champ de données de la carte de configuration est **<image_stream_name>_<image_stream_tag_name>**.

Lors d'une installation déconnectée d'OpenShift Container Platform, le statut de l'opérateur Cluster Samples est défini sur **Removed**. Si vous choisissez de le changer en **Managed**, il installe les échantillons.



NOTE

L'utilisation d'échantillons dans un environnement à réseau restreint ou interrompu peut nécessiter l'accès à des services externes à votre réseau. Voici quelques exemples de services : Github, Maven Central, npm, RubyGems, PyPi et autres. Il peut y avoir des étapes supplémentaires à franchir pour permettre aux objets des opérateurs d'échantillons de grappes d'accéder aux services dont ils ont besoin.

Vous pouvez utiliser cette carte de configuration comme référence pour savoir quelles images doivent être mises en miroir pour que vos flux d'images soient importés.

- Lorsque l'opérateur d'échantillonnage de cluster est défini sur **Removed**, vous pouvez créer votre registre en miroir ou déterminer le registre en miroir existant que vous souhaitez utiliser.
- Mettez en miroir les échantillons que vous souhaitez dans le registre mis en miroir en utilisant la nouvelle carte de configuration comme guide.
- Ajoutez tous les flux d'images que vous n'avez pas mis en miroir à la liste **skippedImagestreams** de l'objet de configuration Cluster Samples Operator.
- Définir **samplesRegistry** de l'objet de configuration Cluster Samples Operator sur le registre en miroir.
- Définissez ensuite l'opérateur d'échantillonnage de cluster sur **Managed** pour installer les flux d'images que vous avez mis en miroir.

Pour une procédure détaillée, voir Utilisation des flux d'images de l'opérateur Cluster Samples avec des registres alternatifs ou en miroir.

CHAPITRE 4. CRÉATION D'IMAGES

Apprenez à créer vos propres images de conteneurs, basées sur des images préconstruites qui sont prêtes à vous aider. Le processus comprend l'apprentissage des meilleures pratiques pour l'écriture des images, la définition des métadonnées pour les images, le test des images et l'utilisation d'un flux de travail de construction personnalisé pour créer des images à utiliser avec OpenShift Container Platform. Après avoir créé une image, vous pouvez la pousser vers le registre d'images OpenShift.

4.1. APPRENDRE LES MEILLEURES PRATIQUES EN MATIÈRE DE CONTENEURS

Lors de la création d'images de conteneurs à exécuter sur OpenShift Container Platform, il y a un certain nombre de meilleures pratiques à prendre en compte en tant qu'auteur d'images pour assurer une bonne expérience pour les consommateurs de ces images. Les images étant destinées à être immuables et utilisées telles quelles, les directives suivantes permettent de s'assurer que vos images sont hautement consommables et faciles à utiliser sur OpenShift Container Platform.

4.1.1. Lignes directrices générales concernant les images de conteneurs

Les directives suivantes s'appliquent à la création d'une image de conteneur en général, et sont indépendantes de l'utilisation des images sur OpenShift Container Platform.

Réutiliser les images
Dans la mesure du possible, basez votre image sur une image en amont appropriée en utilisant la déclaration **FROM**. Cela permet à votre image de récupérer facilement les correctifs de sécurité d'une image en amont lorsqu'elle est mise à jour, plutôt que d'avoir à mettre à jour vos dépendances directement.

En outre, utilisez des balises dans l'instruction **FROM**, par exemple **rhel:rhel7**, pour indiquer clairement aux utilisateurs la version d'une image sur laquelle votre image est basée. L'utilisation d'une balise autre que **latest** permet de s'assurer que votre image n'est pas soumise à des modifications qui pourraient être apportées à la version **latest** d'une image en amont.

Maintenir la compatibilité entre les balises
Lorsque vous marquez vos propres images, essayez de maintenir la compatibilité ascendante au sein d'une balise. Par exemple, si vous fournissez une image nommée **foo** et qu'elle inclut actuellement la version **1.0**, vous pouvez fournir une balise **foo:v1**. Lorsque vous mettez à jour l'image, tant qu'elle reste compatible avec l'image originale, vous pouvez continuer à baliser la nouvelle image **foo:v1**, et les consommateurs en aval de cette balise sont en mesure d'obtenir des mises à jour sans être interrompus.

Si vous publiez ultérieurement une mise à jour incompatible, passez à une nouvelle balise, par exemple **foo:v2**. Cela permet aux consommateurs en aval de passer à la nouvelle version à leur guise, sans être interrompus par inadvertance par la nouvelle image incompatible. Tout consommateur en aval qui utilise **foo:latest** prend le risque que des changements incompatibles soient introduits.

Éviter les processus multiples
Ne démarrez pas plusieurs services, tels qu'une base de données et **SSHD**, à l'intérieur d'un même conteneur. Cela n'est pas nécessaire car les conteneurs sont légers et peuvent être facilement reliés entre eux pour orchestrer plusieurs processus. OpenShift Container Platform vous permet de colocaliser et de cogérer facilement des images connexes en les regroupant dans un seul pod.

Cette colocation garantit que les conteneurs partagent un espace de noms réseau et un espace de stockage pour la communication. Les mises à jour sont également moins perturbantes, car chaque image peut être mise à jour moins fréquemment et indépendamment. Les flux de traitement des signaux

sont également plus clairs avec un processus unique, car il n'est pas nécessaire de gérer l'acheminement des signaux vers les processus créés.

Utiliser exec dans les scripts d'accompagnement
De nombreuses images utilisent des scripts enveloppants pour effectuer certaines configurations avant de lancer un processus pour le logiciel en cours d'exécution. Si votre image utilise un tel script, celui-ci utilise **exec** afin que le processus du script soit remplacé par votre logiciel. Si vous n'utilisez pas **exec**, les signaux envoyés par le moteur d'exécution de votre conteneur sont transmis à votre script d'encapsulation au lieu du processus de votre logiciel. Ce n'est pas ce que vous souhaitez.

Si vous avez un script wrapper qui démarre un processus pour un serveur. Vous démarrez votre conteneur, par exemple, à l'aide de **podman run -i**, qui exécute le script wrapper, lequel démarre à son tour votre processus. Si vous souhaitez fermer votre conteneur à l'aide de **CTRL C**, si votre script wrapper a utilisé **exec** pour démarrer le processus du serveur, **podman** envoie SIGINT au processus du serveur et tout se passe comme prévu. Si vous n'avez pas utilisé **exec** dans votre script wrapper, **podman** envoie SIGINT au processus du script wrapper et votre processus continue à fonctionner comme si de rien n'était.

Notez également que votre processus s'exécute en tant que **PID 1** lorsqu'il est exécuté dans un conteneur. Cela signifie que si votre processus principal se termine, l'ensemble du conteneur est arrêté, ce qui annule tous les processus enfants que vous avez lancés à partir de votre processus **PID 1**.

Nettoyer les fichiers temporaires
Supprimez tous les fichiers temporaires créés au cours du processus de construction. Cela inclut également tous les fichiers ajoutés avec la commande **ADD**. Par exemple, exécutez la commande **yum clean** après avoir effectué les opérations **yum install**.

Vous pouvez éviter que le cache **yum** ne se retrouve dans une couche d'image en créant votre déclaration **RUN** de la manière suivante :

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clean all -y
```

Notez que si vous écrivez à la place :

```
RUN yum -y install mypackage
RUN yum -y install myotherpackage && yum clean all -y
```

La première invocation de **yum** laisse alors des fichiers supplémentaires dans cette couche, et ces fichiers ne peuvent pas être supprimés lorsque l'opération **yum clean** est exécutée ultérieurement. Les fichiers supplémentaires ne sont pas visibles dans l'image finale, mais ils sont présents dans les couches sous-jacentes.

Le processus actuel de construction des conteneurs ne permet pas à une commande exécutée dans une couche ultérieure de réduire l'espace utilisé par l'image lorsque quelque chose a été supprimé dans une couche antérieure. Toutefois, cela pourrait changer à l'avenir. Cela signifie que si vous exécutez une commande **rm** dans une couche ultérieure, bien que les fichiers soient cachés, cela ne réduit pas la taille globale de l'image à télécharger. Par conséquent, comme dans l'exemple **yum clean**, il est préférable de supprimer les fichiers dans la même commande que celle qui les a créés, dans la mesure du possible, afin qu'ils ne finissent pas par être écrits dans un calque.

En outre, l'exécution de plusieurs commandes dans une seule déclaration **RUN** réduit le nombre de couches de votre image, ce qui améliore le temps de téléchargement et d'extraction.

Placez les instructions dans l'ordre approprié
Le constructeur de conteneurs lit le site **Dockerfile** et exécute les instructions de haut en bas. Chaque instruction exécutée avec succès crée une couche qui peut être réutilisée lors de la prochaine

construction de cette image ou d'une autre. Il est très important de placer les instructions qui changent rarement au sommet de votre **Dockerfile**. Cela garantit que les prochaines constructions de la même image seront très rapides, car le cache n'est pas invalidé par les modifications des couches supérieures.

Par exemple, si vous travaillez sur un site **Dockerfile** qui contient une commande **ADD** pour installer un fichier sur lequel vous êtes en train d'itérer, et une commande **RUN** pour **yum install** un paquet, il est préférable de placer la commande **ADD** en dernier :

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

Ainsi, chaque fois que vous modifiez **myfile** et que vous exécutez à nouveau **podman build** ou **docker build**, le système réutilise la couche mise en cache pour la commande **yum** et ne génère la nouvelle couche que pour l'opération **ADD**.

Si, au lieu de cela, vous écriviez **Dockerfile** comme suit :

```
FROM foo
ADD myfile /test/myfile
RUN yum -y install mypackage && yum clean all -y
```

Ensuite, chaque fois que vous modifiez **myfile** et que vous exécutez à nouveau **podman build** ou **docker build**, l'opération **ADD** invalide le cache de la couche **RUN**, de sorte que l'opération **yum** doit également être exécutée à nouveau.

Marquer les ports importants
L'instruction EXPOSE rend un port du conteneur accessible au système hôte et aux autres conteneurs. Bien qu'il soit possible de spécifier qu'un port doit être exposé avec une invocation **podman run**, l'utilisation de l'instruction EXPOSE dans un **Dockerfile** facilite l'utilisation de votre image par les humains et les logiciels en déclarant explicitement les ports dont votre logiciel a besoin pour fonctionner :

- Les ports exposés apparaissent sous **podman ps** associés aux conteneurs créés à partir de votre image.
- Les ports exposés sont présents dans les métadonnées de l'image renvoyée par **podman inspect**.
- Les ports exposés sont liés lorsque vous reliez un conteneur à un autre.

Définir les variables d'environnement
Il est conseillé de définir des variables d'environnement à l'aide de l'instruction **ENV**. Un exemple est la définition de la version de votre projet. Cela permet aux gens de trouver facilement la version sans avoir à consulter l'instruction **Dockerfile**. Un autre exemple consiste à annoncer un chemin sur le système qui pourrait être utilisé par un autre processus, tel que **JAVA_HOME**.

Éviter les mots de passe par défaut
Évitez de définir des mots de passe par défaut. De nombreuses personnes étendent l'image et oublient de supprimer ou de modifier le mot de passe par défaut. Cela peut entraîner des problèmes de sécurité si un utilisateur en production se voit attribuer un mot de passe bien connu. Les mots de passe peuvent être configurés à l'aide d'une variable d'environnement.

Si vous choisissez de définir un mot de passe par défaut, veillez à ce qu'un message d'avertissement approprié s'affiche au démarrage du conteneur. Ce message doit informer l'utilisateur de la valeur du mot de passe par défaut et lui expliquer comment le modifier, par exemple en lui indiquant la variable

d'environnement à définir.

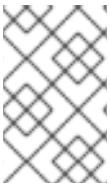
Éviter sshd
Il est préférable d'éviter d'exécuter **sshd** dans votre image. Vous pouvez utiliser la commande **podman exec** ou **docker exec** pour accéder aux conteneurs qui s'exécutent sur l'hôte local. Vous pouvez également utiliser la commande **oc exec** ou la commande **oc rsh** pour accéder aux conteneurs qui s'exécutent sur le cluster OpenShift Container Platform. L'installation et l'exécution de **sshd** dans votre image ouvre des vecteurs d'attaque supplémentaires et des exigences en matière de correctifs de sécurité.

Utiliser des volumes pour les données persistantes
Les images utilisent un volume pour les données persistantes. De cette façon, OpenShift Container Platform monte le stockage réseau sur le nœud qui exécute le conteneur, et si le conteneur est déplacé vers un nouveau nœud, le stockage est rattaché à ce nœud. En utilisant le volume pour tous les besoins de stockage persistant, le contenu est préservé même si le conteneur est redémarré ou déplacé. Si votre image écrit des données à des emplacements arbitraires dans le conteneur, ce contenu ne pourra pas être préservé.

Toutes les données qui doivent être conservées même après la destruction du conteneur doivent être écrites sur un volume. Les moteurs de conteneurs prennent en charge un indicateur **readonly** pour les conteneurs, qui peut être utilisé pour appliquer strictement les bonnes pratiques concernant l'absence d'écriture de données sur un stockage éphémère dans un conteneur. En concevant votre image autour de cette capacité dès maintenant, il sera plus facile d'en tirer parti ultérieurement.

La définition explicite des volumes dans votre site **Dockerfile** permet aux utilisateurs de l'image de comprendre facilement quels volumes ils doivent définir lors de l'exécution de votre image.

Voir la documentation Kubernetes pour plus d'informations sur l'utilisation des volumes dans OpenShift Container Platform.



NOTE

Même avec les volumes persistants, chaque instance de votre image possède son propre volume, et le système de fichiers n'est pas partagé entre les instances. Cela signifie que le volume ne peut pas être utilisé pour partager l'état dans un cluster.

4.1.2. Directives spécifiques à OpenShift Container Platform

Les directives suivantes s'appliquent lors de la création d'images de conteneurs spécifiquement destinées à être utilisées sur OpenShift Container Platform.

4.1.2.1. Activer les images pour la conversion source-image (S2I)

Pour les images destinées à exécuter un code d'application fourni par un tiers, comme une image Ruby conçue pour exécuter un code Ruby fourni par un développeur, vous pouvez permettre à votre image de fonctionner avec l'outil de construction Source-to-Image (S2I). S2I est un cadre qui facilite l'écriture d'images qui prennent le code source d'une application en entrée et produisent une nouvelle image qui exécute l'application assemblée en sortie.

4.1.2.2. Prise en charge d'identifiants d'utilisateurs arbitraires

Par défaut, OpenShift Container Platform exécute les conteneurs en utilisant un identifiant utilisateur attribué arbitrairement. Cela fournit une sécurité supplémentaire contre les processus qui s'échappent du conteneur en raison d'une vulnérabilité du moteur de conteneur et qui obtiennent ainsi des permissions accrues sur le nœud hôte.

Pour qu'une image puisse être exécutée par un utilisateur arbitraire, les répertoires et les fichiers sur lesquels les processus de l'image écrivent doivent appartenir au groupe root et être accessibles en lecture/écriture par ce groupe. Les fichiers à exécuter doivent également disposer des autorisations d'exécution du groupe.

L'ajout de ce qui suit à votre fichier Docker définit les autorisations de répertoire et de fichier pour permettre aux utilisateurs du groupe root d'y accéder dans l'image construite :

```
RUN chgrp -R 0 /some/directory && \
    chmod -R g=u /some/directory
```

L'utilisateur du conteneur étant toujours membre du groupe root, il peut lire et écrire ces fichiers.

AVERTISSEMENT

Il convient d'être prudent lorsque l'on modifie les répertoires et les autorisations de fichiers des zones sensibles d'un conteneur, ce qui n'est pas différent d'un système normal.

S'il est appliqué à des zones sensibles, telles que **/etc/passwd**, il peut permettre la modification de ces fichiers par des utilisateurs involontaires, exposant ainsi potentiellement le conteneur ou l'hôte. CRI-O prend en charge l'insertion d'identifiants d'utilisateurs arbitraires dans le conteneur **/etc/passwd**, de sorte qu'il n'est jamais nécessaire de modifier les autorisations.

En outre, les processus exécutés dans le conteneur ne doivent pas écouter les ports privilégiés, c'est-à-dire les ports inférieurs à 1024, puisqu'ils ne sont pas exécutés en tant qu'utilisateur privilégié.

IMPORTANT

Si votre image S2I n'inclut pas de déclaration **USER** avec un utilisateur numérique, vos constructions échouent par défaut. Pour permettre aux images qui utilisent des utilisateurs nommés ou l'utilisateur root **0** de construire dans OpenShift Container Platform, vous pouvez ajouter le compte de service de construction du projet, **system:serviceaccount:<your-project>:builder**, à la contrainte de contexte de sécurité (SCC) **anyuid**. Vous pouvez également autoriser toutes les images à s'exécuter sous n'importe quel utilisateur.

4.1.2.3. Utiliser des services pour la communication entre images

Dans les cas où votre image doit communiquer avec un service fourni par une autre image, telle qu'une image frontale web qui doit accéder à une image de base de données pour stocker et récupérer des données, votre image consomme un service OpenShift Container Platform. Les services fournissent un point d'accès statique qui ne change pas lorsque les conteneurs sont arrêtés, démarrés ou déplacés. En outre, les services assurent l'équilibrage de la charge pour les demandes.

4.1.2.4. Fournir des bibliothèques communes

Pour les images destinées à exécuter un code d'application fourni par un tiers, assurez-vous que votre image contient les bibliothèques couramment utilisées pour votre plate-forme. En particulier, fournissez

des pilotes de bases de données pour les bases de données courantes utilisées avec votre plateforme. Par exemple, fournissez des pilotes JDBC pour MySQL et PostgreSQL si vous créez une image Java Framework. En procédant ainsi, il n'est pas nécessaire de télécharger des dépendances communes au moment de l'assemblage de l'application, ce qui accélère la création d'images d'application. Cela simplifie également le travail des développeurs d'applications qui doivent s'assurer que toutes leurs dépendances sont respectées.

4.1.2.5. Utiliser les variables d'environnement pour la configuration

Les utilisateurs de votre image peuvent la configurer sans avoir à créer une image en aval basée sur votre image. Cela signifie que la configuration de l'exécution est gérée à l'aide de variables d'environnement. Pour une configuration simple, le processus en cours d'exécution peut consommer directement les variables d'environnement. Pour une configuration plus compliquée ou pour les processus d'exécution qui ne le supportent pas, configurez le processus d'exécution en définissant un fichier de configuration modèle qui est traité lors du démarrage. Au cours de ce traitement, les valeurs fournies par les variables d'environnement peuvent être substituées dans le fichier de configuration ou utilisées pour décider des options à définir dans le fichier de configuration.

Il est également possible et recommandé de transmettre des secrets tels que des certificats et des clés dans le conteneur à l'aide de variables d'environnement. Cela permet de s'assurer que les valeurs secrètes ne se retrouvent pas engagées dans une image et ne fuient pas dans un registre d'image de conteneur.

Le fait de fournir des variables d'environnement permet aux consommateurs de votre image de personnaliser le comportement, comme les paramètres de la base de données, les mots de passe et le réglage des performances, sans avoir à introduire une nouvelle couche au-dessus de votre image. Au lieu de cela, ils peuvent simplement définir des valeurs de variables d'environnement lors de la définition d'un pod et modifier ces paramètres sans reconstruire l'image.

Pour des scénarios extrêmement complexes, la configuration peut également être fournie à l'aide de volumes qui seront montés dans le conteneur au moment de l'exécution. Toutefois, si vous optez pour cette solution, vous devez vous assurer que votre image fournit des messages d'erreur clairs au démarrage lorsque le volume ou la configuration nécessaire n'est pas présent.

Cette rubrique est liée à la rubrique Using Services for Inter-image Communication en ce sens que la configuration comme les datasources est définie en termes de variables d'environnement qui fournissent les informations sur le point de terminaison du service. Cela permet à une application de consommer dynamiquement un service de source de données défini dans l'environnement OpenShift Container Platform sans modifier l'image de l'application.

En outre, le réglage est effectué en inspectant les paramètres **cgroups** pour le conteneur. Cela permet à l'image de s'adapter à la mémoire, au processeur et aux autres ressources disponibles. Par exemple, les images basées sur Java adaptent leur tas en fonction du paramètre de mémoire maximale de **cgroup** afin de s'assurer qu'elles ne dépassent pas les limites et n'obtiennent pas d'erreur de mémoire insuffisante.

4.1.2.6. Définir les métadonnées de l'image

Définir des métadonnées d'image aide OpenShift Container Platform à mieux consommer vos images de conteneurs, permettant à OpenShift Container Platform de créer une meilleure expérience pour les développeurs qui utilisent votre image. Par exemple, vous pouvez ajouter des métadonnées pour fournir des descriptions utiles de votre image, ou offrir des suggestions sur d'autres images qui sont nécessaires.

4.1.2.7. Regroupement

Vous devez bien comprendre ce que signifie l'exécution de plusieurs instances de votre image. Dans le cas le plus simple, la fonction d'équilibrage de charge d'un service gère le routage du trafic vers toutes les instances de votre image. Cependant, de nombreux frameworks doivent partager des informations pour procéder à l'élection d'un leader ou à un basculement, par exemple dans le cadre de la réplication d'une session.

Réfléchissez à la manière dont vos instances réalisent cette communication lorsqu'elles fonctionnent sur OpenShift Container Platform. Bien que les pods puissent communiquer directement entre eux, leurs adresses IP changent à chaque fois que le pod démarre, s'arrête ou est déplacé. Il est donc important que votre schéma de clustering soit dynamique.

4.1.2.8. Enregistrement

Il est préférable d'envoyer toute la journalisation vers la sortie standard. OpenShift Container Platform collecte la sortie standard des conteneurs et l'envoie au service de journalisation centralisé où elle peut être consultée. Si vous devez séparer le contenu des journaux, préfixez la sortie avec un mot-clé approprié, ce qui permet de filtrer les messages.

Si votre image enregistre les données dans un fichier, les utilisateurs doivent utiliser des opérations manuelles pour entrer dans le conteneur en cours d'exécution et récupérer ou consulter le fichier journal.

4.1.2.9. Sondages sur l'état d'esprit et l'état de préparation

Documenter des exemples de sondes de disponibilité et de préparation qui peuvent être utilisées avec votre image. Ces sondes permettent aux utilisateurs de déployer leur image avec la certitude que le trafic n'est pas acheminé vers le conteneur tant qu'il n'est pas prêt à le gérer, et que le conteneur est redémarré si le processus se trouve dans un état malsain.

4.1.2.10. Modèles

Pensez à fournir un modèle d'exemple avec votre image. Un modèle donne aux utilisateurs un moyen facile de déployer rapidement votre image avec une configuration fonctionnelle. Votre modèle doit inclure les sondes de disponibilité et de préparation que vous avez documentées avec l'image, par souci d'exhaustivité.

4.2. INCLURE DES MÉTADONNÉES DANS LES IMAGES

Définir des métadonnées d'image aide OpenShift Container Platform à mieux consommer vos images de conteneurs, permettant à OpenShift Container Platform de créer une meilleure expérience pour les développeurs qui utilisent votre image. Par exemple, vous pouvez ajouter des métadonnées pour fournir des descriptions utiles de votre image, ou offrir des suggestions sur d'autres images qui pourraient également être nécessaires.

Cette rubrique ne définit que les métadonnées nécessaires à la série actuelle de cas d'utilisation. D'autres métadonnées ou cas d'utilisation pourront être ajoutés à l'avenir.

4.2.1. Définition des métadonnées de l'image

Vous pouvez utiliser l'instruction **LABEL** dans une page **Dockerfile** pour définir les métadonnées de l'image. Les étiquettes sont similaires aux variables d'environnement en ce sens qu'il s'agit de paires clé-valeur attachées à une image ou à un conteneur. Les étiquettes sont différentes des variables d'environnement en ce sens qu'elles ne sont pas visibles par l'application en cours d'exécution et qu'elles peuvent également être utilisées pour la recherche rapide d'images et de conteneurs.

Documentation Docker pour plus d'informations sur l'instruction **LABEL**.

Les noms des étiquettes sont généralement associés à un espace de noms. L'espace de noms est défini en conséquence pour refléter le projet qui va récupérer les étiquettes et les utiliser. Pour OpenShift Container Platform, l'espace de noms est défini sur **io.openshift** et pour Kubernetes, l'espace de noms est **io.k8s**.

Voir la documentation sur les métadonnées personnalisées de Docker pour plus de détails sur le format.

Tableau 4.1. Métadonnées prises en charge

Variable	Description
io.openshift.tags	Cette étiquette contient une liste de balises représentées sous la forme d'une liste de valeurs séparées par des virgules. Les balises permettent de classer les images de conteneurs dans de vastes domaines de fonctionnalité. Les balises aident les outils d'interface utilisateur et de génération à suggérer des images de conteneur pertinentes au cours du processus de création de l'application. `LABEL io.openshift.tags mongodb,mongodb24,nosql`
io.openshift.wants	Spécifie une liste de balises que les outils de génération et l'interface utilisateur utilisent pour fournir des suggestions pertinentes si vous n'avez pas déjà les images de conteneur avec les balises spécifiées. Par exemple, si l'image de conteneur veut **mysql** et **redis** et que vous n'avez pas l'image de conteneur avec la balise **redis**, l'interface utilisateur peut vous suggérer d'ajouter cette image dans votre déploiement. `LABEL io.openshift.wants mongodb,redis`
io.k8s.description	Cette étiquette peut être utilisée pour donner aux consommateurs de l'image de conteneur des informations plus détaillées sur le service ou la fonctionnalité que cette image fournit. L'interface utilisateur peut alors utiliser cette description avec le nom de l'image de conteneur pour fournir des informations plus conviviales aux utilisateurs finaux. `LABEL io.k8s.description The MySQL 5.5 Server with master-slave replication support`
io.openshift.non-scalable	Une image peut utiliser cette variable pour indiquer qu'elle ne prend pas en charge la mise à l'échelle. L'interface utilisateur le communique alors aux consommateurs de cette image. Le fait que l'image ne soit pas extensible signifie que la valeur de **replicas** ne doit pas être supérieure à celle de**1**. `LABEL io.openshift.non-scalable true`

Variable	Description

io.openshift.min-memory et **io.openshift.min-cpu**	Cette étiquette indique la quantité de ressources dont l'image du conteneur a besoin pour fonctionner correctement. L'interface utilisateur peut avertir l'utilisateur que le déploiement de cette image de conteneur peut dépasser son quota. Les valeurs doivent être compatibles avec la quantité de Kubernetes.
	`LABEL io.openshift.min-memory 16Gi`
	`LABEL io.openshift.min-cpu 4`

4.3. CRÉER DES IMAGES À PARTIR DU CODE SOURCE AVEC SOURCE-TO-IMAGE

Source-to-image (S2I) est un cadre qui facilite l'écriture d'images qui prennent le code source d'une application en entrée et produisent une nouvelle image qui exécute l'application assemblée en sortie.

Le principal avantage de l'utilisation de S2I pour la construction d'images de conteneurs reproductibles est la facilité d'utilisation pour les développeurs. En tant qu'auteur d'une image de construction, vous devez comprendre deux concepts de base pour que vos images offrent les meilleures performances S2I : le processus de construction et les scripts S2I.

4.3.1. Comprendre le processus de construction de la source à l'image

Le processus de construction se compose des trois éléments fondamentaux suivants, qui sont combinés pour former une image finale du conteneur :

- Sources d'information
- Scripts source-image (S2I)
- Image du constructeur

S2I génère un Dockerfile avec l'image du constructeur comme première instruction **FROM**. Le fichier Docker généré par S2I est ensuite transmis à Buildah.

4.3.2. Comment écrire des scripts source-image ?

Vous pouvez écrire des scripts source-image (S2I) dans n'importe quel langage de programmation, à condition que les scripts soient exécutables dans l'image du constructeur. S2I prend en charge plusieurs options fournissant des scripts **assemble**/**run**/**save-artifacts**. Tous ces emplacements sont vérifiés lors de chaque compilation dans l'ordre suivant :

1. Un script spécifié dans la configuration de la construction.
2. Un script trouvé dans le répertoire source de l'application **.s2i**/**bin**.
3. Un script trouvé à l'URL de l'image par défaut avec l'étiquette **io.openshift.s2i.scripts-url**.

L'étiquette **io.openshift.s2i.scripts-url** spécifiée dans l'image et le script spécifié dans une configuration de compilation peuvent prendre l'une des formes suivantes :

- **image:///path_to_scripts_dir**chemin absolu à l'intérieur de l'image vers un répertoire où se trouvent les scripts S2I.

- **file:///path_to_scripts_dir**chemin d'accès : chemin relatif ou absolu vers un répertoire de l'hôte où se trouvent les scripts S2I.
- **http(s)://path_to_scripts_dir**: URL d'un répertoire où se trouvent les scripts S2I.

Tableau 4.2. Scénarios S2I

Le scénario	Description
assemble	Le script **assemble** construit les artefacts de l'application à partir d'une source et les place dans les répertoires appropriés à l'intérieur de l'image. Ce script est obligatoire. Le flux de travail pour ce script est le suivant : 1. Optionnel : Restaurer les artefacts de construction. Si vous souhaitez prendre en charge les constructions incrémentielles, veillez à définir également **save-artifacts**. 2. Placez la source d'application à l'endroit souhaité. 3. Construire les artefacts de l'application. 4. Installer les artefacts dans des endroits appropriés pour qu'ils fonctionnent.
run	Le script **run** exécute votre application. Ce script est nécessaire.
save-artifacts	Le script **save-artifacts** rassemble toutes les dépendances qui peuvent accélérer les processus de construction qui suivent. Ce script est facultatif. Par exemple : - Pour Ruby, **gems** installé par Bundler. - Pour Java, **.m2** contents. Ces dépendances sont rassemblées dans un fichier **tar** et transmises à la sortie standard.
usage	Le script **usage** vous permet d'informer l'utilisateur sur la manière d'utiliser correctement votre image. Ce script est facultatif.

Le scénario	Description
test/run	Le script **test/run** vous permet de créer un processus pour vérifier si l'image fonctionne correctement. Ce script est facultatif. Le flux proposé pour ce processus est le suivant 1. Construire l'image. 2. Exécutez l'image pour vérifier le script **usage**. 3. Exécutez **s2i build** pour vérifier le script **assemble**. 4. Facultatif : Exécutez à nouveau **s2i build** pour vérifier que les scripts **save-artifacts** et **assemble** sauvegardent et restaurent les artefacts. 5. Exécutez l'image pour vérifier que l'application de test fonctionne.  NOTE L'emplacement suggéré pour placer l'application de test construite par votre script **test/run** est le répertoire **test/test-app** de votre référentiel d'images.

Example S2I scripts

Les exemples de scripts S2I suivants sont écrits en Bash. Chaque exemple suppose que le contenu de **tar** est décompressé dans le répertoire **/tmp/s2i**.

assemble le scénario :

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run le scénario :

```
#!/bin/bash
```

```
# run the application
/opt/application/run.sh
```

save-artifacts le scénario :

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage le scénario :

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

Ressources complémentaires

- Tutoriel de création d'images S2I

4.4. À PROPOS DU TEST DES IMAGES SOURCE-IMAGE

En tant qu'auteur d'une image Source-to-Image (S2I), vous pouvez tester votre image S2I localement et utiliser le système de construction d'OpenShift Container Platform pour les tests automatisés et l'intégration continue.

S2I exige que les scripts **assemble** et **run** soient présents pour exécuter avec succès la construction de S2I. Le fait de fournir le script **save-artifacts** permet de réutiliser les artefacts de construction, et le fait de fournir le script **usage** permet de s'assurer que les informations d'utilisation sont imprimées sur la console lorsque quelqu'un exécute l'image du conteneur en dehors du S2I.

L'objectif du test d'une image S2I est de s'assurer que toutes les commandes décrites fonctionnent correctement, même si l'image du conteneur de base a changé ou si l'outil utilisé par les commandes a été mis à jour.

4.4.1. Comprendre les exigences en matière de tests

L'emplacement standard du script **test** est **test/run**. Ce script est invoqué par le constructeur d'images S2I de OpenShift Container Platform et peut être un simple script Bash ou un binaire Go statique.

Le script **test/run** effectue la construction de S2I, vous devez donc avoir le binaire S2I disponible dans votre site **$PATH**. Si nécessaire, suivez les instructions d'installation dans le README de S2I.

S2I combine le code source de l'application et l'image du constructeur. Pour le tester, vous avez donc besoin d'un exemple de source d'application afin de vérifier que la source se transforme avec succès en une image de conteneur exécutable. L'exemple d'application doit être simple, mais il doit permettre

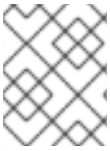
d'exercer les étapes cruciales des scripts **assemble** et **run**.

4.4.2. Générer des scripts et des outils

L'outillage S2I est fourni avec de puissants outils de génération pour accélérer le processus de création d'une nouvelle image S2I. La commande **s2i create** produit tous les scripts et outils de test S2I nécessaires avec la commande **Makefile**:

```
$ s2i create _<image nom>_ _<destination répertoire>_
```

Le script **test/run** généré doit être ajusté pour être utile, mais il constitue un bon point de départ pour commencer à développer.



NOTE

Le script **test/run** produit par la commande **s2i create** exige que les sources de l'application exemple se trouvent dans le répertoire **test/test-app**.

4.4.3. Tester localement

La manière la plus simple d'exécuter localement les tests d'image S2I est d'utiliser la version générée de **Makefile**.

Si vous n'avez pas utilisé la commande **s2i create**, vous pouvez copier le modèle suivant **Makefile** et remplacer le paramètre **IMAGE_NAME** par le nom de votre image.

Échantillon Makefile

```
IMAGE_NAME = openshift/ruby-20-centos7
CONTAINER_ENGINE := $(shell command -v podman 2> /dev/null | echo docker)

build:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME) .

.PHONY: test
test:
 ${CONTAINER_ENGINE} build -t $(IMAGE_NAME)-candidate .
 IMAGE_NAME=$(IMAGE_NAME)-candidate test/run
```

4.4.4. Flux de travail des tests de base

Le script **test** suppose que vous avez déjà créé l'image que vous souhaitez tester. Si nécessaire, construisez d'abord l'image S2I. Exécutez l'une des commandes suivantes :

- Si vous utilisez Podman, exécutez la commande suivante :

  ```
  $ podman build -t <nom_de_l'image_du_constructeur>
  ```

- Si vous utilisez Docker, exécutez la commande suivante :

  ```
  $ docker build -t <nom_de_l'image_du_constructeur>
  ```

Les étapes suivantes décrivent le flux de travail par défaut pour tester les constructeurs d'images S2I :

1. Vérifiez que le script **usage** fonctionne :

   - Si vous utilisez Podman, exécutez la commande suivante :

     ```
     $ podman run <builder_image_name> .
     ```

   - Si vous utilisez Docker, exécutez la commande suivante :

     ```
     $ docker run <builder_image_name> .
     ```

2. Construire l'image :

   ```
   $ s2i build file:///path-to-sample-app _<BUILDER_IMAGE_NAME>_
   _<OUTPUT_APPLICATION_IMAGE_NAME>_
   _<OUTPUT_APPLICATION_IMAGE_NAME>_ _YRFFGUNA-BUILDER_IMAGE_NAME>_
   ```

3. Facultatif : si vous prenez en charge **save-artifacts**, exécutez l'étape 2 une nouvelle fois pour vérifier que l'enregistrement et la restauration des artefacts fonctionnent correctement.

4. Exécuter le conteneur :

   - Si vous utilisez Podman, exécutez la commande suivante :

     ```
     $ podman run <nom_de_l'image_de_l'application_de_sortie>
     ```

   - Si vous utilisez Docker, exécutez la commande suivante :

     ```
     $ docker run <output_application_image_name>
     ```

5. Vérifiez que le conteneur fonctionne et que l'application répond.

L'exécution de ces étapes suffit généralement à déterminer si l'image du constructeur fonctionne comme prévu.

4.4.5. Utilisation d'OpenShift Container Platform pour la construction de l'image

Une fois que vous avez **Dockerfile** et les autres artefacts qui composent votre nouvelle image S2I builder, vous pouvez les mettre dans un dépôt git et utiliser OpenShift Container Platform pour construire et pousser l'image. Définissez un build Docker qui pointe vers votre dépôt.

Si votre instance OpenShift Container Platform est hébergée sur une adresse IP publique, le build peut être déclenché à chaque fois que vous poussez dans votre dépôt GitHub S2I builder image.

Vous pouvez également utiliser le site **ImageChangeTrigger** pour déclencher une reconstruction de vos applications basées sur l'image du constructeur S2I que vous avez mise à jour.

CHAPITRE 5. GESTION DES IMAGES

5.1. APERÇU DE LA GESTION DES IMAGES

Avec OpenShift Container Platform, vous pouvez interagir avec des images et mettre en place des flux d'images, en fonction de l'emplacement des registres d'images, des exigences d'authentification autour de ces registres, et de la façon dont vous voulez que vos constructions et déploiements se comportent.

5.1.1. Aperçu des images

Un flux d'images comprend un nombre quelconque d'images de conteneurs identifiées par des balises. Il présente une vue virtuelle unique des images associées, similaire à un référentiel d'images de conteneurs.

En observant un flux d'images, les constructions et les déploiements peuvent recevoir des notifications lorsque de nouvelles images sont ajoutées ou modifiées et réagir en effectuant une construction ou un déploiement, respectivement.

5.2. MARQUAGE DES IMAGES

Les sections suivantes fournissent une vue d'ensemble et des instructions pour l'utilisation des balises d'image dans le contexte des images de conteneur pour travailler avec les flux d'images OpenShift Container Platform et leurs balises.

5.2.1. Balises d'image

Une balise d'image est une étiquette appliquée à une image conteneur dans un référentiel qui distingue une image spécifique des autres images dans un flux d'images. Généralement, la balise représente une sorte de numéro de version. Par exemple, ici **:v3.11.59-2** est la balise :

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

Vous pouvez ajouter des balises supplémentaires à une image. Par exemple, une image peut se voir attribuer les balises **:v3.11.59-2** et **:latest**.

OpenShift Container Platform propose la commande **oc tag**, qui est similaire à la commande **docker tag**, mais qui opère sur les flux d'images au lieu d'agir directement sur les images.

5.2.2. Conventions relatives aux balises d'image

Les images évoluent au fil du temps et leurs balises le reflètent. En général, une balise d'image pointe toujours vers la dernière image construite.

Si le nom d'une balise contient trop d'informations, comme **v2.0.1-may-2019**, la balise pointe vers une seule révision de l'image et n'est jamais mise à jour. En utilisant les options d'élagage par défaut, une telle image n'est jamais supprimée. Dans les très grands clusters, le schéma de création de nouvelles balises pour chaque image révisée pourrait éventuellement remplir le datastore etcd avec des métadonnées de balises excédentaires pour des images qui sont dépassées depuis longtemps.

Si la balise est nommée **v2.0**, les révisions d'images sont plus probables. L'historique de la balise est donc plus long et, par conséquent, l'élagueur d'images est plus enclin à supprimer les images anciennes et inutilisées.

Bien que la convention de dénomination des balises soit laissée à votre discrétion, voici quelques exemples au format **<image_name>:<image_tag>**:

Tableau 5.1. Conventions de dénomination des balises d'image

Description	Exemple
Révision	**myimage:v2.0.1**
L'architecture	**myimage:v2.0-x86_64**
Image de base	**myimage:v1.2-centos7**
Dernière version (potentiellement instable)	**myimage:latest**
Dernière écurie	**myimage:stable**

Si vous avez besoin de dates dans les noms de balises, inspectez périodiquement les images anciennes et non prises en charge, ainsi que **istags**, et supprimez-les. Dans le cas contraire, vous risquez d'être confronté à une utilisation accrue des ressources en raison de la conservation d'anciennes images.

5.2.3. Ajouter des balises aux flux d'images

Un flux d'images dans OpenShift Container Platform comprend zéro ou plusieurs images de conteneurs identifiées par des balises.

Il existe différents types de balises. Le comportement par défaut utilise une balise **permanent**, qui pointe vers une image spécifique dans le temps. Si la balise **permanent** est utilisée et que la source change, la balise ne change pas pour la destination.

Une balise **tracking** signifie que les métadonnées de la balise de destination sont mises à jour lors de l'importation de la balise source.

Procédure

- Vous pouvez ajouter des balises à un flux d'images à l'aide de la commande **oc tag**:

  ```
  $ oc tag <source> <destination>
  ```

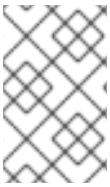
  Par exemple, pour configurer la balise **ruby** image stream **static-2.0** afin qu'elle fasse toujours référence à l'image actuelle pour la balise **ruby** image stream **2.0**:

  ```
  $ oc tag ruby:2.0 ruby:static-2.0
  ```

  Cette opération crée une nouvelle balise de flux d'images nommée **static-2.0** dans le flux d'images **ruby**. La nouvelle balise fait directement référence à l'identifiant de l'image que la balise du flux d'images **ruby:2.0** pointait au moment de l'exécution de **oc tag**, et l'image qu'elle pointe ne change jamais.

- Pour s'assurer que la balise de destination est mise à jour lorsque la balise source change, utilisez l'indicateur **--alias=true**:

```
oc tag --alias=true <source> <destination> $ oc tag --alias=true <source> <destination>
```



NOTE

Utilisez une balise de suivi pour créer des alias permanents, par exemple **latest** ou **stable**. La balise ne fonctionne correctement qu'à l'intérieur d'un seul flux d'images. Toute tentative de création d'un alias entre flux d'images produit une erreur.

- Vous pouvez également ajouter l'indicateur **--scheduled=true** pour que la balise de destination soit rafraîchie, ou réimportée, périodiquement. La période est configurée globalement au niveau du système.

- L'option **--reference** crée une balise de flux d'images qui n'est pas importée. La balise pointe vers l'emplacement de la source, de manière permanente.
  Si vous souhaitez demander à OpenShift Container Platform de toujours récupérer l'image étiquetée à partir du registre intégré, utilisez **--reference-policy=local**. Le registre utilise la fonction pull-through pour servir l'image au client. Par défaut, les blobs d'image sont mis en miroir localement par le registre. Ils peuvent donc être extraits plus rapidement la prochaine fois qu'ils sont nécessaires. Ce drapeau permet également d'extraire des registres non sécurisés sans avoir à fournir **--insecure-registry** à l'exécution du conteneur tant que le flux d'images a une annotation non sécurisée ou que la balise a une politique d'importation non sécurisée.

5.2.4. Suppression des balises dans les flux d'images

Vous pouvez supprimer les balises d'un flux d'images.

Procédure

- Pour supprimer complètement une balise d'un flux d'images, exécutez :

```
$ oc delete istag/ruby:latest
```

ou :

```
$ oc tag -d ruby:latest
```

5.2.5. Référencement d'images dans les flux d'images

Vous pouvez utiliser des balises pour référencer des images dans des flux d'images à l'aide des types de référence suivants.

Tableau 5.2. Types de référence de flux d'images

Type de référence	Description
ImageStreamTag	Un site **ImageStreamTag** est utilisé pour référencer ou récupérer une image pour un flux d'images et une balise donnés.

Type de référence	Description
ImageStreamImage	Un site **ImageStreamImage** est utilisé pour référencer ou récupérer une image pour un flux d'images et un identifiant **sha** donnés.
DockerImage	Un **DockerImage** est utilisé pour référencer ou récupérer une image pour un registre externe donné. Il utilise le nom standard de Docker **pull specification**.

Lorsque vous consultez des exemples de définitions de flux d'images, vous pouvez remarquer qu'ils contiennent des définitions de **ImageStreamTag** et des références à **DockerImage**, mais rien concernant **ImageStreamImage**.

En effet, les objets **ImageStreamImage** sont automatiquement créés dans OpenShift Container Platform lorsque vous importez ou marquez une image dans le flux d'images. Vous ne devriez jamais avoir à définir explicitement un objet **ImageStreamImage** dans une définition de flux d'images que vous utilisez pour créer des flux d'images.

Procédure

- Pour référencer une image pour un flux d'images et une balise donnés, utilisez **ImageStreamTag**:

```
<image_stream_name>:<tag>
```

- Pour référencer une image pour un flux d'images et un ID d'image **sha** donnés, utilisez **ImageStreamImage**:

```
<image_stream_name>@<id>
```

  Le site **<id>** est un identifiant immuable pour une image spécifique, également appelé "digest".

- Pour référencer ou récupérer une image pour un registre externe donné, utilisez **DockerImage**:

```
openshift/ruby-20-centos7:2.0
```

NOTE

Si aucune balise n'est spécifiée, on suppose que la balise **latest** est utilisée.

  Vous pouvez également faire référence à un registre tiers :

```
registry.redhat.io/rhel7:latest
```

  Ou une image avec un résumé :

```
centos/ruby-22-centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
```

5.3. POLITIQUE D'EXTRACTION D'IMAGES

Chaque conteneur d'un pod possède une image de conteneur. Après avoir créé une image et l'avoir transférée dans un registre, vous pouvez y faire référence dans le module.

5.3.1. Vue d'ensemble de la politique de retrait d'images

Lorsque OpenShift Container Platform crée des conteneurs, il utilise le conteneur **imagePullPolicy** pour déterminer si l'image doit être extraite avant de démarrer le conteneur. Il existe trois valeurs possibles pour **imagePullPolicy**:

Tableau 5.3. imagePullPolicy valeurs

Valeur	Description
Always	Tirez toujours l'image.
IfNotPresent	Ne tirez l'image que si elle n'existe pas déjà sur le nœud.
Never	Ne jamais tirer sur l'image.

Si le paramètre **imagePullPolicy** n'est pas spécifié, OpenShift Container Platform le définit en fonction de la balise de l'image :

1. Si la balise est **latest**, OpenShift Container Platform propose par défaut **imagePullPolicy** à **Always**.
2. Dans le cas contraire, OpenShift Container Platform propose par défaut **imagePullPolicy** à **IfNotPresent**.

5.4. UTILISATION DES SECRETS DE TIRAGE DES IMAGES

Si vous utilisez le registre d'images OpenShift et que vous tirez des flux d'images situés dans le même projet, alors votre compte de service pod devrait déjà avoir les permissions correctes et aucune action supplémentaire ne devrait être nécessaire.

Cependant, pour d'autres scénarios, tels que le référencement d'images à travers les projets OpenShift Container Platform ou à partir de registres sécurisés, des étapes de configuration supplémentaires sont nécessaires.

Vous pouvez obtenir le secret d'extraction de l'image à partir du Red Hat OpenShift Cluster Manager . Ce secret d'extraction est appelé **pullSecret**.

Vous utilisez ce secret pour vous authentifier auprès des services fournis par les autorités incluses, Quay.io et registry.redhat.io, qui servent les images de conteneurs pour les composants d'OpenShift Container Platform.

5.4.1. Permettre aux pods de référencer des images à travers les projets

Lors de l'utilisation du registre d'images OpenShift, pour permettre aux pods de **project-a** de référencer les images de **project-b**, un compte de service de **project-a** doit être lié au rôle de **system:image-puller** de **project-b**.

NOTE

Lorsque vous créez un compte de service pod ou un espace de noms, attendez que le compte de service soit provisionné avec un secret docker pull ; si vous créez un pod avant que son compte de service ne soit entièrement provisionné, le pod ne parvient pas à accéder au registre d'images OpenShift.

Procédure

1. Pour permettre aux pods de **project-a** de faire référence aux images de **project-b**, liez un compte de service de **project-a** au rôle de **system:image-puller** de **project-b**:

```
$ oc policy add-role-to-user \
    system:image-puller system:serviceaccount:project-a:default \
    --namespace=project-b
```

   Après avoir ajouté ce rôle, les pods de **project-a** qui font référence au compte de service par défaut sont en mesure d'extraire des images de **project-b**.

2. Pour autoriser l'accès à n'importe quel compte de service sur **project-a**, utilisez le groupe :

```
$ oc policy add-role-to-group \
    system:image-puller system:serviceaccounts:project-a \
    --namespace=project-b
```

5.4.2. Permettre aux pods de référencer des images provenant d'autres registres sécurisés

Le fichier **.dockercfg $HOME/.docker/config.json** pour les clients Docker est un fichier d'informations d'identification Docker qui stocke vos informations d'authentification si vous vous êtes précédemment connecté à un registre sécurisé ou non.

Pour extraire une image de conteneur sécurisée qui ne provient pas du registre d'images d'OpenShift, vous devez créer un secret d'extraction à partir de vos informations d'identification Docker et l'ajouter à votre compte de service.

Le fichier d'informations d'identification Docker et le secret d'extraction associé peuvent contenir plusieurs références au même registre, chacune avec son propre ensemble d'informations d'identification.

Exemple de fichier config.json

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
```

```
      }
    }
}
```

Exemple de secret de tirage

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

Procédure

- Si vous disposez déjà d'un fichier **.dockercfg** pour le registre sécurisé, vous pouvez créer un secret à partir de ce fichier en exécutant :

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockercfg=<path/to/.dockercfg> \
    --type=kubernetes.io/dockercfg
```

- Ou si vous avez un fichier **$HOME/.docker/config.json**:

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

- Si vous n'avez pas encore de fichier d'informations d'identification Docker pour le registre sécurisé, vous pouvez créer un secret en exécutant :

```
$ oc create secret docker-registry <pull_secret_name> \
    --docker-server=<registry_server> \
    --docker-username=<user_name> \
    --docker-password=<password> \
    --docker-email=<email>
```

- Pour utiliser un secret afin d'extraire des images pour les pods, vous devez ajouter le secret à votre compte de service. Le nom du compte de service dans cet exemple doit correspondre au nom du compte de service utilisé par le pod. Le compte de service par défaut est **default**:

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5.4.2.1. Utilisation de registres privés avec authentification déléguée

Un registre privé peut déléguer l'authentification à un service distinct. Dans ce cas, des secrets d'extraction d'image doivent être définis à la fois pour l'authentification et pour les points finaux du registre.

Procédure

1. Créez un secret pour le serveur d'authentification délégué :

```
$ oc create secret docker-registry \
    --docker-server=sso.redhat.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    redhat-connect-sso

secret/redhat-connect-sso
```

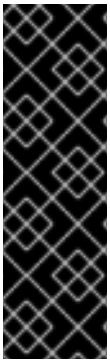
2. Créer un secret pour le registre privé :

```
$ oc create secret docker-registry \
    --docker-server=privateregistry.example.com \
    --docker-username=developer@example.com \
    --docker-password=******** \
    --docker-email=unused \
    private-registry

secret/private-registry
```

5.4.3. Mise à jour du secret d'extraction du cluster global

Vous pouvez mettre à jour le secret d'extraction global pour votre cluster en remplaçant le secret d'extraction actuel ou en ajoutant un nouveau secret d'extraction.



IMPORTANT

Pour transférer votre cluster à un autre propriétaire, vous devez d'abord initier le transfert dans OpenShift Cluster Manager Hybrid Cloud Console , puis mettre à jour le pull secret sur le cluster. La mise à jour du secret d'extraction d'un cluster sans initier le transfert dans OpenShift Cluster Manager entraîne l'arrêt du reporting des métriques de télémétrie dans OpenShift Cluster Manager.

Pour plus d'informations sur le transfert de la propriété d'un cluster , voir "Transferring cluster ownership" dans la documentation de Red Hat OpenShift Cluster Manager.

Conditions préalables

- Vous avez accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.

Procédure

1. Facultatif : Pour ajouter un nouveau secret d'extraction au secret d'extraction existant, procédez comme suit :

    a. Entrez la commande suivante pour télécharger le secret d'extraction :

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' ><pull_secret_location> 1
```

1 Indiquer le chemin d'accès au fichier de secret d'extraction.

b. Entrez la commande suivante pour ajouter le nouveau secret d'extraction :

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

1 Indiquez le nouveau registre. Vous pouvez inclure plusieurs référentiels dans le même registre, par exemple : **--registry="<registry/my-namespace/my-repository>"**.

2 Fournir les informations d'identification du nouveau registre.

3 Indiquer le chemin d'accès au fichier de secret d'extraction.

Vous pouvez également procéder à une mise à jour manuelle du fichier "pull secret".

2. Entrez la commande suivante pour mettre à jour le secret d'extraction global pour votre cluster :

```
oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=
<pull_secret_location> 1
```

1 Indiquez le chemin d'accès au nouveau fichier de secret d'extraction.

Cette mise à jour est déployée sur tous les nœuds, ce qui peut prendre un certain temps en fonction de la taille de votre cluster.

NOTE

Depuis OpenShift Container Platform 4.7.4, les modifications apportées au secret de tirage global ne déclenchent plus la vidange ou le redémarrage d'un nœud.

CHAPITRE 6. GESTION DES FLUX D'IMAGES

Les flux d'images permettent de créer et de mettre à jour des images de conteneurs de manière continue. Au fur et à mesure que des améliorations sont apportées à une image, des balises peuvent être utilisées pour attribuer de nouveaux numéros de version et assurer le suivi des modifications. Ce document décrit la gestion des flux d'images.

6.1. POURQUOI UTILISER LES IMAGESTREAMS ?

Un flux d'images et ses balises associées fournissent une abstraction pour référencer les images de conteneurs à partir d'OpenShift Container Platform. Le flux d'images et ses balises vous permettent de voir quelles images sont disponibles et de vous assurer que vous utilisez l'image spécifique dont vous avez besoin, même si l'image dans le référentiel change.

Les flux d'images ne contiennent pas de données d'images réelles, mais présentent une vue virtuelle unique d'images apparentées, à l'instar d'un référentiel d'images.

Vous pouvez configurer les constructions et les déploiements de manière à ce qu'ils surveillent un flux d'images pour recevoir des notifications lorsque de nouvelles images sont ajoutées et qu'ils réagissent en effectuant une construction ou un déploiement, respectivement.

Par exemple, si un déploiement utilise une certaine image et qu'une nouvelle version de cette image est créée, un déploiement pourrait être automatiquement effectué pour récupérer la nouvelle version de l'image.

Toutefois, si la balise de flux d'images utilisée par le déploiement ou la construction n'est pas mise à jour, même si l'image du conteneur dans le registre d'images du conteneur est mise à jour, la construction ou le déploiement continue d'utiliser l'image précédente, vraisemblablement connue et bonne.

Les images sources peuvent être stockées dans l'un des endroits suivants :

- Le registre intégré d'OpenShift Container Platform.
- Un registre externe, par exemple registry.redhat.io ou quay.io.
- Autres flux d'images dans le cluster OpenShift Container Platform.

Lorsque vous définissez un objet qui fait référence à une balise de flux d'images, comme une configuration de construction ou de déploiement, vous pointez vers une balise de flux d'images et non vers le référentiel. Lorsque vous construisez ou déployez votre application, OpenShift Container Platform interroge le référentiel à l'aide de la balise de flux d'images pour localiser l'ID associé à l'image et utilise cette image exacte.

Les métadonnées du flux d'images sont stockées dans l'instance etcd avec d'autres informations sur le cluster.

L'utilisation de flux d'images présente plusieurs avantages importants :

- Vous pouvez baliser, revenir en arrière et traiter rapidement les images, sans avoir à les repousser à l'aide de la ligne de commande.
- Vous pouvez déclencher des constructions et des déploiements lorsqu'une nouvelle image est poussée vers le registre. OpenShift Container Platform dispose également de déclencheurs génériques pour d'autres ressources, telles que les objets Kubernetes.
- Vous pouvez marquer une balise pour qu'elle soit réimportée périodiquement. Si l'image source

a changé, ce changement est pris en compte et reflété dans le flux d'images, ce qui déclenche le flux de construction ou de déploiement, en fonction de la configuration de la construction ou du déploiement.

- Vous pouvez partager des images à l'aide d'un contrôle d'accès précis et les distribuer rapidement à vos équipes.
- Si l'image source change, la balise du flux d'images pointe toujours vers une version connue de l'image, ce qui garantit que votre application n'est pas interrompue de manière inattendue.
- Vous pouvez configurer la sécurité concernant la visualisation et l'utilisation des images au moyen d'autorisations sur les objets du flux d'images.
- Les utilisateurs qui n'ont pas l'autorisation de lire ou de lister des images au niveau du cluster peuvent toujours récupérer les images marquées dans un projet à l'aide des flux d'images.

6.2. CONFIGURATION DES FLUX D'IMAGES

Un fichier objet **ImageStream** contient les éléments suivants.

Définition de l'objet Imagestream

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample (2)
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d (3)
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 (4)
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest (5)
```

(1) Le nom du flux d'images.

(2) Chemin du dépôt Docker où les nouvelles images peuvent être poussées pour les ajouter ou les mettre à jour dans ce flux d'images.

3. L'identifiant SHA auquel ce tag de flux d'images fait actuellement référence. Les ressources qui font référence à cette balise de flux d'images utilisent cet identifiant.

4. L'identifiant SHA que ce tag de flux d'images a précédemment référencé. Peut être utilisé pour revenir à une image plus ancienne.

5. Nom de la balise du flux d'images.

6.3. FLUX D'IMAGES IMAGES

Une image de flux d'images pointe à l'intérieur d'un flux d'images vers un identifiant d'image particulier.

Les images de flux d'images vous permettent de récupérer des métadonnées sur une image à partir d'un flux d'images particulier où elle est étiquetée.

Les objets d'image de flux d'images sont automatiquement créés dans OpenShift Container Platform chaque fois que vous importez ou marquez une image dans le flux d'images. Vous ne devriez jamais avoir à définir explicitement un objet image de flux d'images dans une définition de flux d'images que vous utilisez pour créer des flux d'images.

L'image du flux d'images se compose du nom du flux d'images et de l'ID de l'image du référentiel, délimités par le signe **@**:

```
<image-stream-name>@<imageid>
```

Pour faire référence à l'image dans l'exemple de l'objet **ImageStream**, l'image du flux d'images se présente comme suit :

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.4. BALISES DE FLUX D'IMAGES

Une balise de flux d'images est un pointeur nommé vers une image dans un flux d'images. Il est abrégé en **istag**. Une balise de flux d'images est utilisée pour référencer ou récupérer une image pour un flux d'images et une balise donnés.

Les balises de flux d'images peuvent faire référence à n'importe quelle image locale ou gérée en externe. Elles contiennent un historique des images représenté sous la forme d'une pile de toutes les images vers lesquelles la balise a pointé. Chaque fois qu'une image nouvelle ou existante est étiquetée sous une balise de flux d'images particulière, elle est placée en première position dans la pile de l'historique. L'image qui occupait précédemment la première position est disponible en deuxième position. Cela permet de revenir facilement en arrière pour que les balises pointent à nouveau vers des images historiques.

La balise de flux d'images suivante provient d'un objet **ImageStream**:

Balise de flux d'images avec deux images dans son historique

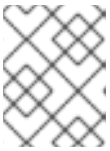
```
tags:
- items:
  - created: 2017-09-02T10:15:09Z
    dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
```

```
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
    - created: 2017-09-01T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
    tag: latest
```

Les balises de flux d'images peuvent être des balises permanentes ou des balises de suivi.

- Les balises permanentes sont des balises spécifiques à une version qui renvoient à une version particulière d'une image, telle que Python 3.5.

- Les balises de suivi sont des balises de référence qui suivent une autre balise de flux d'images et peuvent être mises à jour pour changer l'image qu'elles suivent, comme un lien symbolique. La compatibilité ascendante de ces nouveaux niveaux n'est pas garantie.
  Par exemple, les balises de flux d'images **latest** fournies avec OpenShift Container Platform sont des balises de suivi. Cela signifie que les consommateurs de la balise de flux d'images **latest** sont mis à jour au niveau le plus récent du cadre fourni par l'image lorsqu'un nouveau niveau est disponible. Une balise de flux d'images **latest** vers **v3.10** peut être changée en **v3.11** à tout moment. Il est important de savoir que ces balises de flux d'images **latest** se comportent différemment de la balise Docker **latest**. La balise de flux d'images **latest**, dans ce cas, ne pointe pas vers la dernière image du référentiel Docker. Il pointe vers une autre balise de flux d'images, qui peut ne pas être la dernière version d'une image. Par exemple, si la balise de flux d'images **latest** pointe vers **v3.10** d'une image, lorsque la version **3.11** est publiée, la balise **latest** n'est pas automatiquement mise à jour vers **v3.11** et reste à **v3.10** jusqu'à ce qu'elle soit manuellement mise à jour pour pointer vers une balise de flux d'images **v3.11**.



NOTE

Les balises de suivi sont limitées à un seul flux d'images et ne peuvent pas faire référence à d'autres flux d'images.

Vous pouvez créer vos propres balises de flux d'images en fonction de vos besoins.

La balise de flux d'images est composée du nom du flux d'images et d'une balise, séparés par deux points :

```
<imagestream nom>:<tag>
```

Par exemple, pour faire référence à l'image **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** dans l'exemple d'objet **ImageStream**, la balise de flux d'images serait la suivante :

```
origin-ruby-sample:latest
```

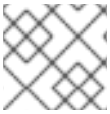
6.5. DÉCLENCHEURS DE CHANGEMENT DE FLUX D'IMAGES

Les déclencheurs de flux d'images permettent à vos constructions et déploiements d'être automatiquement invoqués lorsqu'une nouvelle version d'une image en amont est disponible.

Par exemple, les constructions et les déploiements peuvent être lancés automatiquement lorsqu'une balise de flux d'images est modifiée. Pour ce faire, il suffit de surveiller cette balise de flux d'images et de notifier la construction ou le déploiement lorsqu'une modification est détectée.

6.6. CARTOGRAPHIE DES FLUX D'IMAGES

Lorsque le registre intégré reçoit une nouvelle image, il crée et envoie un flux d'images à OpenShift Container Platform, en fournissant le projet, le nom, la balise et les métadonnées de l'image.



NOTE

La configuration des mappages de flux d'images est une fonction avancée.

Ces informations sont utilisées pour créer une nouvelle image, si elle n'existe pas déjà, et pour marquer l'image dans le flux d'images. OpenShift Container Platform stocke des métadonnées complètes sur chaque image, telles que les commandes, le point d'entrée et les variables d'environnement. Les images dans OpenShift Container Platform sont immuables et la longueur maximale du nom est de 63 caractères.

L'exemple suivant de mappage de flux d'images permet à une image d'être étiquetée comme **test/origin-ruby-sample:latest**:

Définition de l'objet de mappage du flux d'images

```
apiVersion: image.openshift.io/v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
```

```
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Labels:
        build-date: 2015-12-23
        io.k8s.description: Platform for building and running Ruby 2.2 applications
        io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
        io.openshift.build.commit.author: Ben Parees <bparees@users.noreply.github.com>
        io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
        io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
        io.openshift.build.commit.message: 'Merge pull request #51 from php-coder/fix_url_and_sti'
        io.openshift.build.commit.ref: master
        io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
        io.openshift.build.source-location: https://github.com/openshift/ruby-hello-world.git
        io.openshift.builder-base-version: 8d95148
        io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
        io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
        io.openshift.tags: builder,ruby,ruby22
        io.s2i.scripts-url: image:///usr/libexec/s2i
        license: GPLv2
        name: CentOS Base Image
        vendor: CentOS
      User: "1001"
      WorkingDir: /opt/app-root/src
    Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
    ContainerConfig:
      AttachStdout: true
      Cmd:
      - /bin/sh
      - -c
      - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
      - STI_SCRIPTS_URL=image:///usr/libexec/s2i
```

```
      - STI_SCRIPTS_PATH=/usr/libexec/s2i
      - HOME=/opt/app-root/src
      - BASH_ENV=/opt/app-root/etc/scl_enable
      - ENV=/opt/app-root/etc/scl_enable
      - PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
      - RUBY_VERSION=2.2
      ExposedPorts:
        8080/tcp: {}
      Hostname: ruby-sample-build-1-build
      Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
      OpenStdin: true
      StdinOnce: true
      User: "1001"
      WorkingDir: /opt/app-root/src
    Created: 2016-01-29T13:40:00Z
    DockerVersion: 1.8.2.fc21
    Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
    Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
    Size: 441976279
    apiVersion: "1.0"
    kind: DockerImage
  dockerImageMetadataVersion: "1.0"
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.7. TRAVAILLER AVEC DES FLUX D'IMAGES

Les sections suivantes décrivent comment utiliser les flux d'images et les balises de flux d'images.

6.7.1. Obtenir des informations sur les flux d'images

Vous pouvez obtenir des informations générales sur le flux d'images et des informations détaillées sur toutes les balises vers lesquelles il pointe.

Procédure

- Obtenir des informations générales sur le flux d'images et des informations détaillées sur toutes les balises vers lesquelles il pointe :

  ```
  $ oc describe is/<image-nom>
  ```

  Par exemple :

  ```
  $ oc describe is/python
  ```

  Exemple de sortie

  ```
  Name:   python
  Namespace:  default
  Created:  About a minute ago
  Labels:   <none>
  Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
  ```

```
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   1

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago
```

- Obtenir toutes les informations disponibles sur une étiquette de flux d'images particulière :

```
$ oc describe istag/<image-stream>:<tag-name>
```

Par exemple :

```
$ oc describe istag/python:latest
```

Exemple de sortie

```
Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name:  sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created: 2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author:  <none>
Arch:  amd64
Entrypoint: container-entrypoint
Command: /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
User:  1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```

NOTE

Le nombre d'informations émises est supérieur au nombre d'informations affichées.

6.7.2. Ajout de balises à un flux d'images

Vous pouvez ajouter des balises supplémentaires aux flux d'images.

Procédure

- Ajouter une balise qui pointe vers l'une des balises existantes en utilisant la commande `oc tag` :

```
$ oc tag <image-name:tag1> <image-name:tag2>
```

Par exemple :

```
$ oc tag python:3.5 python:latest
```

Exemple de sortie

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

- Confirmez que le flux d'images comporte deux balises, l'une, **3.5**, pointant vers l'image du conteneur externe et une autre, **latest**, pointant vers la même image parce qu'elle a été créée sur la base de la première balise.

```
$ oc describe is/python
```

Exemple de sortie

```
Name:   python
Namespace:  default
Created:  5 minutes ago
Labels:   <none>
Annotations:  openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup:  local=false
Unique Images:  1
Tags:   2

latest
  tagged from
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      About a minute ago

3.5
  tagged from centos/python-35-centos7

  * centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
      5 minutes ago
```

6.7.3. Ajout de balises pour une image externe

Vous pouvez ajouter des balises pour les images externes.

Procédure

- Ajouter des balises pointant vers des images internes ou externes, en utilisant la commande **oc tag** pour toutes les opérations liées aux balises :

```
$ oc tag <repository/image> <image-name:tag>
```

Par exemple, cette commande associe l'image **docker.io/python:3.6.0** à la balise **3.6** dans le flux d'images **python**.

```
$ oc tag docker.io/python:3.6.0 python:3.6
```

Exemple de sortie

```
Tag python:3.6 set to docker.io/python:3.6.0.
```

Si l'image externe est sécurisée, vous devez créer un secret avec des informations d'identification pour accéder à ce registre.

6.7.4. Mise à jour des balises de flux d'images

Vous pouvez mettre à jour une balise pour refléter une autre balise dans un flux d'images.

Procédure

- Mettre à jour une balise :

```
$ oc tag <image-name:tag> <image-name:latest>
```

Par exemple, le texte suivant met à jour la balise **latest** pour refléter la balise **3.6** dans un flux d'images :

```
$ oc tag python:3.6 python:latest
```

Exemple de sortie

```
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

6.7.5. Suppression des balises de flux d'images

Vous pouvez supprimer les anciennes étiquettes d'un flux d'images.

Procédure

- Supprime les anciennes étiquettes d'un flux d'images :

```
oc tag -d <image-name:tag> $ oc tag -d <image-name:tag>
```

Par exemple :

```
$ oc tag -d python:3.5
```

Exemple de sortie

```
Deleted tag default/python:3.5.
```

Voir Suppression des balises de flux d'images obsolètes de Cl uster Samples Operator pour plus d'informations sur la façon dont Cluster Samples Operator gère les balises de flux d'images obsolètes.

6.7.6. Configuration de l'importation périodique de balises de flux d'images

Lorsque vous travaillez avec un registre externe d'images de conteneurs, pour réimporter périodiquement une image, par exemple pour obtenir les dernières mises à jour de sécurité, vous pouvez utiliser le drapeau **--scheduled**.

Procédure

1. Planifier l'importation d'images :

```
$ oc tag <repository/image> <image-name:tag> --scheduled
```

Par exemple :

```
$ oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

Exemple de sortie

```
Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

Cette commande permet à OpenShift Container Platform de mettre à jour périodiquement cette balise de flux d'images. Cette période est un paramètre à l'échelle du cluster, fixé par défaut à 15 minutes.

2. Supprimez la vérification périodique, exécutez à nouveau la commande ci-dessus mais omettez le drapeau **--scheduled**. Cela rétablira le comportement par défaut.

```
$ oc tag <repositiory/image> <image-name:tag>
```

6.8. IMPORTATION D'IMAGES ET DE FLUX D'IMAGES À PARTIR DE REGISTRES PRIVÉS

Un flux d'images peut être configuré pour importer des métadonnées de balises et d'images à partir de registres d'images privés nécessitant une authentification. Cette procédure s'applique si vous modifiez le registre utilisé par l'opérateur d'échantillonnage de cluster pour extraire du contenu et le remplacer par autre chose que registry.redhat.io.

NOTE

Lors de l'importation à partir de registres non sécurisés ou sécurisés, l'URL du registre définie dans le secret doit inclure le suffixe du port **:80**, sinon le secret n'est pas utilisé lors de la tentative d'importation à partir du registre.

Procédure

1. Vous devez créer un objet **secret** utilisé pour stocker vos informations d'identification en entrant la commande suivante :

```
$ oc create secret generic <secret_name> --from-file=.dockerconfigjson=
<file_absolute_path> --type=kubernetes.io/dockerconfigjson
```

2. Une fois le secret configuré, créez le nouveau flux d'images ou entrez la commande **oc import-image**:

```
$ oc import-image <imagestreamtag> --from=<image> --confirm
```

Au cours du processus d'importation, OpenShift Container Platform récupère les secrets et les fournit à la partie distante.

6.8.1. Permettre aux pods de référencer des images provenant d'autres registres sécurisés

Le fichier **.dockercfg $HOME/.docker/config.json** pour les clients Docker est un fichier d'informations d'identification Docker qui stocke vos informations d'authentification si vous vous êtes précédemment connecté à un registre sécurisé ou non.

Pour extraire une image de conteneur sécurisée qui ne provient pas du registre d'images d'OpenShift, vous devez créer un secret d'extraction à partir de vos informations d'identification Docker et l'ajouter à votre compte de service.

Le fichier d'informations d'identification Docker et le secret d'extraction associé peuvent contenir plusieurs références au même registre, chacune avec son propre ensemble d'informations d'identification.

Exemple de fichier config.json

```
{
  "auths":{
    "cloud.openshift.com":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}
```

Exemple de secret de tirage

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
```

```
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

Procédure

- Si vous disposez déjà d'un fichier **.dockercfg** pour le registre sécurisé, vous pouvez créer un secret à partir de ce fichier en exécutant :

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockercfg=<path/to/.dockercfg> \
    --type=kubernetes.io/dockercfg
```

- Ou si vous avez un fichier **$HOME/.docker/config.json**:

```
$ oc create secret generic <pull_secret_name> \
    --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
    --type=kubernetes.io/dockerconfigjson
```

- Si vous n'avez pas encore de fichier d'informations d'identification Docker pour le registre sécurisé, vous pouvez créer un secret en exécutant :

```
$ oc create secret docker-registry <pull_secret_name> \
    --docker-server=<registry_server> \
    --docker-username=<user_name> \
    --docker-password=<password> \
    --docker-email=<email>
```

- Pour utiliser un secret afin d'extraire des images pour les pods, vous devez ajouter le secret à votre compte de service. Le nom du compte de service dans cet exemple doit correspondre au nom du compte de service utilisé par le pod. Le compte de service par défaut est **default**:

```
$ oc secrets link default <pull_secret_name> --for=pull
```

6.9. IMPORTATION D'UNE LISTE DE MANIFESTES PAR IMAGESTREAMIMPORT

Vous pouvez utiliser la ressource **ImageStreamImport** pour trouver et importer dans le cluster des manifestes d'images provenant d'autres registres d'images de conteneurs. Il est possible d'importer des images individuelles ou un référentiel d'images complet.

La procédure suivante permet d'importer une liste de manifestes via l'objet **ImageStreamImport** avec la valeur **importMode**.

Procédure

1. Créez un fichier YAML **ImageStreamImport** et définissez le paramètre **importMode** sur **PreserveOriginal** pour les balises que vous allez importer en tant que liste de manifestes :

```
apiVersion: image.openshift.io/v1
```

```
kind: ImageStreamImport
metadata:
  name: app
  namespace: myapp
spec:
  import: true
  images:
  - from:
      kind: DockerImage
      name: <registry>/<user_name>/<image_name>
    to:
      name: latest
    referencePolicy:
      type: Source
    importPolicy:
      importMode: "PreserveOriginal"
```

2. Créez le site **ImageStreamImport** en exécutant la commande suivante :

```
oc create -f <your_imagestreamimport.yaml>
```

6.9.1. champs de configuration importMode

Le tableau suivant décrit les champs de configuration disponibles pour la valeur **importMode**:

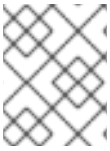
Paramètres	Description
Legacy	La valeur par défaut de **importMode**. Lorsqu'elle est active, la liste des manifestes est rejetée et un seul sous-manifeste est importé. La plate-forme est choisie dans l'ordre de priorité suivant : 1. Annotations d'étiquettes 2. Architecture du plan de contrôle 3. Linux/AMD64 4. Le premier manifeste de la liste
PreserveOriginal	Lorsqu'il est actif, le manifeste original est préservé. Pour les listes de manifestes, la liste de manifestes et tous ses sous-manifestes sont importés.

CHAPITRE 7. UTILISATION DE FLUX D'IMAGES AVEC DES RESSOURCES KUBERNETES

Les flux d'images, étant des ressources natives d'OpenShift Container Platform, fonctionnent d'emblée avec toutes les autres ressources natives disponibles dans OpenShift Container Platform, telles que les builds ou les déploiements. Il est également possible de les faire fonctionner avec les ressources natives de Kubernetes, telles que les travaux, les contrôleurs de réplication, les ensembles de répliques ou les déploiements Kubernetes.

7.1. ACTIVATION DES FLUX D'IMAGES AVEC LES RESSOURCES KUBERNETES

Lorsque vous utilisez des flux d'images avec des ressources Kubernetes, vous ne pouvez référencer que des flux d'images qui résident dans le même projet que la ressource. La référence à un flux d'images doit être composée d'une seule valeur de segment, par exemple **ruby:2.5**, où **ruby** est le nom d'un flux d'images ayant une balise nommée **2.5** et résidant dans le même projet que la ressource faisant la référence.



NOTE

Cette fonctionnalité ne peut pas être utilisée dans l'espace de noms **default**, ni dans aucun espace de noms **openshift-** ou **kube-**.

Il existe deux façons d'activer les flux d'images avec les ressources Kubernetes :

- Activation de la résolution de flux d'images sur une ressource spécifique. Cela permet à cette seule ressource d'utiliser le nom du flux d'images dans le champ image.
- Activation de la résolution d'un flux d'images sur un flux d'images. Cela permet à toutes les ressources pointant vers ce flux d'images de l'utiliser dans le champ de l'image.

Procédure

Vous pouvez utiliser **oc set image-lookup** pour activer la résolution de flux d'images sur une ressource spécifique ou la résolution de flux d'images sur un flux d'images.

1. Pour permettre à toutes les ressources de faire référence au flux d'images nommé **mysql**, entrez la commande suivante :

```
$ oc set image-lookup mysql
```

   Le champ **Imagestream.spec.lookupPolicy.local** prend la valeur "true".

   Flux d'images avec recherche d'images activée

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/display-name: mysql
  name: mysql
  namespace: myproject
```

```
spec:
  lookupPolicy:
    local: true
```

Lorsque cette option est activée, le comportement est activé pour toutes les balises du flux d'images.

2. Vous pouvez ensuite interroger les flux d'images et vérifier si l'option est activée :

```
$ oc set image-lookup imagestream --list
```

Vous pouvez activer la recherche d'images sur une ressource spécifique.

- Pour permettre au déploiement Kubernetes nommé **mysql** d'utiliser des flux d'images, exécutez la commande suivante :

```
$ oc set image-lookup deploy/mysql
```

Ceci définit l'annotation **alpha.image.policy.openshift.io/resolve-names** sur le déploiement.

Déploiement avec recherche d'image activée

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: myproject
spec:
  replicas: 1
  template:
    metadata:
      annotations:
        alpha.image.policy.openshift.io/resolve-names: '*'
    spec:
      containers:
      - image: mysql:latest
        imagePullPolicy: Always
        name: mysql
```

Vous pouvez désactiver la recherche d'images.

- Pour désactiver la recherche d'images, passez **--enabled=false**:

```
$ oc set image-lookup deploy/mysql --enabled=false
```

CHAPITRE 8. DÉCLENCHEMENT DE MISES À JOUR EN CAS DE MODIFICATION DU FLUX D'IMAGES

Lorsqu'un tag de flux d'images est mis à jour pour pointer vers une nouvelle image, OpenShift Container Platform peut automatiquement prendre des mesures pour déployer la nouvelle image vers les ressources qui utilisaient l'ancienne image. Vous configurez ce comportement de différentes manières en fonction du type de ressource qui fait référence à la balise de flux d'images.

8.1. RESSOURCES OPENSHIFT CONTAINER PLATFORM

Les configurations de déploiement et de construction d'OpenShift Container Platform peuvent être automatiquement déclenchées par des changements dans les balises de flux d'images. L'action déclenchée peut être exécutée en utilisant la nouvelle valeur de l'image référencée par la balise de flux d'images mise à jour.

8.2. DÉCLENCHEMENT DES RESSOURCES KUBERNETES

Les ressources Kubernetes n'ont pas de champs pour le déclenchement, contrairement aux configurations de déploiement et de construction, qui incluent dans leur définition d'API un ensemble de champs pour contrôler les déclencheurs. Au lieu de cela, vous pouvez utiliser des annotations dans OpenShift Container Platform pour demander un déclenchement.

L'annotation est définie comme suit :

```
Key: image.openshift.io/triggers
Value:
[
 {
  "from": {
   "kind": "ImageStreamTag", 1
   "name": "example:latest", 2
   "namespace": "myapp" 3
  },
  "fieldPath": "spec.template.spec.containers[?(@.name==\"web\")].image", 4
  "paused": false 5
 },
 ...
]
```

1. Requis : **kind** est la ressource à partir de laquelle le déclencheur doit être **ImageStreamTag**.
2. Obligatoire : **name** doit être le nom d'une balise de flux d'images.
3. Facultatif : **namespace** indique par défaut l'espace de noms de l'objet.
4. Obligatoire : **fieldPath** est le chemin JSON à modifier. Ce champ est limité et n'accepte qu'une expression de chemin JSON qui correspond précisément à un conteneur par ID ou index. Pour les pods, le chemin JSON est "spec.containers[?(@.name='web')].image".
5. Facultatif : **paused** indique si le déclencheur est mis en pause ou non, la valeur par défaut étant **false**. Définissez **paused** à **true** pour désactiver temporairement ce déclencheur.

Lorsqu'une des ressources principales de Kubernetes contient à la fois un modèle de pod et cette annotation, OpenShift Container Platform tente de mettre à jour l'objet en utilisant l'image actuellement associée à la balise de flux d'images qui est référencée par le déclencheur. La mise à jour est effectuée par rapport à l'adresse **fieldPath** spécifiée.

Voici quelques exemples de ressources de base de Kubernetes qui peuvent contenir à la fois un modèle de pod et une annotation :

- **CronJobs**
- **Deployments**
- **StatefulSets**
- **DaemonSets**
- **Jobs**
- **ReplicationControllers**
- **Pods**

8.3. DÉFINITION DU DÉCLENCHEUR D'IMAGE SUR LES RESSOURCES KUBERNETES

Lorsque vous ajoutez un déclencheur d'image aux déploiements, vous pouvez utiliser la commande **oc set triggers**. Par exemple, l'exemple de commande de cette procédure ajoute un déclencheur de changement d'image au déploiement nommé **example** de sorte que lorsque la balise de flux d'image **example:latest** est mise à jour, le conteneur **web** à l'intérieur du déploiement est mis à jour avec la nouvelle valeur d'image. Cette commande définit l'annotation **image.openshift.io/triggers** correcte sur la ressource de déploiement.

Procédure

- Déclenchez les ressources Kubernetes en entrant la commande **oc set triggers**:

```
$ oc set triggers deploy/example --from-image=example:latest -c web
```

À moins que le déploiement ne soit interrompu, cette mise à jour du modèle de pod entraîne automatiquement un déploiement avec la nouvelle valeur de l'image.

CHAPITRE 9. RESSOURCES POUR LA CONFIGURATION DES IMAGES

La procédure suivante permet de configurer les registres d'images.

9.1. PARAMÈTRES DE CONFIGURATION DU CONTRÔLEUR D'IMAGES

La ressource **image.config.openshift.io/cluster** contient des informations à l'échelle du cluster sur la manière de gérer les images. Le nom canonique, et le seul valide, est **cluster**. La ressource **spec** offre les paramètres de configuration suivants.

NOTE

Les paramètres tels que **DisableScheduledImport**, **MaxImagesBulkImportedPerRepository**, **MaxScheduledImportsPerMinute**, **ScheduledImageImportMinimumIntervalSeconds**, **InternalRegistryHostname** ne sont pas configurables.

Paramètres	Description
allowedRegistriesForImport	Limite les registres d'images de conteneurs à partir desquels les utilisateurs normaux peuvent importer des images. Définissez cette liste en fonction des registres dont vous pensez qu'ils contiennent des images valides et à partir desquels vous souhaitez que les applications puissent importer des images. Les utilisateurs autorisés à créer des images ou **ImageStreamMappings** à partir de l'API ne sont pas concernés par cette politique. En général, seuls les administrateurs de clusters disposent des autorisations appropriées. Chaque élément de cette liste contient un emplacement du registre spécifié par le nom de domaine du registre. **domainName**: Spécifie un nom de domaine pour le registre. Si le registre utilise un port non standard **80** ou **443**, le port doit également être inclus dans le nom de domaine. **insecure**: Insécurisé indique si le registre est sécurisé ou non. Par défaut, si rien d'autre n'est spécifié, le registre est supposé être sécurisé.
additionalTrustedCA	Une référence à une carte de configuration contenant des autorités de certification supplémentaires qui doivent être approuvées pendant **image stream import**, **pod image pull**, **openshift-image-registry pullthrough**, et les constructions. L'espace de noms de cette carte de configuration est **openshift-config**. Le format de la carte de configuration consiste à utiliser le nom d'hôte du registre comme clé et le certificat encodé PEM comme valeur, pour chaque autorité de certification de registre supplémentaire à laquelle il faut faire confiance.

Paramètres	Description
externalRegistryHostnames	Fournit les noms d'hôtes pour le registre d'images externe par défaut. Le nom d'hôte externe ne doit être défini que lorsque le registre d'images est exposé à l'extérieur. La première valeur est utilisée dans le champ **publicDockerImageRepository** des flux d'images. La valeur doit être au format **hostname[:port]**.
registrySources	Contient la configuration qui détermine comment le runtime du conteneur doit traiter les registres individuels lors de l'accès aux images pour les builds et les pods. Par exemple, l'autorisation ou non d'un accès non sécurisé. Il ne contient pas de configuration pour le registre interne du cluster. **insecureRegistries**: Les registres qui n'ont pas de certificat TLS valide ou qui ne prennent en charge que les connexions HTTP. Pour spécifier tous les sous-domaines, ajoutez le caractère générique astérisque (*) comme préfixe au nom de domaine. Par exemple, ***.example.com**. Vous pouvez spécifier un dépôt individuel au sein d'un registre. Par exemple : **reg1.io/myrepo/myapp:latest**. **blockedRegistries**: Registres pour lesquels les actions d'extraction et de poussée d'images sont refusées. Pour spécifier tous les sous-domaines, ajoutez le caractère générique astérisque (*) comme préfixe au nom de domaine. Par exemple, ***.example.com**. Vous pouvez spécifier un dépôt individuel au sein d'un registre. Par exemple : **reg1.io/myrepo/myapp:latest**. Tous les autres registres sont autorisés. **allowedRegistries**: Registres pour lesquels les actions d'extraction et de poussée d'images sont autorisées. Pour spécifier tous les sous-domaines, ajoutez le caractère générique astérisque (*) comme préfixe au nom de domaine. Par exemple, ***.example.com**. Vous pouvez spécifier un dépôt individuel au sein d'un registre. Par exemple : **reg1.io/myrepo/myapp:latest**. Tous les autres registres sont bloqués. **containerRuntimeSearchRegistries**: Registres pour lesquels les actions de traction et de poussée d'images sont autorisées à l'aide de noms courts d'images. Tous les autres registres sont bloqués. Il est possible de définir soit **blockedRegistries**, soit **allowedRegistries**, mais pas les deux.

AVERTISSEMENT

Lorsque le paramètre **allowedRegistries** est défini, tous les registres, y compris les registres **registry.redhat.io** et **quay.io** et le registre d'image OpenShift par défaut, sont bloqués sauf s'ils sont explicitement listés. Lors de l'utilisation du paramètre, pour éviter un échec du pod, ajoutez tous les registres, y compris les registres **registry.redhat.io** et **quay.io** et la liste **internalRegistryHostname** à **allowedRegistries**, car ils sont requis par les images de charge utile dans votre environnement. Pour les clusters déconnectés, les registres miroirs doivent également être ajoutés.

Le champ **status** de la ressource **image.config.openshift.io/cluster** contient les valeurs observées de la grappe.

Paramètres	Description
internalRegistryHostname	Défini par l'opérateur de registre d'images, qui contrôle le registre d'images **internalRegistryHostname**. Il définit le nom d'hôte pour le registre d'images par défaut d'OpenShift. La valeur doit être au format **hostname[:port]**. Pour une compatibilité ascendante, vous pouvez toujours utiliser la variable d'environnement **OPENSHIFT_DEFAULT_REGISTRY**, mais ce paramètre remplace la variable d'environnement.
externalRegistryHostnames	Défini par l'opérateur du registre d'images, il fournit les noms d'hôtes externes pour le registre d'images lorsqu'il est exposé à l'extérieur. La première valeur est utilisée dans le champ **publicDockerImageRepository** des flux d'images. Les valeurs doivent être au format **hostname[:port]**.

9.2. CONFIGURATION DES PARAMÈTRES DU REGISTRE DES IMAGES

Vous pouvez configurer les paramètres du registre d'images en modifiant la ressource personnalisée (CR) **image.config.openshift.io/cluster**. Lorsque les modifications apportées au registre sont appliquées à la CR **image.config.openshift.io/cluster**, l'opérateur de configuration de la machine (MCO) effectue les actions séquentielles suivantes :

1. Cordons du nœud
2. Applique les modifications en redémarrant CRI-O
3. Uncordons le nœud

NOTE

Le MCO ne redémarre pas les nœuds lorsqu'il détecte des changements.

Procédure

1. Modifiez la ressource personnalisée **image.config.openshift.io/cluster**:

```
$ oc edit image.config.openshift.io/cluster
```

Voici un exemple de **image.config.openshift.io/cluster** CR :

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
```

```
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: 2
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. **Image**: Contient des informations à l'échelle du cluster sur la manière de gérer les images. Le nom canonique, et le seul valide, est **cluster**.

2. **allowedRegistriesForImport**: Limite les registres d'images de conteneurs à partir desquels les utilisateurs normaux peuvent importer des images. Définissez cette liste en fonction des registres dont vous pensez qu'ils contiennent des images valides et à partir desquels vous souhaitez que les applications puissent importer des images. Les utilisateurs autorisés à créer des images ou **ImageStreamMappings** à partir de l'API ne sont pas concernés par cette politique. En général, seuls les administrateurs de clusters disposent des autorisations appropriées.

3. **additionalTrustedCA**: Une référence à une carte de configuration contenant des autorités de certification (AC) supplémentaires qui sont approuvées lors de l'importation de flux d'images, de l'extraction d'images de pods, de l'extraction de **openshift-image-registry** et des constructions. L'espace de noms pour cette carte de configuration est **openshift-config**. Le format de la carte de configuration consiste à utiliser le nom d'hôte du registre comme clé et le certificat PEM comme valeur, pour chaque autorité de certification de registre supplémentaire à laquelle il faut faire confiance.

4. **registrySources**: Contient la configuration qui détermine si le runtime du conteneur autorise ou bloque les registres individuels lors de l'accès aux images pour les builds et les pods. Le paramètre **allowedRegistries** ou le paramètre **blockedRegistries** peut être défini, mais pas les deux. Vous pouvez également définir s'il faut ou non autoriser l'accès aux registres non sécurisés ou aux registres qui autorisent les registres utilisant des noms courts d'images. Cet exemple utilise le paramètre **allowedRegistries**, qui définit les registres dont l'utilisation est autorisée. Le registre non sécurisé **insecure.com** est également autorisé. Le paramètre **registrySources** ne contient pas de configuration pour le registre interne du cluster.

NOTE

Lorsque le paramètre **allowedRegistries** est défini, tous les registres, y compris les registres registry.redhat.io et quay.io et le registre d'image OpenShift par défaut, sont bloqués sauf s'ils sont explicitement répertoriés. Si vous utilisez ce paramètre, pour éviter une défaillance du pod, vous devez ajouter les registres **registry.redhat.io** et **quay.io** et la liste **internalRegistryHostname** à **allowedRegistries**, car ils sont requis par les images de charge utile dans votre environnement. N'ajoutez pas les registres **registry.redhat.io** et **quay.io** à la liste **blockedRegistries**.

En utilisant les paramètres **allowedRegistries**, **blockedRegistries** ou **insecureRegistries**, vous pouvez spécifier un référentiel individuel au sein d'un registre. Par exemple : **reg1.io/myrepo/myapp:latest**.

Les registres externes non sécurisés doivent être évités afin de réduire les risques éventuels pour la sécurité.

2. Pour vérifier que les modifications ont été appliquées, dressez la liste de vos nœuds :

```
$ oc get nodes
```

Exemple de sortie

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker                 65m   v1.25.4+77bec7a
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.25.4+77bec7a
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.25.4+77bec7a
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.25.4+77bec7a
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.25.4+77bec7a
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.25.4+77bec7a
```

9.2.1. Ajout de registres spécifiques

Vous pouvez ajouter une liste de registres, et éventuellement un référentiel individuel au sein d'un registre, qui sont autorisés pour les actions de traction et de poussée d'image en modifiant la ressource personnalisée (CR) **image.config.openshift.io/cluster**. OpenShift Container Platform applique les modifications de cette CR à tous les nœuds du cluster.

Lors de l'extraction ou de l'importation d'images, le moteur d'exécution du conteneur recherche les registres répertoriés sous le paramètre **registrySources** dans le CR **image.config.openshift.io/cluster**. Si vous avez créé une liste de registres sous le paramètre **allowedRegistries**, le moteur d'exécution du conteneur ne recherche que ces registres. Les registres qui ne figurent pas dans la liste sont bloqués.

AVERTISSEMENT

Lorsque le paramètre **allowedRegistries** est défini, tous les registres, y compris les registres **registry.redhat.io** et **quay.io** et le registre d'image OpenShift par défaut, sont bloqués sauf s'ils sont explicitement listés. Si vous utilisez ce paramètre, pour éviter l'échec d'un pod, ajoutez les registres **registry.redhat.io** et **quay.io** et la liste **internalRegistryHostname** à **allowedRegistries**, car ils sont requis par les images de charge utile dans votre environnement. Pour les clusters déconnectés, des registres miroirs doivent également être ajoutés.

Procédure

1. Modifier le CR **image.config.openshift.io/cluster**:

```
$ oc edit image.config.openshift.io/cluster
```

Voici un exemple de **image.config.openshift.io/cluster** CR avec une liste autorisée :

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    allowedRegistries: 2
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1 Contient des configurations qui déterminent comment le runtime du conteneur doit traiter les registres individuels lors de l'accès aux images pour les builds et les pods. Il ne contient pas de configuration pour le registre interne du cluster.

2 Spécifier les registres, et éventuellement un dépôt dans ce registre, à utiliser pour les actions d'extraction et de poussée d'images. Tous les autres registres sont bloqués.

NOTE

Le paramètre **allowedRegistries** ou le paramètre **blockedRegistries** peut être défini, mais pas les deux.

L'opérateur de configuration de la machine (MCO) surveille la ressource **image.config.openshift.io/cluster** pour détecter toute modification des registres. Lorsque le MCO détecte une modification, il vide les nœuds, applique la modification et déconnecte les nœuds. Une fois que les nœuds sont revenus à l'état **Ready**, la liste des registres autorisés est utilisée pour mettre à jour la stratégie de signature d'image dans le fichier **/host/etc/containers/policy.json** de chaque nœud.

2. Pour vérifier que les registres ont été ajoutés au fichier de stratégie, utilisez la commande suivante sur un nœud :

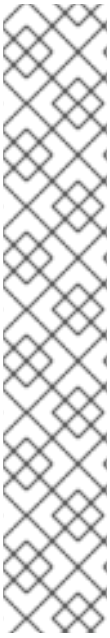
```
$ cat /host/etc/containers/policy.json
```

La politique suivante indique que seules les images provenant des registres example.com, quay.io et registry.redhat.io sont autorisées pour les extractions et les extractions d'images :

Exemple 9.1. Exemple de fichier de politique de signature d'image

```
{
  "default":[
    {
      "type":"reject"
    }
  ],
  "transports":{
    "atomic":{
      "example.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "image-registry.openshift-image-registry.svc:5000":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "insecure.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "quay.io":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "reg4.io/myrepo/myapp:latest":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
```

```
        "registry.redhat.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      },
      "docker":{
        "example.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "image-registry.openshift-image-registry.svc:5000":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "insecure.com":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "quay.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "reg4.io/myrepo/myapp:latest":[
          {
            "type":"insecureAcceptAnything"
          }
        ],
        "registry.redhat.io":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      },
      "docker-daemon":{
        "":[
          {
            "type":"insecureAcceptAnything"
          }
        ]
      }
    }
  }
```



NOTE

Si votre cluster utilise le paramètre **registrySources.insecureRegistries**, assurez-vous que tous les registres non sécurisés sont inclus dans la liste autorisée.

Par exemple :

```
spec:
  registrySources:
    insecureRegistries:
    - insecure.com
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com
    - image-registry.openshift-image-registry.svc:5000
```

9.2.2. Blocage de registres spécifiques

Vous pouvez bloquer n'importe quel registre, et éventuellement un référentiel individuel au sein d'un registre, en modifiant la ressource personnalisée (CR) **image.config.openshift.io/cluster**. OpenShift Container Platform applique les modifications de cette CR à tous les nœuds du cluster.

Lors de l'extraction ou de l'importation d'images, le moteur d'exécution du conteneur recherche les registres répertoriés sous le paramètre **registrySources** dans le CR **image.config.openshift.io/cluster**. Si vous avez créé une liste de registres sous le paramètre **blockedRegistries**, le moteur d'exécution du conteneur ne recherche pas ces registres. Tous les autres registres sont autorisés.

AVERTISSEMENT

Pour éviter un échec du pod, n'ajoutez pas les registres **registry.redhat.io** et **quay.io** à la liste **blockedRegistries**, car ils sont requis par les images de charge utile dans votre environnement.

Procédure

1. Modifier le CR **image.config.openshift.io/cluster**:

```
$ oc edit image.config.openshift.io/cluster
```

Voici un exemple de **image.config.openshift.io/cluster** CR avec une liste de blocage :

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
```

```
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    blockedRegistries: 2
    - untrusted.com
    - reg1.io/myrepo/myapp:latest
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1 Contient des configurations qui déterminent comment le runtime du conteneur doit traiter les registres individuels lors de l'accès aux images pour les builds et les pods. Il ne contient pas de configuration pour le registre interne du cluster.

2 Spécifier les registres, et éventuellement un dépôt dans ce registre, qui ne doivent pas être utilisés pour les actions d'extraction et de poussée d'images. Tous les autres registres sont autorisés.

NOTE

Le registre **blockedRegistries** ou le registre **allowedRegistries** peut être défini, mais pas les deux.

L'opérateur de configuration de la machine (MCO) surveille la ressource **image.config.openshift.io/cluster** pour détecter toute modification des registres. Lorsque le MCO détecte une modification, il draine les nœuds, applique la modification et désenregistre les nœuds. Une fois que les nœuds sont revenus à l'état **Ready**, les modifications apportées aux registres bloqués apparaissent dans le fichier **/etc/containers/registries.conf** de chaque nœud.

2. Pour vérifier que les registres ont été ajoutés au fichier de stratégie, utilisez la commande suivante sur un nœud :

```
$ cat /host/etc/containers/registries.conf
```

L'exemple suivant indique que les images du registre **untrusted.com** ne sont pas autorisées pour les extractions et les extractions d'images :

Exemple de sortie

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "untrusted.com"
  blocked = true
```

9.2.2.1. Blocage d'un registre de charges utiles

Dans une configuration de mise en miroir, vous pouvez bloquer les registres de charge utile en amont dans un environnement déconnecté à l'aide d'un objet **ImageContentSourcePolicy** (ICSP). L'exemple de procédure suivant montre comment bloquer le registre de charge utile **quay.io/openshift-payload**.

Procédure

1. Créez la configuration miroir à l'aide d'un objet **ImageContentSourcePolicy** (ICSP) pour mettre en miroir la charge utile dans un registre de votre instance. L'exemple de fichier ICSP suivant met en miroir la charge utile **internal-mirror.io/openshift-payload**:

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: my-icsp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - internal-mirror.io/openshift-payload
    source: quay.io/openshift-payload
```

2. Une fois l'objet déployé sur vos nœuds, vérifiez que la configuration du miroir est définie en consultant le fichier **/etc/containers/registries.conf**:

Exemple de sortie

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

3. Utilisez la commande suivante pour modifier le fichier de ressources personnalisées **image.config.openshift.io**:

```
$ oc edit image.config.openshift.io cluster
```

4. Pour bloquer le registre des charges utiles, ajoutez la configuration suivante au fichier de ressources personnalisées **image.config.openshift.io**:

```
spec:
  registrySource:
    blockedRegistries:
     - quay.io/openshift-payload
```

Vérification

- Vérifiez que le registre des charges utiles en amont est bloqué en consultant le fichier **/etc/containers/registries.conf** sur le nœud.

Exemple de sortie

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  blocked = true
  mirror-by-digest-only = true
```

```
[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

9.2.3. Autoriser les registres non sécurisés

Vous pouvez ajouter des registres non sécurisés, et éventuellement un référentiel individuel au sein d'un registre, en modifiant la ressource personnalisée (CR) **image.config.openshift.io/cluster**. OpenShift Container Platform applique les modifications de cette CR à tous les nœuds du cluster.

Les registres qui n'utilisent pas de certificats SSL valides ou qui n'exigent pas de connexions HTTPS sont considérés comme non sécurisés.

AVERTISSEMENT

Les registres externes non sécurisés doivent être évités afin de réduire les risques éventuels pour la sécurité.

Procédure

1. Modifier le CR **image.config.openshift.io/cluster**:

```
$ oc edit image.config.openshift.io/cluster
```

Voici un exemple de **image.config.openshift.io/cluster** CR avec une liste de registres non sécurisés :

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: 1
    insecureRegistries: 2
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com 3
    - reg4.io/myrepo/myapp:latest
```

```
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. Contient des configurations qui déterminent comment le runtime du conteneur doit traiter les registres individuels lors de l'accès aux images pour les builds et les pods. Il ne contient pas de configuration pour le registre interne du cluster.
2. Spécifiez un registre non sécurisé. Vous pouvez spécifier un référentiel dans ce registre.
3. Veiller à ce que les registres non sécurisés soient inclus dans la liste **allowedRegistries**.

NOTE

Lorsque le paramètre **allowedRegistries** est défini, tous les registres, y compris les registres registry.redhat.io et quay.io et le registre d'image OpenShift par défaut, sont bloqués sauf s'ils sont explicitement listés. Si vous utilisez ce paramètre, pour éviter une défaillance du pod, ajoutez tous les registres, y compris les registres **registry.redhat.io** et **quay.io**, ainsi que la liste **internalRegistryHostname** à **allowedRegistries**, car ils sont requis par les images de charge utile dans votre environnement. Pour les clusters déconnectés, les registres miroirs doivent également être ajoutés.

L'opérateur de configuration de la machine (MCO) surveille le CR **image.config.openshift.io/cluster** pour détecter toute modification des registres, puis draine et débloque les nœuds lorsqu'il détecte des changements. Une fois que les nœuds sont revenus à l'état **Ready**, les modifications apportées aux registres non sécurisés et bloqués apparaissent dans le fichier **/etc/containers/registries.conf** de chaque nœud.

2. Pour vérifier que les registres ont été ajoutés au fichier de stratégie, utilisez la commande suivante sur un nœud :

```
$ cat /host/etc/containers/registries.conf
```

L'exemple suivant indique que les images provenant du registre **insecure.com** ne sont pas sûres et qu'elles sont autorisées pour les extractions et les extractions d'images.

Exemple de sortie

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "insecure.com"
  insecure = true
```

9.2.4. Ajout de registres autorisant les noms courts d'images

Vous pouvez ajouter des registres pour rechercher un nom court d'image en modifiant la ressource personnalisée (CR) **image.config.openshift.io/cluster**. OpenShift Container Platform applique les modifications de cette CR à tous les nœuds du cluster.

Un nom court d'image vous permet de rechercher des images sans inclure le nom de domaine complet dans la spécification d'extraction. Par exemple, vous pouvez utiliser **rhel7/etcd** au lieu de **registry.access.redhat.com/rhe7/etcd**.

Vous pouvez utiliser des noms courts dans des situations où l'utilisation du chemin complet n'est pas pratique. Par exemple, si votre cluster fait référence à plusieurs registres internes dont le DNS change fréquemment, vous devrez mettre à jour les noms de domaine pleinement qualifiés dans vos spécifications d'extraction à chaque changement. Dans ce cas, l'utilisation d'un nom court d'image peut s'avérer utile.

Lors de l'extraction ou de l'importation d'images, le moteur d'exécution du conteneur recherche les registres répertoriés sous le paramètre **registrySources** dans le CR **image.config.openshift.io/cluster**. Si vous avez créé une liste de registres sous le paramètre **containerRuntimeSearchRegistries**, lors de l'extraction d'une image avec un nom court, le moteur d'exécution du conteneur recherche ces registres.

AVERTISSEMENT

L'utilisation de noms courts d'images avec les registres publics est fortement déconseillée car l'image risque de ne pas être déployée si le registre public exige une authentification. Utilisez des noms d'image pleinement qualifiés avec les registres publics.

Les registres internes ou privés de Red Hat prennent généralement en charge l'utilisation de noms courts d'images.

Si vous répertoriez les registres publics sous le paramètre **containerRuntimeSearchRegistries**, vous exposez vos informations d'identification à tous les registres de la liste et vous vous exposez à des attaques du réseau et du registre.

Vous ne pouvez pas répertorier plusieurs registres publics sous le paramètre **containerRuntimeSearchRegistries** si chaque registre public nécessite des informations d'identification différentes et si un cluster ne répertorie pas le registre public dans le secret d'extraction global.

Pour un registre public nécessitant une authentification, vous ne pouvez utiliser un nom court d'image que si les informations d'identification du registre sont stockées dans le secret d'extraction global.

L'opérateur de configuration de la machine (MCO) surveille la ressource **image.config.openshift.io/cluster** pour détecter toute modification des registres. Lorsque le MCO détecte une modification, il vide les nœuds, applique la modification et désenregistre les nœuds. Après le retour des nœuds à l'état **Ready**, si le paramètre **containerRuntimeSearchRegistries** est ajouté, le MCO crée un fichier dans le répertoire **/etc/containers/registries.conf.d** sur chaque nœud avec les registres répertoriés. Ce fichier remplace la liste par défaut des registres de recherche non qualifiés du fichier **/host/etc/containers/registries.conf**. Il n'existe aucun moyen de revenir à la liste par défaut des registres de recherche non qualifiés.

Le paramètre **containerRuntimeSearchRegistries** ne fonctionne qu'avec les moteurs de conteneurs Podman et CRI-O. Les registres de la liste ne peuvent être utilisés que dans les spécifications des pods, pas dans les builds et les flux d'images.

Procédure

1. Modifiez la ressource personnalisée **image.config.openshift.io/cluster**:

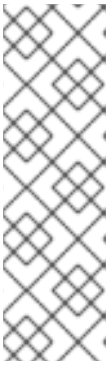
```
$ oc edit image.config.openshift.io/cluster
```

Voici un exemple de **image.config.openshift.io/cluster** CR :

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    containerRuntimeSearchRegistries: 1
    - reg1.io
    - reg2.io
    - reg3.io
    allowedRegistries: 2
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io
    - reg2.io
    - reg3.io
    - image-registry.openshift-image-registry.svc:5000
...
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. Indiquer les registres à utiliser avec les noms courts d'images. Il est conseillé d'utiliser les noms courts d'images uniquement avec des registres internes ou privés afin de réduire les risques éventuels pour la sécurité.

2. Veiller à ce que tous les registres répertoriés sous **containerRuntimeSearchRegistries** soient inclus dans la liste **allowedRegistries**.



NOTE

Lorsque le paramètre **allowedRegistries** est défini, tous les registres, y compris les registres **registry.redhat.io** et **quay.io** et le registre d'image OpenShift par défaut, sont bloqués sauf s'ils sont explicitement listés. Si vous utilisez ce paramètre, pour éviter une défaillance du pod, ajoutez tous les registres, y compris les registres **registry.redhat.io** et **quay.io** et la liste **internalRegistryHostname** à **allowedRegistries**, car ils sont requis par les images de charge utile dans votre environnement. Pour les clusters déconnectés, les registres miroirs doivent également être ajoutés.

2. Pour vérifier que les registres ont été ajoutés, lorsqu'un nœud revient à l'état **Ready**, utilisez la commande suivante sur le nœud :

```
$ cat /host/etc/containers/registries.conf.d/01-image-searchRegistries.conf
```

Exemple de sortie

```
unqualified-search-registries = ['reg1.io', 'reg2.io', 'reg3.io']
```

9.2.5. Configuration de magasins de confiance supplémentaires pour l'accès au registre d'images

La ressource personnalisée **image.config.openshift.io/cluster** peut contenir une référence à une carte de configuration qui contient des autorités de certification supplémentaires à approuver lors de l'accès au registre d'images.

Conditions préalables

- Les autorités de certification (CA) doivent être codées en PEM.

Procédure

Vous pouvez créer une carte de configuration dans l'espace de noms **openshift-config** et utiliser son nom dans **AdditionalTrustedCA** dans la ressource personnalisée **image.config.openshift.io** pour fournir des autorités de certification supplémentaires qui doivent être approuvées lorsqu'elles contactent des registres externes.

La clé de configuration est le nom d'hôte d'un registre avec le port pour lequel cette autorité de certification doit être approuvée, et le contenu du certificat PEM est la valeur, pour chaque autorité de certification de registre supplémentaire à approuver.

Registre d'images Exemple de carte de configuration de l'autorité de certification

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com..5000: | 1
```

```
-----BEGIN CERTIFICATE-----
...
-----END CERTIFICATE-----
```

1 Si le registre comporte le port, tel que **registry-with-port.example.com:5000**, **:** doit être remplacé par **..**.

Vous pouvez configurer des autorités de certification supplémentaires en suivant la procédure suivante.

1. Pour configurer une autorité de certification supplémentaire :

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n
openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

9.2.6. Configuration de la mise en miroir du référentiel du registre d'images

La configuration de la mise en miroir du référentiel du registre des conteneurs vous permet d'effectuer les opérations suivantes :

- Configurez votre cluster OpenShift Container Platform pour rediriger les demandes d'extraction d'images à partir d'un dépôt sur un registre d'images source et les faire résoudre par un dépôt sur un registre d'images miroir.
- Identifier plusieurs référentiels miroirs pour chaque référentiel cible, afin de s'assurer que si un miroir est en panne, un autre peut être utilisé.

Les attributs de la mise en miroir de référentiel dans OpenShift Container Platform sont les suivants :

- Les extractions d'images résistent aux interruptions du registre.
- Les clusters situés dans des environnements déconnectés peuvent extraire des images de sites critiques, tels que quay.io, et demander aux registres situés derrière le pare-feu de l'entreprise de fournir les images demandées.
- Un ordre particulier de registres est essayé lorsqu'une demande d'extraction d'image est faite, le registre permanent étant généralement le dernier essayé.
- Les informations sur le miroir que vous saisissez sont ajoutées au fichier **/etc/containers/registries.conf** sur chaque nœud du cluster OpenShift Container Platform.
- Lorsqu'un nœud demande une image à partir du référentiel source, il essaie chaque référentiel miroir à tour de rôle jusqu'à ce qu'il trouve le contenu demandé. Si tous les miroirs échouent, le cluster essaie le référentiel source. En cas de succès, l'image est transférée au nœud.

La configuration de la mise en miroir du référentiel peut se faire de la manière suivante :

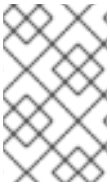
- A l'installation d'OpenShift Container Platform :
  En extrayant les images de conteneurs nécessaires à OpenShift Container Platform, puis en

amenant ces images derrière le pare-feu de votre entreprise, vous pouvez installer OpenShift Container Platform dans un centre de données qui se trouve dans un environnement déconnecté.

- Après l'installation d'OpenShift Container Platform :
  Même si vous ne configurez pas la mise en miroir lors de l'installation d'OpenShift Container Platform, vous pouvez le faire plus tard en utilisant l'objet **ImageContentSourcePolicy**.

La procédure suivante fournit une configuration miroir post-installation, dans laquelle vous créez un objet **ImageContentSourcePolicy** qui identifie :

- La source du référentiel d'images de conteneurs que vous souhaitez mettre en miroir.
- Une entrée distincte pour chaque référentiel miroir dans lequel vous souhaitez proposer le contenu demandé au référentiel source.



NOTE

Vous ne pouvez configurer des secrets de tirage globaux que pour les clusters qui ont un objet **ImageContentSourcePolicy**. Vous ne pouvez pas ajouter un secret d'extraction à un projet.

Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.

Procédure

1. Configurer les référentiels miroirs, en utilisant l'un ou l'autre :
   - La configuration d'un référentiel miroir avec Red Hat Quay, comme décrit dans Red Hat Quay Repository Mirroring. L'utilisation de Red Hat Quay vous permet de copier des images d'un référentiel vers un autre et de synchroniser automatiquement ces référentiels de manière répétée au fil du temps.
   - Utilisation d'un outil tel que **skopeo** pour copier manuellement les images du répertoire source vers le référentiel miroir.
     Par exemple, après avoir installé le paquetage RPM skopeo sur un système Red Hat Enterprise Linux (RHEL) 7 ou RHEL 8, utilisez la commande **skopeo** comme indiqué dans cet exemple :

     ```
     $ skopeo copy \
     docker://registry.access.redhat.com/ubi8/ubi-minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455ebaa6 \
     docker://example.io/example/ubi-minimal
     ```

     Dans cet exemple, vous avez un registre d'images de conteneurs nommé **example.io** avec un dépôt d'images nommé **example** dans lequel vous voulez copier l'image **ubi8/ubi-minimal** à partir de **registry.access.redhat.com**. Après avoir créé le registre, vous pouvez configurer votre cluster OpenShift Container Platform pour rediriger les requêtes faites sur le dépôt source vers le dépôt miroir.
2. Connectez-vous à votre cluster OpenShift Container Platform.

3. Créez un fichier **ImageContentSourcePolicy** (par exemple, **registryrepomirror.yaml**), en remplaçant la source et les miroirs par vos propres paires et images de registres et de référentiels :

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal (1)
    - example.com/example/ubi-minimal (2)
    source: registry.access.redhat.com/ubi8/ubi-minimal (3)
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 (4)
  - mirrors:
    - mirror.example.com
    source: registry.redhat.io (5)
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage (6)
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example (7)
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com (8)
```

(1) Indique le nom du registre d'images et du référentiel.

(2) Indique plusieurs référentiels miroirs pour chaque référentiel cible. Si un miroir est hors service, le référentiel cible peut utiliser un autre miroir.

(3) Indique le registre et le référentiel contenant le contenu qui est mis en miroir.

(4) Vous pouvez configurer un espace de noms à l'intérieur d'un registre pour utiliser n'importe quelle image dans cet espace de noms. Si vous utilisez un domaine de registre comme source, la ressource **ImageContentSourcePolicy** est appliquée à tous les référentiels du registre.

(5) Si vous configurez le nom du registre, la ressource **ImageContentSourcePolicy** est appliquée à tous les référentiels, du registre source au registre miroir.

(6) Tire l'image **mirror.example.net/image@sha256:...**.

(7) Extrait l'image **myimage** dans l'espace de noms du registre source à partir du miroir **mirror.example.net/myimage@sha256:...**.

(8) Extrait l'image **registry.example.com/example/myimage** du registre miroir **mirror.example.net/registry-example-com/example/myimage@sha256:...**. La ressource **ImageContentSourcePolicy** est appliquée à tous les référentiels d'un registre source à un registre miroir **mirror.example.net/registry-example-com**.

4. Créer le nouvel objet **ImageContentSourcePolicy**:

```
$ oc create -f registryrepomirror.yaml
```

Une fois l'objet **ImageContentSourcePolicy** créé, les nouveaux paramètres sont déployés sur chaque nœud et le cluster commence à utiliser le référentiel miroir pour les requêtes vers le référentiel source.

5. Pour vérifier que les paramètres de configuration en miroir sont appliqués, procédez comme suit sur l'un des nœuds.

   a. Dressez la liste de vos nœuds :

```
$ oc get node
```

Exemple de sortie

```
NAME                          STATUS                ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-138-148.ec2.internal   Ready                     master   11m  v1.25.0
ip-10-0-139-122.ec2.internal   Ready                     master   11m  v1.25.0
ip-10-0-147-35.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-153-12.ec2.internal    Ready                    worker   7m   v1.25.0
ip-10-0-154-10.ec2.internal    Ready                    master   11m  v1.25.0
```

La ressource **Imagecontentsourcepolicy** ne redémarre pas les nœuds.

   b. Lancez le processus de débogage pour accéder au nœud :

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

Exemple de sortie

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

   c. Changez votre répertoire racine en /**host**:

```
sh-4.2# chroot /host
```

   d. Vérifiez le fichier /**etc**/**containers**/**registries.conf** pour vous assurer que les changements ont bien été effectués :

```
sh-4.2# cat /etc/containers/registries.conf
```

Exemple de sortie

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
short-name-mode = ""

[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi8/ubi-minimal"
```

```
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal"

  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"

[[registry]]
  prefix = ""
  location = "registry.example.com"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net"

[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.net/image"

[[registry]]
  prefix = ""
  location = "registry.redhat.io"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.com"

[[registry]]
  prefix = ""
  location = "registry.redhat.io/openshift4"
  mirror-by-digest-only = true

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
```

e. Transmet un condensé d'image au nœud à partir de la source et vérifie s'il est résolu par le miroir. Les objets **ImageContentSourcePolicy** ne prennent en charge que les condensés d'image, et non les balises d'image.

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455ebaa6
```

Dépannage de la mise en miroir du référentiel

Si la procédure de mise en miroir du référentiel ne fonctionne pas comme décrit, utilisez les informations suivantes sur le fonctionnement de la mise en miroir du référentiel pour résoudre le problème.

- Le premier miroir de travail est utilisé pour fournir l'image tirée.
- Le registre principal n'est utilisé que si aucun autre miroir ne fonctionne.
- Dans le contexte du système, les drapeaux **Insecure** sont utilisés comme solution de repli.
- Le format du fichier **/etc/containers/registries.conf** a récemment changé. Il s'agit désormais de la version 2 et du format TOML.

Ressources complémentaires

- Pour plus d'informations sur les secrets de tirage globaux, voir Mise à jour du secret de tirage global du cluster.

CHAPITRE 10. UTILISER DES MODÈLES

Les sections suivantes présentent une vue d'ensemble des modèles et expliquent comment les utiliser et les créer.

10.1. COMPRENDRE LES MODÈLES

Un template décrit un ensemble d'objets qui peuvent être paramétrés et traités pour produire une liste d'objets à créer par OpenShift Container Platform. Un modèle peut être traité pour créer tout ce que vous avez le droit de créer dans un projet, par exemple des services, des configurations de construction et des configurations de déploiement. Un modèle peut également définir un ensemble d'étiquettes à appliquer à chaque objet défini dans le modèle.

Vous pouvez créer une liste d'objets à partir d'un modèle à l'aide de la CLI ou, si un modèle a été téléchargé dans votre projet ou dans la bibliothèque de modèles globale, à l'aide de la console web.

10.2. TÉLÉCHARGEMENT D'UN MODÈLE

Si vous disposez d'un fichier JSON ou YAML qui définit un modèle, comme dans cet exemple, vous pouvez télécharger le modèle dans les projets à l'aide de l'interface de programmation. Cela permet d'enregistrer le modèle dans le projet pour une utilisation répétée par tout utilisateur disposant d'un accès approprié à ce projet. Des instructions sur l'écriture de vos propres modèles sont fournies plus loin dans cette rubrique.

Procédure

- Téléchargez un modèle dans la bibliothèque de modèles de votre projet actuel, transmettez le fichier JSON ou YAML à l'aide de la commande suivante :

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

- Téléchargez un modèle dans un autre projet en utilisant l'option **-n** avec le nom du projet :

```
$ oc create -f <filename> -n <projet>
```

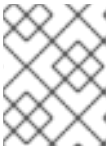
Le modèle est maintenant disponible pour être sélectionné à l'aide de la console web ou de la CLI.

10.3. CRÉATION D'UNE APPLICATION À L'AIDE DE LA CONSOLE WEB

Vous pouvez utiliser la console web pour créer une application à partir d'un modèle.

Procédure

1. Dans le projet souhaité, cliquez sur **Add to Project**.
2. Sélectionnez une image de constructeur dans la liste des images de votre projet ou dans le catalogue de services.



NOTE

Seules les étiquettes de flux d'images dont l'étiquette **builder** figure dans les annotations apparaissent dans cette liste, comme indiqué ici :

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
  dockerImageRepository: "registry.redhat.io/rhscl/ruby-26-rhel7"
  tags:
    -
      name: "2.6"
      annotations:
        description: "Build and run Ruby 2.6 applications"
        iconClass: "icon-ruby"
        tags: "builder,ruby" 1
        supports: "ruby:2.6,ruby"
        version: "2.6"
```

1 Le fait d'inclure **builder** ici garantit que cette balise de flux d'images apparaît dans la console web en tant que constructeur.

3. Modifiez les paramètres de l'écran de la nouvelle application pour configurer les objets en fonction de votre application.

10.4. CRÉATION D'OBJETS À PARTIR DE MODÈLES À L'AIDE DE L'INTERFACE DE PROGRAMMATION

Vous pouvez utiliser la CLI pour traiter les modèles et utiliser la configuration générée pour créer des objets.

10.4.1. Ajout d'étiquettes

Les étiquettes sont utilisées pour gérer et organiser les objets générés, tels que les pods. Les étiquettes spécifiées dans le modèle sont appliquées à chaque objet généré à partir du modèle.

Procédure

- Ajouter des étiquettes dans le modèle à partir de la ligne de commande :

```
$ oc process -f <filename> -l name=otherLabel
```

10.4.2. Paramètres d'inscription

La liste des paramètres que vous pouvez modifier figure dans la section **parameters** du modèle.

Procédure

1. Vous pouvez dresser la liste des paramètres à l'aide de l'interface de gestion en utilisant la commande suivante et en spécifiant le fichier à utiliser :

```
$ oc process --parameters -f <filename>
```

Alternativement, si le modèle est déjà téléchargé :

```
$ oc process --paramètres -n <projet> <nom_du_modèle>
```

Par exemple, l'exemple suivant montre le résultat de la liste des paramètres pour l'un des modèles de démarrage rapide dans le projet par défaut **openshift**:

```
$ oc process --parameters -n openshift rails-postgresql-example
```

Exemple de sortie

```
NAME                     DESCRIPTION
GENERATOR          VALUE
SOURCE_REPOSITORY_URL        The URL of the repository with your application source
code                                    https://github.com/sclorg/rails-ex.git
SOURCE_REPOSITORY_REF        Set this to a branch name, tag or other ref of your
repository if you are not using the default branch
CONTEXT_DIR                  Set this to the relative path to your project if it is not in the root of
your repository
APPLICATION_DOMAIN           The exposed hostname that will route to the Rails service
rails-postgresql-example.openshiftapps.com
GITHUB_WEBHOOK_SECRET        A secret string used to configure the GitHub webhook
expression          [a-zA-Z0-9]{40}
SECRET_KEY_BASE              Your secret key for verifying the integrity of signed cookies
expression          [a-z0-9]{127}
APPLICATION_USER             The application user that is used within the sample application
to authorize access on pages                    openshift
APPLICATION_PASSWORD         The application password that is used within the sample
application to authorize access on pages                secret
DATABASE_SERVICE_NAME        Database service name
postgresql
POSTGRESQL_USER              database username
expression          user[A-Z0-9]{3}
POSTGRESQL_PASSWORD          database password
expression          [a-zA-Z0-9]{8}
POSTGRESQL_DATABASE          database name
root
POSTGRESQL_MAX_CONNECTIONS   database max connections
10
POSTGRESQL_SHARED_BUFFERS    database shared buffers
12MB
```

La sortie identifie plusieurs paramètres qui sont générés à l'aide d'un générateur d'expressions régulières lorsque le modèle est traité.

10.4.3. Générer une liste d'objets

À l'aide de l'interface de programmation, vous pouvez traiter un fichier définissant un modèle pour renvoyer la liste des objets sur la sortie standard.

Procédure

1. Traite un fichier définissant un modèle pour renvoyer la liste des objets sur la sortie standard :

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

Sinon, si le modèle a déjà été téléchargé dans le projet en cours :

```
oc process <nom_du_modèle>
```

2. Créer des objets à partir d'un modèle en traitant le modèle et en acheminant la sortie vers **oc create**:

```
oc process -f <filename> $ oc create -f -
```

Sinon, si le modèle a déjà été téléchargé dans le projet en cours :

```
oc process <template> | oc create -f -
```

3. Vous pouvez remplacer toutes les valeurs de paramètres définies dans le fichier en ajoutant l'option **-p** pour chaque paire **<name>=<value>** que vous souhaitez remplacer. Une référence de paramètre apparaît dans n'importe quel champ de texte à l'intérieur des éléments du modèle.
Par exemple, dans l'exemple suivant, les paramètres **POSTGRESQL_USER** et **POSTGRESQL_DATABASE** d'un modèle sont remplacés pour produire une configuration avec des variables d'environnement personnalisées :

   a. Création d'une liste d'objets à partir d'un modèle

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase
```

   b. Le fichier JSON peut être redirigé vers un fichier ou appliqué directement sans télécharger le modèle en envoyant la sortie traitée à la commande **oc create**:

```
$ oc process -f my-rails-postgresql \
    -p POSTGRESQL_USER=bob \
    -p POSTGRESQL_DATABASE=mydatabase \
    | oc create -f -
```

   c. Si vous avez un grand nombre de paramètres, vous pouvez les stocker dans un fichier et passer ce fichier à **oc process**:

```
$ cat postgres.env
POSTGRESQL_USER=bob
POSTGRESQL_DATABASE=mydatabase
```

```
$ oc process -f my-rails-postgresql --param-file=postgres.env
```

   d. Vous pouvez également lire l'environnement à partir de l'entrée standard en utilisant **"-"** comme argument de **--param-file**:

```
$ sed s/bob/alice/ postgres.env | oc process -f my-rails-postgresql --param-file=-
```

10.5. MODIFIER LES MODÈLES TÉLÉCHARGÉS

Vous pouvez modifier un modèle qui a déjà été téléchargé dans votre projet.

Procédure

- Modifier un modèle qui a déjà été téléchargé :

```
oc edit template <template> $ oc edit template <template>
```

10.6. UTILISATION DE L'APPLICATION INSTANTANÉE ET DES MODÈLES DE DÉMARRAGE RAPIDE

OpenShift Container Platform fournit un certain nombre d'applications instantanées par défaut et de modèles de démarrage rapide pour faciliter la création d'une nouvelle application pour différents langages. Des modèles sont fournis pour Rails (Ruby), Django (Python), Node.js, CakePHP (PHP) et Dancer (Perl). L'administrateur de votre cluster doit créer ces modèles dans le projet global par défaut **openshift** pour que vous puissiez y accéder.

Par défaut, les modèles sont construits à partir d'un dépôt de sources publiques sur GitHub qui contient le code de l'application nécessaire.

Procédure

1. Vous pouvez dresser la liste des applications instantanées par défaut et des modèles de démarrage rapide disponibles avec :

```
$ oc get templates -n openshift
```

2. Pour modifier les sources et créer votre propre version de l'application :

   a. Met en place le référentiel référencé par le paramètre par défaut du modèle **SOURCE_REPOSITORY_URL**.

   b. Remplacer la valeur du paramètre **SOURCE_REPOSITORY_URL** lors de la création à partir du modèle, en spécifiant votre fourchette au lieu de la valeur par défaut.
   Ce faisant, la configuration de construction créée par le modèle pointe désormais vers votre version du code de l'application, et vous pouvez modifier le code et reconstruire l'application à votre guise.

NOTE

Certains modèles d'application instantanée et de démarrage rapide définissent une configuration de déploiement de la base de données. La configuration qu'ils définissent utilise un stockage éphémère pour le contenu de la base de données. Ces modèles ne doivent être utilisés qu'à des fins de démonstration, car toutes les données de la base sont perdues si le module de base de données redémarre pour quelque raison que ce soit.

10.6.1. Modèles de démarrage rapide

Un modèle de démarrage rapide est un exemple de base d'une application fonctionnant sur OpenShift Container Platform. Les démarrages rapides sont disponibles dans une variété de langages et de cadres, et sont définis dans un modèle, qui est construit à partir d'un ensemble de services, de configurations de construction et de configurations de déploiement. Ce modèle référence les images et les dépôts de sources nécessaires pour construire et déployer l'application.

Pour explorer un démarrage rapide, créez une application à partir d'un modèle. Votre administrateur doit avoir déjà installé ces modèles dans votre cluster OpenShift Container Platform, auquel cas vous pouvez simplement le sélectionner à partir de la console web.

Les démarrages rapides font référence à un référentiel de sources qui contient le code source de l'application. Pour personnaliser le démarrage rapide, forker le référentiel et, lors de la création d'une application à partir du modèle, remplacer le nom du référentiel source par défaut par votre référentiel forké. Il en résulte que les constructions sont effectuées en utilisant votre code source au lieu du code source de l'exemple fourni. Vous pouvez ensuite mettre à jour le code dans votre dépôt de sources et lancer une nouvelle compilation pour voir les changements reflétés dans l'application déployée.

10.6.1.1. Modèles de démarrage rapide pour le cadre web

Ces modèles de démarrage rapide fournissent une application de base du cadre et du langage indiqués :

- CakePHP : un cadre web PHP qui inclut une base de données MySQL
- Dancer : un cadre web en Perl qui inclut une base de données MySQL
- Django : un framework web Python qui inclut une base de données PostgreSQL
- NodeJS : une application web NodeJS qui inclut une base de données MongoDB
- Rails : un cadre web Ruby qui inclut une base de données PostgreSQL

10.7. MODÈLES D'ÉCRITURE

Vous pouvez définir de nouveaux modèles pour faciliter la recréation de tous les objets de votre application. Le modèle définit les objets qu'il crée ainsi que certaines métadonnées pour guider la création de ces objets.

Voici un exemple de définition d'un objet modèle simple (YAML) :

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: redis-template
  annotations:
    description: "Description"
    iconClass: "icon-redis"
    tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
```

```
parameters:
- description: Password used for Redis authentication
  from: '[A-Z0-9]{8}'
  generate: expression
  name: REDIS_PASSWORD
labels:
  redis: master
```

10.7.1. Rédaction de la description du modèle

La description du modèle vous informe de ce que fait le modèle et vous aide à le trouver lors d'une recherche dans la console web. L'ajout de métadonnées autres que le nom du modèle est facultatif, mais utile. En plus des informations descriptives générales, les métadonnées comprennent également un ensemble de balises. Les balises utiles comprennent le nom du langage auquel le modèle est lié, par exemple Java, PHP, Ruby, etc.

Voici un exemple de métadonnées de description de modèle :

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: cakephp-mysql-example 1
  annotations:
    openshift.io/display-name: "CakePHP MySQL Example (Ephemeral)" 2
    description: >-
      An example CakePHP application with a MySQL database. For more information
      about using this template, including OpenShift considerations, see
      https://github.com/sclorg/cakephp-ex/blob/master/README.md.


      WARNING: Any data stored will be lost upon pod destruction. Only use this
      template for testing." 3
    openshift.io/long-description: >-
      This template defines resources needed to develop a CakePHP application,
      including a build configuration, application DeploymentConfig, and
      database DeploymentConfig.  The database is stored in
      non-persistent storage, so this configuration should be used for
      experimental purposes only. 4
    tags: "quickstart,php,cakephp" 5
    iconClass: icon-php 6
    openshift.io/provider-display-name: "Red Hat, Inc." 7
    openshift.io/documentation-url: "https://github.com/sclorg/cakephp-ex" 8
    openshift.io/support-url: "https://access.redhat.com" 9
message: "Your admin credentials are ${ADMIN_USERNAME}:${ADMIN_PASSWORD}" 10
```

1. Le nom unique du modèle.
2. Un nom bref et convivial, qui peut être utilisé par les interfaces utilisateurs.
3. Une description du modèle. Elle doit être suffisamment détaillée pour que les utilisateurs comprennent ce qui est déployé et les éventuelles mises en garde qu'ils doivent connaître avant de procéder au déploiement. Elle doit également fournir des liens vers des informations supplémentaires, telles qu'un fichier README. Les nouvelles lignes peuvent être incluses pour créer des paragraphes.

4 Description supplémentaire du modèle. Elle peut être affichée par le catalogue de services, par exemple.

5 Balises à associer au modèle pour la recherche et le regroupement. Ajoutez des balises qui l'incluent dans l'une des catégories de catalogue fournies. Reportez-vous à **id** et **categoryAliases** dans **CATALOG_CATEGORIES** dans le fichier des constantes de la console. Les catégories peuvent également être personnalisées pour l'ensemble du cluster.

6 Une icône à afficher avec votre modèle dans la console web.

Exemple 10.1. Icônes disponibles

- **icon-3scale**
- **icon-aerogear**
- **icon-amq**
- **icon-angularjs**
- **icon-ansible**
- **icon-apache**
- **icon-beaker**
- **icon-camel**
- **icon-capedwarf**
- **icon-cassandra**
- **icon-catalog-icon**
- **icon-clojure**
- **icon-codeigniter**
- **icon-cordova**
- **icon-datagrid**
- **icon-datavirt**
- **icon-debian**
- **icon-decisionserver**
- **icon-django**
- **icon-dotnet**
- **icon-drupal**
- **icon-eap**
- **icon-elastic**

- **icon-erlang**
- **icon-fedora**
- **icon-freebsd**
- **icon-git**
- **icon-github**
- **icon-gitlab**
- **icon-glassfish**
- **icon-go-gopher**
- **icon-golang**
- **icon-grails**
- **icon-hadoop**
- **icon-haproxy**
- **icon-helm**
- **icon-infinispan**
- **icon-jboss**
- **icon-jenkins**
- **icon-jetty**
- **icon-joomla**
- **icon-jruby**
- **icon-js**
- **icon-knative**
- **icon-kubevirt**
- **icon-laravel**
- **icon-load-balancer**
- **icon-mariadb**
- **icon-mediawiki**
- **icon-memcached**
- **icon-mongodb**
- **icon-mssql**

- **icon-mysql-database**
- **icon-nginx**
- **icon-nodejs**
- **icon-openjdk**
- **icon-openliberty**
- **icon-openshift**
- **icon-openstack**
- **icon-other-linux**
- **icon-other-unknown**
- **icon-perl**
- **icon-phalcon**
- **icon-php**
- **icon-play**
- **iconpostgresql**
- **icon-processserver**
- **icon-python**
- **icon-quarkus**
- **icon-rabbitmq**
- **icon-rails**
- **icon-redhat**
- **icon-redis**
- **icon-rh-integration**
- **icon-rh-spring-boot**
- **icon-rh-tomcat**
- **icon-ruby**
- **icon-scala**
- **icon-serverlessfx**
- **icon-shadowman**
- **icon-spring-boot**

- **icon-spring**
- **icon-sso**
- **icon-stackoverflow**
- **icon-suse**
- **icon-symfony**
- **icon-tomcat**
- **icon-ubuntu**
- **icon-vertx**
- **icon-wildfly**
- **icon-windows**
- **icon-wordpress**
- **icon-xamarin**
- **icon-zend**

7 Le nom de la personne ou de l'organisation qui fournit le modèle.

8 Une URL renvoyant à une documentation supplémentaire pour le modèle.

9 URL où l'on peut obtenir de l'aide pour le modèle.

10 Un message d'instruction qui s'affiche lorsque ce modèle est instancié. Ce champ doit informer l'utilisateur sur la manière d'utiliser les ressources nouvellement créées. La substitution des paramètres est effectuée sur le message avant qu'il ne soit affiché afin que les informations d'identification générées et d'autres paramètres puissent être inclus dans la sortie. Inclure des liens vers toute documentation sur les étapes suivantes que les utilisateurs doivent suivre.

10.7.2. Rédaction de modèles d'étiquettes

Les modèles peuvent inclure un ensemble d'étiquettes. Ces étiquettes sont ajoutées à chaque objet créé lorsque le modèle est instancié. La définition d'une étiquette de cette manière permet aux utilisateurs de trouver et de gérer facilement tous les objets créés à partir d'un modèle particulier.

Voici un exemple d'étiquettes d'objets de modèle :

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" 1
  app: "${NAME}" 2
```

1 Une étiquette appliquée à tous les objets créés à partir de ce modèle.

2 Une étiquette paramétrée qui est également appliquée à tous les objets créés à partir de ce modèle. L'expansion des paramètres est effectuée à la fois sur les clés et les valeurs de l'étiquette.

10.7.3. Écriture des paramètres du modèle

Les paramètres permettent à une valeur d'être fournie par vous ou générée lors de l'instanciation du modèle. Cette valeur est ensuite substituée partout où le paramètre est référencé. Les références peuvent être définies dans n'importe quel champ de la liste des objets. Ceci est utile pour générer des mots de passe aléatoires ou pour vous permettre de fournir un nom d'hôte ou une autre valeur spécifique à l'utilisateur qui est nécessaire pour personnaliser le modèle. Les paramètres peuvent être référencés de deux manières :

- Comme une valeur de chaîne en plaçant des valeurs dans le formulaire **${PARAMETER_NAME}** dans n'importe quel champ de chaîne du modèle.
- En tant que valeur JSON ou YAML en plaçant des valeurs dans le formulaire **${{PARAMETER_NAME}}** à la place de n'importe quel champ du modèle.

En utilisant la syntaxe **${PARAMETER_NAME}**, plusieurs références de paramètres peuvent être combinées dans un seul champ et la référence peut être intégrée dans des données fixes, telles que **"http://${PARAMETER_1}${PARAMETER_2}"**. Les deux valeurs des paramètres sont substituées et la valeur résultante est une chaîne de caractères entre guillemets.

Lors de l'utilisation de la syntaxe **${{PARAMETER_NAME}}**, une seule référence de paramètre est autorisée et les caractères de tête et de fin ne sont pas autorisés. La valeur résultante n'est pas mise entre guillemets, sauf si, après substitution, le résultat n'est pas un objet JSON valide. Si le résultat n'est pas une valeur JSON valide, la valeur résultante est citée et traitée comme une chaîne de caractères standard.

Un même paramètre peut être référencé plusieurs fois dans un modèle et il peut être référencé à l'aide des deux syntaxes de substitution dans un même modèle.

Une valeur par défaut peut être fournie, qui est utilisée si vous ne fournissez pas de valeur différente :

Voici un exemple de définition d'une valeur explicite comme valeur par défaut :

```
parameters:
  - name: USERNAME
    description: "The user name for Joe"
    value: joe
```

Les valeurs des paramètres peuvent également être générées sur la base de règles spécifiées dans la définition du paramètre, par exemple en générant une valeur de paramètre :

```
parameters:
  - name: PASSWORD
    description: "The random user password"
    generate: expression
    from: "[a-zA-Z0-9]{12}"
```

Dans l'exemple précédent, le traitement génère un mot de passe aléatoire de 12 caractères composé de toutes les lettres de l'alphabet, majuscules et minuscules, et de chiffres.

La syntaxe disponible n'est pas une syntaxe d'expression régulière complète. Cependant, vous pouvez utiliser les modificateurs **w**, **d**, **a**, et **A**:

- **[\w]{10}** produit 10 caractères alphabétiques, des chiffres et des traits de soulignement. Cette valeur est conforme à la norme PCRE et correspond à **[a-zA-Z0-9_]{10}**.
- **[\d]{10}** produit 10 nombres. Ce nombre est égal à **[0-9]{10}**.
- **[\a]{10}** produit 10 caractères alphabétiques. Cela correspond à **[a-zA-Z]{10}**.
- **[\A]{10}** produces 10 punctuation or symbol characters. This is equal to **[~!@#$%\^&*()\-_+={}\[\]\\|<,>.?/"';:`]{10}**.

NOTE

Selon que le modèle est écrit en YAML ou en JSON, et selon le type de chaîne dans laquelle le modificateur est intégré, il peut être nécessaire d'échapper à la barre oblique inverse par une seconde barre oblique inverse. Les exemples suivants sont équivalents :

Exemple de modèle YAML avec un modificateur

```
parameters:
- name: singlequoted_example
  generate: expression
  from: '[\A]{10}'
- name: doublequoted_example
  generate: expression
  from: "[\\A]{10}"
```

Exemple de modèle JSON avec un modificateur

```
{
    "parameters": [
      {
       "name": "json_example",
       "generate": "expression",
       "from": "[\\A]{10}"
      }
    ]
}
```

Voici un exemple de modèle complet avec les définitions des paramètres et les références :

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
  - kind: BuildConfig
    apiVersion: build.openshift.io/v1
    metadata:
      name: cakephp-mysql-example
      annotations:
        description: Defines how to build the application
    spec:
      source:
        type: Git
```

```
        git:
          uri: "${SOURCE_REPOSITORY_URL}" 1
          ref: "${SOURCE_REPOSITORY_REF}"
        contextDir: "${CONTEXT_DIR}"
    - kind: DeploymentConfig
      apiVersion: apps.openshift.io/v1
      metadata:
        name: frontend
      spec:
        replicas: "${{REPLICA_COUNT}}" 2
parameters:
  - name: SOURCE_REPOSITORY_URL 3
    displayName: Source Repository URL 4
    description: The URL of the repository with your application source code 5
    value: https://github.com/sclorg/cakephp-ex.git 6
    required: true 7
  - name: GITHUB_WEBHOOK_SECRET
    description: A secret string used to configure the GitHub webhook
    generate: expression 8
    from: "[a-zA-Z0-9]{40}" 9
  - name: REPLICA_COUNT
    description: Number of replicas to run
    value: "2"
    required: true
message: "... The GitHub webhook secret is ${GITHUB_WEBHOOK_SECRET} ..." 10
```

1. Cette valeur est remplacée par la valeur du paramètre **SOURCE_REPOSITORY_URL** lors de l'instanciation du modèle.

2. Cette valeur est remplacée par la valeur non citée du paramètre **REPLICA_COUNT** lorsque le modèle est instancié.

3. Le nom du paramètre. Cette valeur est utilisée pour référencer le paramètre dans le modèle.

4. Nom convivial du paramètre. Il est affiché aux utilisateurs.

5. Description du paramètre. Fournissez des informations plus détaillées sur l'objectif du paramètre, y compris toute contrainte sur la valeur attendue. Les descriptions doivent utiliser des phrases complètes afin de respecter les normes textuelles de la console. Cette description ne doit pas être un doublon du nom d'affichage.

6. Une valeur par défaut pour le paramètre qui est utilisée si vous ne remplacez pas la valeur lors de l'instanciation du modèle. Évitez d'utiliser des valeurs par défaut pour des éléments tels que les mots de passe ; utilisez plutôt des paramètres générés en combinaison avec des secrets.

7. Indique que ce paramètre est obligatoire, ce qui signifie que vous ne pouvez pas le remplacer par une valeur vide. Si le paramètre ne fournit pas de valeur par défaut ou générée, vous devez fournir une valeur.

8. Un paramètre dont la valeur est générée.

9. L'entrée du générateur. Dans ce cas, le générateur produit une valeur alphanumérique de 40 caractères comprenant des majuscules et des minuscules.

10. Des paramètres peuvent être inclus dans le message du modèle. Cela vous informe des valeurs générées.

générées.

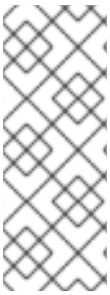
10.7.4. Écriture de la liste des objets du modèle

La partie principale du modèle est la liste des objets créés lors de l'instanciation du modèle. Il peut s'agir de n'importe quel objet API valide, tel qu'une configuration de construction, une configuration de déploiement ou un service. L'objet est créé exactement comme il est défini ici, les valeurs des paramètres étant substituées avant la création. La définition de ces objets peut faire référence à des paramètres définis précédemment.

Voici un exemple de liste d'objets :

```
kind: "Template"
apiVersion: "v1"
metadata:
  name: my-template
objects:
  - kind: "Service" 1
    apiVersion: "v1"
    metadata:
      name: "cakephp-mysql-example"
      annotations:
        description: "Exposes and load balances the application pods"
    spec:
      ports:
        - name: "web"
          port: 8080
          targetPort: 8080
      selector:
        name: "cakephp-mysql-example"
```

1 La définition d'un service, qui est créé par ce modèle.



NOTE

Si les métadonnées d'une définition d'objet comprennent une valeur fixe du champ **namespace**, le champ est supprimé de la définition lors de l'instanciation du modèle. Si le champ **namespace** contient une référence à un paramètre, la substitution normale du paramètre est effectuée et l'objet est créé dans l'espace de noms auquel la substitution du paramètre a résolu la valeur, à condition que l'utilisateur soit autorisé à créer des objets dans cet espace de noms.

10.7.5. Marquer un modèle comme liant

Le Template Service Broker annonce un service dans son catalogue pour chaque objet modèle dont il a connaissance. Par défaut, chacun de ces services est annoncé comme étant liant, ce qui signifie qu'un utilisateur final est autorisé à se lier au service fourni.

Procédure

Les auteurs de modèles peuvent empêcher les utilisateurs finaux de se lier aux services fournis à partir d'un modèle donné.

- Empêcher l'utilisateur final de se lier aux services fournis à partir d'un modèle donné en ajoutant l'annotation **template.openshift.io/bindable: "false"** au modèle.

10.7.6. Exposition des champs de l'objet modèle

Les auteurs de modèles peuvent indiquer que les champs de certains objets d'un modèle doivent être exposés. Le Template Service Broker reconnaît les champs exposés sur les objets **ConfigMap**, **Secret**, **Service**, et **Route**, et renvoie les valeurs des champs exposés lorsqu'un utilisateur lie un service soutenu par le broker.

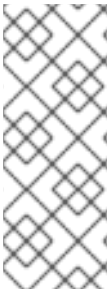
Pour exposer un ou plusieurs champs d'un objet, ajoutez des annotations préfixées par **template.openshift.io/expose-** ou **template.openshift.io/base64-expose-** à l'objet dans le modèle.

Chaque clé d'annotation, dont le préfixe a été supprimé, est transmise pour devenir une clé dans une réponse **bind**.

Chaque valeur d'annotation est une expression JSONPath de Kubernetes, qui est résolue au moment de la liaison pour indiquer le champ d'objet dont la valeur doit être renvoyée dans la réponse **bind**.

NOTE

Bind les paires clé-valeur de réponse peuvent être utilisées dans d'autres parties du système en tant que variables d'environnement. Il est donc recommandé que chaque clé d'annotation dont le préfixe a été supprimé soit un nom de variable d'environnement valide - commençant par un caractère **A-Z**, **a-z**, ou _, et suivi d'au moins zéro caractère **A-Z**, **a-z**, **0-9**, ou _.



NOTE

Unless escaped with a backslash, Kubernetes' JSONPath implementation interprets characters such as **.**, **@**, and others as metacharacters, regardless of their position in the expression. Therefore, for example, to refer to a **ConfigMap** datum named **my.key**, the required JSONPath expression would be **{.data['my\.key']}**. Depending on how the JSONPath expression is then written in YAML, an additional backslash might be required, for example **"{.data['my\\.key']}"**.

Voici un exemple d'exposition des champs de différents objets :

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: ConfigMap
  apiVersion: v1
  metadata:
    name: my-template-config
    annotations:
      template.openshift.io/expose-username: "{.data['my\\.username']}"
  data:
    my.username: foo
- kind: Secret
  apiVersion: v1
  metadata:
    name: my-template-config-secret
```

```
    annotations:
      template.openshift.io/base64-expose-password: "{.data['password']}"
  stringData:
    password: bar
- kind: Service
  apiVersion: v1
  metadata:
    name: my-template-service
    annotations:
      template.openshift.io/expose-service_ip_port: "{.spec.clusterIP}:{.spec.ports[?
(.name==\"web\")].port}"
  spec:
    ports:
    - name: "web"
      port: 8080
- kind: Route
  apiVersion: route.openshift.io/v1
  metadata:
    name: my-template-route
    annotations:
      template.openshift.io/expose-uri: "http://{.spec.host}{.spec.path}"
  spec:
    path: mypath
```

Voici un exemple de réponse à une opération **bind** à partir du modèle partiel ci-dessus :

```
{
  "credentials": {
    "username": "foo",
    "password": "YmFy",
    "service_ip_port": "172.30.12.34:8080",
    "uri": "http://route-test.router.default.svc.cluster.local/mypath"
  }
}
```

Procédure

- Utilisez l'annotation **template.openshift.io/expose-** pour renvoyer la valeur du champ sous forme de chaîne de caractères. C'est pratique, mais cela ne permet pas de gérer des données binaires arbitraires.
- Si vous souhaitez renvoyer des données binaires, utilisez plutôt l'annotation **template.openshift.io/base64-expose-** pour encoder base64 les données avant qu'elles ne soient renvoyées.

10.7.7. En attendant que le modèle soit prêt

Les auteurs de modèles peuvent indiquer que certains objets d'un modèle doivent être attendus avant que l'instanciation d'un modèle par le catalogue de services, le Template Service Broker ou l'API **TemplateInstance** ne soit considérée comme terminée.

Pour utiliser cette fonctionnalité, marquez un ou plusieurs objets du type **Build**, **BuildConfig**, **Deployment**, **DeploymentConfig**, **Job**, ou **StatefulSet** dans un modèle avec l'annotation suivante :

```
"template.alpha.openshift.io/wait-for-ready": "true"
```

L'instanciation du modèle n'est pas terminée tant que tous les objets marqués par l'annotation ne sont pas prêts. De même, si l'un des objets annotés signale un échec ou si le modèle n'est pas prêt dans un délai fixe d'une heure, l'instanciation du modèle échoue.

Pour les besoins de l'instanciation, la disponibilité et l'échec de chaque type d'objet sont définis comme suit :

Genre	Préparation	Échec
Build	La phase des rapports sur les objets est terminée.	L'objet signale une phase annulée, une erreur ou un échec.
BuildConfig	Le dernier objet de construction associé signale que la phase est terminée.	Le dernier objet de construction associé signale une phase annulée, une erreur ou un échec.
Deployment	L'objet signale un nouvel ensemble de répliques et un déploiement disponible. Cela permet d'honorer les sondes de disponibilité définies sur l'objet.	L'objet signale que la condition de progression est fausse.
DeploymentConfig	L'objet signale la disponibilité d'un nouveau contrôleur de réplication et d'un nouveau déploiement. Ceci honore les sondes de disponibilité définies sur l'objet.	L'objet signale que la condition de progression est fausse.
Job	L'objet signale l'achèvement.	L'objet signale qu'une ou plusieurs défaillances se sont produites.
StatefulSet	L'objet signale que toutes les répliques sont prêtes. Cela permet d'honorer les sondes de disponibilité définies sur l'objet.	Sans objet.

Voici un exemple d'extrait de modèle qui utilise l'annotation **wait-for-ready**. D'autres exemples peuvent être trouvés dans les modèles de démarrage rapide de OpenShift Container Platform.

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: ...
    annotations:
      # wait-for-ready used on BuildConfig ensures that template instantiation
      # will fail immediately if build fails
      template.alpha.openshift.io/wait-for-ready: "true"
```

```
  spec:
    ...
- kind: DeploymentConfig
  apiVersion: apps.openshift.io/v1
  metadata:
    name: ...
    annotations:
      template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: Service
  apiVersion: v1
  metadata:
    name: ...
  spec:
    ...
```

Recommandations supplémentaires

- Définissez les tailles par défaut de la mémoire, du processeur et du stockage pour vous assurer que votre application dispose de suffisamment de ressources pour fonctionner correctement.
- Évitez de faire référence à la balise **latest** à partir des images si cette balise est utilisée dans plusieurs versions majeures. Les applications en cours d'exécution risquent en effet de s'interrompre lorsque de nouvelles images sont ajoutées à cette balise.
- Un bon modèle se construit et se déploie proprement, sans nécessiter de modifications après son déploiement.

10.7.8. Création d'un modèle à partir d'objets existants

Plutôt que d'écrire un modèle complet à partir de zéro, vous pouvez exporter des objets existants de votre projet sous forme de YAML, puis modifier le YAML à partir de là en ajoutant des paramètres et d'autres personnalisations sous forme de modèle.

Procédure

- Exporter les objets d'un projet au format YAML :

  ```
  $ oc get -o yaml all > <yaml_filename>
  ```

  Vous pouvez également remplacer **all** par un type de ressource particulier ou par plusieurs ressources. Exécutez **oc get -h** pour plus d'exemples.

  Les types d'objets inclus dans **oc get -o yaml all** sont les suivants :

  - **BuildConfig**
  - **Build**
  - **DeploymentConfig**
  - **ImageStream**
  - **Pod**

- **ReplicationController**
- **Route**
- **Service**

NOTE

L'utilisation de l'alias **all** n'est pas recommandée car son contenu peut varier selon les clusters et les versions. Il est préférable de spécifier toutes les ressources nécessaires.

CHAPITRE 11. UTILISER RUBY ON RAILS

Ruby on Rails est un framework web écrit en Ruby. Ce guide couvre l'utilisation de Rails 4 sur OpenShift Container Platform.

AVERTISSEMENT

Parcourez l'ensemble du tutoriel pour avoir une vue d'ensemble de toutes les étapes nécessaires à l'exécution de votre application sur OpenShift Container Platform. Si vous rencontrez un problème, essayez de lire l'ensemble du tutoriel, puis revenez sur votre problème. Il peut également être utile de revoir les étapes précédentes pour s'assurer que toutes les étapes ont été exécutées correctement.

11.1. CONDITIONS PRÉALABLES

- Connaissances de base en Ruby et Rails.
- Version installée localement de Ruby 2.0.0, Rubygems, Bundler.
- Connaissance de base de Git.
- Instance en cours d'exécution d'OpenShift Container Platform 4.
- Assurez-vous qu'une instance d'OpenShift Container Platform est en cours d'exécution et qu'elle est disponible. Assurez-vous également que votre client **oc** CLI est installé et que la commande est accessible depuis votre shell de commande, afin que vous puissiez l'utiliser pour vous connecter à l'aide de votre adresse e-mail et de votre mot de passe.

11.2. MISE EN PLACE DE LA BASE DE DONNÉES

Les applications Rails sont presque toujours utilisées avec une base de données. Pour le développement local, utilisez la base de données PostgreSQL.

Procédure

1. Installer la base de données :

```
$ sudo yum install -y postgresql postgresql-server postgresql-devel
```

2. Initialiser la base de données :

```
$ sudo postgresql-setup initdb
```

   Cette commande crée le répertoire **/var/lib/pgsql/data**, dans lequel les données sont stockées.

3. Démarrer la base de données :

```
$ sudo systemctl start postgresql.service
```

4. Lorsque la base de données est en cours d'exécution, créez votre utilisateur **rails**:

```
$ sudo -u postgres createuser -s rails
```

Notez que l'utilisateur créé n'a pas de mot de passe.

11.3. RÉDIGER VOTRE DEMANDE

Si vous démarrez votre application Rails à partir de zéro, vous devez d'abord installer la gem Rails. Vous pourrez ensuite écrire votre application.

Procédure

1. Installer la gem Rails :

```
$ gem install rails
```

Exemple de sortie

```
Successfully installed rails-4.3.0
1 gem installed
```

2. Après avoir installé la gem Rails, créez une nouvelle application avec PostgreSQL comme base de données :

```
$ rails new rails-app --database=postgresql
```

3. Allez dans le répertoire de votre nouvelle application :

```
$ cd rails-app
```

4. Si vous avez déjà une application, assurez-vous que la gem **pg** (postgresql) est présente dans votre **Gemfile**. Si ce n'est pas le cas, modifiez votre **Gemfile** en ajoutant la gemme :

```
gem 'pg'
```

5. Générer un nouveau **Gemfile.lock** avec toutes vos dépendances :

```
$ bundle install
```

6. Outre l'utilisation de la base de données **postgresql** avec la gem **pg**, vous devez également vous assurer que **config/database.yml** utilise l'adaptateur **postgresql**.
Assurez-vous d'avoir mis à jour la section **default** dans le fichier **config/database.yml**, de manière à ce qu'elle ressemble à ceci :

```
default: &default
  adapter: postgresql
  encoding: unicode
  pool: 5
  host: localhost
  username: rails
  password:
```

7. Créez les bases de données de développement et de test de votre application :

```
$ rake db:create
```

Cette opération crée les bases de données **development** et **test** dans votre serveur PostgreSQL.

11.3.1. Création d'une page de bienvenue

Étant donné que Rails 4 ne sert plus de page statique **public/index.html** en production, vous devez créer une nouvelle page racine.

Pour créer une page de bienvenue personnalisée, il faut suivre les étapes suivantes :

- Créer un contrôleur avec une action d'indexation.
- Créez une page de visualisation pour l'action d'indexation du contrôleur de bienvenue.
- Créez une route qui dessert la page racine des applications avec le contrôleur et la vue créés.

Rails propose un générateur qui réalise toutes les étapes nécessaires pour vous.

Procédure

1. Exécuter le générateur Rails :

```
$ rails generate controller welcome index
```

Tous les fichiers nécessaires sont créés.

2. modifier la ligne 2 du fichier **config/routes.rb** comme suit :

```
root 'welcome#index'
```

3. Exécutez le serveur rails pour vérifier que la page est disponible :

```
$ rails server
```

Vous devriez voir votre page en visitant http://localhost:3000 dans votre navigateur. Si vous ne voyez pas la page, vérifiez les journaux qui sont envoyés à votre serveur pour le débogage.

11.3.2. Configuration de l'application pour OpenShift Container Platform

Pour que votre application communique avec le service de base de données PostgreSQL fonctionnant dans OpenShift Container Platform, vous devez modifier la section **default** dans votre **config/database.yml** pour utiliser les variables d'environnement, que vous devez définir plus tard, lors de la création du service de base de données.

Procédure

- Modifiez la section **default** dans votre **config/database.yml** avec des variables prédéfinies comme suit :

Exemple de fichier YAML config/database

```
<% user = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ? "root" :
ENV["POSTGRESQL_USER"] %>
<% password = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ?
ENV["POSTGRESQL_ADMIN_PASSWORD"] : ENV["POSTGRESQL_PASSWORD"] %>
<% db_service = ENV.fetch("DATABASE_SERVICE_NAME","").upcase %>

default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see rails configuration guide
  # http://guides.rubyonrails.org/configuring.html#database-pooling
  pool: <%= ENV["POSTGRESQL_MAX_CONNECTIONS"] || 5 %>
  username: <%= user %>
  password: <%= password %>
  host: <%= ENV["#{db_service}_SERVICE_HOST"] %>
  port: <%= ENV["#{db_service}_SERVICE_PORT"] %>
  database: <%= ENV["POSTGRESQL_DATABASE"] %>
```

11.3.3. Stockage de votre application dans Git

La construction d'une application dans OpenShift Container Platform nécessite généralement que le code source soit stocké dans un dépôt git, vous devez donc installer **git** si vous ne l'avez pas déjà.

Conditions préalables

- Installer git.

Procédure

1. Assurez-vous d'être dans le répertoire de votre application Rails en exécutant la commande **ls -1**. La sortie de la commande devrait ressembler à ceci :

```
$ ls -1
```

Exemple de sortie

```
app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.rdoc
test
tmp
vendor
```

2. Exécutez les commandes suivantes dans le répertoire de votre application Rails pour initialiser et valider votre code dans git :

```
$ git init
```

```
$ git add .
```

```
$ git commit -m "initial commit"
```

Une fois que votre application est validée, vous devez la pousser vers un dépôt distant. Compte GitHub, dans lequel vous créez un nouveau dépôt.

3. Définissez la télécommande qui pointe vers votre dépôt **git**:

```
$ git remote add origin git@github.com:<namespace/repository-name>.git
```

4. Poussez votre application vers votre dépôt git distant.

```
$ git push
```

11.4. DÉPLOYER VOTRE APPLICATION SUR OPENSHIFT CONTAINER PLATFORM

Vous pouvez déployer votre application sur OpenShift Container Platform.

Après avoir créé le projet **rails-app**, vous êtes automatiquement transféré dans le nouvel espace de noms du projet.

Le déploiement de votre application dans OpenShift Container Platform se fait en trois étapes :

- Création d'un service de base de données à partir de l'image PostgreSQL d'OpenShift Container Platform.
- Création d'un service frontal à partir de l'image de construction Ruby 2.0 d'OpenShift Container Platform et de votre code source Ruby on Rails, qui sont reliés au service de base de données.
- Création d'un itinéraire pour votre application.

Procédure

- Pour déployer votre application Ruby on Rails, créez un nouveau projet pour l'application :

```
$ oc new-project rails-app --description="My Rails application" --display-name="Rails
Application"
```

11.4.1. Création du service de base de données

Votre application Rails s'attend à ce qu'un service de base de données soit en cours d'exécution. Pour ce service, utilisez l'image de la base de données PostgreSQL.

Pour créer le service de base de données, utilisez la commande **oc new-app**. À cette commande, vous devez transmettre certaines variables d'environnement nécessaires qui sont utilisées dans le conteneur de la base de données. Ces variables d'environnement sont nécessaires pour définir le nom d'utilisateur, le mot de passe et le nom de la base de données. Vous pouvez modifier les valeurs de ces variables d'environnement comme bon vous semble. Les variables sont les suivantes :

- BASE DE DONNÉES POSTGRESQL_
- POSTGRESQL_USER
- MOT DE PASSE POSTGRESQL_

La définition de ces variables garantit :

- Il existe une base de données portant le nom spécifié.
- Il existe un utilisateur portant le nom spécifié.
- L'utilisateur peut accéder à la base de données spécifiée avec le mot de passe spécifié.

Procédure

1. Créer le service de base de données :

```
$ oc new-app postgresql -e POSTGRESQL_DATABASE=db_name -e
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password
```

   Pour définir également le mot de passe de l'administrateur de la base de données, ajoutez à la commande précédente le texte suivant :

```
-e POSTGRESQL_ADMIN_PASSWORD=admin_pw
```

2. Observez les progrès :

```
$ oc get pods --watch
```

11.4.2. Création du service frontal

Pour apporter votre application à OpenShift Container Platform, vous devez spécifier un dépôt dans lequel votre application vit.

Procédure

1. Créez le service frontal et spécifiez les variables d'environnement liées à la base de données qui ont été définies lors de la création du service de base de données :

```
$ oc new-app path/to/source/code --name=rails-app -e POSTGRESQL_USER=username -e
POSTGRESQL_PASSWORD=password -e POSTGRESQL_DATABASE=db_name -e
DATABASE_SERVICE_NAME=postgresql
```

   Avec cette commande, OpenShift Container Platform récupère le code source, configure le constructeur, construit l'image de votre application et déploie l'image nouvellement créée avec les variables d'environnement spécifiées. L'application s'appelle **rails-app**.

2. Vérifiez que les variables d'environnement ont été ajoutées en consultant le document JSON de la configuration du déploiement de **rails-app**:

```
$ oc get dc rails-app -o json
```

   La section suivante devrait s'afficher :

Exemple de sortie

```
env": [
    {
        "name": "POSTGRESQL_USER",
        "value": "username"
    },
    {
        "name": "POSTGRESQL_PASSWORD",
        "value": "password"
    },
    {
        "name": "POSTGRESQL_DATABASE",
        "value": "db_name"
    },
    {
        "name": "DATABASE_SERVICE_NAME",
        "value": "postgresql"
    }

],
```

3. Vérifier le processus de construction :

```
$ oc logs -f build/rails-app-1
```

4. Une fois la construction terminée, regardez les pods en cours d'exécution dans OpenShift Container Platform :

```
$ oc get pods
```

Vous devriez voir une ligne commençant par **myapp-<number>-<hash>**, et c'est votre application qui s'exécute dans OpenShift Container Platform.

5. Avant que votre application ne soit fonctionnelle, vous devez initialiser la base de données en exécutant le script de migration de la base de données. Il y a deux façons de procéder :

- Manuellement à partir du conteneur frontal en cours d'exécution :
  - Exécuter dans le conteneur frontal avec la commande **rsh**:

    ```
    oc rsh <frontend_pod_id>
    ```

  - Exécuter la migration depuis l'intérieur du conteneur :

    ```
    $ RAILS_ENV=production bundle exec rake db:migrate
    ```

    Si vous exécutez votre application Rails dans un environnement **development** ou **test**, vous ne devez pas spécifier la variable d'environnement **RAILS_ENV**.

- En ajoutant des crochets de cycle de vie de pré-déploiement dans votre modèle.

11.4.3. Créer un itinéraire pour votre application

Vous pouvez exposer un service pour créer une route pour votre application.

Procédure

- Pour exposer un service en lui donnant un nom d'hôte accessible de l'extérieur comme **www.example.com**, utilisez OpenShift Container Platform route. Dans votre cas, vous devez exposer le service frontal en tapant :

```
$ oc expose service rails-app --hostname=www.example.com
```

AVERTISSEMENT

Assurez-vous que le nom d'hôte que vous spécifiez se résout dans l'adresse IP du routeur.

CHAPITRE 12. UTILISATION D'IMAGES

12.1. VUE D'ENSEMBLE DE L'UTILISATION DES IMAGES

Utilisez les rubriques suivantes pour découvrir les différentes images Source-to-Image (S2I), les bases de données et les autres images de conteneurs disponibles pour les utilisateurs d'OpenShift Container Platform.

Les images de conteneurs officielles de Red Hat sont fournies dans le Red Hat Registry à l'adresse registry .redhat.io. Les images S2I, base de données et Jenkins prises en charge par OpenShift Container Platform sont fournies dans le référentiel **openshift4** dans le Red Hat Quay Registry. Par exemple, **quay.io/openshift-release-dev/ocp-v4.0-<address>** est le nom de l'image OpenShift Application Platform.

Les images des intergiciels xPaaS sont fournies dans leurs référentiels de produits respectifs sur le Red Hat Registry, mais elles sont suffixées par un **-openshift**. Par exemple, **registry.redhat.io/jboss-eap-6/eap64-openshift** est le nom de l'image JBoss EAP.

Toutes les images prises en charge par Red Hat et couvertes dans cette section sont décrites dans la section Images de conteneurs du catalogue de l'écosystème Red Hat . Pour chaque version de chaque image, vous pouvez trouver des détails sur son contenu et son utilisation. Parcourez ou recherchez l'image qui vous intéresse.

IMPORTANT

Les nouvelles versions des images de conteneurs ne sont pas compatibles avec les versions antérieures d'OpenShift Container Platform. Vérifiez et utilisez la version correcte des images de conteneurs, en fonction de votre version d'OpenShift Container Platform.

12.2. DE LA SOURCE À L'IMAGE

Vous pouvez utiliser les images Red Hat Software Collections comme base pour les applications qui dépendent d'environnements d'exécution spécifiques tels que Node.js, Perl ou Python. Vous pouvez utiliser la documentation Red Hat Java Source-to-Image for OpenShift comme référence pour les environnements d'exécution qui utilisent Java. Des versions spéciales de certaines de ces images de base d'exécution sont appelées images Source-to-Image (S2I). Avec les images S2I, vous pouvez insérer votre code dans un environnement d'image de base qui est prêt à exécuter ce code.

Les images de S2I comprennent :

- .NET
- Java
- Aller
- Node.js
- Perl
- PHP
- Python

- Rubis

Les images S2I sont disponibles pour être utilisées directement depuis la console web d'OpenShift Container Platform en suivant la procédure suivante :

1. Connectez-vous à la console web d'OpenShift Container Platform en utilisant vos identifiants de connexion. La vue par défaut de la console web d'OpenShift Container Platform est la perspective **Administrator**.
2. Utilisez le sélecteur de perspective pour passer à la perspective **Developer**.
3. Dans la vue **Add**, sélectionnez un projet existant dans la liste ou utilisez la liste déroulante **Project** pour créer un nouveau projet.
4. Choisissez **All services** sous la tuile **Developer Catalog**.
5. Sélectionnez le type **Builder Images** et voyez les images S2I disponibles.

Les images S2I sont également disponibles via l'opérateur Configuring the Cluster Samples .

12.2.1. Vue d'ensemble du processus de construction de la source à l'image

Source-to-image (S2I) produit des images prêtes à être exécutées en injectant du code source dans un conteneur qui prépare ce code source à être exécuté. Il exécute les étapes suivantes :

1. Exécute la commande **FROM <builder image>**
2. Copie le code source à un emplacement défini dans l'image du constructeur
3. Exécute le script d'assemblage dans l'image du constructeur
4. Définit le script d'exécution dans l'image du constructeur comme la commande par défaut

Buildah crée ensuite l'image du conteneur.

12.2.2. Ressources complémentaires

- Pour obtenir des instructions sur l'utilisation de l'opérateur d'échantillonnage de grappes, consultez la section Configuration de l'opérateur d'échantillonnage de grappes.
- Pour plus d'informations sur les builds S2I, voir la documentation de la stratégie de builds sur les builds S2I.
- Pour obtenir une aide au dépannage du processus S2I, voir Dépannage du processus Source-to-Image.
- Pour une vue d'ensemble de la création d'images avec S2I, voir Création d'images à partir du code source avec source-to-image.
- Pour une vue d'ensemble des tests des images S2I, voir À propos des tests des images S2I.

12.3. PERSONNALISATION DES IMAGES SOURCE-IMAGE

Les images de construction source-image (S2I) comprennent des scripts d'assemblage et d'exécution, mais le comportement par défaut de ces scripts n'est pas adapté à tous les utilisateurs. Vous pouvez personnaliser le comportement d'un constructeur S2I qui inclut des scripts par défaut.

12.3.1. Invoquer des scripts intégrés dans une image

Les images de construction fournissent leur propre version des scripts source-image (S2I) qui couvrent les cas d'utilisation les plus courants. Si ces scripts ne répondent pas à vos besoins, S2I permet de les remplacer en ajoutant des scripts personnalisés dans le répertoire **.s2i/bin**. Toutefois, ce faisant, vous remplacez complètement les scripts standard. Dans certains cas, le remplacement des scripts est acceptable, mais, dans d'autres scénarios, vous pouvez exécuter quelques commandes avant ou après les scripts tout en conservant la logique du script fourni dans l'image. Pour réutiliser les scripts standard, vous pouvez créer un script enveloppant qui exécute une logique personnalisée et délègue la suite du travail aux scripts par défaut de l'image.

Procédure

1. Regardez la valeur de l'étiquette **io.openshift.s2i.scripts-url** pour déterminer l'emplacement des scripts à l'intérieur de l'image du constructeur :

```
$ podman inspect --format='{{ index .Config.Labels "io.openshift.s2i.scripts-url" }}'
wildfly/wildfly-centos7
```

Exemple de sortie

```
image:///usr/libexec/s2i
```

Vous avez inspecté l'image du constructeur **wildfly/wildfly-centos7** et découvert que les scripts se trouvent dans le répertoire **/usr/libexec/s2i**.

2. Créez un script qui comprend une invocation de l'un des scripts standard enveloppée dans d'autres commandes :

.s2i/bin/assemble scénario

```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "After successful assembling"
else
    echo "After failed assembling"
fi

exit $rc
```

Cet exemple montre un script d'assemblage personnalisé qui imprime le message, exécute le script d'assemblage standard à partir de l'image et imprime un autre message en fonction du code de sortie du script d'assemblage.

IMPORTANT

Lorsque vous enveloppez le script d'exécution, vous devez utiliser **exec** pour l'invoquer afin de vous assurer que les signaux sont traités correctement. L'utilisation de **exec** empêche également d'exécuter des commandes supplémentaires après avoir invoqué le script d'exécution de l'image par défaut.

.s2i/bin/run scénario

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```