OpenShift Dedicated 4

Construits à l’aide de BuildConfig

Contient des informations sur les builds pour OpenShift Dedicated

Last Updated: 2025-05-27

OpenShift Dedicated 4 Construits à l'aide de BuildConfig

Contient des informations sur les builds pour OpenShift Dedicated

Notice légale

Résumé

Constructions pour OpenShift Dedicated clusters.

Table des matières

CHAPITRE 1. COMPRENDRE LES CONSTRUCTIONS D'IMAGES

1.1. CONSTRUCTIONS

La construction est le processus de transformation des paramètres d'entrée en un objet résultant. Le plus souvent, le processus est utilisé pour transformer les paramètres d'entrée ou le code source en une image exécutable. L'objet BuildConfig est la définition de l'ensemble du processus de construction.

La société OpenShift Dedicated utilise Kubernetes en créant des conteneurs à partir de la création d'images et en les poussant vers un registre d'images de conteneur.

Les objets de construction partagent des caractéristiques communes, y compris les entrées pour une construction, l'exigence de compléter un processus de construction, l'enregistrement du processus de construction, la publication de ressources provenant de constructions réussies et la publication du statut final de la construction. Les builds profitent des restrictions de ressources, spécifiant des limites sur les ressources telles que l'utilisation du CPU, l'utilisation de la mémoire et le temps d'exécution de la construction ou du pod.

L'objet résultant d'une construction dépend du constructeur utilisé pour la créer. Dans le cas des constructions docker et S2I, les objets qui en résultent sont des images exécutables. Dans le cas des constructions personnalisées, les objets résultants sont ce que l'auteur de l'image du constructeur a spécifié.

En outre, la stratégie de construction de pipelines peut être utilisée pour mettre en œuvre des flux de travail sophistiqués:

- Intégration continue
- Déploiement continu

1.1.1. Construction de Docker

Le logiciel OpenShift Dedicated utilise Buildah pour construire une image conteneur à partir d'un Dockerfile. Consultez la documentation de référence Dockerfile pour plus d'informations sur les images de conteneurs de construction avec Dockerfile.

ASTUCE

Lorsque vous définissez les arguments de construction Docker à l'aide du tableau buildArgs, voir Comprendre comment ARG et FROM interagissent dans la documentation de référence Dockerfile.

1.1.2. Construction de source à image

La source à l'image (S2I) est un outil pour construire des images de conteneurs reproductibles. Il produit des images prêtes à l'emploi en injectant une source d'application dans une image de conteneur et en assemblant une nouvelle image. La nouvelle image intègre l'image de base, le constructeur et la source construite et est prête à être utilisée avec la commande buildah run. Le S2I prend en charge les constructions incrémentielles, qui réutilisent des dépendances précédemment téléchargées, des artefacts précédemment construits, et ainsi de suite.

1.1.3. Construction de pipelines

IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

La stratégie de construction Pipeline permet aux développeurs de définir un pipeline Jenkins pour une utilisation par le plugin pipeline Jenkins. La construction peut être démarrée, surveillée et gérée par OpenShift Dedicated de la même manière que n'importe quel autre type de construction.

Les flux de travail de pipeline sont définis dans un fichier jenkins, soit intégrés directement dans la configuration de construction, soit fournis dans un référentiel Git et référencés par la configuration de construction.

CHAPITRE 2. COMPRENDRE LES CONFIGURATIONS DE CONSTRUCTION

Les sections suivantes définissent le concept de construction, de configuration de construction et décrivent les stratégies de construction primaires disponibles.

2.1. BUILDCONFIGS

La configuration de build décrit une définition de build unique et un ensemble de déclencheurs pour le moment où une nouvelle construction est créée. Les configurations de build sont définies par un BuildConfig, qui est un objet REST qui peut être utilisé dans un POST vers le serveur API pour créer une nouvelle instance.

La configuration de build, ou BuildConfig, se caractérise par une stratégie de build et une ou plusieurs sources. La stratégie détermine le processus, tandis que les sources fournissent sa contribution.

En fonction de la façon dont vous choisissez de créer votre application en utilisant OpenShift Dedicated, un BuildConfig est généralement généré automatiquement pour vous si vous utilisez la console Web ou CLI, et il peut être modifié à tout moment. Comprendre les pièces qui composent un BuildConfig et leurs options disponibles peut vous aider si vous choisissez de modifier manuellement votre configuration plus tard.

L'exemple suivant BuildConfig se traduit par une nouvelle construction chaque fois qu'une balise d'image conteneur ou le code source change:

Définition d'objet BuildConfig

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" 1
spec:
  runPolicy: "Serial" 2
  triggers: 3
    -
      type: "GitHub"
      github:
        secret: "secret101"
    - type: "Generic"
      generic:
        secret: "secret101"
    -
      type: "ImageChange"
  source: 4
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy: 5
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "ruby-20-centos7:latest"
  output: 6
    to:
      kind: "ImageStreamTag"
```

```
    name: "origin-ruby-sample:latest"
postCommit: 7
    script: "bundle exec rake test"
```

1. Cette spécification crée un nouveau BuildConfig nommé ruby-échantillon-build.
2. Le champ runPolicy contrôle si les builds créés à partir de cette configuration de construction peuvent être exécutés simultanément. La valeur par défaut est Serial, ce qui signifie que les nouvelles versions s'exécutent de manière séquentielle, pas simultanément.
3. Il est possible de spécifier une liste de déclencheurs, ce qui provoque la création d'une nouvelle version.
4. La section source définit la source de la construction. Le type source détermine la source principale d'entrée, et peut être soit Git, pour pointer vers un emplacement de référentiel de code, Dockerfile, à construire à partir d'un Dockerfile en ligne, ou Binary, pour accepter les charges utiles binaires. Il est possible d'avoir plusieurs sources à la fois. Consultez la documentation pour chaque type de source pour plus de détails.
5. La section stratégie décrit la stratégie de construction utilisée pour exécuter le build. Ici, vous pouvez spécifier une stratégie Source, Docker ou Custom. Cet exemple utilise l'image de conteneur ruby-20-centos7 que Source-à-image (S2I) utilise pour la construction de l'application.
6. Après que l'image du conteneur est construite avec succès, elle est poussée dans le référentiel décrit dans la section de sortie.
7. La section postCommit définit un crochet de construction optionnel.

CHAPITRE 3. CRÉATION D'ENTRÉES DE BUILD

Consultez les sections suivantes pour un aperçu des entrées de construction, des instructions sur la façon d'utiliser les entrées pour fournir du contenu source aux builds sur lesquels fonctionner, et comment utiliser des environnements de construction et créer des secrets.

3.1. CONSTRUIRE DES ENTRÉES

L'entrée de build fournit du contenu source sur lequel les builds peuvent fonctionner. Les entrées de build suivantes vous permettent de fournir des sources dans OpenShift Dedicated, listées par ordre de priorité:

- Définitions de Dockerfile en ligne
- Contenu extrait des images existantes
- Dépôts Git
- Entrées binaires (locales)
- Les secrets d'entrée
- Artefacts externes

Il est possible de combiner plusieurs entrées en une seule version. Cependant, comme le Dockerfile en ligne a préséance, il peut écraser n'importe quel autre fichier nommé Dockerfile fourni par une autre entrée. Les dépôts binaires (local) et Git sont des entrées mutuellement exclusives.

Il est possible d'utiliser des secrets d'entrée lorsque vous ne voulez pas que certaines ressources ou informations d'identification utilisées lors d'une construction soient disponibles dans l'image de l'application finale produite par la construction, ou que vous souhaitez consommer une valeur définie dans une ressource secrète. Des artefacts externes peuvent être utilisés pour extraire des fichiers supplémentaires qui ne sont pas disponibles comme l'un des autres types d'entrée de build.

Lorsque vous exécutez une construction:

1. Le répertoire de travail est construit et tout le contenu d'entrée est placé dans le répertoire de travail. À titre d'exemple, le référentiel Git d'entrée est cloné dans le répertoire de travail, et les fichiers spécifiés à partir d'images d'entrée sont copiés dans le répertoire de travail en utilisant le chemin cible.
2. Le processus de construction modifie les répertoires dans le contexteDir, si l'on est défini.
3. Le Dockerfile en ligne, s'il y en a, est écrit dans le répertoire actuel.
4. Le contenu du répertoire actuel est fourni au processus de construction pour référence par le Dockerfile, la logique du constructeur personnalisé ou le script assembler. Cela signifie que tout contenu d'entrée qui réside en dehors du contexteDir est ignoré par la build.

L'exemple suivant d'une définition de source comprend plusieurs types d'entrée et une explication de la façon dont ils sont combinés. Consultez les sections spécifiques pour chaque type d'entrée pour plus de détails sur la façon dont chaque type d'entrée est défini.

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git (1)
```

```
    ref: "master"
  images:
  - from:
      kind: ImageStreamTag
      name: myinputimage:latest
      namespace: mynamespace
    paths:
    - destinationDir: app/dir/injected/dir 2
      sourcePath: /usr/lib/somefile.jar
  contextDir: "app/dir" 3
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" 4
```

1. Le référentiel à cloner dans le répertoire de travail pour la construction.
2. /usr/lib/somefile.jar de myinputimage est stocké dans <workingdir>/app/dir/injected/dir.
3. Le répertoire de travail pour la construction devient <original_workingdir>/app/dir.
4. Dockerfile avec ce contenu est créé dans <original_workingdir>/app/dir, écraser tout fichier existant avec ce nom.

3.2. DOCKERFILE SOURCE

Lorsque vous fournissez une valeur dockerfile, le contenu de ce champ est écrit sur le disque en tant que fichier nommé dockerfile. Cela se fait après le traitement d'autres sources d'entrée, donc si le référentiel source d'entrée contient un Dockerfile dans le répertoire racine, il est écrasé avec ce contenu.

La définition de la source fait partie de la section Spécification dans le BuildConfig:

```
source:
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" 1
```

1. Le champ dockerfile contient un Dockerfile intégré qui est construit.

Ressources supplémentaires

- L'utilisation typique de ce champ est de fournir un Dockerfile à une construction de stratégie docker.

3.3. IMAGE SOURCE

Il est possible d'ajouter des fichiers supplémentaires au processus de construction avec des images. Les images d'entrée sont référencées de la même manière que les cibles From et To image sont définies. Cela signifie que les images de conteneur et les balises de flux d'images peuvent être référencées. En conjonction avec l'image, vous devez fournir une ou plusieurs paires de chemin pour indiquer le chemin des fichiers ou des répertoires pour copier l'image et la destination pour les placer dans le contexte de construction.

Le chemin source peut être n'importe quel chemin absolu dans l'image spécifiée. La destination doit être un chemin d'annuaire relatif. Au moment de la construction, l'image est chargée et les fichiers et répertoires indiqués sont copiés dans le répertoire contextuel du processus de construction. C'est le

même répertoire dans lequel le contenu du référentiel source est cloné. Lorsque le chemin source se termine dans /. alors le contenu du répertoire est copié, mais le répertoire lui-même n'est pas créé à la destination.

Les entrées d'image sont spécifiées dans la définition source du BuildConfig:

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git
    ref: "master"
  images: 1
  - from: 2
      kind: ImageStreamTag
      name: myinputimage:latest
      namespace: mynamespace
    paths: 3
    - destinationDir: injected/dir 4
      sourcePath: /usr/lib/somefile.jar 5
  - from:
      kind: ImageStreamTag
      name: myotherinputimage:latest
      namespace: myothernamespace
    pullSecret: mysecret 6
    paths:
    - destinationDir: injected/dir
      sourcePath: /usr/lib/somefile.jar
```

1. Ensemble d'une ou de plusieurs images et fichiers d'entrée.
2. La référence à l'image contenant les fichiers à copier.
3. Ensemble de chemins source/destination.
4. Le répertoire relatif à la racine de construction où le processus de construction peut accéder au fichier.
5. L'emplacement du fichier à copier à partir de l'image référencée.
6. Le secret facultatif est fourni si des informations d'identification sont nécessaires pour accéder à l'image d'entrée.

NOTE

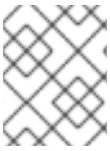
Lorsque votre cluster utilise un objet ImageDigestMirrorSet, ImageTagMirrorSet ou ImageContentSourcePolicy pour configurer la mise en miroir de référentiels, vous pouvez utiliser uniquement des secrets de traction globaux pour les registres miroirs. Il est impossible d'ajouter un secret d'attraction à un projet.

Images qui nécessitent des secrets de traction

Lorsque vous utilisez une image d'entrée qui nécessite un secret de traction, vous pouvez lier le secret de traction au compte de service utilisé par la construction. Les builds utilisent par défaut le compte de service constructeur. Le secret de traction est automatiquement ajouté à la construction si le secret

contient un identifiant qui correspond au référentiel hébergeant l'image d'entrée. Afin de lier un pull secret au compte de service utilisé par la construction, exécutez:

```
$ oc secrets link builder dockerhub
```



NOTE

Cette fonctionnalité n'est pas prise en charge pour les builds utilisant la stratégie personnalisée.

Images sur des registres miroirs qui nécessitent des secrets de tirage

Lorsque vous utilisez une image d'entrée à partir d'un registre en miroir, si vous obtenez une erreur de construction: ne pas tirer le message image, vous pouvez résoudre l'erreur en utilisant l'une des méthodes suivantes:

- Créez un secret d'entrée qui contient les informations d'authentification pour le référentiel de l'image du constructeur et tous les miroirs connus. Dans ce cas, créez un secret d'attraction pour les informations d'identification du registre des images et de ses miroirs.
- Le secret d'entrée est utilisé comme secret d'attraction sur l'objet BuildConfig.

3.4. GIT SOURCE

Lorsqu'il est spécifié, le code source est récupéré à partir de l'emplacement fourni.

Lorsque vous fournissez un Dockerfile en ligne, il écrase le Dockerfile dans le contexteDir du référentiel Git.

La définition de la source fait partie de la section Spécification dans le BuildConfig:

```
source:
  git: 1
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
  contextDir: "app/dir" 2
  dockerfile: "FROM openshift/ruby-22-centos7\nUSER example" 3
```

1. Le champ git contient l'identifiant de ressource uniforme (URI) dans le référentiel Git distant du code source. Il faut spécifier la valeur du champ ref pour vérifier une référence Git spécifique. Il peut s'agir d'une balise SHA1 ou d'un nom de branche. La valeur par défaut du champ ref est master.
2. Le champ contextDir vous permet de remplacer l'emplacement par défaut à l'intérieur du référentiel de code source où la build recherche le code source de l'application. Lorsque votre application existe à l'intérieur d'un sous-répertoire, vous pouvez remplacer l'emplacement par défaut (le dossier racine) en utilisant ce champ.
3. Lorsque le champ dockerfile optionnel est fourni, il devrait s'agir d'une chaîne contenant un Dockerfile qui écrase tout Dockerfile pouvant exister dans le référentiel source.

Dans le cas où le champ ref indique une requête de traction, le système utilise une opération de récupération de git, puis la vérification FETCH_HEAD.

Lorsqu'aucune valeur réf n'est fournie, OpenShift Dedicated effectue un clone peu profond (--profondeur=1). Dans ce cas, seuls les fichiers associés au plus récent commit sur la branche par défaut (généralement maître) sont téléchargés. Cela se traduit par le téléchargement des dépôts plus rapidement, mais sans l'historique complet de l'engagement. Afin d'effectuer un clone git complet de la branche par défaut d'un référentiel spécifié, définissez ref au nom de la branche par défaut (par exemple principal).

AVERTISSEMENT

Les opérations de clone de Git qui passent par un proxy qui exécute l'homme au milieu (MITM) TLS détournement ou recryptage de la connexion proxiée ne fonctionnent pas.

3.4.1. À l'aide d'un proxy

Lorsque votre dépôt Git ne peut être consulté qu'à l'aide d'un proxy, vous pouvez définir le proxy à utiliser dans la section source de la configuration de construction. Il est possible de configurer un proxy HTTP et HTTPS à utiliser. Les deux champs sont facultatifs. Les domaines pour lesquels aucun proxying ne doit être effectué peuvent également être spécifiés dans le champ NoProxy.

NOTE

L'URI source doit utiliser le protocole HTTP ou HTTPS pour que cela fonctionne.

```
source:
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
    ref: "master"
    httpProxy: http://proxy.example.com
    httpsProxy: https://proxy.example.com
    noProxy: somedomain.com, otherdomain.com
```

NOTE

En ce qui concerne les constructions de stratégie Pipeline, compte tenu des restrictions actuelles avec le plugin Git pour Jenkins, toutes les opérations Git via le plugin Git ne tirent pas parti du proxy HTTP ou HTTPS défini dans le BuildConfig. Le plugin Git utilise uniquement le proxy configuré dans l'interface utilisateur Jenkins dans le panneau Gestionnaire de plugins. Ce proxy est ensuite utilisé pour toutes les interactions git au sein de Jenkins, dans tous les emplois.

Ressources supplémentaires

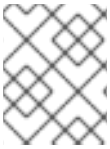
- À JenkinsBehindProxy, vous trouverez des instructions sur la configuration des proxys via l'interface utilisateur Jenkins.

3.4.2. Les secrets de Clone

Les pods de builder nécessitent l'accès à tous les référentiels Git définis comme source pour une construction. Les secrets de clone source sont utilisés pour fournir à la pod constructeur l'accès auquel il n'aurait normalement pas accès, tels que des dépôts privés ou des dépôts avec des certificats SSL autosignés ou non fiables.

Les configurations secrètes de clone source suivantes sont prises en charge:

- Fichier .gitconfig
- Authentification de base
- Authentification des clés SSH
- Autorités de certification de confiance



NOTE

Il est également possible d'utiliser des combinaisons de ces configurations pour répondre à vos besoins spécifiques.

3.4.2.1. Ajout automatique d'un secret de clone source à une configuration de build

Lorsqu'un BuildConfig est créé, OpenShift Dedicated peut remplir automatiquement sa référence secrète de clone source. Ce comportement permet aux builds résultants d'utiliser automatiquement les informations d'identification stockées dans le secret référencé pour s'authentifier vers un référentiel Git distant, sans nécessiter de configuration supplémentaire.

Afin d'utiliser cette fonctionnalité, un secret contenant les informations d'identification du référentiel Git doit exister dans l'espace de noms dans lequel le BuildConfig est créé ultérieurement. Ces secrets doivent inclure une ou plusieurs annotations préfixées avec build.openshift.io/source-secret-match-uri- . La valeur de chacune de ces annotations est un modèle Uniform Resource Identifier (URI), qui est défini comme suit. Lorsqu'un BuildConfig est créé sans référence secrète de clone source et que son URI source Git correspond à un modèle URI dans une annotation secrète, OpenShift Dedicated insère automatiquement une référence à ce secret dans le BuildConfig.

Conditions préalables

Le modèle URI doit consister en:

- A schéma valide: *://, git://, http://, https:/ ou ssh:///
- Hôte: *' ou un nom d'hôte valide ou une adresse IP éventuellement précédée de *.
- Chemin: /* ou / suivi de tous les caractères optionnellement incluant * caractères

Dans tout ce qui précède, un caractère * est interprété comme un wildcard.

IMPORTANT

Les modèles URI doivent correspondre aux URI source Git qui sont conformes à la RFC3986. Il ne faut pas inclure un composant nom d'utilisateur (ou mot de passe) dans un modèle URI.

Ainsi, si vous utilisez ssh://git@bitbucket.atlassian.com:7999/ATLASSIAN jira.git pour une URL de dépôt git, le secret source doit être spécifié comme ssh://bitbucket.atlassian.com:7999/* (et non ssh://git@bitbucket.atlassian.com:7999/*).

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=ssh://bitbucket.atlassian.com:7999/*'
```

Procédure

Lorsque plusieurs secrets correspondent à l'URI Git d'un BuildConfig particulier, OpenShift Dedicated sélectionne le secret avec le plus long match. Cela permet l'écrasement de base, comme dans l'exemple suivant.

Le fragment suivant montre deux secrets de clones sources partiels, le premier correspondant à n'importe quel serveur dans le domaine mycorp.com accessible par HTTPS, et le second accès principal aux serveurs mydev1.mycorp.com et mydev2.mycorp.com:

```
kind: Secret
apiVersion: v1
metadata:
  name: matches-all-corporate-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://*.mycorp.com/*
data:
  ...
---
kind: Secret
apiVersion: v1
metadata:
  name: override-for-my-dev-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://mydev1.mycorp.com/*
    build.openshift.io/source-secret-match-uri-2: https://mydev2.mycorp.com/*
data:
  ...
```

- Ajouter une annotation build.openshift.io/source-secret-match-uri- à un secret préexistant en utilisant:

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=https://*.mycorp.com/*'
```

3.4.2.2. Ajout manuel d'un clone source secret

Les secrets de clone source peuvent être ajoutés manuellement à une configuration de build en ajoutant un champ sourceSecret à la section source à l'intérieur du BuildConfig et le paramétrer au nom du secret que vous avez créé. Dans cet exemple, c'est le secret de base.

```
apiVersion: "build.openshift.io/v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
  source:
    git:
      uri: "https://github.com/user/app.git"
    sourceSecret:
      name: "basicsecret"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "python-33-centos7:latest"
```

Procédure

Il est également possible d'utiliser la commande oc set build-secret pour définir le secret du clone source sur une configuration de build existante.

- Afin de définir le secret du clone source sur une configuration de build existante, entrez la commande suivante:

```
$ oc set build-secret --source bc/sample-build basicsecret
```

3.4.2.3. Créer un secret à partir d'un fichier .gitconfig

Lorsque le clonage de votre application dépend d'un fichier .gitconfig, vous pouvez créer un secret qui le contient. Ajoutez-le au compte de service constructeur, puis à votre BuildConfig.

Procédure

- Créer un secret à partir d'un fichier .gitconfig:

```
$ oc create secret generic <secret_name> --from-file=<path/to/.gitconfig>
```

NOTE

La vérification SSL peut être désactivée si sslVerify=false est défini pour la section http dans votre fichier .gitconfig:

```
[http]
        sslVerify=false
```

3.4.2.4. Créer un secret à partir d'un fichier .gitconfig pour Git sécurisé

Lorsque votre serveur Git est sécurisé avec SSL bidirectionnel et nom d'utilisateur avec mot de passe, vous devez ajouter les fichiers de certificat à votre création de source et ajouter des références aux fichiers de certificat dans le fichier .gitconfig.

Conditions préalables

- Il faut avoir des informations d'identification Git.

Procédure

Ajoutez les fichiers de certificat à votre source de création et ajoutez des références aux fichiers de certificat dans le fichier .gitconfig.

1. Ajoutez les fichiers client.crt, cacert.crt et client.key dans le dossier /var/run/secrets/openshift.io/source/ dans le code source de l'application.

2. Dans le fichier .gitconfig pour le serveur, ajoutez la section [http] affichée dans l'exemple suivant:

```
# cat .gitconfig
```

Exemple de sortie

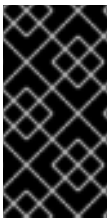
```
[user]
        name = <name>
        email = <email>
[http]
        sslVerify = false
        sslCert = /var/run/secrets/openshift.io/source/client.crt
        sslKey = /var/run/secrets/openshift.io/source/client.key
        sslCaInfo = /var/run/secrets/openshift.io/source/cacert.crt
```

3. Créer le secret:

```
$ oc create secret generic <secret_name> \
--from-literal=username=<user_name> \ 1
--from-literal=password=<password> \ 2
--from-file=.gitconfig=.gitconfig \
--from-file=client.crt=/var/run/secrets/openshift.io/source/client.crt \
--from-file=cacert.crt=/var/run/secrets/openshift.io/source/cacert.crt \
--from-file=client.key=/var/run/secrets/openshift.io/source/client.key
```

1. Le nom d'utilisateur Git de l'utilisateur.

2. Le mot de passe pour cet utilisateur.



IMPORTANT

Afin d'éviter d'avoir à saisir à nouveau votre mot de passe, assurez-vous de spécifier l'image source à image (S2I) dans vos builds. Cependant, si vous ne pouvez pas cloner le référentiel, vous devez toujours spécifier votre nom d'utilisateur et votre mot de passe pour promouvoir la construction.

Ressources supplémentaires

- /Var/run/secrets/openshift.io/source/ dossier dans le code source de l'application.

3.4.2.5. Créer un secret à partir de l'authentification de base du code source

L'authentification de base nécessite soit une combinaison de --username et --password, soit un jeton à authentifier par rapport au serveur de gestion de la configuration du logiciel (SCM).

Conditions préalables

- Le nom d'utilisateur et le mot de passe pour accéder au référentiel privé.

Procédure

1. Créez le secret d'abord avant d'utiliser le --username et --password pour accéder au référentiel privé:

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --type=kubernetes.io/basic-auth
```

2. Créer un secret d'authentification de base avec un jeton:

```
$ oc create secret generic <secret_name> \
    --from-literal=password=<token> \
    --type=kubernetes.io/basic-auth
```

3.4.2.6. Création d'un secret à partir du code source authentification de clé SSH

L'authentification basée sur les clés SSH nécessite une clé SSH privée.

Les clés de dépôt sont généralement situées dans le répertoire $HOME/.ssh/, et sont nommées id_dsa.pub, id_ecdsa.pub, id_ed25519.pub, ou id_rsa.pub par défaut.

Procédure

1. Générer des identifiants clés SSH:

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

NOTE

La création d'une phrase de passe pour la clé SSH empêche OpenShift Dedicated de construire. Lorsqu'il est demandé une phrase de passe, laissez-le vide.

Deux fichiers sont créés: la clé publique et une clé privée correspondante (un de id_dsa, id_ecdsa, id_ed25519 ou id_rsa). Avec ces deux en place, consultez le manuel de votre système de gestion du contrôle source (SCM) sur la façon de télécharger la clé publique. La clé privée est utilisée pour accéder à votre dépôt privé.

2. Avant d'utiliser la clé SSH pour accéder au référentiel privé, créez le secret:

```
$ oc create secret generic <secret_name> \
    --from-file=ssh-privatekey=<path/to/ssh/private/key> \
    --from-file=<path/to/known_hosts> \ 1
    --type=kubernetes.io/ssh-auth
```

1 Facultatif : L'ajout de ce champ permet une vérification stricte de la clé hôte du serveur.

AVERTISSEMENT

En sautant le fichier connu_hosts tout en créant le secret, la construction est vulnérable à une attaque potentielle de l'homme dans le milieu (MITM).

NOTE

Assurez-vous que le fichierknown_hosts inclut une entrée pour l'hôte de votre code source.

3.4.2.7. Créer un secret à partir des autorités de certification fiables du code source

L'ensemble des autorités de certification Transport Layer Security (TLS) qui sont fiables lors d'une opération de clone Git sont intégrés dans les images d'infrastructure dédiées à OpenShift. Lorsque votre serveur Git utilise un certificat autosigné ou signé par une autorité qui ne fait pas confiance à l'image, vous pouvez créer un secret qui contient le certificat ou désactiver la vérification TLS.

Lorsque vous créez un secret pour le certificat CA, OpenShift Dedicated l'utilise pour accéder à votre serveur Git lors de l'opération de clone Git. Cette méthode est nettement plus sécurisée que la désactivation de la vérification SSL Git, qui accepte tout certificat TLS qui est présenté.

Procédure

Créez un secret avec un fichier de certificat CA.

1. Dans le cas où votre CA utilise des autorisations de certification intermédiaires, combinez les certificats pour toutes les AC dans un fichier ca.crt. Entrez la commande suivante:

```
$ cat intermediateCA.crt intermediateCA.crt rootCA.crt > ca.crt
```

2. Créez le secret en entrant la commande suivante:

```
$ oc create secret generic mycert --from-file=ca.crt=</path/to/file> 1
```

1 Il faut utiliser le nom de clé ca.crt.

3.4.2.8. Combinaisons de sources secrètes

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques.

3.4.2.8.1. Création d'un secret d'authentification SSH avec un fichier .gitconfig

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques, tels qu'un secret d'authentification SSH avec un fichier .gitconfig.

Conditions préalables

- Authentification SSH
- Fichier .gitconfig

Procédure

- Afin de créer un secret d'authentification SSH avec un fichier .gitconfig, entrez la commande suivante:

```
$ oc create secret generic <secret_name> \
    --from-file=ssh-privatekey=<path/to/ssh/private/key> \
    --from-file=<path/to/.gitconfig> \
    --type=kubernetes.io/ssh-auth
```

3.4.2.8.2. Créer un secret qui combine un fichier .gitconfig et un certificat CA

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques, comme un secret qui combine un fichier .gitconfig et un certificat d'autorité de certification (CA).

Conditions préalables

- Fichier .gitconfig
- Certificat CA

Procédure

- Afin de créer un secret qui combine un fichier .gitconfig et un certificat CA, entrez la commande suivante:

```
$ oc create secret generic <secret_name> \
    --from-file=ca.crt=<path/to/certificate> \
    --from-file=<path/to/.gitconfig>
```

3.4.2.8.3. Créer un secret d'authentification de base avec un certificat CA

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques, comme un secret qui combine une autorité d'authentification de base et un certificat d'autorité de certification (CA).

Conditions préalables

- Informations d'authentification de base

- Certificat CA

Procédure

- Afin de créer un secret d'authentification de base avec un certificat CA, entrez la commande suivante:

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=ca-cert=</path/to/file> \
    --type=kubernetes.io/basic-auth
```

3.4.2.8.4. Créer un secret d'authentification de base avec un fichier de configuration Git

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques, comme un secret qui combine une authentification de base et un fichier .gitconfig.

Conditions préalables

- Informations d'authentification de base
- Fichier .gitconfig

Procédure

- Afin de créer un secret d'authentification de base avec un fichier .gitconfig, entrez la commande suivante:

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=</path/to/.gitconfig> \
    --type=kubernetes.io/basic-auth
```

3.4.2.8.5. Créer un secret d'authentification de base avec un fichier .gitconfig et un certificat CA

Il est possible de combiner les différentes méthodes de création de secrets de clone source pour vos besoins spécifiques, comme un secret qui combine une authentification de base, un fichier .gitconfig et un certificat d'autorité de certification (CA).

Conditions préalables

- Informations d'authentification de base
- Fichier .gitconfig
- Certificat CA

Procédure

- Afin de créer un secret d'authentification de base avec un fichier .gitconfig et un certificat CA, entrez la commande suivante:

```
$ oc create secret generic <secret_name> \
    --from-literal=username=<user_name> \
    --from-literal=password=<password> \
    --from-file=</path/to/.gitconfig> \
    --from-file=ca-cert=</path/to/file> \
    --type=kubernetes.io/basic-auth
```

3.5. BINAIRE (LOCAL) SOURCE

Le contenu en streaming d'un système de fichiers local vers le constructeur est appelé une construction de type binaire. La valeur correspondante de BuildConfig.spec.source.type est binaire pour ces builds.

Ce type de source est unique en ce qu'il est exploité uniquement en fonction de votre utilisation de la construction de démarrage oc.

NOTE

Les constructions de type binaire nécessitent que le contenu soit diffusé à partir du système de fichiers local, de sorte que le déclenchement automatique d'une construction de type binaire, comme un déclencheur de changement d'image, n'est pas possible. C'est parce que les fichiers binaires ne peuvent pas être fournis. De même, vous ne pouvez pas lancer des constructions de type binaire à partir de la console Web.

Afin d'utiliser des builds binaires, invoquez oc start-build avec l'une de ces options:

- --from-file: Le contenu du fichier que vous spécifiez est envoyé en tant que flux binaire au constructeur. Il est également possible de spécifier une URL vers un fichier. Ensuite, le constructeur stocke les données dans un fichier avec le même nom en haut du contexte de construction.

- --from-dir et --from-repo: Les contenus sont archivés et envoyés en tant que flux binaire au constructeur. Ensuite, le constructeur extrait le contenu de l'archive dans le répertoire contextuel de construction. Avec --from-dir, vous pouvez également spécifier une URL vers une archive, qui est extraite.

- --from-archive: L'archive que vous spécifiez est envoyée au constructeur, où elle est extraite dans le répertoire contextuel de construction. Cette option se comporte comme --from-dir; une archive est créée sur votre hôte d'abord, chaque fois que l'argument de ces options est un répertoire.

Dans chacun des cas énumérés précédemment:

- Lorsque votre BuildConfig a déjà un type de source binaire défini, il est effectivement ignoré et remplacé par ce que le client envoie.

- Lorsque votre BuildConfig a un type de source Git défini, il est dynamiquement désactivé, puisque Binary et Git sont mutuellement exclusifs, et les données dans le flux binaire fourni au constructeur ont préséance.

Au lieu d'un nom de fichier, vous pouvez passer une URL avec un schéma HTTP ou HTTPS à --from-file et --from-archive. Lorsque vous utilisez --from-file avec une URL, le nom du fichier dans l'image du constructeur est déterminé par l'en-tête Content-Disposition envoyé par le serveur Web, ou le dernier composant du chemin d'URL si l'en-tête n'est pas présent. Aucune forme d'authentification n'est prise en charge et il n'est pas possible d'utiliser un certificat TLS personnalisé ou de désactiver la validation du certificat.

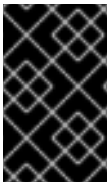
Lorsque vous utilisez oc new-build --binary=true, la commande veille à ce que les restrictions associées aux builds binaires soient appliquées. Le BuildConfig résultant a un type source de binaire, ce qui signifie que la seule façon valide d'exécuter une build pour ce BuildConfig est d'utiliser oc start-build avec l'une des options pour fournir les données binaires requises.

Les options source Dockerfile et contextDir ont une signification particulière avec les builds binaires.

Dockerfile peut être utilisé avec n'importe quelle source de construction binaire. Lorsque Dockerfile est utilisé et que le flux binaire est une archive, son contenu sert de Dockerfile de remplacement à n'importe quel Dockerfile dans l'archive. Lorsque Dockerfile est utilisé avec l'argument --from-file, et que l'argument de fichier est nommé Dockerfile, la valeur de Dockerfile remplace la valeur du flux binaire.

Dans le cas du flux binaire encapsulant le contenu d'archive extrait, la valeur du champ contextDir est interprétée comme un sous-répertoire dans l'archive, et, si elle est valide, le constructeur change dans ce sous-répertoire avant d'exécuter la build.

3.6. ENTRÉES SECRETS ET CONFIGURATION DES CARTES



IMPORTANT

Afin d'éviter que le contenu des secrets d'entrée et de configuration des cartes apparaisse dans les images de conteneurs de sortie de construction, utilisez les volumes de build dans vos stratégies de build Docker et source-à-image.

Dans certains scénarios, les opérations de construction nécessitent des informations d'identification ou d'autres données de configuration pour accéder aux ressources dépendantes, mais il est indésirable que ces informations soient placées dans le contrôle de la source. À cet effet, vous pouvez définir des secrets d'entrée et configurer des cartes d'entrée.

Lors de la création d'une application Java avec Maven, par exemple, vous pouvez configurer un miroir privé de Maven Central ou JCenter accessible par des clés privées. Afin de télécharger des bibliothèques à partir de ce miroir privé, vous devez fournir ce qui suit:

1. Fichier settings.xml configuré avec l'URL et les paramètres de connexion du miroir.
2. Clé privée référencée dans le fichier de paramètres, comme ~/.ssh/id_rsa.

« pour des raisons de sécurité, vous ne voulez pas exposer vos informations d'identification dans l'image de l'application.

Cet exemple décrit une application Java, mais vous pouvez utiliser la même approche pour ajouter des certificats SSL dans le répertoire /etc/ssl/certs, les clés API ou les jetons, les fichiers de licence, et plus encore.

3.6.1. C'est quoi un secret?

Le type d'objet Secret fournit un mécanisme pour contenir des informations sensibles telles que les mots de passe, les fichiers de configuration client dédiés OpenShift, les fichiers dockercfg, les informations d'identification de référentiels de source privée, etc. Les secrets découplent le contenu sensible des gousses. Il est possible de monter des secrets dans des conteneurs à l'aide d'un plugin de volume ou le système peut utiliser des secrets pour effectuer des actions au nom d'un pod.

Définition d'objet secret YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque (1)
data: (2)
  username: <username> (3)
  password: <password>
stringData: (4)
  hostname: myapp.mydomain.com (5)
```

1. Indique la structure des noms et valeurs clés du secret.
2. Le format autorisé pour les clés dans le champ de données doit respecter les directives de la valeur DNS_SUBDOMAIN dans le glossaire des identifiants Kubernetes.
3. La valeur associée aux clés dans la carte de données doit être encodée base64.
4. Les entrées dans la carte des chaînes de données sont converties en base64 et l'entrée est ensuite déplacée sur la carte des données automatiquement. Ce champ est écrit seulement. La valeur n'est retournée que par le champ de données.
5. La valeur associée aux touches de la carte stringData est composée de chaînes de texte brut.

3.6.1.1. Les propriétés des secrets

Les principales propriétés comprennent:

- Les données secrètes peuvent être référencées indépendamment de leur définition.
- Les volumes de données secrètes sont soutenus par des installations temporaires de stockage de fichiers (tmpfs) et ne viennent jamais se reposer sur un nœud.
- Les données secrètes peuvent être partagées dans un espace de noms.

3.6.1.2. Les types de secrets

La valeur dans le champ type indique la structure des noms et valeurs clés du secret. Le type peut être utilisé pour faire appliquer la présence de noms d'utilisateur et de clés dans l'objet secret. Lorsque vous ne voulez pas de validation, utilisez le type opaque, qui est la valeur par défaut.

Indiquez l'un des types suivants pour déclencher une validation minimale du côté serveur afin d'assurer la présence de noms clés spécifiques dans les données secrètes:

- Kubernetes.io/service-account-token. Utilise un jeton de compte de service.
- Kubernetes.io/dockercfg. Utilise le fichier .dockercfg pour les informations d'identification Docker requises.
- Kubernetes.io/dockerconfigjson. Utilise le fichier .docker/config.json pour les informations d'identification Docker requises.
- Kubernetes.io/auth de base. À utiliser avec l'authentification de base.

- Kubernetes.io/ssh-auth. Utiliser avec l'authentification de clé SSH.
- Kubernetes.io/tls. À utiliser avec les autorités de certification TLS.

Indiquez type= Opaque si vous ne voulez pas de validation, ce qui signifie que le secret ne prétend pas se conformer à une convention pour les noms ou les valeurs clés. Le secret opaque permet des paires de valeurs non structurées qui peuvent contenir des valeurs arbitraires.

NOTE

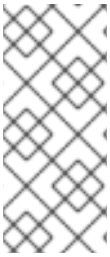
Il est possible de spécifier d'autres types arbitraires, tels que example.com/my-secret-type. Ces types ne sont pas imposés côté serveur, mais indiquent que le créateur du secret destiné à se conformer aux exigences clé / valeur de ce type.

3.6.1.3. Les mises à jour des secrets

Lorsque vous modifiez la valeur d'un secret, la valeur utilisée par une gousse déjà en cours d'exécution ne change pas dynamiquement. Afin de changer un secret, vous devez supprimer la gousse d'origine et créer un nouveau pod, dans certains cas avec un PodSpec identique.

La mise à jour d'un secret suit le même flux de travail que le déploiement d'une nouvelle image de conteneur. La commande kubectl roll-update peut être utilisée.

La valeur resourceVersion dans un secret n'est pas spécifiée lorsqu'elle est référencée. Donc, si un secret est mis à jour en même temps que les pods commencent, la version du secret qui est utilisé pour le pod n'est pas définie.



NOTE

Actuellement, il n'est pas possible de vérifier la version de ressource d'un objet secret qui a été utilisé lors de la création d'un pod. Il est prévu que les pods rapportent ces informations, afin qu'un contrôleur puisse redémarrer ceux à l'aide d'une ancienne ressourceVersion. Dans l'intervalle, ne mettez pas à jour les données des secrets existants, mais en créez de nouveaux avec des noms distincts.

3.6.2. Créer des secrets

Il faut créer un secret avant de créer les gousses qui dépendent de ce secret.

Lors de la création de secrets:

- Créez un objet secret avec des données secrètes.
- Actualisez le compte de service pod pour permettre la référence au secret.
- Créez un pod, qui consomme le secret comme variable d'environnement ou comme fichier à l'aide d'un volume secret.

Procédure

- Afin de créer un objet secret à partir d'un fichier JSON ou YAML, entrez la commande suivante:

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

  À titre d'exemple, vous pouvez créer un secret à partir de votre fichier local .docker/config.json:

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

Cette commande génère une spécification JSON du secret nommé dockerhub et crée l'objet.

Définition d'objet secret YAML Opaque

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque 1
data:
  username: <username>
  password: <password>
```

1 Indique un secret opaque.

Docker Configuration JSON Fichier Secret Object Définition

```
apiVersion: v1
kind: Secret
metadata:
  name: aregistrykey
  namespace: myapps
type: kubernetes.io/dockerconfigjson 1
data:

.dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXlzCg== 2
```

1 Indique que le secret utilise un fichier JSON de configuration docker.

2 La sortie d'un fichier JSON de configuration docker codé de base64.

3.6.3. En utilisant des secrets

Après avoir créé des secrets, vous pouvez créer un pod pour référencer votre secret, obtenir des journaux et supprimer le pod.

Procédure

1. Créez le pod pour faire référence à votre secret en entrant la commande suivante:

```
$ oc create -f <your_yaml_file>.yaml
```

2. Accédez aux journaux en entrant la commande suivante:

```
$ oc logs secret-example-pod
```

3. Effacer le pod en entrant la commande suivante:

```
$ oc delete pod secret-example-pod
```

Ressources supplémentaires

- Exemple de fichiers YAML avec des données secrètes:

Fichier YAML d'un secret qui créera quatre fichiers

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: <username> 1
  password: <password> 2
stringData:
  hostname: myapp.mydomain.com 3
  secret.properties: |- 4
    property1=valueA
    property2=valueB
```

1. Le fichier contient des valeurs décodées.
2. Le fichier contient des valeurs décodées.
3. Le fichier contient la chaîne fournie.
4. Le fichier contient les données fournies.

Fichier YAML d'un pod populant des fichiers dans un volume avec des données secrètes

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
          # name must match the volume name below
          - name: secret-volume
            mountPath: /etc/secret-volume
            readOnly: true
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret
  restartPolicy: Never
```

Fichier YAML d'un pod populer des variables d'environnement avec des données secrètes

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef:
              name: test-secret
              key: username
  restartPolicy: Never
```

Fichier YAML d'un objet BuildConfig qui peuple des variables d'environnement avec des données secrètes

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username
```

3.6.4. Ajout de secrets d'entrée et configuration des cartes

Afin de fournir des informations d'identification et d'autres données de configuration à une compilation sans les placer dans le contrôle source, vous pouvez définir des secrets d'entrée et des cartes de configuration d'entrée.

Dans certains scénarios, les opérations de construction nécessitent des informations d'identification ou d'autres données de configuration pour accéder aux ressources dépendantes. Afin de rendre ces informations disponibles sans le placer dans le contrôle source, vous pouvez définir des secrets d'entrée et des cartes de configuration d'entrée.

Procédure

Ajouter un secret d'entrée, configurer des cartes ou les deux à un objet BuildConfig existant:

1. Lorsque l'objet ConfigMap n'existe pas, créez-le en entrant la commande suivante:

```
$ oc create configmap settings-mvn \
    --from-file=settings.xml=<path/to/settings.xml>
```

Cela crée une nouvelle carte de configuration nommée settings-mvn, qui contient le contenu en texte brut du fichier settings.xml.

ASTUCE

Alternativement, vous pouvez appliquer le YAML suivant pour créer la carte de configuration:

```
apiVersion: core/v1
kind: ConfigMap
metadata:
  name: settings-mvn
data:
  settings.xml: |
    <settings>
    … # Insert maven settings here
    </settings>
```

2. Dans le cas où l'objet Secret n'existe pas, créez-le en entrant la commande suivante:

```
$ oc create secret generic secret-mvn \
    --from-file=ssh-privatekey=<path/to/.ssh/id_rsa> \
    --type=kubernetes.io/ssh-auth
```

Cela crée un nouveau secret nommé secret-mvn, qui contient le contenu codé de base64 de la clé privée id_rsa.

ASTUCE

Alternativement, vous pouvez appliquer le YAML suivant pour créer le secret d'entrée:

```
apiVersion: core/v1
kind: Secret
metadata:
  name: secret-mvn
type: kubernetes.io/ssh-auth
data:
  ssh-privatekey: |
    # Insert ssh private key, base64 encoded
```

3. Ajoutez la carte de configuration et le secret à la section source de l'objet BuildConfig existant:

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
```

```
  secrets:
    - secret:
        name: secret-mvn
```

4. Afin d'inclure la carte secrète et de configuration dans un nouvel objet BuildConfig, entrez la commande suivante:

```
$ oc new-build \
    openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
    --context-dir helloworld --build-secret “secret-mvn” \
    --build-config-map "settings-mvn"
```

Au cours de la construction, le processus de construction copie les fichiers settings.xml et id_rsa dans le répertoire où se trouve le code source. Dans OpenShift Dedicated S2I builder images, il s'agit du répertoire de travail d'image, qui est défini à l'aide de l'instruction WORKDIR dans le Dockerfile. Dans le cas où vous souhaitez spécifier un autre répertoire, ajoutez une destinationDir à la définition:

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
    - configMap:
        name: settings-mvn
      destinationDir: ".m2"
  secrets:
    - secret:
        name: secret-mvn
      destinationDir: ".ssh"
```

Lorsque vous créez un nouvel objet BuildConfig, vous pouvez également spécifier le répertoire de destination en entrant la commande suivante:

```
$ oc new-build \
    openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
    --context-dir helloworld --build-secret “secret-mvn:.ssh” \
    --build-config-map "settings-mvn:.m2"
```

Dans les deux cas, le fichier settings.xml est ajouté au répertoire ./.m2 de l'environnement de construction, et la clé id_rsa est ajoutée au répertoire ./.ssh.

3.6.5. La stratégie source à image

Lors de l'utilisation d'une stratégie Source, tous les secrets d'entrée définis sont copiés dans leur destination respectiveDir. Lorsque vous avez laissé destinationDir vide, les secrets sont placés dans le répertoire de travail de l'image du constructeur.

La même règle est utilisée lorsqu'une destinationDir est un chemin relatif. Les secrets sont placés dans les chemins qui sont relatifs au répertoire de travail de l'image. Le répertoire final du chemin destinationDir est créé s'il n'existe pas dans l'image du constructeur. L'ensemble des répertoires précédents de destinationDir doit exister, ou une erreur se produira.

NOTE

Les secrets d'entrée sont ajoutés sous forme d'écriture mondiale, ont 0666 permissions, et sont tronqués à la taille zéro après l'exécution du script d'assemblage. Cela signifie que les fichiers secrets existent dans l'image résultante, mais ils sont vides pour des raisons de sécurité.

Les cartes de configuration d'entrée ne sont pas tronquées une fois le script d'assemblage terminé.

3.7. ARTEFACTS EXTERNES

Il n'est pas recommandé de stocker des fichiers binaires dans un référentiel source. C'est pourquoi vous devez définir une version qui tire des fichiers supplémentaires, tels que les dépendances Java .jar, pendant le processus de construction. La façon dont cela est fait dépend de la stratégie de construction que vous utilisez.

Dans le cas d'une stratégie de création de source, vous devez mettre les commandes shell appropriées dans le script assembler:

fichier .s2i/bin/assembler

```
#!/bin/sh
APP_VERSION=1.0
wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar
```

fichier .s2i/bin/run

```
#!/bin/sh
exec java -jar app.jar
```

Dans le cas d'une stratégie de construction Docker, vous devez modifier le Dockerfile et invoquer les commandes shell avec l'instruction RUN:

Extrait de Dockerfile

```
FROM jboss/base-jdk:8

ENV APP_VERSION 1.0
RUN wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar

EXPOSE 8080
CMD [ "java", "-jar", "app.jar" ]
```

Dans la pratique, vous voudrez peut-être utiliser une variable d'environnement pour l'emplacement du fichier afin que le fichier spécifique à télécharger puisse être personnalisé à l'aide d'une variable d'environnement définie sur le BuildConfig, plutôt que de mettre à jour le fichier Dockerfile ou assembler le script.

Il est possible de choisir entre différentes méthodes de définition des variables d'environnement:

- En utilisant le fichier .s2i/environnement (uniquement pour une stratégie de création de source)
- Définir les variables dans l'objet BuildConfig

- Fournir les variables explicitement à l'aide de la commande oc start-build --env (uniquement pour les builds qui sont déclenchés manuellement)

3.8. L'UTILISATION DES INFORMATIONS D'IDENTIFICATION DOCKER POUR LES REGISTRES PRIVÉS

Il est possible de fournir des builds avec un fichier .docker/config.json avec des informations d'identification valides pour les registres de conteneurs privés. Cela vous permet de pousser l'image de sortie dans un registre d'image de conteneur privé ou de tirer une image de constructeur du registre d'image de conteneur privé qui nécessite une authentification.

Il est possible de fournir des informations d'identification pour plusieurs référentiels au sein d'un même registre, chacun avec des informations d'identification spécifiques à ce chemin de registre.

NOTE

Ce n'est pas nécessaire pour le registre d'images de conteneur OpenShift Dedicated, car les secrets sont générés automatiquement pour vous par OpenShift Dedicated.

Le fichier .docker/config.json se trouve dans votre répertoire d'accueil par défaut et a le format suivant:

```
auths:
  index.docker.io/v1/: 1
    auth: "YWRfbGzhcGU6R2labnRib21ifTE=" 2
    email: "user@example.com" 3
  docker.io/my-namespace/my-user/my-image: 4
    auth: "GzhYWRGU6R2fbclabnRgbkSp=""
    email: "user@example.com"
  docker.io/my-namespace: 5
    auth: "GzhYWRGU6R2deesfrRgbkSp=""
    email: "user@example.com"
```

1. L'URL du registre.
2. Le mot de passe crypté.
3. Adresse e-mail pour la connexion.
4. L'URL et les informations d'identification pour une image spécifique dans un espace de noms.
5. L'URL et les informations d'identification pour un espace de noms de registre.

Il est possible de définir plusieurs registres d'images de conteneur ou de définir plusieurs référentiels dans le même registre. Alternativement, vous pouvez également ajouter des entrées d'authentification à ce fichier en exécutant la commande de connexion docker. Le fichier sera créé s'il n'existe pas.

Kubernetes fournit des objets secrets, qui peuvent être utilisés pour stocker la configuration et les mots de passe.

Conditions préalables

- Il faut avoir un fichier .docker/config.json.

Procédure

1. Créez le secret à partir de votre fichier .docker/config.json local en entrant la commande suivante:

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

   Cela génère une spécification JSON du secret nommé dockerhub et crée l'objet.

2. Ajoutez un champ pushSecret dans la section de sortie du BuildConfig et définissez-le au nom du secret que vous avez créé, qui dans l'exemple précédent est dockerhub:

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "private.registry.com/org/private-image:latest"
    pushSecret:
      name: "dockerhub"
```

   Il est possible d'utiliser la commande oc set build-secret pour définir le secret de poussée sur la configuration de construction:

```
$ oc set build-secret --push bc/sample-build dockerhub
```

   Il est également possible de lier le secret push au compte de service utilisé par la construction au lieu de spécifier le champ pushSecret. Les builds utilisent par défaut le compte de service constructeur. Le secret push est automatiquement ajouté à la construction si le secret contient un identifiant qui correspond au référentiel hébergeant l'image de sortie de la construction.

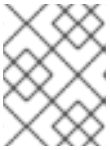
```
$ oc secrets link builder dockerhub
```

3. Extrayez l'image du conteneur constructeur à partir d'un registre d'images de conteneur privé en spécifiant le champ pullSecret, qui fait partie de la définition de stratégie de construction:

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "docker.io/user/private_repository"
    pullSecret:
      name: "dockerhub"
```

   Il est possible d'utiliser la commande oc set build-secret pour définir le secret de traction sur la configuration de construction:

```
$ oc set build-secret --pull bc/sample-build dockerhub
```



NOTE

Cet exemple utilise pullSecret dans une version Source, mais il est également applicable dans les builds Docker et Custom.

Il est également possible de lier le pull secret au compte de service utilisé par la construction au lieu de spécifier le champ pullSecret. Les builds utilisent par défaut le compte de service constructeur. Le secret de traction est automatiquement ajouté à la construction si le secret contient un identifiant qui correspond au référentiel hébergeant l'image d'entrée de la build. Afin de lier le pull secret au compte de service utilisé par la build au lieu de spécifier le champ pullSecret, entrez la commande suivante:

```
$ oc secrets link builder dockerhub
```

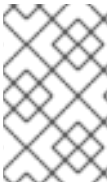
NOTE

Dans la spécification BuildConfig, vous devez spécifier une image à partir de l'image pour profiter de cette fonctionnalité. Les builds de stratégie Docker générés par oc new-build ou oc new-app peuvent ne pas le faire dans certaines situations.

3.9. CONSTRUIRE DES ENVIRONNEMENTS

Comme pour les variables d'environnement de pod, les variables d'environnement de build peuvent être définies en termes de références à d'autres ressources ou variables à l'aide de l'API Downward. Il y a quelques exceptions, qui sont notées.

Il est également possible de gérer les variables d'environnement définies dans le BuildConfig avec la commande oc set env.

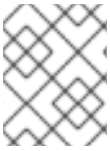


NOTE

Le référencement des ressources de conteneurs à l'aide de valeursDepuis les variables d'environnement de construction n'est pas pris en charge car les références sont résolues avant la création du conteneur.

3.9.1. En utilisant les champs de construction comme variables d'environnement

Il est possible d'injecter des informations sur l'objet de construction en définissant la source de variable d'environnement fieldPath sur le JsonPath du champ à partir duquel vous êtes intéressé à obtenir la valeur.



NOTE

La stratégie de Jenkins Pipeline ne prend pas en charge la valeurDe la syntaxe des variables d'environnement.

Procédure

- Définissez la source de variable d'environnement fieldPath sur le JsonPath du champ à partir duquel vous êtes intéressé à obtenir la valeur:

```
env:
  - name: FIELDREF_ENV
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
```

3.9.2. En utilisant les secrets comme variables d'environnement

Les valeurs clés des secrets peuvent être disponibles en tant que variables d'environnement à l'aide de la syntaxe valueFrom.

IMPORTANT

Cette méthode montre les secrets sous forme de texte clair dans la sortie de la console de dose de construction. Afin d'éviter cela, utilisez les secrets d'entrée et configurez plutôt les cartes.

Procédure

- Afin d'utiliser un secret comme variable d'environnement, définissez la valeurDe la syntaxe:

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: MYVAL
        valueFrom:
          secretKeyRef:
            key: myval
            name: mysecret
```

Ressources supplémentaires

- Entrées secrets et configuration des cartes

3.10. DES SECRETS DE CERTIFICAT DE SERVICE

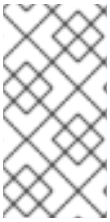
Les secrets de certificat de service sont destinés à prendre en charge les applications complexes de middleware qui ont besoin de certificats out-of-the-box. Il a les mêmes paramètres que les certificats de serveur générés par l'outil administrateur pour les nœuds et les maîtres.

Procédure

Afin de sécuriser la communication à votre service, demandez au cluster de générer une paire de certificats / clé de service signée dans un secret dans votre espace de noms.

- Définissez l'annotation de nom service.beta.openshift.io/serving-cert-secret sur votre service avec la valeur définie au nom que vous souhaitez utiliser pour votre secret.
  Ensuite, votre PodSpec peut monter ce secret. Lorsqu'il est disponible, votre pod fonctionne.
  Le certificat est bon pour le service interne DNS nom,
  <service.name>.<service.namespace>.svc.

  Le certificat et la clé sont au format PEM, stockés respectivement dans tls.crt et tls.key. La paire de certificats/clés est automatiquement remplacée lorsqu'elle est proche de l'expiration.
  Afficher la date d'expiration dans le service.beta.openshift.io/expiry annotation sur le secret, qui est au format RFC3339.



NOTE

Dans la plupart des cas, le nom DNS du service <service.name>.<service.namespace>.svc n'est pas routable à l'extérieur. L'utilisation principale de <service.name>.<service.namespace>.svc est pour la communication intracluster ou intraservice, et avec re-crypter les routes.

D'autres pods peuvent faire confiance aux certificats créés en cluster, qui ne sont signés que pour les noms DNS internes, en utilisant le paquet d'autorité de certification (CA) dans le fichier /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt qui est automatiquement monté dans leur pod.

L'algorithme de signature de cette fonctionnalité est x509.SHA256WithRSA. Afin de tourner manuellement, supprimer le secret généré. Création d'un nouveau certificat.

3.11. LES RESTRICTIONS DES SECRETS

Afin d'utiliser un secret, un pod doit faire référence au secret. Le secret peut être utilisé avec un pod de trois façons:

- De remplir des variables d'environnement pour les conteneurs.
- En tant que fichiers dans un volume monté sur un ou plusieurs de ses conteneurs.
- En kubelet lorsque vous tirez des images pour le pod.

Les secrets de type de volume écrivent des données dans le conteneur en tant que fichier en utilisant le mécanisme de volume. imagePullSecrets utilise les comptes de service pour l'injection automatique du secret dans tous les pods dans un espace de noms.

Lorsqu'un modèle contient une définition secrète, la seule façon pour le modèle d'utiliser le secret fourni est de s'assurer que les sources de volume secrètes sont validées et que la référence d'objet spécifié pointe réellement vers un objet de type Secret. C'est pourquoi un secret doit être créé avant les pods qui en dépendent. Le moyen le plus efficace de s'assurer que cela est de le faire injecter automatiquement grâce à l'utilisation d'un compte de service.

Les objets API secrets résident dans un espace de noms. Ils ne peuvent être référencés que par des pods dans ce même espace de noms.

Les secrets individuels sont limités à 1 Mo de taille. C'est pour décourager la création de grands secrets qui épuiseraient la mémoire apiserver et kubelet. Cependant, la création d'un certain nombre de petits secrets pourrait également épuiser la mémoire.

CHAPITRE 4. GESTION DE LA PRODUCTION DE BUILD

Consultez les sections suivantes pour une vue d'ensemble et des instructions pour gérer les sorties de construction.

4.1. CONSTRUIRE LA SORTIE

Les constructions qui utilisent la stratégie source à image (S2I) aboutissent à la création d'une nouvelle image conteneur. L'image est ensuite poussée vers le registre d'image du conteneur spécifié dans la section de sortie de la spécification Build.

Lorsque le type de sortie est ImageStreamTag, l'image sera poussée vers le registre d'images OpenShift intégré et marquée dans le flux d'images spécifié. Lorsque la sortie est de type DockerImage, le nom de la référence de sortie sera utilisé comme spécification de poussée docker. La spécification peut contenir un registre ou sera par défaut à DockerHub si aucun registre n'est spécifié. Lorsque la section de sortie de la spécification de construction est vide, l'image ne sera pas poussée à la fin de la construction.

Sortie vers un ImageStreamTag

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
```

Sortie vers un docker Push Spécification

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "my-registry.mycompany.com:5000/myimages/myimage:tag"
```

4.2. LES VARIABLES D'ENVIRONNEMENT D'IMAGE DE SORTIE

la stratégie source à image (S2I) établit les variables d'environnement suivantes sur les images de sortie:

La variable	Description
AJOUTER AU PANIER OPENSHIFT_BUILD_NAME	Le nom de la construction
ESPACE OPENSHIFT_BUILD_NAMESPACE	Espace de noms de la construction
À PROPOS DE OPENSHIFT_BUILD_SOURCE	L'URL source de la build
À PROPOS DE OPENSHIFT_BUILD_REFERENCE	La référence Git utilisée dans la construction
À PROPOS DE OPENSHIFT_BUILD_COMMIT	Commit source utilisé dans la construction

En outre, toute variable d'environnement définie par l'utilisateur, par exemple celles configurées avec les options de stratégie S2I, fera également partie de la liste des variables d'environnement d'image de sortie.

4.3. ÉTIQUETTES D'IMAGE DE SORTIE

la source à l'image (S2I) crée les étiquettes suivantes sur les images de sortie:

Étiquette	Description
IO.openshift.build.commit. Author	Auteur de la source commit utilisée dans la construction
IO.openshift.build.commit.date	Date de l'engagement de la source utilisé dans la construction
IO.openshift.build.commit.id	Hachage de la source commit utilisé dans la construction
IO.openshift.build.commit.message	Le message de la source commit utilisé dans la construction
IO.openshift.build.commit.ref	Branche ou référence spécifiée dans la source
IO.openshift.build.source-localisation	URL source pour la construction

Il est également possible d'utiliser le champ BuildConfig.spec.output.imageLabels pour spécifier une liste d'étiquettes personnalisées qui seront appliquées à chaque image construite à partir de la configuration de construction.

Étiquettes personnalisées pour les images construites

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "my-image:latest"
    imageLabels:
    - name: "vendor"
      value: "MyCompany"
    - name: "authoritative-source-url"
      value: "registry.mycompany.com"
```

CHAPITRE 5. EN UTILISANT DES STRATÉGIES DE CONSTRUCTION

Les sections suivantes définissent les stratégies de construction principales prises en charge et comment les utiliser.

5.1. CONSTRUCTION DE DOCKER

Le logiciel OpenShift Dedicated utilise Buildah pour construire une image conteneur à partir d'un Dockerfile. Consultez la documentation de référence Dockerfile pour plus d'informations sur les images de conteneurs de construction avec Dockerfile.

ASTUCE

Lorsque vous définissez les arguments de construction Docker à l'aide du tableau buildArgs, voir Comprendre comment ARG et FROM interagissent dans la documentation de référence Dockerfile.

5.1.1. Le remplacement de l'image Dockerfile FROM

Il est possible de remplacer l'instruction FROM du Dockerfile par les paramètres de l'objet BuildConfig. Lorsque le Dockerfile utilise des builds multi-étapes, l'image de la dernière instruction FROM sera remplacée.

Procédure

- Afin de remplacer l'instruction FROM du Dockerfile par les paramètres de l'objet BuildConfig, ajoutez les paramètres suivants à l'objet BuildConfig:

```
strategy:
  dockerStrategy:
    from:
      kind: "ImageStreamTag"
      name: "debian:latest"
```

5.1.2. En utilisant le chemin Dockerfile

Docker builds utilise un Dockerfile situé à la racine du contexte spécifié dans le champ BuildConfig.spec.source.contextDir.

Le champ dockerfilePath permet à la construction d'utiliser un chemin différent pour localiser votre Dockerfile, par rapport au champ BuildConfig.spec.source.contextDir. Il peut s'agir d'un nom de fichier différent du Dockerfile par défaut, tel que MyDockerfile, ou d'un chemin vers un Dockerfile dans un sous-répertoire, tel que dockerfiles/app1/Dockerfile.

Procédure

- Définissez le champ dockerfilePath pour que la construction utilise un chemin différent pour localiser votre Dockerfile:

```
strategy:
  dockerStrategy:
    dockerfilePath: dockerfiles/app1/Dockerfile
```

5.1.3. En utilisant des variables d'environnement docker

Afin de rendre les variables d'environnement disponibles pour le processus de construction docker et l'image résultante, vous pouvez ajouter des variables d'environnement à la définition de dockerStrategy de la configuration de construction.

Les variables d'environnement définies là sont insérées comme une seule instruction ENV Dockerfile juste après l'instruction FROM, de sorte qu'elle puisse être référencée plus tard dans le Dockerfile.

Les variables sont définies pendant la construction et restent dans l'image de sortie, elles seront donc présentes dans n'importe quel conteneur qui exécute cette image aussi bien.

À titre d'exemple, définir un proxy HTTP personnalisé à utiliser pendant la construction et l'exécution:

```
dockerStrategy:
...
  env:
    - name: "HTTP_PROXY"
      value: "http://myproxy.net:5187/"
```

Il est également possible de gérer les variables d'environnement définies dans la configuration de build avec la commande oc set env.

5.1.4. Ajout d'arguments de construction Docker

Il est possible de définir les arguments Docker build à l'aide du tableau buildArgs. Les arguments de construction sont transmis à Docker lorsqu'une build est lancée.

ASTUCE

Découvrez comment ARG et FROM interagissent dans la documentation de référence Dockerfile.

Procédure

- Afin de définir les arguments de construction Docker, ajoutez des entrées au tableau buildArgs, qui se trouve dans la définition dockerStrategy de l'objet BuildConfig. À titre d'exemple:

```
dockerStrategy:
...
  buildArgs:
    - name: "version"
      value: "latest"
```

NOTE

Les champs nom et valeur sont pris en charge. Les paramètres sur le champ valueFrom sont ignorés.

5.1.5. Couches d'écaillement avec docker builds

Docker crée normalement un calque représentant chaque instruction dans un Dockerfile. Définir l'imageOptimizationPolicy sur SkipLayers fusionne toutes les instructions en un seul calque au-dessus de l'image de base.

Procédure

- Définir l'imageOptimizationPolicy sur SkipLayers:

```
strategy:
  dockerStrategy:
    imageOptimizationPolicy: SkipLayers
```

5.1.6. En utilisant des volumes de construction

Les volumes de build peuvent être montés pour donner aux builds en cours l'accès à des informations que vous ne voulez pas persister dans l'image du conteneur de sortie.

Les volumes de construction fournissent des informations sensibles, telles que des informations d'identification de référentiel, dont l'environnement de construction ou la configuration n'a besoin qu'au moment de la construction. Les volumes de construction sont différents des entrées de build, dont les données peuvent persister dans l'image du conteneur de sortie.

Les points de montage des volumes de construction, à partir desquels le build en cours d'exécution lit les données, sont fonctionnellement similaires aux montures de volume pod.

Conditions préalables

- Il a ajouté un secret d'entrée, une carte de configuration ou les deux à un objet BuildConfig.

Procédure

- Dans la définition dockerStrategy de l'objet BuildConfig, ajoutez n'importe quel volume de build au tableau des volumes. À titre d'exemple:

```
spec:
  dockerStrategy:
    volumes:
      - name: secret-mvn (1)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh (2)
        source:
          type: Secret (3)
          secret:
            secretName: my-secret (4)
      - name: settings-mvn (5)
        mounts:
        - destinationPath: /opt/app-root/src/.m2 (6)
        source:
          type: ConfigMap (7)
          configMap:
            name: my-config (8)
```

(1) (5) C'est nécessaire. C'est un nom unique.

(2) (6) C'est nécessaire. Le chemin absolu du point de montage. Il ne doit pas contenir .. ou : et n'entre pas en collision avec le chemin de destination généré par le constructeur. Le /opt/app-root/src est le répertoire d'accueil par défaut pour de nombreuses images compatibles Red Hat S2I.

3 7 C'est nécessaire. Le type de source, ConfigMap, Secret ou CSI.

4 8 C'est nécessaire. Le nom de la source.

Ressources supplémentaires

- Construire des entrées
- Entrées secrets et configuration des cartes

5.2. CONSTRUCTION DE SOURCE À IMAGE

La source à l'image (S2I) est un outil pour construire des images de conteneurs reproductibles. Il produit des images prêtes à l'emploi en injectant une source d'application dans une image de conteneur et en assemblant une nouvelle image. La nouvelle image intègre l'image de base, le constructeur et la source construite et est prête à être utilisée avec la commande buildah run. Le S2I prend en charge les constructions incrémentielles, qui réutilisent des dépendances précédemment téléchargées, des artefacts précédemment construits, et ainsi de suite.

5.2.1. Effectuer des constructions incrémentielles source-à-image

La source à l'image (S2I) peut effectuer des constructions incrémentielles, ce qui signifie qu'elle réutilise des artefacts à partir d'images précédemment construites.

Procédure

- Afin de créer une construction incrémentale, créez un avec la modification suivante de la définition de stratégie:

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "incremental-image:latest" 1
    incremental: true 2
```

1 Indiquez une image qui prend en charge les constructions incrémentielles. Consultez la documentation de l'image du constructeur pour déterminer si elle prend en charge ce comportement.

2 Ce drapeau contrôle si une construction incrémentale est tentée. Lorsque l'image du constructeur ne prend pas en charge les constructions incrémentielles, la construction réussira toujours, mais vous obtiendrez un message journal indiquant que la construction incrémentale n'a pas été couronnée de succès en raison d'un script manquant d'artéfacts de sauvegarde.

Ressources supplémentaires

- Consultez les Exigences S2I pour obtenir des informations sur la façon de créer une image de constructeur prenant en charge les constructions incrémentales.

5.2.2. Créer des scripts d'image source-à-image

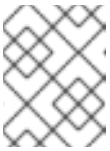
Il est possible de remplacer les scripts d'assemblage, d'exécution et de sauvegarde des artefacts source-à-image (S2I) fournis par l'image du constructeur.

Procédure

- Afin de remplacer les scripts S2I d'assemblage, d'exécution et de sauvegarde fournis par l'image du constructeur, complétez l'une des actions suivantes:
  - Fournissez un script assembler, exécuter ou enregistrer des artefacts dans le répertoire .s2i/bin de votre référentiel source d'applications.
  - Fournir une URL d'un répertoire contenant les scripts dans le cadre de la définition de stratégie dans l'objet BuildConfig. À titre d'exemple:

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "builder-image:latest"
    scripts: "http://somehost.com/scripts_directory" 1
```

1 Le processus de construction append l'exécution, l'assemblage et les sauvegardes-artefacts sur le chemin. En cas d'existence d'un ou de tous les scripts avec ces noms, le processus de construction utilise ces scripts à la place des scripts portant le même nom que celui fourni dans l'image.



NOTE

Les fichiers situés à l'URL des scripts ont préséance sur les fichiers situés dans .s2i/bin du référentiel source.

5.2.3. Les variables d'environnement source à image

Il existe deux façons de rendre les variables d'environnement disponibles pour le processus de création de source et l'image résultante: les fichiers d'environnement et les valeurs d'environnement BuildConfig. Les variables que vous fournissez en utilisant l'une ou l'autre méthode seront présentes pendant le processus de construction et dans l'image de sortie.

5.2.3.1. En utilisant des fichiers d'environnement source à image

La source build vous permet de définir des valeurs d'environnement, une par ligne, à l'intérieur de votre application, en les spécifiant dans un fichier .s2i/environnement dans le référentiel source. Les variables d'environnement spécifiées dans ce fichier sont présentes pendant le processus de construction et dans l'image de sortie.

Lorsque vous fournissez un fichier .s2i/environnement dans votre référentiel source, source-à-image (S2I) lit ce fichier pendant la construction. Cela permet la personnalisation du comportement de construction car le script assemble peut utiliser ces variables.

Procédure

À titre d'exemple, désactiver la compilation d'actifs pour votre application Rails pendant la construction:

- Ajouter DISABLE_ASSET_COMPILATION=true dans le fichier .s2i/environnement.

En plus des builds, les variables d'environnement spécifiées sont également disponibles dans l'application en cours d'exécution elle-même. À titre d'exemple, faire démarrer l'application Rails en mode développement au lieu de la production:

- Ajouter RAILS_ENV=développement au fichier .s2i/environnement.

La liste complète des variables d'environnement prises en charge est disponible dans la section d'utilisation des images pour chaque image.

5.2.3.2. En utilisant l'environnement de configuration source à image

Il est possible d'ajouter des variables d'environnement à la définition sourceStrategy de la configuration de construction. Les variables d'environnement définies là sont visibles lors de l'exécution du script d'assemblage et seront définies dans l'image de sortie, les rendant également disponibles pour le script d'exécution et le code d'application.

Procédure

- À titre d'exemple, désactiver la compilation d'actifs pour votre application Rails:

```
sourceStrategy:
...
  env:
    - name: "DISABLE_ASSET_COMPILATION"
      value: "true"
```

Ressources supplémentaires

- La section environnement de construction fournit des instructions plus avancées.
- Il est également possible de gérer les variables d'environnement définies dans la configuration de build avec la commande oc set env.

5.2.4. Ignorer les fichiers source source source-à-image

La source à image (S2I) prend en charge un fichier .s2iignore, qui contient une liste de modèles de fichiers qui doivent être ignorés. Les fichiers dans le répertoire de travail de construction, tels que fournis par les différentes sources d'entrée, qui correspondent à un modèle trouvé dans le fichier .s2iignore ne seront pas mis à la disposition du script assembler.

5.2.5. Création d'images à partir du code source avec source-à-image

La source à image (S2I) est un framework qui facilite l'écriture d'images qui prennent le code source de l'application comme entrée et produisent une nouvelle image qui exécute l'application assemblée en sortie.

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

5.2.5.1. Comprendre le processus de création de source à image

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

- Les sources
- Scripts source à image (S2I)
- Image du constructeur

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

5.2.5.2. Comment écrire des scripts source à image

Il est possible d'écrire des scripts source à image (S2I) dans n'importe quel langage de programmation, tant que les scripts sont exécutables à l'intérieur de l'image du constructeur. Le S2I prend en charge plusieurs options fournissant des scripts assemble/run/save-artefacts. L'ensemble de ces emplacements sont vérifiés sur chaque construction dans l'ordre suivant:

1. Le script spécifié dans la configuration de build.
2. Le script trouvé dans le répertoire source de l'application .s2i/bin.
3. 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 build 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 relatif ou absolu vers un répertoire sur l'hôte où se trouvent les scripts S2I.
- HTTP(s)://path_to_scripts_dir: URL vers un répertoire où se trouvent les scripts S2I.

Tableau 5.1. Scripts S2I

Le script	Description
assembler	Le script assemble les artefacts de l'application à partir d'une source et les place dans des répertoires appropriés à l'intérieur de l'image. Ce script est requis. Le flux de travail de ce script est: 1. Facultatif: Restaurer des artefacts de construction. Lorsque vous souhaitez prendre en charge les constructions incrémentielles, assurez-vous de définir également des objets de sauvegarde. 2. Placez la source de l'application à l'emplacement souhaité. 3. Construisez les artefacts de l'application. 4. Installez les artefacts dans des endroits appropriés pour qu'ils puissent fonctionner.
courir	Le script d'exécution exécute votre application. Ce script est requis.

Le script	Description
des objets de sauvegarde	Le script sauvegarde-artefacts rassemble toutes les dépendances qui peuvent accélérer les processus de construction qui suivent. Ce script est facultatif. À titre d'exemple: • Avec Ruby, gemmes installées par Bundler. • Le contenu de Java, .m2. Ces dépendances sont rassemblées dans un fichier goudron et diffusées vers la sortie standard.
l'utilisation	Le script d'utilisation vous permet d'informer l'utilisateur comment utiliser correctement votre image. Ce script est facultatif.
essai / course	Le script test/exécution vous permet de créer un processus pour vérifier si l'image fonctionne correctement. Ce script est facultatif. Le déroulement proposé de ce processus est: 1. Construisez l'image. 2. Exécutez l'image pour vérifier le script d'utilisation. 3. Exécutez s2i build pour vérifier le script assembler. 4. Facultatif: Exécuter s2i à nouveau pour vérifier les sauvegardes-artefacts et assembler les scripts sauvegarder et restaurer la fonctionnalité des artefacts. 5. Exécutez l'image pour vérifier que l'application de test fonctionne.  **NOTE** L'emplacement suggéré pour mettre l'application de test construite par votre script test/exécution est le répertoire test/test-app dans votre référentiel d'images.

Exemples de scripts S2I

Les exemples de scripts S2I suivants sont écrits en Bash. Chaque exemple suppose que son contenu de goudron est déballé dans le répertoire /tmp/s2i.

assembler le script:

```
#!/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
```

exécutez le script:

```
#!/bin/bash

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

script de sauvegarde-artefacts:

```
#!/bin/bash

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

script d'utilisation:

```
#!/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 supplémentaires

- Didacticiel de création d'images S2I

5.2.6. En utilisant des volumes de construction

Les volumes de build peuvent être montés pour donner aux builds en cours l'accès à des informations que vous ne voulez pas persister dans l'image du conteneur de sortie.

Les volumes de construction fournissent des informations sensibles, telles que des informations d'identification de référentiel, dont l'environnement de construction ou la configuration n'a besoin qu'au moment de la construction. Les volumes de construction sont différents des entrées de build, dont les données peuvent persister dans l'image du conteneur de sortie.

Les points de montage des volumes de construction, à partir desquels le build en cours d'exécution lit les données, sont fonctionnellement similaires aux montures de volume pod.

Conditions préalables

- Il a ajouté un secret d'entrée, une carte de configuration ou les deux à un objet BuildConfig.

Procédure

- Dans la définition sourceStrategy de l'objet BuildConfig, ajoutez n'importe quel volume de build au tableau des volumes. À titre d'exemple:

```
spec:
  sourceStrategy:
    volumes:
      - name: secret-mvn (1)
        mounts:
        - destinationPath: /opt/app-root/src/.ssh (2)
        source:
          type: Secret (3)
          secret:
            secretName: my-secret (4)
      - name: settings-mvn (5)
        mounts:
        - destinationPath: /opt/app-root/src/.m2 (6)
        source:
          type: ConfigMap (7)
          configMap:
            name: my-config (8)
```

(1) (5) C'est nécessaire. C'est un nom unique.

(2) (6) C'est nécessaire. Le chemin absolu du point de montage. Il ne doit pas contenir .. ou : et n'entre pas en collision avec le chemin de destination généré par le constructeur. Le /opt/app-root/src est le répertoire d'accueil par défaut pour de nombreuses images compatibles Red Hat S2I.

(3) (7) C'est nécessaire. Le type de source, ConfigMap, Secret ou CSI.

(4) (8) C'est nécessaire. Le nom de la source.

Ressources supplémentaires

- Construire des entrées
- Entrées secrets et configuration des cartes

5.3. CONSTRUCTION DE PIPELINES

IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

La stratégie de construction Pipeline permet aux développeurs de définir un pipeline Jenkins pour une utilisation par le plugin pipeline Jenkins. La construction peut être démarrée, surveillée et gérée par OpenShift Dedicated de la même manière que n'importe quel autre type de construction.

Les flux de travail de pipeline sont définis dans un fichier jenkins, soit intégrés directement dans la configuration de construction, soit fournis dans un référentiel Git et référencés par la configuration de construction.

5.3.1. Comprendre les pipelines dédiés à OpenShift

IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

Les pipelines vous donnent le contrôle de la construction, du déploiement et de la promotion de vos applications sur OpenShift Dedicated. En utilisant une combinaison de la stratégie de construction de Jenkins Pipeline, des jenkinsfiles et du langage spécifique de domaine dédié OpenShift (DSL) fourni par le plugin client Jenkins, vous pouvez créer une construction avancée, tester, déployer et promouvoir des pipelines pour n'importe quel scénario.

Jenkins Sync Plugin dédié à OpenShift

Le plugin de synchronisation Jenkins dédié OpenShift maintient la configuration de construction et construit des objets en synchronisation avec les tâches et les builds Jenkins, et fournit les éléments suivants:

- Création dynamique d'emplois et de gestion à Jenkins.
- Création dynamique de modèles de pod d'agent à partir de flux d'images, de balises de flux d'images ou de configuration de cartes.
- Injection de variables d'environnement.
- La visualisation de pipeline dans la console Web dédiée OpenShift.
- Intégration avec le plugin Jenkins Git, qui passe des informations de commit depuis OpenShift Dedicated builds au plugin Jenkins Git.
- La synchronisation des secrets dans les entrées d'identification Jenkins.

Jenkins Client Plugin dédié à OpenShift

Le plugin client Jenkins dédié OpenShift est un plugin Jenkins qui vise à fournir une syntaxe de pipeline Jenkins lisible, concise, complète et fluide pour des interactions riches avec un serveur API dédié OpenShift. Le plugin utilise l'outil de ligne de commande OpenShift Dedicated, oc, qui doit être disponible sur les nœuds exécutant le script.

Le plugin client Jenkins doit être installé sur votre master Jenkins afin que le DSL dédié OpenShift soit disponible à utiliser dans le fichier jenkins pour votre application. Ce plugin est installé et activé par défaut lors de l'utilisation de l'image OpenShift Dedicated Jenkins.

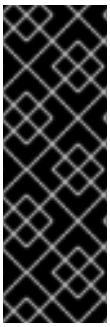
Dans le cas des pipelines dédiés à OpenShift dans votre projet, vous devrez utiliser la stratégie de construction de pipelines Jenkins. Cette stratégie par défaut à l'utilisation d'un jenkinsfile à la racine de votre référentiel source, mais fournit également les options de configuration suivantes:

- Le champ jenkinsfile en ligne dans votre configuration de construction.
- Champ jenkinsfilePath dans votre configuration de construction qui fait référence à l'emplacement du jenkinsfile à utiliser par rapport au contexte sourceDir.

NOTE

Le champ jenkinsfilePath optionnel spécifie le nom du fichier à utiliser, par rapport au contexte sourceDir. En cas d'omission de contextDir, il par défaut à la racine du référentiel. En cas d'omission de jenkinsfilePath, il devient jenkinsfile par défaut.

5.3.2. Fournir le fichier Jenkins pour les constructions de pipelines



IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

Le jenkinsfile utilise la syntaxe standard du langage groovy pour permettre un contrôle fin de la configuration, de la construction et du déploiement de votre application.

Il est possible de fournir le jenkinsfile de l'une des façons suivantes:

- Fichier situé dans votre référentiel de code source.
- Intégré dans le cadre de votre configuration de build en utilisant le champ jenkinsfile.

Lors de l'utilisation de la première option, le jenkinsfile doit être inclus dans le référentiel de code source de vos applications à l'un des endroits suivants:

- Fichier nommé jenkinsfile à la racine de votre référentiel.
- Fichier nommé jenkinsfile à la racine du contexte sourceDir de votre référentiel.
- Le nom de fichier spécifié via le champ jenkinsfilePath de la section JenkinsPipelineStrategy de votre BuildConfig, qui est relatif au contexte sourceDir si fourni, sinon il par défaut à la racine du référentiel.

Le fichier jenkinsfile est exécuté sur le pod d'agent Jenkins, qui doit avoir les binaires du client OpenShift Dedicated si vous avez l'intention d'utiliser le DSL dédié OpenShift.

Procédure

Afin de fournir le fichier Jenkins, vous pouvez soit:

- Intégrez le fichier Jenkins dans la configuration de construction.

- Inclure dans la configuration de construction une référence au référentiel Git qui contient le fichier Jenkins.

Définition intégrée

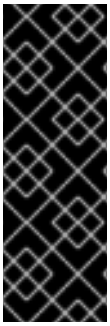
```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('agent') {
          stage 'build'
          openshiftBuild(buildConfig: 'ruby-sample-build', showBuildLogs: 'true')
          stage 'deploy'
          openshiftDeploy(deploymentConfig: 'frontend')
        }
```

La référence à Git Repository

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  source:
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfilePath: some/repo/dir/filename 1
```

1 Le champ jenkinsfilePath optionnel spécifie le nom du fichier à utiliser, par rapport au contexte sourceDir. En cas d'omission de contextDir, il par défaut à la racine du référentiel. En cas d'omission de jenkinsfilePath, il devient jenkinsfile par défaut.

5.3.3. En utilisant des variables d'environnement pour les constructions de pipelines



IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

Afin de rendre les variables d'environnement disponibles pour le processus de construction de pipelines, vous pouvez ajouter des variables d'environnement à la définition jenkinsPipelineStrategy de la configuration de construction.

Les variables d'environnement seront définies comme paramètres pour n'importe quel travail Jenkins associé à la configuration de construction.

Procédure

- Afin de définir les variables d'environnement à utiliser pendant la construction, modifiez le fichier YAML:

```
jenkinsPipelineStrategy:
...
  env:
    - name: "FOO"
      value: "BAR"
```

Il est également possible de gérer les variables d'environnement définies dans la configuration de build avec la commande oc set env.

5.3.3.1. Cartographie entre les variables d'environnement BuildConfig et les paramètres de travail Jenkins

Lorsqu'un travail Jenkins est créé ou mis à jour en fonction des modifications apportées à une configuration de construction de stratégie de pipeline, toutes les variables d'environnement de la configuration de construction sont cartographiées aux définitions des paramètres de travail Jenkins, où les valeurs par défaut pour les définitions des paramètres de travail Jenkins sont les valeurs actuelles des variables d'environnement associées.

Après la création initiale de l'emploi Jenkins, vous pouvez toujours ajouter des paramètres supplémentaires à la tâche à partir de la console Jenkins. Les noms des paramètres diffèrent des noms des variables d'environnement dans la configuration de construction. Les paramètres sont honorés lorsque les constructions sont lancées pour ces emplois Jenkins.

La façon dont vous démarrez des builds pour le travail Jenkins dicte comment les paramètres sont définis.

- Lorsque vous démarrez avec oc start-build, les valeurs des variables d'environnement dans la configuration de build sont les paramètres définis pour l'instance de travail correspondante. Les modifications que vous apportez aux valeurs par défaut des paramètres de la console Jenkins sont ignorées. Les valeurs de configuration de build ont préséance.

- Lorsque vous démarrez avec oc start-build -e, les valeurs pour les variables d'environnement spécifiées dans l'option -e ont préséance.

    - Lorsque vous spécifiez une variable d'environnement non listée dans la configuration de construction, elles seront ajoutées sous forme de définitions de paramètres de travail Jenkins.

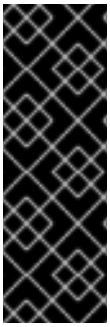
    - Les modifications que vous apportez de la console Jenkins aux paramètres correspondant aux variables d'environnement sont ignorées. La configuration de la construction et ce que vous spécifiez avec oc start-build -e ont préséance.

- Lorsque vous démarrez le travail Jenkins avec la console Jenkins, vous pouvez contrôler le réglage des paramètres avec la console Jenkins dans le cadre du démarrage d'une construction pour le travail.

NOTE

Il est recommandé de spécifier dans la configuration de construction toutes les variables d'environnement possibles à associer aux paramètres de travail. Cela réduit les E/S sur disque et améliore les performances pendant le traitement de Jenkins.

5.3.4. Didacticiel de construction de pipeline



IMPORTANT

La stratégie de construction du pipeline est dépréciée dans OpenShift Dedicated 4. Des fonctionnalités équivalentes et améliorées sont présentes dans les pipelines dédiés OpenShift basés sur Tekton.

Les images Jenkins sur OpenShift Dedicated sont entièrement prises en charge et les utilisateurs doivent suivre la documentation utilisateur de Jenkins pour définir leur fichier jenkins dans un travail ou le stocker dans un système de gestion de contrôle de source.

Cet exemple montre comment créer un pipeline dédié OpenShift qui va construire, déployer et vérifier une application Node.js/MongoDB en utilisant le modèle nodejs-mongodb.json.

Procédure

1. Créer le maître Jenkins:

```
$ oc project <project_name>
```

Choisissez le projet que vous souhaitez utiliser ou créez un nouveau projet avec oc new-project <project_name>.

```
$ oc new-app jenkins-ephemeral 1
```

À la place, si vous souhaitez utiliser un stockage persistant, utilisez des jenkins-persistents.

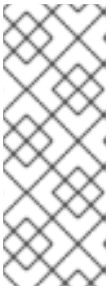
2. Créez un fichier nommé nodejs-sample-pipeline.yaml avec le contenu suivant:

NOTE

Cela crée un objet BuildConfig qui utilise la stratégie de pipeline Jenkins pour construire, déployer et mettre à l'échelle l'application d'exemple Node.js/MongoDB.

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "nodejs-sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: <pipeline content from below>
    type: JenkinsPipeline
```

3. Après avoir créé un objet BuildConfig avec un jenkinsPipelineStrategy, dites au pipeline quoi faire en utilisant un jenkinsfile en ligne:



NOTE

Cet exemple ne met pas en place un référentiel Git pour l'application.

Le contenu jenkinsfile suivant est écrit dans Groovy en utilisant OpenShift Dedicated DSL. Dans cet exemple, incluez du contenu en ligne dans l'objet BuildConfig en utilisant le style YAML Literal, bien que l'inclusion d'un fichier jenkins dans votre référentiel source soit la méthode préférée.

```
def templatePath = 'https://raw.githubusercontent.com/openshift/nodejs-
ex/master/openshift/templates/nodejs-mongodb.json' 1
def templateName = 'nodejs-mongodb-example' 2
pipeline {
  agent {
    node {
      label 'nodejs' 3
    }
  }
  options {
    timeout(time: 20, unit: 'MINUTES') 4
  }
  stages {
    stage('preamble') {
        steps {
            script {
                openshift.withCluster() {
                    openshift.withProject() {
                        echo "Using project: ${openshift.project()}"
                    }
                }
            }
        }
    }
    stage('cleanup') {
      steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                  openshift.selector("all", [ template : templateName ]).delete() 5
                  if (openshift.selector("secrets", templateName).exists()) { 6
                    openshift.selector("secrets", templateName).delete()
                  }
                }
            }
        }
      }
    }
    stage('create') {
      steps {
        script {
            openshift.withCluster() {
```

```
            openshift.withProject() {
              openshift.newApp(templatePath) 7
            }
          }
        }
      }
      stage('build') {
        steps {
          script {
            openshift.withCluster() {
              openshift.withProject() {
                def builds = openshift.selector("bc", templateName).related('builds')
                timeout(5) { 8
                  builds.untilEach(1) {
                    return (it.object().status.phase == "Complete")
                  }
                }
              }
            }
          }
        }
      }
      stage('deploy') {
        steps {
          script {
            openshift.withCluster() {
              openshift.withProject() {
                def rm = openshift.selector("dc", templateName).rollout()
                timeout(5) { 9
                  openshift.selector("dc", templateName).related('pods').untilEach(1) {
                    return (it.object().status.phase == "Running")
                  }
                }
              }
            }
          }
        }
      }
      stage('tag') {
        steps {
          script {
            openshift.withCluster() {
              openshift.withProject() {
                openshift.tag("${templateName}:latest", "${templateName}-staging:latest") 10
              }
            }
          }
        }
      }
    }
  }
}
```

1 Chemin du modèle à utiliser.

1 2 Le nom du modèle qui sera créé.

3 Lancez un pod d'agent node.js sur lequel exécuter cette construction.

4 Fixez un délai de 20 minutes pour ce pipeline.

5 Effacer tout avec cette étiquette de modèle.

6 Effacer tous les secrets avec cette étiquette de modèle.

7 Créez une nouvelle application à partir du templatePath.

8 Attendez jusqu'à cinq minutes pour que la construction soit terminée.

9 Attendez jusqu'à cinq minutes pour que le déploiement se termine.

10 Lorsque tout le reste réussit, marquez $ {templateName}:dernière image comme $ {templateName}-staging:lastest. La configuration de construction de pipeline pour l'environnement de mise en scène peut regarder pour $ {templateName}-staging:dernière image à changer, puis la déployer dans l'environnement de mise en scène.

NOTE

L'exemple précédent a été écrit en utilisant le style de pipeline déclaratif, mais l'ancien style de pipeline scripté est également pris en charge.

4. Créez le Pipeline BuildConfig dans votre cluster OpenShift dédié:

```
$ oc create -f nodejs-sample-pipeline.yaml
```

   a. Dans le cas où vous ne souhaitez pas créer votre propre fichier, vous pouvez utiliser l'échantillon du référentiel Origin en exécutant:

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/jenkins/pipeline/nodejs-
sample-pipeline.yaml
```

5. Démarrez le pipeline:

```
$ oc start-build nodejs-sample-pipeline
```

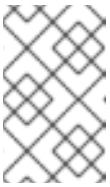
NOTE

Alternativement, vous pouvez démarrer votre pipeline avec la console Web dédiée OpenShift en naviguant vers la section Builds → Pipeline et en cliquant sur Démarrer le pipeline, ou en visitant la console Jenkins, en naviguant vers le pipeline que vous avez créé, et en cliquant sur Construire maintenant.

Lorsque le pipeline est lancé, vous devriez voir les actions suivantes effectuées dans le cadre de votre projet:

- L'instance d'emploi est créée sur le serveur Jenkins.

- La gousse d'agent est lancée, si votre pipeline en a besoin.
- Le pipeline coule sur la gousse d'agent, ou le capitaine si aucun agent n'est requis.
  - Les ressources précédemment créées avec l'étiquette template=nodejs-mongodb-example seront supprimées.
  - Le modèle nodejs-mongodb-exemple de nodejs-mongodb-exemple créera une nouvelle application et toutes ses ressources associées.
  - La construction sera lancée à l'aide de l'exemple nodejs-mongodb BuildConfig.
    - Le pipeline attendra jusqu'à ce que la construction soit terminée pour déclencher la prochaine étape.
  - Le déploiement sera lancé à l'aide de la configuration de déploiement nodejs-mongodb-exemple.
    - Le pipeline attendra jusqu'à ce que le déploiement soit terminé pour déclencher la prochaine étape.
  - En cas de succès de la construction et du déploiement, l'image nodejs-mongodb-example:dernière image sera marquée comme nodejs-mongodb-example:stage.
- La gousse d'agent est supprimée, si l'on en avait besoin pour le pipeline.



NOTE

La meilleure façon de visualiser l'exécution du pipeline est de l'afficher dans la console Web dédiée OpenShift. Consultez vos pipelines en vous connectant à la console Web et en naviguant vers Builds → Pipelines.

5.4. AJOUT DE SECRETS AVEC CONSOLE WEB

Ajoutez un secret à votre configuration de build afin qu'il puisse accéder à un référentiel privé.

Procédure

Ajouter un secret à votre configuration de build afin qu'il puisse accéder à un référentiel privé à partir de la console Web OpenShift Dedicated:

1. Créez un nouveau projet OpenShift dédié.
2. Créez un secret qui contient des informations d'identification pour accéder à un référentiel de code source privé.
3. Créer une configuration de build.
4. Dans la page d'éditeur de configuration de build ou dans l'application de création à partir de la page d'image du constructeur de la console Web, définissez le secret source.
5. Cliquez sur **Save**.

5.5. ACTIVER LA TRACTION ET LA POUSSÉE

Il est possible de tirer sur un registre privé en réglant le secret d'attraction et en poussant en plaçant le secret de poussée dans la configuration de la construction.

Procédure

Activer l'accès à un registre privé:

- Définissez le secret de traction dans la configuration de construction.

Afin d'activer la poussée:

- Définissez le secret de poussée dans la configuration de la construction.

CHAPITRE 6. EXÉCUTER ET CONFIGURER DES BUILDS DE BASE

Les sections suivantes fournissent des instructions pour les opérations de construction de base, y compris le démarrage et l'annulation des builds, l'édition de BuildConfigs, la suppression de BuildConfigs, l'affichage des détails de construction et l'accès aux journaux de construction.

6.1. DÉMARRAGE D'UNE CONSTRUCTION

Dans votre projet actuel, vous pouvez démarrer manuellement une nouvelle construction à partir d'une configuration de construction existante.

Procédure

- Lorsque vous démarrez une compilation manuellement, entrez la commande suivante:

```
$ oc start-build <buildconfig_name>
```

6.1.1. Faire redémarrer une construction

Il est possible de redémarrer manuellement une construction à l'aide du drapeau --from-build.

Procédure

- Afin de redémarrer manuellement un build, entrez la commande suivante:

```
$ oc start-build --from-build=<build_name>
```

6.1.2. Journaux de construction en streaming

Il est possible de spécifier le drapeau --follow pour diffuser les journaux de la construction dans stdout.

Procédure

- Afin de diffuser manuellement les journaux d'une build dans stdout, entrez la commande suivante:

```
$ oc start-build <buildconfig_name> --follow
```

6.1.3. Définir des variables d'environnement lors du démarrage d'une build

Il est possible de spécifier le drapeau --env pour définir n'importe quelle variable d'environnement souhaitée pour la construction.

Procédure

- Afin de spécifier une variable d'environnement souhaitée, entrez la commande suivante:

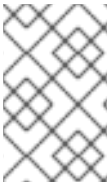
```
$ oc start-build <buildconfig_name> --env=<key>=<value>
```

6.1.4. Démarrage d'une construction avec source

Au lieu de compter sur un pull source Git pour une build, vous pouvez également démarrer une construction en poussant directement votre source, qui pourrait être le contenu d'un répertoire de travail Git ou SVN, un ensemble d'artefacts binaires pré-construits que vous souhaitez déployer, ou un seul fichier. Cela peut être fait en spécifiant l'une des options suivantes pour la commande Start-Build:

L'option	Description
--from-dir=<répertoire>	Indique un répertoire qui sera archivé et utilisé comme entrée binaire pour la construction.
--from-file=<file>	Indique un seul fichier qui sera le seul fichier dans la source de construction. Le fichier est placé dans la racine d'un répertoire vide avec le même nom de fichier que le fichier original fourni.
--from-repo=<local_source_repo >	Indique un chemin vers un référentiel local à utiliser comme entrée binaire pour une build. Ajoutez l'option --commit pour contrôler quelle branche, balise ou commit est utilisé pour la construction.

Lorsque vous passez l'une de ces options directement à la construction, les contenus sont diffusés dans la construction et remplacent les paramètres de source de construction actuels.



NOTE

Les builds déclenchés par l'entrée binaire ne préserveront pas la source sur le serveur, de sorte que les reconstructions déclenchées par des modifications d'image de base utiliseront la source spécifiée dans la configuration de construction.

Procédure

- Afin de démarrer une compilation à partir d'un référentiel de code source et d'envoyer le contenu d'un référentiel Git local sous forme d'archive à partir de la balise v2, entrez la commande suivante:

```
$ oc start-build hello-world --from-repo=../hello-world --commit=v2
```

6.2. ANNULATION D'UNE CONSTRUCTION

Il est possible d'annuler une construction à l'aide de la console Web ou de la commande CLI suivante.

Procédure

- Afin d'annuler manuellement une construction, entrez la commande suivante:

```
$ oc cancel-build <build_name>
```

6.2.1. Annulation de plusieurs builds

Avec la commande CLI suivante, vous pouvez annuler plusieurs builds.

Procédure

- Afin d'annuler manuellement plusieurs builds, entrez la commande suivante:

```
$ oc cancel-build <build1_name> <build2_name> <build3_name>
```

6.2.2. Annuler toutes les constructions

Avec la commande CLI suivante, vous pouvez annuler toutes les versions de la configuration de construction.

Procédure

- Afin d'annuler toutes les versions, entrez la commande suivante:

```
$ oc cancel-build bc/<buildconfig_name>
```

6.2.3. Annuler toutes les constructions dans un état donné

Il est possible d'annuler toutes les constructions dans un état donné, tels que nouveaux ou en attente, tout en ignorant les constructions dans d'autres états.

Procédure

- Afin d'annuler tout dans un état donné, entrez la commande suivante:

```
$ oc cancel-build bc/<buildconfig_name>
```

6.3. ÉDITION D'UN BUILDCONFIG

Afin de modifier vos configurations de build, vous utilisez l'option Edit BuildConfig dans la vue Builds de la perspective Développeur.

Il est possible d'utiliser l'une ou l'autre des vues suivantes pour modifier un BuildConfig:

- La vue Formulaire vous permet d'éditer votre BuildConfig à l'aide des champs de formulaire standard et des cases à cocher.
- La vue YAML vous permet d'éditer votre BuildConfig avec un contrôle total sur les opérations.

Il est possible de basculer entre la vue Formulaire et YAML sans perdre de données. Les données de la vue Formulaire sont transférées à la vue YAML et vice versa.

Procédure

1. Dans la vue Builds de la perspective Développeur, cliquez sur le menu Options pour voir l'option Edit BuildConfig.
2. Cliquez sur Modifier BuildConfig pour afficher l'option Formulaire.
3. Dans la section Git, entrez l'URL du référentiel Git pour la base de code que vous souhaitez utiliser pour créer une application. L'URL est ensuite validée.
    - Facultatif: Cliquez sur Afficher les options avancées de Git pour ajouter des détails tels que:

- Git Référence pour spécifier une branche, une balise ou un commit qui contient du code que vous souhaitez utiliser pour construire l'application.
- Context Dir pour spécifier le sous-répertoire qui contient le code que vous souhaitez utiliser pour construire l'application.
- Créer un nom secret avec des informations d'identification pour tirer votre code source d'un référentiel privé.

4. Dans la section Build from, sélectionnez l'option à partir de laquelle vous souhaitez créer. Les options suivantes peuvent être utilisées:
   - Image Stream balise référence une image pour un flux d'image et une balise donnée. Entrez le projet, le flux d'images et l'étiquette de l'emplacement que vous souhaitez construire et poussez vers.
   - Image Stream image référence une image pour un flux d'image donné et le nom de l'image. Entrez l'image de flux d'image à partir de laquelle vous souhaitez construire. Entrez également le projet, le flux d'images et le tag à pousser.
   - Image Docker: L'image Docker est référencée par un référentiel d'images Docker. Il vous faudra également entrer le projet, le flux d'images et le tag pour vous référer à l'endroit où vous souhaitez pousser.
5. Facultatif: Dans la section Variables d'environnement, ajoutez les variables d'environnement associées au projet en utilisant les champs Nom et Valeur. Afin d'ajouter plus de variables d'environnement, utilisez Add Value, ou Add from ConfigMap and Secret.
6. Facultatif: Pour personnaliser davantage votre application, utilisez les options avancées suivantes:

   Déclencheur

   Déclenche une nouvelle construction d'image lorsque l'image du constructeur change. Ajoutez d'autres déclencheurs en cliquant sur Ajouter Trigger et en sélectionnant le type et le secret.

   Les secrets

   Ajoute des secrets pour votre application. Ajoutez plus de secrets en cliquant sur Ajouter secret et en sélectionnant le point Secret et Montage.

   La politique

   Cliquez sur Exécuter la stratégie pour sélectionner la stratégie d'exécution de construction. La stratégie sélectionnée détermine l'ordre dans lequel les builds créés à partir de la configuration de build doivent s'exécuter.

   Crochets

   Choisissez Exécuter les crochets de construction après que l'image soit construite pour exécuter des commandes à la fin de la construction et vérifier l'image. Ajoutez le type de crochet, la commande et les arguments à ajouter à la commande.
7. Cliquez sur Enregistrer pour enregistrer le BuildConfig.

6.4. LA SUPPRESSION D'UN BUILDCONFIG

Il est possible de supprimer un BuildConfig à l'aide de la commande suivante.

Procédure

- Afin de supprimer un BuildConfig, entrez la commande suivante:

```
$ oc delete bc <BuildConfigName>
```

  Cela supprime également toutes les versions qui ont été instanciées à partir de ce BuildConfig.

- Afin de supprimer un BuildConfig et de conserver les builds instatés à partir du BuildConfig, spécifiez le drapeau --cascade=false lorsque vous entrez la commande suivante:

```
$ oc delete --cascade=false bc <BuildConfigName>
```

6.5. AFFICHAGE DES DÉTAILS DE CONSTRUCTION

Il est possible d'afficher les détails de construction avec la console Web ou en utilisant la commande CLI de description d'oc.

Ceci affiche des informations, y compris:

- La source de construction.
- La stratégie de construction.
- La destination de sortie.
- Digérer de l'image dans le registre de destination.
- Comment la construction a été créée.

Dans le cas où la construction utilise la stratégie Source, la sortie décrit oc inclut également des informations sur la révision de la source utilisée pour la construction, y compris l'ID de commit, l'auteur, le committer et le message.

Procédure

- Afin d'afficher les détails de construction, entrez la commande suivante:

```
$ oc describe build <build_name>
```

6.6. ACCÈS AUX JOURNAUX DE CONSTRUCTION

Il est possible d'accéder aux journaux de construction à l'aide de la console Web ou du CLI.

Procédure

- Afin de diffuser les journaux à l'aide de la build directement, entrez la commande suivante:

```
$ oc describe build <build_name>
```

6.6.1. Accès aux journaux BuildConfig

Il est possible d'accéder aux journaux BuildConfig à l'aide de la console Web ou du CLI.

Procédure

- Afin de diffuser les journaux de la dernière version pour un BuildConfig, entrez la commande suivante:

```
$ oc logs -f bc/<buildconfig_name>
```

6.6.2. Accéder aux journaux BuildConfig pour une version donnée

Il est possible d'accéder aux journaux d'une version donnée pour un BuildConfig à l'aide de la console Web ou du CLI.

Procédure

- Afin de diffuser les journaux d'une version donnée pour un BuildConfig, entrez la commande suivante:

```
$ oc logs --version=<number> bc/<buildconfig_name>
```

6.6.3. Activer la verbosité du journal

Il est possible d'activer une sortie plus verbeuse en passant la variable d'environnement BUILD_LOGLEVEL dans le cadre de la sourceStrategy dans un BuildConfig.

NOTE

L'administrateur peut définir la verbosité de construction par défaut pour toute l'instance OpenShift Dedicated en configurant env/BUILD_LOGLEVEL. Cette valeur par défaut peut être remplacée en spécifiant BUILD_LOGLEVEL dans un BuildConfig donné. Il est possible de spécifier une priorité supérieure sur la ligne de commande pour les builds non binaires en passant --build-loglevel à oc start-build.

Les niveaux de journal disponibles pour les compilations de sources sont les suivants:

De niveau 0	Produit la sortie à partir de conteneurs exécutant le script assembler et toutes les erreurs rencontrées. C'est la valeur par défaut.
De niveau 1	Il produit des informations de base sur le processus exécuté.
De niveau 2	Il produit des informations très détaillées sur le processus exécuté.
De niveau 3	Il produit des informations très détaillées sur le processus exécuté et une liste des contenus d'archives.
De niveau 4	Actuellement produit les mêmes informations que le niveau 3.
De niveau 5	Il produit tout ce qui est mentionné sur les niveaux précédents et fournit en outre des messages push docker.

Procédure

- Afin d'activer plus de sortie de verbose, passez la variable d'environnement BUILD_LOGLEVEL dans le cadre de la sourceStrategy ou dockerStrategy dans un BuildConfig:

```
sourceStrategy:
...
  env:
    - name: "BUILD_LOGLEVEL"
      value: "2" 1
```

1 Ajustez cette valeur au niveau de journal souhaité.

CHAPITRE 7. DÉCLENCHEMENT ET MODIFICATION DES BUILDS

Les sections suivantes décrivent comment déclencher des builds et modifier les builds en utilisant des crochets de construction.

7.1. CRÉER DES DÉCLENCHEURS

Lorsque vous définissez un BuildConfig, vous pouvez définir des déclencheurs pour contrôler les circonstances dans lesquelles le BuildConfig doit être exécuté. Les déclencheurs de build suivants sont disponibles:

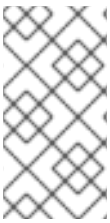
- À propos de Webhook
- Changement d'image
- Changement de configuration

7.1.1. Déclencheurs Webhook

Les déclencheurs Webhook vous permettent de déclencher une nouvelle construction en envoyant une demande au point de terminaison OpenShift Dedicated API. Ces déclencheurs peuvent être définis à l'aide de webhooks GitHub, GitLab, Bitbucket ou génériques.

Actuellement, OpenShift Dedicated webhooks ne prend en charge que les versions analogues de l'événement push pour chacun des systèmes de gestion du code source (SCM) basés sur Git. Les autres types d'événements sont ignorés.

Lorsque les événements push sont traités, l'hôte de plan de contrôle dédié OpenShift confirme si la référence de branche à l'intérieur de l'événement correspond à la référence de branche dans le BuildConfig correspondant. Dans ce cas, il vérifie alors la référence exacte de commit notée dans l'événement webhook sur la construction OpenShift Dedicated. Lorsqu'ils ne correspondent pas, aucune construction n'est déclenchée.



NOTE

les nouvelles applications OC et oc new-build créent automatiquement les déclencheurs GitHub et Generic webhook, mais tous les autres déclencheurs de webhook nécessaires doivent être ajoutés manuellement. Il est possible d'ajouter manuellement des déclencheurs en paramétrant les déclencheurs.

Dans tous les webhooks, vous devez définir un secret avec une clé nommée WebHookSecretKey et la valeur étant la valeur à fournir lors de l'invocation du webhook. La définition du webhook doit alors faire référence au secret. Le secret assure l'unicité de l'URL, empêchant les autres de déclencher la construction. La valeur de la clé est comparée au secret fourni lors de l'invocation du webhook.

C'est par exemple un webhook GitHub avec une référence à un secret nommé mysecret:

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```

Le secret est alors défini comme suit. A noter que la valeur du secret est encodée base64 comme requis pour tout champ de données d'un objet secret.

```
- kind: Secret
  apiVersion: v1
  metadata:
    name: mysecret
    creationTimestamp:
  data:
    WebHookSecretKey: c2VjcmV0dmFsdWUx
```

7.1.1.1. Ajout d'utilisateurs non authentifiés à la liaison du rôle system:webhook

En tant qu'administrateur de cluster, vous pouvez ajouter des utilisateurs non authentifiés à la liaison de rôle system:webhook dans OpenShift Dedicated pour des espaces de noms spécifiques. La liaison de rôle system:webhook permet aux utilisateurs de déclencher des builds à partir de systèmes externes qui n'utilisent pas de mécanisme d'authentification dédié OpenShift. Les utilisateurs non authentifiés n'ont pas accès par défaut à des liaisons de rôle non publiques. Il s'agit d'un changement par rapport aux versions dédiées d'OpenShift avant 4.16.

L'ajout d'utilisateurs non authentifiés au lien de rôle system:webhook est nécessaire pour déclencher avec succès des builds à partir de GitHub, GitLab et Bitbucket.

Lorsqu'il est nécessaire d'autoriser les utilisateurs non authentifiés à accéder à un cluster, vous pouvez le faire en ajoutant des utilisateurs non authentifiés à la liaison de rôle system:webhook dans chaque espace de noms requis. Cette méthode est plus sûre que d'ajouter des utilisateurs non authentifiés à la liaison de rôle de cluster system:webhook. Cependant, si vous avez un grand nombre d'espaces de noms, il est possible d'ajouter des utilisateurs non authentifiés à la liaison de rôle de cluster system:webhook qui appliquerait la modification à tous les espaces de noms.

IMPORTANT

Assurez-vous toujours de respecter les normes de sécurité de votre organisation lors de la modification de l'accès non authentifié.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle cluster-admin.
- L'OpenShift CLI (oc) a été installé.

Procédure

1. Créez un fichier YAML nommé add-webhooks-unauth.yaml et ajoutez le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  name: webhook-access-unauthenticated
  namespace: <namespace> (1)
roleRef:
  apiGroup: rbac.authorization.k8s.io
```

```
  kind: Role
  name: "system:webhook"
subjects:
  - apiGroup: rbac.authorization.k8s.io
    kind: Group
    name: "system:unauthenticated"
```

1 L'espace de noms de votre BuildConfig.

2. Appliquer la configuration en exécutant la commande suivante:

```
$ oc apply -f add-webhooks-unauth.yaml
```

Ressources supplémentaires

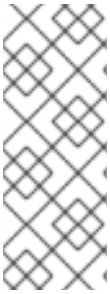
- Liaisons de rôles de cluster pour les groupes non authentifiés

7.1.1.2. En utilisant GitHub webhooks

Les webhooks GitHub gèrent l'appel effectué par GitHub lorsqu'un dépôt est mis à jour. Lors de la définition du déclencheur, vous devez spécifier un secret, qui fait partie de l'URL que vous fournissez à GitHub lors de la configuration du webhook.

Exemple de définition GitHub webhook:

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```



NOTE

Le secret utilisé dans la configuration du déclencheur webhook n'est pas le même que le champ secret que vous rencontrez lors de la configuration du webhook dans GitHub UI. Le secret de la configuration du déclencheur webhook rend l'URL Webhook unique et difficile à prédire. Le secret configuré dans l'interface utilisateur GitHub est un champ de chaîne optionnel qui est utilisé pour créer un condensateur HMAC hex du corps, qui est envoyé sous la forme d'un en-tête X-Hub-Signature.

L'URL de charge utile est retournée sous forme d'URL GitHub Webhook par la commande de description oc (voir Affichage des URL Webhook), et est structurée comme suit:

Exemple de sortie

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

Conditions préalables

- Créez un BuildConfig à partir d'un référentiel GitHub.

- le système: non authentifié a accès au rôle system:webhook dans les espaces de noms requis. En outre, system:unauthenticated a accès au rôle de cluster system:webhook.

Procédure

1. Configurez un GitHub Webhook.

   a. Après avoir créé un objet BuildConfig à partir d'un référentiel GitHub, exécutez la commande suivante:

   ```
   $ oc describe bc/<name_of_your_BuildConfig>
   ```

   Cette commande génère une URL webhook GitHub.

   Exemple de sortie

   ```
   https://api.starter-us-east-
   1.openshift.com:443/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<na
   me>/webhooks/<secret>/github
   ```

   b. Découpez et collez cette URL dans GitHub, à partir de la console Web GitHub.

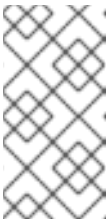
   c. Dans votre dépôt GitHub, sélectionnez Ajouter Webhook dans Paramètres → Webhooks.

   d. Collez la sortie URL dans le champ URL de charge utile.

   e. Changez le type de contenu de l'application par défaut de GitHub/x-www-form-urlencoded en application/json.

   f. Cliquez sur Ajouter webhook.
   Il faut voir un message de GitHub indiquant que votre webhook a été configuré avec succès.

   Lorsque vous poussez un changement dans votre référentiel GitHub, une nouvelle version démarre automatiquement, et lors d'une construction réussie, un nouveau déploiement commence.

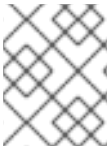
   NOTE

   Gogs prend en charge le même format de charge utile webhook que GitHub. Ainsi, si vous utilisez un serveur Gogs, vous pouvez définir un déclencheur GitHub webhook sur votre BuildConfig et le déclencher par votre serveur Gogs.

2. Compte tenu d'un fichier contenant une charge utile JSON valide, telle que payload.json, vous pouvez déclencher manuellement le webhook avec la commande curl suivante:

   ```
   $ curl -H "X-GitHub-Event: push" -H "Content-Type: application/json" -k -X POST --data-
   binary @payload.json
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
   gs/<name>/webhooks/<secret>/github
   ```

   L'argument -k n'est nécessaire que si votre serveur API n'a pas de certificat signé correctement.



NOTE

La construction ne sera déclenchée que si la valeur ref de l'événement GitHub webhook correspond à la valeur ref spécifiée dans le champ source.git de la ressource BuildConfig.

Ressources supplémentaires

- Gogs

7.1.1.3. En utilisant GitLab webhooks

Les webhooks GitLab gèrent l'appel effectué par GitLab lorsqu'un dépôt est mis à jour. Comme avec les déclencheurs GitHub, vous devez spécifier un secret. L'exemple suivant est une définition de déclenchement YAML dans le BuildConfig:

```
type: "GitLab"
gitlab:
  secretReference:
    name: "mysecret"
```

L'URL de charge utile est retournée comme l'URL GitLab Webhook par la commande oc décrit, et est structurée comme suit:

Exemple de sortie

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

Conditions préalables

- le système: non authentifié a accès au rôle system:webhook dans les espaces de noms requis. En outre, system:unauthenticated a accès au rôle de cluster system:webhook.

Procédure

1. Configurez un GitLab Webhook.
   a. Accédez à l'URL Webhook en entrant la commande suivante:

      ```
      $ oc describe bc <name>
      ```

   b. Copiez l'URL Webhook en remplaçant <secret> par votre valeur secrète.
   c. Suivez les instructions de configuration de GitLab pour coller l'URL Webhook dans les paramètres de votre dépôt GitLab.
2. Compte tenu d'un fichier contenant une charge utile JSON valide, telle que payload.json, vous pouvez déclencher manuellement le webhook avec la commande curl suivante:

   ```
   $ curl -H "X-GitLab-Event: Push Hook" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
   ```

L'argument -k n'est nécessaire que si votre serveur API n'a pas de certificat signé correctement.

7.1.1.4. En utilisant Bitbucket webhooks

Bitbucket webhooks gère l'appel effectué par Bitbucket lorsqu'un dépôt est mis à jour. Comme les déclencheurs GitHub et GitLab, vous devez spécifier un secret. L'exemple suivant est une définition de déclenchement YAML dans le BuildConfig:

```
type: "Bitbucket"
bitbucket:
  secretReference:
    name: "mysecret"
```

L'URL de charge utile est retournée sous forme d'URL Bitbucket Webhook par la commande de description oc, et est structurée comme suit:

Exemple de sortie

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

Conditions préalables

- le système: non authentifié a accès au rôle system:webhook dans les espaces de noms requis. En outre, system:unauthenticated a accès au rôle de cluster system:webhook.

Procédure

1. Configurez un Bitbucket Webhook.
   a. Accédez à l'URL Webhook en entrant la commande suivante:

      ```
      $ oc describe bc <name>
      ```

   b. Copiez l'URL Webhook en remplaçant <secret> par votre valeur secrète.

   c. Suivez les instructions de configuration de Bitbucket pour coller l'URL Webhook dans vos paramètres de dépôt Bitbucket.

2. Étant donné un fichier contenant une charge utile JSON valide, telle que payload.json, vous pouvez déclencher manuellement le webhook en entrant la commande curl suivante:

   ```
   $ curl -H "X-Event-Key: repo:push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
   ```

   L'argument -k n'est nécessaire que si votre serveur API n'a pas de certificat signé correctement.

7.1.1.5. En utilisant des webhooks génériques

Les webhooks génériques sont appelés à partir de n'importe quel système capable de faire une

demande web. Comme pour les autres webhooks, vous devez spécifier un secret, qui fait partie de l'URL que l'appelant doit utiliser pour déclencher la construction. Le secret assure l'unicité de l'URL, empêchant les autres de déclencher la construction. Ce qui suit est un exemple de définition de déclencheur YAML dans le BuildConfig:

```
type: "Generic"
generic:
  secretReference:
    name: "mysecret"
  allowEnv: true 1
```

1 Défini sur true pour permettre à un webhook générique de passer dans des variables d'environnement.

Procédure

1. Afin de configurer l'appelant, fournissez au système d'appel l'URL du point de terminaison webhook générique pour votre build.

   Exemple d'URL générique de point de terminaison webhook

   ```
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
   gs/<name>/webhooks/<secret>/generic
   ```

   L'appelant doit appeler le webhook comme une opération POST.

2. Afin d'appeler le webhook manuellement, entrez la commande curl suivante:

   ```
   $ curl -X POST -k
   https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
   gs/<name>/webhooks/<secret>/generic
   ```

   Le verbe HTTP doit être défini sur POST. Le drapeau -k non sécurisé est spécifié pour ignorer la validation du certificat. Ce second drapeau n'est pas nécessaire si votre cluster a correctement signé des certificats.

   Le point de terminaison peut accepter une charge utile optionnelle avec le format suivant:

   ```
   git:
     uri: "<url to git repository>"
     ref: "<optional git reference>"
     commit: "<commit hash identifying a specific git commit>"
     author:
       name: "<author name>"
       email: "<author e-mail>"
     committer:
       name: "<committer name>"
       email: "<committer e-mail>"
     message: "<commit message>"
   env: 1
     - name: "<variable name>"
       value: "<variable value>"
   ```

1 Comme les variables d'environnement BuildConfig, les variables d'environnement définies ici sont mises à la disposition de votre build. Lorsque ces variables entrent en collision avec

3. Afin de passer cette charge utile à l'aide de curl, définissez-la dans un fichier nommé payload_file.yaml et exécutez la commande suivante:

```
$ curl -H "Content-Type: application/yaml" --data-binary @payload_file.yaml -X POST -k
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

Les arguments sont les mêmes que l'exemple précédent avec l'ajout d'un en-tête et d'une charge utile. L'argument -H définit l'en-tête Content-Type sur application/yaml ou application/json en fonction de votre format de charge utile. L'argument --data-binary est utilisé pour envoyer une charge utile binaire avec des nouvelles lignes intactes avec la demande POST.

NOTE

Le logiciel OpenShift Dedicated permet aux builds d'être déclenchés par le webhook générique même si une charge utile de requête invalide est présentée, par exemple, le type de contenu invalide, le contenu non comparable ou invalide, etc. Ce comportement est maintenu pour la rétrocompatibilité. Lorsqu'une charge utile de requête invalide est présentée, OpenShift Dedicated renvoie un avertissement au format JSON dans le cadre de sa réponse HTTP 200 OK.

7.1.1.6. Affichage des URL Webhook

La commande oc described permet d'afficher les URL Webhook associées à une configuration de build. Lorsque la commande n'affiche pas d'URL Webhook, aucun déclencheur webhook n'est actuellement défini pour cette configuration de construction.

Procédure

- Afin d'afficher les URL Webhook associées à un BuildConfig, exécutez la commande suivante:

```
$ oc describe bc <name>
```

7.1.2. En utilisant les déclencheurs de changement d'image

En tant que développeur, vous pouvez configurer votre build pour s'exécuter automatiquement chaque fois qu'une image de base change.

Il est possible d'utiliser des déclencheurs de changement d'image pour invoquer automatiquement votre build lorsqu'une nouvelle version d'une image en amont est disponible. À titre d'exemple, si une build est basée sur une image RHEL, vous pouvez déclencher ce build pour exécuter n'importe quand l'image RHEL change. En conséquence, l'image de l'application est toujours en cours d'exécution sur la dernière image de base RHEL.

NOTE

Les flux d'images qui pointent vers les images de conteneur dans les registres de conteneurs v1 ne déclenchent une compilation qu'une fois lorsque la balise de flux d'image devient disponible et non sur les mises à jour d'image ultérieures. Cela est dû à l'absence d'images uniques identifiables dans les registres de conteneurs v1.

Procédure

1. Définissez une ImageStream qui pointe vers l'image en amont que vous souhaitez utiliser comme déclencheur:

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby-20-centos7"
```

   Ceci définit le flux d'images qui est lié à un référentiel d'images conteneur situé à <system-registry>/<namespace>/ruby-20-centos7. Le <system-registry> est défini comme un service avec le nom docker-registry s'exécutant dans OpenShift Dedicated.

2. Dans le cas où un flux d'images est l'image de base de la construction, définissez le champ de la stratégie de construction pour pointer vers ImageStream:

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
```

   Dans ce cas, la définition sourceStrategy consomme la dernière balise du flux d'images nommé ruby-20-centos7 situé dans cet espace de noms.

3. Définissez une build avec un ou plusieurs déclencheurs qui pointent vers ImageStreams:

```
type: "ImageChange" (1)
imageChange: {}
type: "ImageChange" (2)
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
```

   (1) Déclencheur de changement d'image qui surveille ImageStream et Tag tel que défini par la stratégie de construction à partir du champ. L'objet imageChange ici doit être vide.

   (2) Déclencheur de changement d'image qui surveille un flux d'images arbitraire. La partie imageChange, dans ce cas, doit inclure un champ qui fait référence à ImageStreamTag pour surveiller.

Lors de l'utilisation d'un déclencheur de changement d'image pour le flux d'images de stratégie, la build générée est fournie avec une balise docker immuable qui pointe vers la dernière image correspondant à cette balise. Cette nouvelle référence d'image est utilisée par la stratégie lorsqu'elle s'exécute pour la construction.

Dans le cas d'autres déclencheurs de changement d'image qui ne font pas référence au flux d'images de stratégie, une nouvelle version est lancée, mais la stratégie de construction n'est pas mise à jour avec une référence d'image unique.

Comme cet exemple a un déclencheur de changement d'image pour la stratégie, la construction résultante est:

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "172.30.17.3:5001/mynamespace/ruby-20-centos7:<immutableid>"
```

Cela garantit que la construction déclenchée utilise la nouvelle image qui vient d'être poussée vers le référentiel, et la construction peut être redémarrée à tout moment avec les mêmes entrées.

Il est possible de mettre en pause un déclencheur de changement d'image pour autoriser plusieurs modifications sur le flux d'image référencé avant le démarrage d'une build. Lors de l'ajout initial d'un ImageChangeTrigger à un BuildConfig, vous pouvez également définir l'attribut mis en pause à true afin d'éviter qu'une build ne soit immédiatement déclenchée.

```
type: "ImageChange"
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
  paused: true
```

Lorsqu'une build est déclenchée en raison d'un déclencheur de webhook ou d'une requête manuelle, la construction créée utilise le <immutableid> résolu à partir de l'imageStream référencée par la Stratégie. Cela garantit que les builds sont effectués en utilisant des balises d'image cohérentes pour faciliter la reproduction.

Ressources supplémentaires

- registres des conteneurs v1

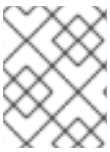
7.1.3. Identifier le déclencheur de changement d'image d'une construction

En tant que développeur, si vous avez des déclencheurs de changement d'image, vous pouvez identifier quel changement d'image a initié la dernière version. Cela peut être utile pour le débogage ou le dépannage des builds.

Exemple BuildConfig

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: bc-ict-example
  namespace: bc-ict-example-namespace
spec:

# ...

  triggers:
```

```
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input:latest
        namespace: bc-ict-example-namespace
  - imageChange:
      from:
        kind: ImageStreamTag
        name: input2:latest
        namespace: bc-ict-example-namespace
    type: ImageChange
status:
  imageChangeTriggers:
  - from:
      name: input:latest
      namespace: bc-ict-example-namespace
    lastTriggerTime: "2021-06-30T13:47:53Z"
    lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-
namespace/input@sha256:0f88ffbeb9d25525720bfa3524cb1bf0908b7f791057cf1acfae917b11266a69

  - from:
      name: input2:latest
      namespace: bc-ict-example-namespace
    lastTriggeredImageID:  image-registry.openshift-image-registry.svc:5000/bc-ict-example-
namespace/input2@sha256:0f88ffbeb9d25525720bfa3524cb2ce0908b7f791057cf1acfae917b11266a6
9

  lastVersion: 1
```



NOTE

Cet exemple omet les éléments qui ne sont pas liés aux déclencheurs de changement d'image.

Conditions préalables

- Il y a plusieurs déclencheurs de changement d'image. Ces déclencheurs ont déclenché une ou plusieurs builds.

Procédure

1. Dans le BuildConfig CR, dans status.imageChangeTriggers, identifiez le lastTriggerTime qui a le dernier horodatage.
   Cette imageChangeTriggerStatus

   > Then you use the `name` and `namespace` from that build to find the corresponding image change trigger in `buildConfig.spec.triggers`.

2. Dans imageChangeTriggers, comparez les horodatages pour identifier les derniers

Déclencheurs de changement d'image

Dans votre configuration de build, buildConfig.spec.triggers est un ensemble de stratégies de déclenchement de build, BuildTriggerPolicy.

Chaque BuildTriggerPolicy a un champ de type et un ensemble de champs de pointeurs. Chaque champ de pointeur correspond à l'une des valeurs autorisées pour le champ type. En tant que tel, vous ne pouvez définir que BuildTriggerPolicy sur un seul champ de pointeur.

Dans le cas des déclencheurs de changement d'image, la valeur du type est ImageChange. Ensuite, le champ imageChange est le pointeur d'un objet ImageChangeTrigger, qui a les champs suivants:

- LastTriggeredImageID: Ce champ, qui n'est pas affiché dans l'exemple, est déprécié dans OpenShift Dedicated 4.8 et sera ignoré dans une future version. Il contient la référence d'image résolue pour ImageStreamTag lorsque la dernière version a été déclenchée à partir de ce BuildConfig.
- en pause : Vous pouvez utiliser ce champ, qui n'est pas affiché dans l'exemple, pour désactiver temporairement ce déclencheur de changement d'image particulier.
- à partir de: Utilisez ce champ pour référencer ImageStreamTag qui pilote ce déclencheur de changement d'image. Il est le type Kubernetes de base, OwnerReference.

Le champ à partir du champ a les champs de note suivants:

- genre: Pour les déclencheurs de changement d'image, la seule valeur prise en charge est ImageStreamTag.
- espace de noms : Utilisez ce champ pour spécifier l'espace de noms de ImageStreamTag.
- le nom : Utilisez ce champ pour spécifier ImageStreamTag.

Changement d'image statut de déclencheur

Dans votre configuration de build, buildConfig.status.imageChangeTriggers est un ensemble d'éléments ImageChangeTriggerStatus. Chaque élément ImageChangeTriggerStatus inclut les éléments from, lastTriggeredImageID et lastTriggerTime affichés dans l'exemple précédent.

L'imageChangeTriggerStatus qui a la dernière version lastTriggerTime a déclenché la version la plus récente. Il utilise son nom et son espace de noms pour identifier le déclencheur de changement d'image dans buildConfig.spec.triggers qui a déclenché la construction.

Le dernierTriggerTime avec l'horodatage le plus récent signifie le ImageChangeTriggerStatus de la dernière version. Cette ImageChangeTriggerStatus a le même nom et espace de noms que le déclencheur de changement d'image dans buildConfig.spec.triggers qui a déclenché la construction.

Ressources supplémentaires

- registres des conteneurs v1

7.1.4. Déclencheurs de changement de configuration

Le déclencheur de changement de configuration permet d'invoquer automatiquement une build dès la création d'un nouveau BuildConfig.

Ce qui suit est un exemple de définition de déclencheur YAML dans le BuildConfig:

```
type: "ConfigChange"
```

NOTE

Les déclencheurs de changement de configuration ne fonctionnent actuellement que lors de la création d'un nouveau BuildConfig. Dans une version ultérieure, les déclencheurs de changement de configuration seront également en mesure de lancer une build chaque fois qu'un BuildConfig est mis à jour.

7.1.4.1. Configurer les déclencheurs manuellement

Les déclencheurs peuvent être ajoutés et supprimés des configurations de build avec des déclencheurs oc set.

Procédure

- Afin de définir un déclencheur GitHub webhook sur une configuration de build, entrez la commande suivante:

```
$ oc set triggers bc <name> --from-github
```

- Afin de définir un déclencheur de changement d'image, entrez la commande suivante:

```
$ oc set triggers bc <name> --from-image='<image>'
```

- Afin de supprimer un déclencheur, entrez la commande suivante:

```
$ oc set triggers bc <name> --from-bitbucket --remove
```

NOTE

Lorsqu'un déclencheur webhook existe déjà, l'ajouter à nouveau régénère le secret du webhook.

Consultez la documentation d'aide en saisissant la commande suivante:

```
$ oc set triggers --help
```

7.2. CONSTRUIRE DES CROCHETS

Les crochets de construction permettent d'injecter le comportement dans le processus de construction.

Le champ postCommit d'un objet BuildConfig exécute des commandes à l'intérieur d'un conteneur temporaire qui exécute l'image de sortie de build. Le crochet est exécuté immédiatement après que la dernière couche de l'image a été commise et avant que l'image ne soit poussée à un registre.

Le répertoire de travail actuel est défini sur le WORKDIR de l'image, qui est le répertoire de travail par défaut de l'image conteneur. Dans la plupart des images, c'est là que se trouve le code source.

Le crochet échoue si le script ou la commande renvoie un code de sortie non nul ou si le démarrage du conteneur temporaire échoue. Lorsque le crochet échoue, il marque la construction comme ayant échoué et l'image n'est pas poussée vers un registre. La raison de l'échec peut être inspectée en regardant les journaux de construction.

Les crochets de construction peuvent être utilisés pour exécuter des tests unitaires pour vérifier l'image

avant que la construction ne soit marquée et que l'image soit disponible dans un registre. Lorsque tous les tests passent et que le coureur d'essai retourne avec le code de sortie 0, la construction est marquée avec succès. En cas de défaillance du test, la construction est marquée comme ayant échoué. Dans tous les cas, le journal de construction contient la sortie du coureur de test, qui peut être utilisé pour identifier les tests échoués.

Le crochet postCommit est non seulement limité à l'exécution de tests, mais peut également être utilisé pour d'autres commandes. Comme il fonctionne dans un conteneur temporaire, les modifications apportées par le crochet ne persistent pas, ce qui signifie que l'exécution du crochet ne peut pas affecter l'image finale. Ce comportement permet, entre autres, l'installation et l'utilisation de dépendances de test qui sont automatiquement éliminées et ne sont pas présentes dans l'image finale.

7.2.1. Configuration des crochets de construction de post commit

Il existe différentes façons de configurer le crochet post-construction. Dans les exemples suivants, tous les formulaires sont équivalents et exécutent le test de rake de bundle exec --verbose.

Procédure

- Employez l'une des options suivantes pour configurer les crochets post-build:

L'option	Description

Le script shell	`postCommit:`
	`  script: "bundle exec rake test --verbose"`
	
	La valeur du script est un script shell à exécuter avec /bin/sh -ic. Cette option est utilisée lorsqu'un script shell est approprié pour exécuter le crochet de construction. À titre d'exemple, pour l'exécution de tests unitaires comme ci-dessus. Contrôler le point d'entrée de l'image ou si l'image n'a pas /bin/sh, utiliser la commande, ou args, ou les deux.
	
	 
	NOTE
	
	Le drapeau -i supplémentaire a été introduit pour améliorer l'expérience de travail avec les images CentOS et RHEL, et pourrait être supprimé dans une version ultérieure.

L'option	Description

Commande comme point d'entrée d'image	`postCommit:`
	`  command: ["/bin/bash", "-c", "bundle exec rake test --verbose"]`
	
	Dans cette forme, la commande est la commande à exécuter, qui remplace le point d'entrée de l'image dans le formulaire exec, comme documenté dans la référence Dockerfile. Ceci est nécessaire si l'image n'a pas /bin/sh, ou si vous ne voulez pas utiliser un shell. Dans tous les autres cas, l'utilisation du script peut être plus pratique.

Commande avec arguments	`postCommit:`
	`  command: ["bundle", "exec", "rake", "test"]`
	`  args: ["--verbose"]`
	
	Ce formulaire est équivalent à l'ajout des arguments à la commande.

NOTE

Fournir à la fois le script et la commande crée simultanément un crochet de construction invalide.

7.2.2. En utilisant le CLI pour définir les crochets de construction de post commit

La commande oc set build-hook peut être utilisée pour définir le crochet de construction pour une configuration de construction.

Procédure

1. Effectuer l'une des actions suivantes:

   - Afin de définir une commande en tant que crochet post-commit build, entrez la commande suivante:

     ```
     $ oc set build-hook bc/mybc \
         --post-commit \
         --command \
         -- bundle exec rake test --verbose
     ```

   - Afin de définir un script en tant que crochet post-commit build, entrez la commande suivante:

     ```
     $ oc set build-hook bc/mybc --post-commit --script="bundle exec rake test --verbose"
     ```

CHAPITRE 8. EFFECTUER DES CONSTRUCTIONS AVANCÉES

Il est possible de définir les ressources de construction et la durée maximale, d'attribuer des builds à des nœuds, de construire des chaînes, de tailler des builds et de configurer des stratégies d'exécution de construction.

8.1. CONFIGURATION DES RESSOURCES DE CONSTRUCTION

Les builds par défaut sont complétés par des pods utilisant des ressources non liées, telles que la mémoire et le CPU. Ces ressources peuvent être limitées.

Procédure

Il est possible de limiter l'utilisation des ressources de deux manières:

- Limitez l'utilisation des ressources en spécifiant les limites de ressources dans les limites de conteneur par défaut d'un projet.
- Limitez l'utilisation des ressources en spécifiant les limites de ressources dans le cadre de la configuration de construction.
  - Dans l'exemple suivant, chacune des ressources, cpu et paramètres de mémoire sont facultatifs:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  resources:
    limits:
      cpu: "100m" 1
      memory: "256Mi" 2
```

1 CPU est en unités CPU: 100m représente 0,1 unité CPU (100 * 1e-3).

2 la mémoire est en octets: 256Mi représente 268435456 octets (256 * 2 ^ 20).

Cependant, si un quota a été défini pour votre projet, l'un des deux éléments suivants est requis:

- D'une section de ressources assortie d'une demande explicite:

```
resources:
  requests: 1
    cpu: "100m"
    memory: "256Mi"
```

1 L'objet demande contient la liste des ressources qui correspondent à la liste des ressources dans le quota.

- Limite définie dans votre projet, où les valeurs par défaut de l'objet LimitRange s'appliquent aux pods créés au cours du processus de construction.

Dans le cas contraire, la création de la gousse de construction échouera, invoquant un défaut de satisfaire les quotas.

8.2. DÉFINITION DE LA DURÉE MAXIMALE

Lors de la définition d'un objet BuildConfig, vous pouvez définir sa durée maximale en définissant le champ CompleteDeadlineSeconds. Il est spécifié en secondes et n'est pas défini par défaut. Lorsqu'il n'est pas défini, il n'y a pas de durée maximale appliquée.

La durée maximale est comptée à partir du moment où un groupe de construction est programmé dans le système, et définit combien de temps il peut être actif, y compris le temps nécessaire pour tirer l'image du constructeur. Après avoir atteint le délai spécifié, la construction est terminée par OpenShift Dedicated.

Procédure

- Afin de définir la durée maximale, spécifiez l'achèvementDeadlineSeconds dans votre BuildConfig. L'exemple suivant montre la partie d'un BuildConfig spécifiant l'achèvement du champDeadlineSeconds pendant 30 minutes:

```
spec:
  completionDeadlineSeconds: 1800
```

NOTE

Ce paramètre n'est pas pris en charge avec l'option Stratégie des pipelines.

8.3. ASSIGNER DES BUILDS À DES NŒUDS SPÉCIFIQUES

Les builds peuvent être ciblés pour s'exécuter sur des nœuds spécifiques en spécifiant les étiquettes dans le champ nodeSelector d'une configuration de build. La valeur nodeSelector est un ensemble de paires clé-valeur qui sont appariées aux étiquettes Node lors de la planification de la gousse de construction.

La valeur nodeSelector peut également être contrôlée par des valeurs par défaut et outrepassées à l'échelle du cluster. Les valeurs par défaut ne seront appliquées que si la configuration de build ne définit pas de paires clé-valeur pour le nodeSelector et ne définit pas non plus une valeur cartographique explicitement vide de nodeSelector:{}. Les valeurs de dépassement remplaceront les valeurs dans la configuration de build sur une base clé par clé.

NOTE

Lorsque le NodeSelector spécifié ne peut pas être apparié à un nœud avec ces étiquettes, la construction reste indéfiniment dans l'état en attente.

Procédure

- Assignez des builds à exécuter sur des nœuds spécifiques en attribuant des étiquettes dans le champ nodeSelector du BuildConfig, par exemple:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
```

```
  name: "sample-build"
spec:
  nodeSelector: 1
    key1: value1
    key2: value2
```

1 Les builds associés à cette configuration de build s'exécutent uniquement sur les nœuds avec les étiquettes key1=value2 et key2=value2.

8.4. CONSTRUCTIONS ENCHAÎNÉES

Dans le cas des langages compilés tels que Go, C++ et Java, y compris les dépendances nécessaires à la compilation dans l'image de l'application, pourraient augmenter la taille de l'image ou introduire des vulnérabilités qui peuvent être exploitées.

Afin d'éviter ces problèmes, deux constructions peuvent être enchaînées ensemble. D'une construction qui produit l'artefact compilé, et une seconde qui place cet artéfact dans une image séparée qui exécute l'artefact.

8.5. CONSTRUCTIONS D'ÉLAGAGE

Les constructions qui ont terminé leur cycle de vie sont maintenues indéfiniment. Il est possible de limiter le nombre de versions précédentes qui sont conservées.

Procédure

1. Limitez le nombre de versions précédentes qui sont conservées en fournissant une valeur entière positive pour réussirBuildsHistoryLimit ou failBuildsHistoryLimit dans votre BuildConfig, par exemple:

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  successfulBuildsHistoryLimit: 2 1
  failedBuildsHistoryLimit: 2 2
```

1 SuccessBuildsHistoryLimit conservera jusqu'à deux versions avec un statut terminé.

2 failBuildsHistoryLimit conservera jusqu'à deux versions avec un statut d'échec, d'annulation ou d'erreur.

2. Déclenchez l'élagage de construction par l'une des actions suivantes:
   - La mise à jour d'une configuration de build.
   - En attendant qu'une construction complète son cycle de vie.

Les constructions sont triées par leur horodatage de création avec les plus anciennes constructions élaguées en premier.

8.6. CONSTRUIRE UNE STRATÉGIE D'EXÉCUTION

La stratégie d'exécution de construction décrit l'ordre dans lequel les builds créés à partir de la configuration de construction doivent s'exécuter. Cela peut être fait en modifiant la valeur du champ runPolicy dans la section Spécifications de la spécification Build.

Il est également possible de modifier la valeur runPolicy pour les configurations de build existantes, en:

- Changer de parallèle en série ou sérieLatest Seulement et déclencher une nouvelle construction à partir de cette configuration provoque la nouvelle construction à attendre que toutes les constructions parallèles soient terminées car la construction en série ne peut fonctionner que seule.

- Changer de série en SerialLatestOnly et déclencher une nouvelle construction provoque l'annulation de toutes les versions existantes dans la file d'attente, à l'exception de la version en cours d'exécution et de la version la plus récente. La construction la plus récente se déroule ensuite.

CHAPITRE 9. EN UTILISANT LES ABONNEMENTS RED HAT DANS LES BUILDS

Les sections suivantes permettent d'installer le contenu d'abonnement Red Hat dans OpenShift Dedicated builds.

9.1. CRÉATION D'UNE BALISE DE FLUX D'IMAGES POUR L'IMAGE DE BASE UNIVERSELLE RED HAT

Afin d'installer les paquets Red Hat Enterprise Linux (RHEL) dans une version, vous pouvez créer une balise de flux d'images pour faire référence à l'image de base universelle Red Hat (UBI).

Afin de rendre l'UBI disponible dans chaque projet du cluster, ajoutez la balise flux d'image à l'espace de noms openshift. Dans le cas contraire, pour le rendre disponible dans un projet spécifique, ajoutez la balise flux d'image à ce projet.

Les balises de flux d'images permettent l'accès à l'UBI en utilisant les identifiants Registry.redhat.io qui sont présents dans le secret d'installation, sans exposer le secret d'attraction à d'autres utilisateurs. Cette méthode est plus pratique que d'exiger que chaque développeur installe des secrets de traction avec des identifiants Registry.redhat.io dans chaque projet.

Procédure

- Afin de créer une ressource ImageStreamTag dans un seul projet, entrez la commande suivante:

```
$ oc tag --source=docker registry.redhat.io/ubi9/ubi:latest ubi:latest
```

ASTUCE

Alternativement, vous pouvez appliquer le YAML suivant pour créer une ressource ImageStreamTag dans un seul projet:

```
apiVersion: image.openshift.io/v1
kind: ImageStream
metadata:
  name: ubi9
spec:
  tags:
  - from:
      kind: DockerImage
      name: registry.redhat.io/ubi9/ubi:latest
    name: latest
    referencePolicy:
      type: Source
```

9.2. AJOUT DE DROITS D'ABONNEMENT EN TANT QUE SECRET DE CONSTRUCTION

Les builds qui utilisent des abonnements Red Hat pour installer du contenu doivent inclure les clés de droit comme secret de construction.

Conditions préalables

.

- Il faut avoir accès au cluster en tant qu'utilisateur avec le rôle cluster-admin ou avoir la permission d'accéder à des secrets dans le projet openshift-config-gaged.

Procédure

1. Copiez le secret de droit à partir de l'espace de noms openshift-config-gaged dans l'espace de noms de la construction en entrant les commandes suivantes:

```
$ cat << EOF > secret-template.txt
kind: Secret
apiVersion: v1
metadata:
  name: etc-pki-entitlement
type: Opaque
data: {{ range \$key, \$value := .data }}
  {{ \$key }}: {{ \$value }} {{ end }}
EOF
$ oc get secret etc-pki-entitlement -n openshift-config-managed -o=go-template-file --
template=secret-template.txt | oc apply -f -
```

2. Ajoutez le secret de propriété etc-pki en tant que volume de construction dans la stratégie Docker de la configuration de construction:

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
      name: ubi9:latest
    volumes:
    - name: etc-pki-entitlement
      mounts:
      - destinationPath: /etc/pki/entitlement
      source:
        type: Secret
        secret:
          secretName: etc-pki-entitlement
```

9.3. EXÉCUTION DE BUILDS AVEC LE GESTIONNAIRE D'ABONNEMENT

9.3.1. Docker construit à l'aide du Gestionnaire d'abonnement

Les builds de stratégie Docker peuvent utiliser yum ou dnf pour installer des paquets Red Hat Enterprise Linux (RHEL) supplémentaires.

Conditions préalables

- Les clés d'admissibilité doivent être ajoutées en tant que volumes de stratégie de construction.

Procédure

- Comme exemple Dockerfile, utilisez ce qui suit pour installer du contenu avec le Gestionnaire d'abonnement:

```
FROM registry.redhat.io/ubi9/ubi:latest
RUN rm -rf /etc/rhsm-host (1)
RUN yum --enablerepo=codeready-builder-for-rhel-9-x86_64-rpms install \ (2)
   nss_wrapper \
   uid_wrapper -y && \
   yum clean all -y
RUN ln -s /run/secrets/rhsm /etc/rhsm-host (3)
```

(1) Il faut inclure la commande pour supprimer le répertoire /etc/rhsm-host et tout son contenu dans votre Dockerfile avant d'exécuter les commandes yum ou dnf.

(2) Le navigateur de paquets Red Hat vous permet de trouver les référentiels appropriés pour vos paquets installés.

(3) Il faut restaurer le lien symbolique /etc/rhsm-host pour garder votre image compatible avec d'autres images de conteneur Red Hat.

9.4. EXÉCUTION DE BUILDS AVEC LES ABONNEMENTS RED HAT SATELLITE

9.4.1. Ajout de configurations Red Hat Satellite aux builds

Les constructions qui utilisent Red Hat Satellite pour installer du contenu doivent fournir des configurations appropriées pour obtenir du contenu des dépôts Satellite.

Conditions préalables

- Il vous faut fournir ou créer un fichier de configuration de référentiel compatible yum qui télécharge du contenu depuis votre instance Satellite.

  Configuration de référentiel d'échantillons

```
[test-<name>]
name=test-<number>
baseurl = https://satellite.../content/dist/rhel/server/7/7Server/x86_64/os
enabled=1
gpgcheck=0
sslverify=0
sslclientkey = /etc/pki/entitlement/...-key.pem
sslclientcert = /etc/pki/entitlement/....pem
```

Procédure

1. Créez un objet ConfigMap contenant le fichier de configuration du dépôt Satellite en entrant la commande suivante:

```
$ oc create configmap yum-repos-d --from-file /path/to/satellite.repo
```

2. Ajoutez la configuration du dépôt Satellite et la clé de droit en tant que volume de construction:

```
strategy:
  dockerStrategy:
    from:
      kind: ImageStreamTag
      name: ubi9:latest
    volumes:
    - name: yum-repos-d
      mounts:
      - destinationPath: /etc/yum.repos.d
      source:
        type: ConfigMap
        configMap:
          name: yum-repos-d
    - name: etc-pki-entitlement
      mounts:
      - destinationPath: /etc/pki/entitlement
      source:
        type: Secret
        secret:
          secretName: etc-pki-entitlement
```

9.4.2. Docker construit à l'aide d'abonnements Red Hat Satellite

Les builds de stratégie Docker peuvent utiliser les dépôts Red Hat Satellite pour installer du contenu d'abonnement.

Conditions préalables

- En tant que volumes de construction, vous avez ajouté les clés de droit et les configurations de dépôt Satellite.

Procédure

- À l'aide de l'exemple suivant, créez un Dockerfile pour installer du contenu avec Satellite:

```
FROM registry.redhat.io/ubi9/ubi:latest
RUN rm -rf /etc/rhsm-host (1)
RUN yum --enablerepo=codeready-builder-for-rhel-9-x86_64-rpms install \ (2)
    nss_wrapper \
    uid_wrapper -y && \
    yum clean all -y
RUN ln -s /run/secrets/rhsm /etc/rhsm-host (3)
```

(1) Il faut inclure la commande pour supprimer le répertoire /etc/rhsm-host et tout son contenu dans votre Dockerfile avant d'exécuter les commandes yum ou dnf.

(2) Contactez votre administrateur système Satellite pour trouver les référentiels appropriés pour les paquets installés de la construction.

(3) Il faut restaurer le lien symbolique /etc/rhsm-host pour garder votre image compatible avec d'autres images de conteneur Red Hat.

Ressources supplémentaires

- Comment utiliser les builds avec les abonnements Red Hat Satellite et quel certificat utiliser

9.5. RESSOURCES SUPPLÉMENTAIRES

- Gestion des flux d'images
- Construire des stratégies

CHAPITRE 10. DÉPANNAGE DES CONSTRUCTIONS

Faites ce qui suit pour résoudre les problèmes de construction.

10.1. LA RÉSOLUTION DU REFUS D'ACCÈS AUX RESSOURCES

En cas de refus de votre demande d'accès aux ressources:

La question

La construction échoue avec:

```
requested access to the resource is denied
```

La résolution

« vous avez dépassé l'un des quotas d'image fixés sur votre projet. Consultez votre quota actuel et vérifiez les limites appliquées et le stockage en cours d'utilisation:

```
$ oc describe quota
```

10.2. ÉCHEC DE GÉNÉRATION DE CERTIFICAT DE SERVICE

En cas de refus de votre demande d'accès aux ressources:

La question

En cas d'échec d'une génération de certificat de service (l'annotation d'erreur service.beta.openshift.io/serving-cert-generation-error annotation contient):

Exemple de sortie

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

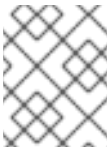
La résolution

Le service qui a généré le certificat n'existe plus, ou a un serviceUID différent. Il faut forcer la régénération des certificats en supprimant l'ancien secret et en éliminant les annotations suivantes sur le service: service.beta.openshift.io/serving-cert-generation-error et service.beta.openshift.io/serving-cert-generation-error-num. Afin d'effacer les annotations, entrez les commandes suivantes:

```
$ oc delete secret <secret_name>
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-num-
```



NOTE

La commande supprimant une annotation a un - après le nom d'annotation à supprimer.