OpenShift Dedicated 4

Opérateurs

Opérateurs dédiés à OpenShift

Last Updated: 2025-05-27

OpenShift Dedicated 4 Opérateurs

Opérateurs dédiés à OpenShift

Notice légale

Résumé

Comment les opérateurs aident à emballer, déployer et gérer les services sur le plan de contrôle.

Table des matières

CHAPITRE 1. APERÇU DES OPÉRATEURS

Les opérateurs sont parmi les composants les plus importants d'OpenShift Dedicated. Ils sont la méthode préférée d'emballage, de déploiement et de gestion des services sur le plan de contrôle. Ils peuvent également fournir des avantages aux applications que les utilisateurs exécutent.

Les opérateurs s'intègrent aux API Kubernetes et aux outils CLI tels que kubectl et OpenShift CLI (oc). Ils fournissent les moyens de surveiller les applications, d'effectuer des contrôles de santé, de gérer les mises à jour en direct (OTA) et de s'assurer que les applications restent dans votre état spécifié.

Les opérateurs sont conçus spécifiquement pour les applications natives de Kubernetes afin d'implémenter et d'automatiser les opérations courantes de jour 1, telles que l'installation et la configuration. Les opérateurs peuvent également automatiser les opérations du Jour 2, telles que l'autoscalisation vers le haut ou vers le bas et la création de sauvegardes. L'ensemble de ces activités est dirigée par un logiciel fonctionnant sur votre cluster.

Bien que les deux suivent des concepts et des objectifs similaires, les opérateurs dans OpenShift Dedicated sont gérés par deux systèmes différents, en fonction de leur objectif:

Les opérateurs de clusters

Géré par l'opérateur de versions de cluster (CVO) et installé par défaut pour effectuer des fonctions de cluster.

Opérateurs complémentaires optionnels

Géré par Operator Lifecycle Manager (OLM) et peut être rendu accessible pour les utilisateurs à exécuter dans leurs applications. Également connu sous le nom d'opérateurs basés sur OLM.

1.1. DESTINÉ AUX DÉVELOPPEURS

En tant qu'auteur de l'opérateur, vous pouvez effectuer les tâches de développement suivantes pour les opérateurs basés sur OLM:

- Installez l'opérateur SDK CLI.
- Créer des Opérateurs Go-based, des Opérateurs Ansibles et des Opérateurs basés sur Helm.
- Employez le SDK de l'opérateur pour construire, tester et déployer un opérateur.
- Créez une application à partir d'un opérateur installé via la console Web.

1.2. ADMINISTRATEURS POUR LES ADMINISTRATEURS

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez effectuer les tâches suivantes:

- Gérez des catalogues personnalisés.
- Installez un opérateur à partir de OperatorHub.
- Afficher l'état de l'opérateur.
- Gérer les conditions de l'opérateur.
- La mise à niveau des opérateurs installés.
- Effacer les opérateurs installés.

- Configurer le support proxy.

1.3. LES PROCHAINES ÉTAPES

En savoir plus sur les opérateurs, voir Qu'est-ce que les opérateurs?

CHAPITRE 2. COMPRENDRE LES OPÉRATEURS

2.1. EN QUOI CONSISTENT LES OPÉRATEURS?

Conceptuellement, les opérateurs prennent les connaissances opérationnelles humaines et les encodent dans des logiciels plus facilement partagés avec les consommateurs.

Les opérateurs sont des logiciels qui facilitent la complexité opérationnelle de l'exécution d'un autre logiciel. Ils agissent comme une extension de l'équipe d'ingénierie du fournisseur de logiciels, en surveillant un environnement Kubernetes (comme OpenShift Dedicated) et en utilisant son état actuel pour prendre des décisions en temps réel. Les opérateurs avancés sont conçus pour gérer les mises à niveau de manière transparente, réagir aux pannes automatiquement et ne pas prendre de raccourcis, comme sauter un processus de sauvegarde logiciel pour gagner du temps.

Les opérateurs sont plus techniquement une méthode d'emballage, de déploiement et de gestion d'une application Kubernetes.

L'application Kubernetes est une application qui est déployée sur Kubernetes et gérée à l'aide des API Kubernetes et des outils kubectl ou oc. Afin de pouvoir tirer le meilleur parti des Kubernetes, vous avez besoin d'un ensemble d'API cohérentes à étendre afin d'assurer le service et la gestion de vos applications qui s'exécutent sur Kubernetes. Considérez les Opérateurs comme le runtime qui gère ce type d'application sur Kubernetes.

2.1.1. Comment utiliser les opérateurs?

Les opérateurs fournissent:

- La répétabilité de l'installation et de la mise à niveau.
- Contrôles de santé constants de chaque composant du système.
- Des mises à jour en direct (OTA) pour les composants OpenShift et le contenu ISV.
- C'est un endroit pour encapsuler les connaissances des ingénieurs de terrain et les diffuser à tous les utilisateurs, pas seulement un ou deux.

Comment déployer sur Kubernetes?

Kubernetes (et par extension, OpenShift Dedicated) contient tous les primitifs nécessaires pour construire des systèmes distribués complexes - gestion secrète, équilibrage de charge, découverte de services, autoscaling - qui fonctionnent sur site et fournisseurs de cloud.

Comment gérer votre application avec les API Kubernetes et les outils kubectl?

Ces API sont riches en fonctionnalités, ont des clients pour toutes les plates-formes et se branchent dans le contrôle d'accès / audit du cluster. L'opérateur utilise le mécanisme d'extension Kubernetes, les définitions de ressources personnalisées (CRD), de sorte que votre objet personnalisé, par exemple MongoDB, ressemble et agit comme les objets Kubernetes natifs intégrés.

Comment les opérateurs se comparent-ils aux courtiers de services?

Le courtier de services est une étape vers la découverte programmatique et le déploiement d'une application. Cependant, parce qu'il ne s'agit pas d'un processus de longue durée, il ne peut pas exécuter les opérations de jour 2 comme la mise à niveau, le basculement ou la mise à l'échelle. Les personnalisations et paramétrages des réglages sont fournis au moment de l'installation, par rapport à un opérateur qui surveille constamment l'état actuel de votre cluster. Les services hors-clus sont un bon match pour un courtier de services, bien que les opérateurs existent également pour ceux-ci.

2.1.2. Cadre de l'opérateur

Le Cadre d'opérateur est une famille d'outils et de capacités pour fournir l'expérience client décrite ci-dessus. Il ne s'agit pas seulement d'écrire du code; tester, livrer et mettre à jour les opérateurs est tout aussi important. Les composants du cadre opérateur sont constitués d'outils open source pour résoudre ces problèmes:

Le SDK de l'opérateur

Le SDK de l'opérateur assiste les auteurs de l'opérateur dans le démarrage, la construction, l'essai et l'emballage de leur propre opérateur en fonction de leur expertise sans nécessiter de connaissance des complexités de l'API Kubernetes.

Gestionnaire du cycle de vie de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) contrôle l'installation, la mise à niveau et le contrôle d'accès basé sur les rôles (RBAC) des opérateurs dans un cluster. Il est déployé par défaut dans OpenShift Dedicated 4.

Registre de l'opérateur

Le Registre des opérateurs stocke les versions de services de cluster (CSV) et les définitions de ressources personnalisées (CRD) pour la création dans un cluster et stocke les métadonnées de l'opérateur sur les paquets et les canaux. Il s'exécute dans un cluster Kubernetes ou OpenShift pour fournir les données du catalogue Opérateur à OLM.

L'opérateurHub

OperatorHub est une console Web permettant aux administrateurs de cluster de découvrir et de sélectionner Opérateurs à installer sur leur cluster. Il est déployé par défaut dans OpenShift Dedicated.

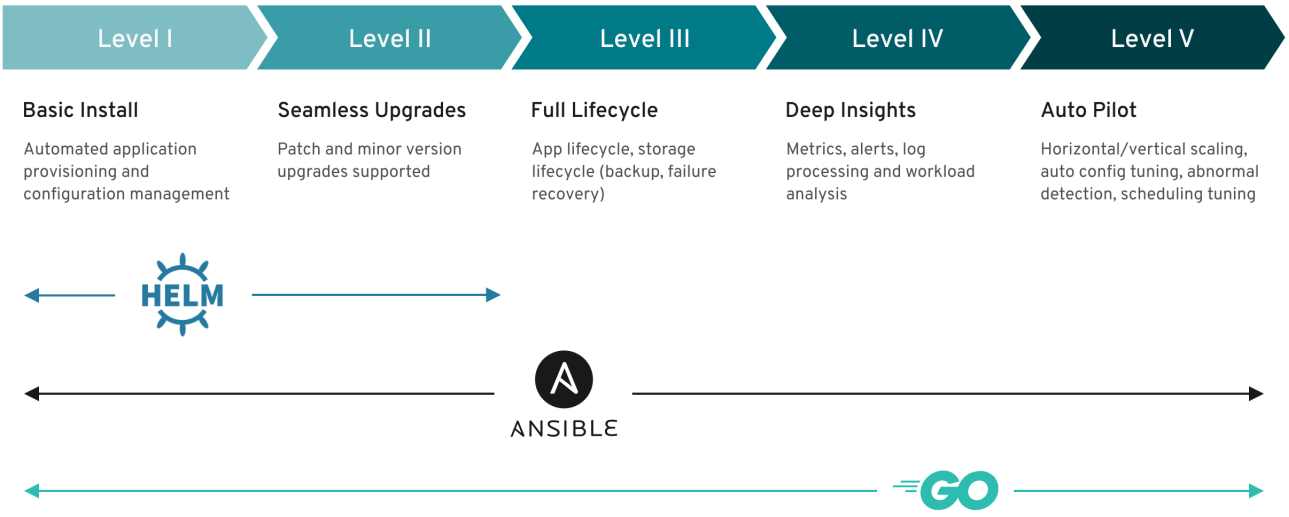
Ces outils sont conçus pour être composables, de sorte que vous pouvez utiliser tous ceux qui vous sont utiles.

2.1.3. Le modèle de maturité de l'opérateur

Le niveau de sophistication de la logique de gestion encapsulé au sein d'un opérateur peut varier. Cette logique est également en général fortement dépendante du type de service représenté par l'Opérateur.

Cependant, on peut généraliser l'échelle de maturité des opérations encapsulées d'un opérateur pour certains ensembles de capacités que la plupart des opérateurs peuvent inclure. À cette fin, le modèle de maturité de l'opérateur suivant définit cinq phases de maturité pour les opérations génériques de jour 2 d'un opérateur:

Figure 2.1. Le modèle de maturité de l'opérateur



Le modèle ci-dessus montre également comment ces capacités peuvent être développées au mieux grâce aux capacités Helm, Go et Ansible du SDK Opérateur.

2.2. FORMAT D'EMBALLAGE DU CADRE OPÉRATEUR

Ce guide décrit le format d'emballage pour les opérateurs pris en charge par Operator Lifecycle Manager (OLM) dans OpenShift Dedicated.

2.2.1. Format de paquet

Le format groupé pour les opérateurs est un format d'emballage introduit par le Cadre de l'opérateur. Afin d'améliorer l'évolutivité et de permettre aux utilisateurs en amont d'héberger leurs propres catalogues, la spécification de format groupé simplifie la distribution des métadonnées de l'opérateur.

Le bundle Opérateur représente une seule version d'un Opérateur. Les manifestes de paquets sur disque sont conteneurisés et expédiés sous forme d'image de paquet, qui est une image de conteneur non-runnable qui stocke les manifestes Kubernetes et les métadonnées de l'opérateur. Le stockage et la distribution de l'image de paquet sont ensuite gérés à l'aide d'outils de conteneurs existants tels que podman et docker et des registres de conteneurs tels que Quay.

Les métadonnées de l'opérateur peuvent inclure:

- Informations qui identifient l'opérateur, par exemple son nom et sa version.
- Des informations supplémentaires qui pilotent l'interface utilisateur, par exemple son icône et quelques exemples de ressources personnalisées (CRs).
- API requise et fournies.
- Images connexes.

Lorsque le chargement se manifeste dans la base de données du Registre de l'opérateur, les exigences suivantes sont validées:

- Le faisceau doit avoir au moins un canal défini dans les annotations.

- Chaque paquet a exactement une version de service cluster (CSV).
- Lorsqu'un CSV possède une définition de ressource personnalisée (CRD), ce CRD doit exister dans le paquet.

2.2.1.1. Les manifestes

Les manifestes de paquets se réfèrent à un ensemble de manifestes Kubernetes qui définissent le déploiement et le modèle RBAC de l'opérateur.

Le bundle comprend un CSV par répertoire et généralement les CRD qui définissent les API appartenant au CSV dans son répertoire /manifestes.

Exemple de mise en page de format de paquet

```
etcd
├── manifests
│   ├── etcdcluster.crd.yaml
│   └── etcdoperator.clusterserviceversion.yaml
│   └── secret.yaml
│   └── configmap.yaml
└── metadata
    └── annotations.yaml
    └── dependencies.yaml
```

Autres objets pris en charge
Les types d'objets suivants peuvent également être inclus en option dans le répertoire /manifestes d'un bundle:

Les types d'objets optionnels pris en charge

- **ClusterRole**
- **ClusterRoleBinding**
- **ConfigMap**
- **ConsoleCLIDownload**
- **ConsoleLink**
- **ConsoleQuickStart**
- **ConsoleYamlSample**
- **Budget de perturbation de Pod**
- **Classe de priorité**
- **À propos de PrometheusRule**
- **Le rôle**
- **À propos de RoleBinding**
- **Le secret**

- **Le service**
- **Compte de ServiceAccount**
- **À propos de ServiceMonitor**
- **À propos de VerticalPodAutoscaler**

Lorsque ces objets optionnels sont inclus dans un paquet, Operator Lifecycle Manager (OLM) peut les créer à partir du paquet et gérer leur cycle de vie avec le CSV:

Cycle de vie pour les objets optionnels

- Lorsque le CSV est supprimé, OLM supprime l'objet optionnel.
- Lorsque le CSV est mis à jour:
  - Lorsque le nom de l'objet optionnel est le même, OLM le met à jour en place.
  - Lorsque le nom de l'objet optionnel a changé d'une version à l'autre, OLM le supprime et le recrée.

2.2.1.2. Annotations

Le bundle inclut également un fichier annotations.yaml dans son répertoire /metadata. Ce fichier définit des données agrégées de niveau supérieur qui aident à décrire le format et les informations du paquet sur la façon dont le paquet devrait être ajouté dans un index de paquets:

Exemple annotations.yaml

```
annotations:
  operators.operatorframework.io.bundle.mediatype.v1: "registry+v1" 1
  operators.operatorframework.io.bundle.manifests.v1: "manifests/" 2
  operators.operatorframework.io.bundle.metadata.v1: "metadata/" 3
  operators.operatorframework.io.bundle.package.v1: "test-operator" 4
  operators.operatorframework.io.bundle.channels.v1: "beta,stable" 5
  operators.operatorframework.io.bundle.channel.default.v1: "stable" 6
```

1. Le type de média ou le format du paquet Opérateur. Le format Register+v1 signifie qu'il contient un CSV et ses objets Kubernetes associés.
2. Le chemin dans l'image vers le répertoire qui contient l'opérateur se manifeste. Cette étiquette est réservée à une utilisation future et actuellement par défaut aux manifestes/. La valeur manifeste.v1 implique que le paquet contient des manifestes d'opérateur.
3. Le chemin dans l'image vers le répertoire qui contient des fichiers de métadonnées sur le paquet. Cette étiquette est réservée à une utilisation future et actuellement par défaut aux métadonnées/. La valeur métadonnées.v1 implique que ce paquet possède les métadonnées de l'opérateur.
4. Le nom du paquet.
5. La liste des canaux auxquels le bundle s'abonne lorsqu'elle est ajoutée dans un Registre d'opérateur.
6. Le canal par défaut auquel un opérateur doit être souscrit lorsqu'il est installé à partir d'un registre.

NOTE

En cas d'inadéquation, le fichier annotations.yaml fait autorité car le registre de l'opérateur sur le groupe qui s'appuie sur ces annotations n'a accès qu'à ce fichier.

2.2.1.3. Dépendances

Les dépendances d'un opérateur sont listées dans un fichier de dépendances.yaml dans le dossier métadonnées d'un bundle. Ce fichier est facultatif et actuellement utilisé uniquement pour spécifier des dépendances explicites de version d'opérateur.

La liste de dépendances contient un champ de type pour chaque élément pour spécifier quel type de dépendance il s'agit. Les types de dépendances des opérateurs suivants sont pris en charge:

emballage OLM.

Ce type indique une dépendance pour une version spécifique de l'opérateur. Les informations de dépendance doivent inclure le nom du paquet et la version du paquet au format semver. À titre d'exemple, vous pouvez spécifier une version exacte telle que 0.5.2 ou une gamme de versions telles que >0.5.1.

à propos de OLM.gvk

Avec ce type, l'auteur peut spécifier une dépendance avec des informations de groupe/version/type (GVK), similaire à l'utilisation existante CRD et API dans un CSV. Il s'agit d'un chemin permettant aux auteurs de l'opérateur de consolider toutes les dépendances, API ou versions explicites, pour être au même endroit.

limite OLM.

Ce type déclare des contraintes génériques sur les propriétés arbitraires de l'opérateur.

Dans l'exemple suivant, les dépendances sont spécifiées pour un opérateur Prometheus et etcd CRD:

Exemple de fichier dépendances.yaml

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
      group: etcd.database.coreos.com
      kind: EtcdCluster
      version: v1beta2
```

Ressources supplémentaires

- La résolution de dépendance du gestionnaire de cycle de vie de l'opérateur

2.2.1.4. À propos de l'opm CLI

L'outil Opm CLI est fourni par le Cadre d'opérateur pour une utilisation avec le format de paquet Opm. Cet outil vous permet de créer et de maintenir des catalogues d'opérateurs à partir d'une liste de paquets d'opérateurs similaires aux référentiels de logiciels. Le résultat est une image de conteneur qui peut être stockée dans un registre de conteneurs puis installée sur un cluster.

Le catalogue contient une base de données de pointeurs vers l'opérateur manifeste du contenu qui peut être interrogé via une API incluse qui est servie lorsque l'image du conteneur est exécutée. Dans OpenShift Dedicated, Operator Lifecycle Manager (OLM) peut référencer l'image dans une source de catalogue, définie par un objet CatalogSource, qui sonne l'image à intervalles réguliers pour permettre des mises à jour fréquentes aux opérateurs installés sur le cluster.

- Consultez les outils CLI pour les étapes d'installation de l'opm CLI.

2.2.2. Faits saillants

Les catalogues basés sur des fichiers sont la dernière itération du format de catalogue dans Operator Lifecycle Manager (OLM). Il s'agit d'un texte simple (JSON ou YAML) et d'une évolution de configuration déclarative du format de base de données SQLite antérieur, et il est entièrement rétrocompatible. L'objectif de ce format est d'activer l'édition de catalogue de l'opérateur, la composabilité et l'extensibilité.

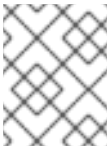
Édition de l'édition

Avec les catalogues basés sur des fichiers, les utilisateurs qui interagissent avec le contenu d'un catalogue sont en mesure d'apporter des modifications directes au format et de vérifier que leurs modifications sont valides. Comme ce format est en texte brut JSON ou YAML, les mainteneurs de catalogue peuvent facilement manipuler les métadonnées de catalogue à la main ou avec des outils JSON ou YAML largement connus et pris en charge, tels que le jq CLI.
Cette modifiabilité permet les fonctionnalités suivantes et les extensions définies par l'utilisateur:

- La promotion d'un faisceau existant vers un nouveau canal
- Changer le canal par défaut d'un paquet
- Algorithmes personnalisés pour ajouter, mettre à jour et supprimer les chemins de mise à niveau

Composabilité

Les catalogues basés sur des fichiers sont stockés dans une hiérarchie de répertoire arbitraire, ce qui permet la composition du catalogue. À titre d'exemple, envisagez deux répertoires de catalogues distincts basés sur des fichiers : catalogA et catalogB. Le mainteneur de catalogue peut créer un nouveau catalogue combiné en créant un nouveau catalogue C et en copiant le catalogueA et le catalogueB.
Cette composabilité permet des catalogues décentralisés. Le format permet aux auteurs de l'opérateur de maintenir des catalogues spécifiques à l'opérateur, et il permet aux mainteneurs de construire trivialement un catalogue composé de catalogues individuels de l'opérateur. Les catalogues basés sur des fichiers peuvent être composés en combinant plusieurs autres catalogues, en extrayant des sous-ensembles d'un catalogue, ou une combinaison de ces deux catalogues.



NOTE

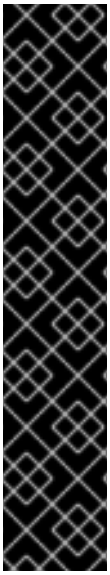
Les paquets en double et les paquets dupliqués dans un paquet ne sont pas autorisés. La commande de validation opm renvoie une erreur si des doublons sont trouvés.

Les auteurs de l'opérateur étant plus familiers avec leur opérateur, ses dépendances et sa compatibilité de mise à niveau, ils sont en mesure de maintenir leur propre catalogue spécifique à l'opérateur et d'avoir un contrôle direct sur son contenu. Avec les catalogues basés sur des fichiers, les auteurs de l'opérateur possèdent la tâche de construire et de maintenir leurs paquets dans un catalogue. Cependant, les mainteneurs de catalogues composites ne possèdent que la tâche de gérer les paquets dans leur catalogue et de le publier aux utilisateurs.

Extensibilité

La spécification de catalogue basée sur des fichiers est une représentation de bas niveau d'un catalogue. Bien qu'il puisse être maintenu directement sous sa forme de bas niveau, les mainteneurs de catalogue peuvent construire des extensions intéressantes sur le dessus qui peuvent être utilisées par leur propre outil personnalisé pour effectuer un certain nombre de mutations.
À titre d'exemple, un outil pourrait traduire une API de haut niveau, telle que (mode=semver), jusqu'au format de catalogue de bas niveau basé sur des fichiers pour les chemins de mise à niveau. Il se peut qu'un mainteneur de catalogue doive personnaliser toutes les métadonnées du paquet en ajoutant une nouvelle propriété à des paquets répondant à certains critères.

Bien que cette extensibilité permette de développer des outils officiels supplémentaires en plus des API de bas niveau pour les futures versions dédiées à OpenShift, l'avantage majeur est que les responsables du catalogue ont également cette capacité.



IMPORTANT

À partir d'OpenShift Dedicated 4.11, le catalogue de l'opérateur par défaut Red Hat est publié dans le format de catalogue basé sur des fichiers. Les catalogues d'opérateurs Red Hat fournis par défaut pour OpenShift Dedicated 4.6 à 4.10 publiés dans le format de base de données SQLite obsolète.

Les sous-commandes opm, les drapeaux et les fonctionnalités liés au format de base de données SQLite sont également obsolètes et seront supprimés dans une version ultérieure. Les fonctionnalités sont toujours prises en charge et doivent être utilisées pour les catalogues utilisant le format de base de données SQLite obsolète.

La plupart des sous-commandes et des drapeaux opm pour travailler avec le format de base de données SQLite, tels que le prune de l'index opm, ne fonctionnent pas avec le format de catalogue basé sur des fichiers. Consultez Gérer les catalogues personnalisés pour plus d'informations sur le travail avec les catalogues basés sur les fichiers.

2.2.2.1. La structure des répertoires

Les catalogues basés sur des fichiers peuvent être stockés et chargés à partir de systèmes de fichiers basés sur des répertoires. L'opm CLI charge le catalogue en marchant le répertoire racine et en récursant dans des sous-répertoires. Le CLI tente de charger tous les fichiers qu'il trouve et échoue si des erreurs se produisent.

Les fichiers non catalog peuvent être ignorés en utilisant les fichiers .indexignore, qui ont les mêmes règles pour les modèles et la préséance que les fichiers .gitignore.

Exemple de fichier .indexignore

```
# Ignore everything except non-object .json and .yaml files
**/*
!*.json
!*.yaml
**/objects/*.json
**/objects/*.yaml
```

Les responsables du catalogue ont la flexibilité de choisir la disposition souhaitée, mais il est recommandé de stocker les blobs de catalogue de chaque paquet dans des sous-répertoires distincts. Chaque fichier individuel peut être soit JSON ou YAML; il n'est pas nécessaire que chaque fichier d'un catalogue utilise le même format.

La structure de base recommandée

```
catalog
├── packageA
│   └── index.yaml
├── packageB
│   ├── .indexignore
│   ├── index.yaml
│   └── objects
│       └── packageB.v0.1.0.clusterserviceversion.yaml
└── packageC
    └── index.json
    └── deprecations.yaml
```

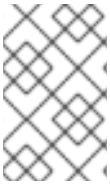
Cette structure recommandée a la propriété que chaque sous-répertoire de la hiérarchie des répertoires est un catalogue autonome, ce qui rend les opérations triviales de système de fichiers triviales de composition, de découverte et de navigation du catalogue. Le catalogue peut également être inclus dans un catalogue parent en le copiant dans le répertoire racine du catalogue parent.

2.2.2.2. Les schémas

Les catalogues basés sur des fichiers utilisent un format, basé sur la spécification du langage CUE, qui peut être étendu avec des schémas arbitraires. Le schéma _Meta CUE suivant définit le format auquel tous les blobs de catalogues basés sur des fichiers doivent adhérer:

_Méta schema

```
_Meta: {
  // schema is required and must be a non-empty string
  schema: string & !=""

  // package is optional, but if it's defined, it must be a non-empty string
  package?: string & !=""

  // properties is optional, but if it's defined, it must be a list of 0 or more properties
  properties?: [... #Property]
}

#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}
```



NOTE

Aucun schéma CUE listé dans cette spécification ne doit être considéré comme exhaustif. La commande de validation opm a des validations supplémentaires qui sont difficiles ou impossibles à exprimer de manière concise dans CUE.

Le catalogue de gestion du cycle de vie de l'opérateur (OLM) utilise actuellement trois schémas (olm.package, olm.channel et olm.bundle), qui correspondent aux concepts de paquets et de paquets existants d'OLM.

Chaque paquet Opérateur dans un catalogue nécessite exactement un blob olm.package, au moins un blob olm.channel et un ou plusieurs blobs olm.bundle.

NOTE

Les schémas Olm.* sont réservés aux schémas définis par OLM. Les schémas personnalisés doivent utiliser un préfixe unique, tel qu'un domaine que vous possédez.

2.2.2.2.1. schéma OLM.package

Le schéma olm.package définit les métadonnées au niveau du paquet pour un opérateur. Cela inclut son nom, sa description, son canal par défaut et son icône.

Exemple 2.1. schéma OLM.package

```
#Package: {
  schema: "olm.package"

  // Package name
  name: string & !=""

  // A description of the package
  description?: string

  // The package's default channel
  defaultChannel: string & !=""

  // An optional icon
  icon?: {
    base64data: string
    mediatype:  string
  }
}
```

2.2.2.2.2. schéma OLM.canal

Le schéma olm.channel définit un canal à l'intérieur d'un paquet, les entrées de paquet qui sont membres du canal et les chemins de mise à niveau pour ces paquets.

Lorsqu'une entrée de paquet représente un bord dans plusieurs blobs olm.canal, elle ne peut apparaître qu'une seule fois par canal.

Il est valable pour une valeur de remplacement d'une entrée de référencer un autre nom de paquet qui ne peut pas être trouvé dans ce catalogue ou un autre catalogue. Cependant, tous les autres invariants de canal doivent tenir vrai, comme un canal n'ayant pas plusieurs têtes.

Exemple 2.2. schéma OLM.canal

```
#Channel: {
```

```
  schema: "olm.channel"
  package: string & !=""
  name: string & !=""
  entries: [...#ChannelEntry]
}

#ChannelEntry: {
  // name is required. It is the name of an `olm.bundle` that
  // is present in the channel.
  name: string & !=""

  // replaces is optional. It is the name of bundle that is replaced
  // by this entry. It does not have to be present in the entry list.
  replaces?: string & !=""

  // skips is optional. It is a list of bundle names that are skipped by
  // this entry. The skipped bundles do not have to be present in the
  // entry list.
  skips?: [...string & !=""]

  // skipRange is optional. It is the semver range of bundle versions
  // that are skipped by this entry.
  skipRange?: string & !=""
}
```

AVERTISSEMENT

Lors de l'utilisation du champ skipRange, les versions de l'opérateur sauté sont taillées à partir du graphique de mise à jour et sont plus longues installables par les utilisateurs avec la propriété spec.startingCSV des objets d'abonnement.

Il est possible de mettre à jour progressivement un opérateur tout en gardant les versions précédemment installées à la disposition des utilisateurs pour l'installation future en utilisant à la fois le champ skipRange et le champ de remplacement. Assurez-vous que le champ remplace la version précédente immédiate de la version de l'opérateur en question.

2.2.2.2.3. schéma OLM.bundle

Exemple 2.3. schéma OLM.bundle

```
#Bundle: {
  schema: "olm.bundle"
  package: string & !=""
  name: string & !=""
  image: string & !=""
  properties: [...#Property]
  relatedImages?: [...#RelatedImage]
}
```

```
#Property: {
  // type is required
  type: string & !=""

  // value is required, and it must not be null
  value: !=null
}

#RelatedImage: {
  // image is the image reference
  image: string & !=""

  // name is an optional descriptive name for an image that
  // helps identify its purpose in the context of the bundle
  name?: string & !=""
}
```

2.2.2.2.4. schéma OLM.déprécations

Le schéma optionnel olm.deprecations définit les informations de déprécation pour les paquets, les paquets et les canaux dans un catalogue. Les auteurs d'opérateurs peuvent utiliser ce schéma pour fournir des messages pertinents sur leurs opérateurs, tels que l'état du support et les chemins de mise à niveau recommandés, aux utilisateurs exécutant ces opérateurs à partir d'un catalogue.

Lorsque ce schéma est défini, la console Web OpenShift Dedicated affiche des badges d'avertissement pour les éléments affectés de l'opérateur, y compris les messages de déprécation personnalisés, sur les pages de pré- et de post-installation de l'opérateur.

L'entrée de schéma olm.deprecations contient un ou plusieurs des types de référence suivants, qui indique la portée de déprécation. Après l'installation de l'opérateur, tous les messages spécifiés peuvent être considérés comme des conditions d'état sur l'objet d'abonnement associé.

Tableau 2.1. Les types de référence de déprécation

Le type	Champ d'application	État du statut
emballage OLM.	Représente l'ensemble du paquet	**Forfait déprécié**
canal OLM.	Il représente un canal	**ChannelDeprécated**
à propos de OLM.bundle	Il représente une version de bundle	**BundleDeprecated**

Chaque type de référence a ses propres exigences, comme détaillé dans l'exemple suivant.

Exemple 2.4. Exemple de schéma olm.deprecations avec chaque type de référence

```
schema: olm.deprecations
package: my-operator 1
entries:
  - reference:
```

```
    schema: olm.package 2
  message: | 3
  The 'my-operator' package is end of life. Please use the
  'my-operator-new' package for support.
- reference:
    schema: olm.channel
    name: alpha 4
  message: |
  The 'alpha' channel is no longer supported. Please switch to the
  'stable' channel.
- reference:
    schema: olm.bundle
    name: my-operator.v1.68.0 5
  message: |
  my-operator.v1.68.0 is deprecated. Uninstall my-operator.v1.68.0 and
  install my-operator.v1.72.0 for support.
```

1. Chaque schéma de déprécation doit avoir une valeur de paquet, et cette référence de paquet doit être unique dans tout le catalogue. Il ne doit pas y avoir de champ de nom associé.
2. Le schéma olm.package ne doit pas inclure un champ nom, car il est déterminé par le champ de paquet défini plus tôt dans le schéma.
3. Les champs de messages, pour tout type de référence, doivent être d'une longueur non nulle et représenté comme un blob de texte opaque.
4. Le champ nom du schéma olm.channel est requis.
5. Le champ nom du schéma olm.bundle est requis.

NOTE

La fonction de déprécation ne tient pas compte du chevauchement de la déprécation, par exemple le paquet par rapport au canal par rapport au faisceau.

Les auteurs de l'opérateur peuvent enregistrer les entrées de schéma olm.deprecations en tant que fichier deprecations.yaml dans le même répertoire que le fichier index.yaml du paquet:

Exemple de structure de répertoire pour un catalogue avec déprécations

```
my-catalog
└── my-operator
    ├── index.yaml
    └── deprecations.yaml
```

Ressources supplémentaires

- La mise à jour ou le filtrage d'une image de catalogue basée sur des fichiers

2.2.2.3. Les propriétés

Les propriétés sont des morceaux arbitraires de métadonnées qui peuvent être attachés à des schémas

de catalogue basés sur des fichiers. Le champ type est une chaîne qui spécifie efficacement la signification sémantique et syntaxique du champ de valeur. La valeur peut être tout JSON ou YAML arbitraire.

L'OLM définit une poignée de types de propriétés, encore une fois en utilisant le préfixe Olm.* réservé.

2.2.2.3.1. immobilier OLM.package

La propriété olm.package définit le nom et la version du paquet. Il s'agit d'une propriété requise sur les paquets, et il doit y avoir exactement l'une de ces propriétés. Le champ packageName doit correspondre au champ paquet de première classe du paquet, et le champ de version doit être une version sémantique valide.

Exemple 2.5. immobilier OLM.package

```
#PropertyPackage: {
  type: "olm.package"
  value: {
    packageName: string & !=""
    version: string & !=""
  }
}
```

2.2.2.3.2. immobilier OLM.gvk

La propriété olm.gvk définit le groupe/version/type (GVK) d'une API Kubernetes fournie par ce paquet. Cette propriété est utilisée par OLM pour résoudre un paquet avec cette propriété en tant que dépendance pour d'autres bundles qui répertorient le même GVK qu'une API requise. Le GVK doit adhérer aux validations Kubernetes GVK.

Exemple 2.6. immobilier OLM.gvk

```
#PropertyGVK: {
  type: "olm.gvk"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

2.2.2.3.3. conditions d'utilisation OLM.package.required

La propriété olm.package.required définit le nom du paquet et la gamme de versions d'un autre paquet que ce paquet nécessite. Dans chaque propriété de paquet requise, une liste de paquets, OLM s'assure qu'un opérateur est installé sur le cluster pour le paquet listé et dans la gamme de versions requise. Le champ versionRange doit être une plage sémantique valide (semver).

Exemple 2.7. la propriété OLM.package.required

```
#PropertyPackageRequired: {
```

```
  type: "olm.package.required"
  value: {
    packageName: string & !=""
    versionRange: string & !=""
  }
}
```

2.2.2.3.4. conditions d'utilisation OLM.gvk.required

La propriété olm.gvk.required définit le groupe/version/type (GVK) d'une API Kubernetes que ce paquet nécessite. Dans chaque propriété GVK requise, une liste de paquets, OLM s'assure qu'un opérateur est installé sur le cluster qui le fournit. Le GVK doit adhérer aux validations Kubernetes GVK.

Exemple 2.8. la propriété OLM.gvk.required

```
#PropertyGVKRequired: {
  type: "olm.gvk.required"
  value: {
    group: string & !=""
    version: string & !=""
    kind: string & !=""
  }
}
```

2.2.2.4. Exemple de catalogue

Avec les catalogues basés sur des fichiers, les responsables du catalogue peuvent se concentrer sur la curation et la compatibilité de l'opérateur. Étant donné que les auteurs de l'opérateur ont déjà produit des catalogues spécifiques à l'opérateur pour leurs opérateurs, les responsables du catalogue peuvent construire leur catalogue en rendant chaque catalogue Opérateur dans un sous-répertoire du répertoire racine du catalogue.

Il existe de nombreuses façons possibles de créer un catalogue basé sur des fichiers; les étapes suivantes décrivent une approche simple:

1. Conserver un fichier de configuration unique pour le catalogue, contenant des références d'image pour chaque opérateur dans le catalogue:

   Exemple de fichier de configuration du catalogue

   ```
   name: community-operators
   repo: quay.io/community-operators/catalog
   tag: latest
   references:
   - name: etcd-operator
     image: quay.io/etcd-
   operator/index@sha256:5891b5b522d5df086d0ff0b110fbd9d21bb4fc7163af34d08286a2e846f
   6be03
   - name: prometheus-operator
     image: quay.io/prometheus-
   operator/index@sha256:e258d248fda94c63753607f7c4494ee0fcbe92f1a76bfdac795c9d84101
   eb317
   ```

2. Exécutez un script qui analyse le fichier de configuration et crée un nouveau catalogue à partir de ses références:

   Exemple de script

```
name=$(yq eval '.name' catalog.yaml)
mkdir "$name"
yq eval '.name + "/" + .references[].name' catalog.yaml | xargs mkdir
for l in $(yq e '.name as $catalog | .references[] | .image + "|" + $catalog + "/" + .name +
"/index.yaml"' catalog.yaml); do
  image=$(echo $l | cut -d'|' -f1)
  file=$(echo $l | cut -d'|' -f2)
  opm render "$image" > "$file"
done
opm generate dockerfile "$name"
indexImage=$(yq eval '.repo + ":" + .tag' catalog.yaml)
docker build -t "$indexImage" -f "$name.Dockerfile" .
docker push "$indexImage"
```

2.2.2.5. Lignes directrices

Considérez les directives suivantes lors de la maintenance des catalogues basés sur des fichiers.

2.2.2.5.1. Des paquets immuables

L'avis général de Operator Lifecycle Manager (OLM) est que les images groupées et leurs métadonnées doivent être traitées comme immuables.

Lorsqu'un paquet cassé a été poussé à un catalogue, vous devez supposer qu'au moins un de vos utilisateurs a été mis à niveau vers ce paquet. Basé sur cette hypothèse, vous devez libérer un autre paquet avec un chemin de mise à niveau à partir du paquet cassé pour s'assurer que les utilisateurs avec le paquet cassé installé reçoivent une mise à niveau. L'OLM ne réinstallera pas un paquet installé si le contenu de ce paquet est mis à jour dans le catalogue.

Cependant, il y a certains cas où un changement dans les métadonnées du catalogue est préféré:

- Canal promotion: Si vous avez déjà publié un paquet et décidez plus tard que vous souhaitez l'ajouter à un autre canal, vous pouvez ajouter une entrée pour votre paquet dans un autre blob olm.channel.

- De nouveaux chemins de mise à niveau: Si vous publiez une nouvelle version de paquet 1.2.z, par exemple 1.2.4, mais 1.3.0 est déjà publié, vous pouvez mettre à jour les métadonnées du catalogue pour 1.3.0 pour sauter 1.2.4.

2.2.2.5.2. Contrôle des sources

Les métadonnées du catalogue doivent être stockées dans le contrôle de la source et traitées comme source de vérité. Les mises à jour des images de catalogue devraient inclure les étapes suivantes:

1. Actualisez le répertoire du catalogue contrôlé par la source avec un nouveau commit.

2. Créez et poussez l'image du catalogue. Employez une taxonomie de marquage cohérente, telle que :dernière ou :<target_cluster_version>, afin que les utilisateurs puissent recevoir des mises à jour d'un catalogue au fur et à mesure qu'ils deviennent disponibles.

2.2.2.6. L'utilisation de CLI

En ce qui concerne les instructions concernant la création de catalogues basés sur des fichiers à l'aide de l'opm CLI, voir Gérer les catalogues personnalisés. En ce qui concerne la documentation de référence sur les commandes opm CLI liées à la gestion des catalogues basés sur des fichiers, consultez les outils CLI.

2.2.2.7. Automatisation

Les auteurs d'opérateurs et les responsables du catalogue sont encouragés à automatiser la maintenance de leur catalogue avec des flux de travail CI/CD. Les mainteneurs de catalogues peuvent encore s'améliorer à ce sujet en construisant l'automatisation GitOps pour accomplir les tâches suivantes:

- Les auteurs de la demande de tirage (PR) sont autorisés à apporter les modifications demandées, par exemple en mettant à jour la référence d'image de leur paquet.
- Assurez-vous que les mises à jour du catalogue passent la commande opm valid.
- Assurez-vous que les références mises à jour du paquet ou de l'image du catalogue existent, que les images du catalogue s'exécutent avec succès dans un cluster, et que les opérateurs de ce paquet peuvent être installés avec succès.
- Fusionner automatiquement les PR qui passent les vérifications précédentes.
- Automatiquement reconstruire et republier l'image du catalogue.

2.3. GLOSSAIRE DU CADRE OPÉRATEUR DES TERMES COMMUNS

Cette rubrique fournit un glossaire des termes communs liés au Cadre d'exploitation, y compris le gestionnaire du cycle de vie de l'opérateur (OLM) et le SDK de l'opérateur.

2.3.1. Le paquet

Dans le format de paquet, un bundle est une collection d'opérateurs CSV, manifestes et métadonnées. Ensemble, ils forment une version unique d'un opérateur qui peut être installé sur le cluster.

2.3.2. Image de paquet

Dans le format de paquet, une image de paquet est une image de conteneur qui est construite à partir des manifestes de l'opérateur et qui contient un paquet. Les images groupées sont stockées et distribuées par les registres de conteneurs spec Open Container Initiative (OCI), tels que Quay.io ou DockerHub.

2.3.3. Catalogue source

La source d'un catalogue représente une réserve de métadonnées que OLM peut interroger pour découvrir et installer des opérateurs et leurs dépendances.

2.3.4. Canal

Le canal définit un flux de mises à jour pour un opérateur et est utilisé pour déployer des mises à jour pour les abonnés. La tête pointe vers la dernière version de ce canal. À titre d'exemple, un canal stable aurait toutes les versions stables d'un opérateur disposées du plus tôt au dernier.

L'opérateur peut avoir plusieurs canaux, et un abonnement lié à un certain canal ne chercherait que des mises à jour dans ce canal.

2.3.5. Tête de canal

La tête de canal fait référence à la dernière mise à jour connue d'un canal particulier.

2.3.6. Cluster de service version

La version de service de cluster (CSV) est un manifeste YAML créé à partir des métadonnées de l'opérateur qui aide OLM à exécuter l'opérateur dans un cluster. Ce sont les métadonnées qui accompagnent une image de conteneur d'opérateur, utilisée pour peupler les interfaces utilisateur avec des informations telles que son logo, sa description et sa version.

C'est aussi une source d'informations techniques qui est nécessaire pour exécuter l'opérateur, comme les règles RBAC qu'il exige et de quelles ressources personnalisées (CR) il gère ou dépend.

2.3.7. Dépendance

L'opérateur peut dépendre de la présence d'un autre opérateur dans le cluster. À titre d'exemple, l'opérateur Vault dépend de l'opérateur etcd pour sa couche de persistance des données.

L'OLM résout les dépendances en veillant à ce que toutes les versions spécifiées des opérateurs et des CRD soient installées sur le cluster pendant la phase d'installation. Cette dépendance est résolue par la recherche et l'installation d'un opérateur dans un catalogue qui satisfait à l'API CRD requise, et n'est pas lié à des paquets ou des paquets.

2.3.8. Extension

Les extensions permettent aux administrateurs de clusters d'étendre les capacités des utilisateurs sur leur cluster dédié OpenShift. Les extensions sont gérées par Operator Lifecycle Manager (OLM) v1.

L'API ClusterExtension simplifie la gestion des extensions installées, qui inclut des opérateurs via le format de paquet Register+v1, en consolidant les API face à l'utilisateur en un seul objet. Les administrateurs et les SRE peuvent utiliser l'API pour automatiser les processus et définir les états souhaités en utilisant les principes GitOps.

2.3.9. Image de l'index

Dans le format de paquet, une image d'index se réfère à une image d'une base de données (un instantané de base de données) qui contient des informations sur les paquets Opérateur, y compris les CSV et les CRD de toutes les versions. Cet index peut héberger un historique d'opérateurs sur un cluster et être maintenu en ajoutant ou en supprimant des opérateurs à l'aide de l'outil Opm CLI.

2.3.10. Installer le plan

Le plan d'installation est une liste calculée de ressources à créer pour installer ou mettre à niveau automatiquement un CSV.

2.3.11. La multitenance

Le locataire dans OpenShift Dedicated est un utilisateur ou un groupe d'utilisateurs qui partagent un accès et des privilèges communs pour un ensemble de charges de travail déployées, généralement représentées par un espace de noms ou un projet. Il est possible d'utiliser les locataires pour assurer un

niveau d'isolement entre différents groupes ou équipes.

Lorsqu'un cluster est partagé par plusieurs utilisateurs ou groupes, il est considéré comme un cluster multilocataires.

2.3.12. Exploitant

Les opérateurs constituent une méthode de mise en package, de déploiement et de gestion d'une application Kubernetes. L'application Kubernetes est une application qui est déployée sur Kubernetes et gérée à l'aide des API Kubernetes et des outils kubectl ou oc.

Dans Operator Lifecycle Manager (OLM) v1, l'API ClusterExtension simplifie la gestion des extensions installées, qui inclut les opérateurs via le format de paquet Register+v1.

2.3.13. Groupe d'opérateurs

Le groupe d'opérateurs configure tous les opérateurs déployés dans le même espace de noms que l'objet OperatorGroup pour surveiller leur CR dans une liste d'espaces de noms ou de clusters.

2.3.14. Forfait

Dans le format de paquet, un paquet est un répertoire qui contient tout l'historique publié d'un opérateur avec chaque version. La version publiée d'un opérateur est décrite dans un manifeste CSV aux côtés des CRD.

2.3.15. Registry

Le registre est une base de données qui stocke des images groupées d'opérateurs, chacune avec toutes ses versions les plus récentes et historiques dans tous les canaux.

2.3.16. Abonnement

L'abonnement maintient les CSV à jour en suivant un canal dans un paquet.

2.3.17. Graphique de mise à jour

Le graphique de mise à jour relie les versions de CSV ensemble, comme le graphique de mise à jour de tout autre logiciel emballé. Les opérateurs peuvent être installés de manière séquentielle, ou certaines versions peuvent être ignorées. Le graphique de mise à jour devrait croître uniquement en tête avec de nouvelles versions ajoutées.

Également connu sous le nom de bords de mise à jour ou chemins de mise à jour.

2.4. GESTIONNAIRE DU CYCLE DE VIE DE L'OPÉRATEUR (OLM)

2.4.1. Concepts et ressources du gestionnaire du cycle de vie de l'opérateur

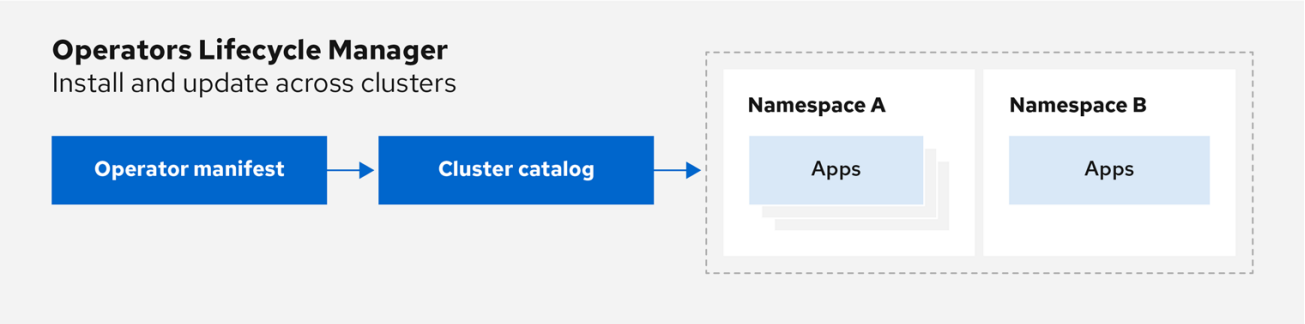
Ce guide donne un aperçu des concepts qui pilotent le gestionnaire de cycle de vie de l'opérateur (OLM) dans OpenShift Dedicated.

2.4.1.1. En quoi consiste le gestionnaire de cycle de vie de l'opérateur (OLM) Classic?

Opérateur Lifecycle Manager (OLM) Classic aide les utilisateurs à installer, mettre à jour et gérer le cycle

de vie des applications natives Kubernetes (Operators) et de leurs services associés fonctionnant sur leurs clusters dédiés OpenShift. Il s'inscrit dans le cadre de l'opérateur, une boîte à outils open source conçue pour gérer les opérateurs de manière efficace, automatisée et évolutive.

Figure 2.2. Flux de travail OLM (Classic)



L'OLM fonctionne par défaut dans OpenShift Dedicated 4, qui aide les administrateurs à jouer le rôle d'administrateur dédié dans l'installation, la mise à niveau et l'octroi d'un accès aux opérateurs fonctionnant sur leur cluster. La console Web dédiée OpenShift fournit des écrans de gestion aux administrateurs dédiés pour installer des opérateurs, ainsi que l'accès à des projets spécifiques pour utiliser le catalogue des opérateurs disponibles sur le cluster.

En ce qui concerne les développeurs, une expérience en libre-service permet de fournir et de configurer des instances de bases de données, de surveillance et de services de big data sans avoir à être des experts en la matière, car l'opérateur dispose de ces connaissances.

2.4.1.2. Les ressources OLM

Les définitions de ressources personnalisées (DCR) suivantes sont définies et gérées par Operator Lifecycle Manager (OLM):

Tableau 2.2. CRDS géré par OLM et les opérateurs de catalogue

A) Ressources	Court nom	Description
ClusterService Version (CSV)	**CSV**	Les métadonnées de l'application. Exemple : nom, version, icône, ressources requises.
CatalogueSource	**catsrc**	Dépôt de CSV, CRD et packages qui définissent une application.
Abonnement	**a) Sous**	Garde les CSVs à jour en suivant un canal dans un paquet.
InstallPlan	**IP**	Liste calculée des ressources à créer pour installer ou mettre à jour automatiquement un CSV.
Groupe d'opérateurs	**à propos de OG**	Configure tous les opérateurs déployés dans le même espace de noms que l'objet OperatorGroup pour surveiller leur ressource personnalisée (CR) dans une liste d'espaces de noms ou à l'échelle du cluster.

A) Ressources	Court nom	Description
Conditions de l'opérateur	-	Crée un canal de communication entre OLM et un opérateur qu'il gère. Les opérateurs peuvent écrire dans le tableau Status.Conditions pour communiquer des états complexes à OLM.

2.4.1.2.1. Cluster de service version

La version de service cluster (CSV) représente une version spécifique d'un opérateur en cours d'exécution sur un cluster dédié OpenShift. Il s'agit d'un manifeste YAML créé à partir des métadonnées de l'opérateur qui aide le gestionnaire de cycle de vie de l'opérateur (OLM) à exécuter l'opérateur dans le cluster.

L'OLM exige que ces métadonnées concernant un opérateur puissent être maintenues en toute sécurité sur un cluster et pour fournir des informations sur la façon dont les mises à jour devraient être appliquées au fur et à mesure que de nouvelles versions de l'opérateur sont publiées. Ceci est similaire au logiciel d'emballage pour un système d'exploitation traditionnel; pensez à l'étape d'emballage pour OLM comme le stade auquel vous faites votre rpm, deb ou apk bundle.

Le CSV comprend les métadonnées qui accompagnent une image de conteneur d'opérateur, utilisée pour remplir les interfaces utilisateur avec des informations telles que son nom, sa version, sa description, ses étiquettes, son lien de dépôt et son logo.

Le CSV est également une source d'informations techniques nécessaires à l'exécution de l'opérateur, telles que les ressources personnalisées (CR) dont il gère ou dépend, les règles RBAC, les exigences en cluster et les stratégies d'installation. Ces informations indiquent à OLM comment créer les ressources requises et configurer l'opérateur en tant que déploiement.

2.4.1.2.2. Catalogue source

La source du catalogue représente un magasin de métadonnées, généralement en faisant référence à une image d'index stockée dans un registre de conteneurs. Le gestionnaire de cycle de vie de l'opérateur (OLM) interroge les sources de catalogue pour découvrir et installer les opérateurs et leurs dépendances. OperatorHub dans la console Web dédiée OpenShift affiche également les opérateurs fournis par les sources du catalogue.

ASTUCE

Les administrateurs de clusters peuvent afficher la liste complète des opérateurs fournis par une source de catalogue activée sur un cluster à l'aide de la page Administration → Paramètres du cluster → Configuration → OperatorHub page dans la console Web.

La spécification d'un objet CatalogSource indique comment construire un pod ou comment communiquer avec un service qui sert l'API GRPC du Registre de l'opérateur.

Exemple 2.9. Exemple d'objet CatalogSource

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog (1)
  namespace: openshift-marketplace (2)
```

```
  annotations:
    olm.catalogImageTemplate: 3
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}.
{kube_patch_version}"
spec:
  displayName: Example Catalog 4
  image: quay.io/example-org/example-catalog:v1 5
  priority: -400 6
  publisher: Example Org
  sourceType: grpc 7
  grpcPodConfig:
    securityContextConfig: <security_mode> 8
    nodeSelector: 9
      custom_label: <label>
    priorityClassName: system-cluster-critical 10
    tolerations: 11
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
  updateStrategy:
    registryPoll: 12
      interval: 30m0s
status:
  connectionState:
    address: example-catalog.openshift-marketplace.svc:50051
    lastConnect: 2021-08-26T18:14:31Z
    lastObservedState: READY 13
  latestImageRegistryPoll: 2021-08-26T18:46:25Z 14
  registryService: 15
    createdAt: 2021-08-26T16:16:37Z
    port: 50051
    protocol: grpc
    serviceName: example-catalog
    serviceNamespace: openshift-marketplace
```

1. Le nom de l'objet CatalogSource. Cette valeur est également utilisée dans le nom du pod associé qui est créé dans l'espace de noms demandé.

2. Espace de noms pour créer le catalogue. Afin de rendre le catalogue disponible à l'échelle du cluster dans tous les espaces de noms, définissez cette valeur sur openshift-marketplace. Les sources de catalogue Red Hat par défaut utilisent également l'espace de noms openshift-marketplace. Dans le cas contraire, définissez la valeur d'un espace de noms spécifique pour rendre l'opérateur disponible uniquement dans cet espace de noms.

3. Facultatif: Pour éviter les mises à niveau de cluster potentiellement laissant les installations de l'opérateur dans un état non pris en charge ou sans chemin de mise à jour continu, vous pouvez activer la modification automatique de la version d'image d'index de votre catalogue d'opérateur dans le cadre des mises à niveau de cluster.

   Définissez l'annotation olm.catalogImageTemplate sur votre nom d'image d'index et utilisez une ou plusieurs des variables de version du cluster Kubernetes comme indiqué lors de la construction du modèle pour la balise d'image. L'annotation écrase le champ spec.image au

moment de l'exécution. Consultez la section « Modèle d'image pour les sources de catalogue personnalisées » pour plus de détails.

4 Afficher le nom du catalogue dans la console Web et CLI.

5 Index de l'image pour le catalogue. En option, peut être omis lors de l'utilisation de l'annotation olm.catalogImageTemplate, qui définit la spécification de traction au moment de l'exécution.

6 Le poids pour la source du catalogue. L'OMM utilise le poids pour la priorisation lors de la résolution de dépendance. Le poids plus élevé indique que le catalogue est préféré aux catalogues moins pondérés.

7 Les types de sources comprennent les éléments suivants:

- GRPC avec une référence d'image: OLM tire l'image et exécute le pod, qui est censé servir une API conforme.
- GRPC avec un champ d'adresse: OLM tente de contacter l'API gRPC à l'adresse donnée. Cela ne devrait pas être utilisé dans la plupart des cas.
- ConfigMap: OLM analyse la configuration des données cartographiques et exécute un pod qui peut servir l'API gRPC.

8 Indiquez la valeur de l'héritage ou de la restriction. Lorsque le champ n'est pas défini, la valeur par défaut est héritée. Dans une version ultérieure d'OpenShift Dedicated, il est prévu que la valeur par défaut soit limitée. Dans le cas où votre catalogue ne peut pas fonctionner avec des autorisations restreintes, il est recommandé de définir manuellement ce champ sur l'héritage.

9 Facultatif: Pour les sources de catalogue de type grpc, remplace le sélecteur de nœud par défaut pour le pod servant le contenu dans spec.image, si défini.

10 Facultatif: Pour les sources de catalogue de type grpc, remplace le nom de classe de priorité par défaut pour le pod servant le contenu dans spec.image, si défini. Kubernetes fournit par défaut des classes de priorité critiques et critiques du cluster système. Définir le champ à vide ("") attribue à la pod la priorité par défaut. D'autres classes prioritaires peuvent être définies manuellement.

11 Facultatif: Pour les sources de catalogue de type grpc, outrepasse les tolérances par défaut pour le pod servant le contenu dans spec.image, si défini.

12 Vérifiez automatiquement les nouvelles versions à un intervalle donné pour rester à jour.

13 Dernier état observé de la connexion au catalogue. À titre d'exemple:

- LIRE: Une connexion est établie avec succès.
- CONNECTING : Une connexion tente d'établir.
- FAILURE TRANSIENTE : Un problème temporaire s'est produit lors de la tentative d'établir une connexion, comme un délai d'attente. L'état finira par revenir à CONNECTING et essayer à nouveau.

Consultez les états de connectivité dans la documentation du GRPC pour plus de détails.

14 La dernière fois que le registre conteneur stockant l'image du catalogue a été sondé pour s'assurer que l'image est à jour.

15 Informations d'état pour le service de registre de l'opérateur du catalogue.

Faire référence au nom d'un objet CatalogSource dans un abonnement indique à OLM où chercher pour trouver un opérateur demandé:

Exemple 2.10. Exemple Objet d'abonnement faisant référence à une source de catalogue

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

Ressources supplémentaires

- Comprendre l'opérateurHub
- Catalogues d'opérateurs Red Hat
- Ajout d'une source de catalogue à un cluster
- La priorité du catalogue
- Affichage de l'état de la source du catalogue de l'opérateur en utilisant le CLI
- Catalogue source de calendrier des pod

2.4.1.2.2.1. Modèle d'image pour les sources de catalogue personnalisées

La compatibilité de l'opérateur avec le cluster sous-jacent peut être exprimée par une source de catalogue de différentes manières. L'une des façons, qui est utilisée pour les sources de catalogue par défaut de Red Hat, est d'identifier les balises d'image pour les images d'index qui sont spécifiquement créées pour une version de la plate-forme particulière, par exemple OpenShift Dedicated 4.

Lors d'une mise à niveau de cluster, la balise d'image d'index pour les sources de catalogue par défaut Red Hat est mise à jour automatiquement par l'opérateur de versions de cluster (CVO) de sorte que le gestionnaire de cycle de vie de l'opérateur (OLM) tire la version mise à jour du catalogue. Lors d'une mise à jour de OpenShift Dedicated 4.17 à 4, le champ spec.image de l'objet CatalogSource pour le catalogue redhat-operators est mis à jour à partir de:

```
registry.redhat.io/redhat/redhat-operator-index:v4.18
```

à:

```
registry.redhat.io/redhat/redhat-operator-index:v4.18
```

Cependant, le CVO ne met pas automatiquement à jour les balises d'image pour les catalogues personnalisés. Afin de s'assurer que les utilisateurs disposent d'une installation d'opérateur compatible et prise en charge après une mise à niveau de cluster, les catalogues personnalisés doivent également être mis à jour pour faire référence à une image d'index mise à jour.

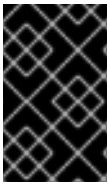
À partir de OpenShift Dedicated 4.9, les administrateurs de clusters peuvent ajouter l'annotation olm.catalogImageTemplate dans l'objet CatalogSource pour les catalogues personnalisés à une référence d'image qui inclut un modèle. Les variables de version Kubernetes suivantes sont prises en charge pour être utilisées dans le modèle:

- **kube_major_version**
- **kube_minor_version**
- **kube_patch_version**

NOTE

Il faut spécifier la version du cluster Kubernetes et non une version de cluster dédié OpenShift, car ce dernier n'est pas actuellement disponible pour templating.

À condition que vous ayez créé et poussé une image d'index avec une balise spécifiant la version mise à jour de Kubernetes, le réglage de cette annotation permet de modifier automatiquement les versions d'image d'index dans les catalogues personnalisés après une mise à niveau de cluster. La valeur d'annotation est utilisée pour définir ou mettre à jour la référence d'image dans le champ spec.image de l'objet CatalogSource. Cela permet d'éviter les mises à niveau de clusters laissant les installations de l'opérateur dans des états non pris en charge ou sans chemin de mise à jour continue.

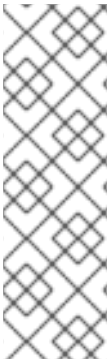


IMPORTANT

Assurez-vous que l'image d'index avec la balise mise à jour, quel que soit le registre dans lequel elle est stockée, est accessible par le cluster au moment de la mise à jour du cluster.

Exemple 2.11. Exemple de source de catalogue avec un modèle d'image

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  generation: 1
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    olm.catalogImageTemplate:
      "quay.io/example-org/example-catalog:v{kube_major_version}.{kube_minor_version}"
spec:
  displayName: Example Catalog
  image: quay.io/example-org/example-catalog:v1.31
  priority: -400
  publisher: Example Org
```



NOTE

Lorsque le champ spec.image et l'annotation olm.catalogImageTemplate sont toutes deux définies, le champ spec.image est écrasé par la valeur résolue de l'annotation. Lorsque l'annotation ne se résout pas à une spécification de traction utilisable, la source du catalogue revient à la valeur spec.image définie.

Lorsque le champ spec.image n'est pas défini et que l'annotation ne se résout pas à une spécification de traction utilisable, OLM arrête la réconciliation de la source du catalogue et le place dans une condition d'erreur lisible par l'homme.

Dans un cluster OpenShift Dedicated 4, qui utilise Kubernetes 1.31, l'annotation olm.catalogImageTemplate dans l'exemple précédent résout la référence d'image suivante:

```
quay.io/example-org/example-catalog:v1.31
```

Dans le cas des futures versions d'OpenShift Dedicated, vous pouvez créer des images d'index mises à jour pour vos catalogues personnalisés qui ciblent la version ultérieure de Kubernetes utilisée par la version ultérieure OpenShift Dedicated. Avec l'annotation olm.catalogImageTemplate définie avant la mise à niveau, la mise à niveau du cluster vers la version ultérieure OpenShift Dedicated mettrait alors automatiquement à jour l'image d'index du catalogue.

2.4.1.2.2.2. Exigences en matière de santé du catalogue

Les catalogues d'opérateurs sur un cluster sont interchangeables du point de vue de la résolution d'installation; un objet d'abonnement peut faire référence à un catalogue spécifique, mais les dépendances sont résolues à l'aide de tous les catalogues du cluster.

À titre d'exemple, si Catalog A est malsain, un référencement d'abonnement Catalog A pourrait résoudre une dépendance dans Catalog B, que l'administrateur du cluster n'aurait peut-être pas attendu, car B avait normalement une priorité de catalogue inférieure à A.

En conséquence, OLM exige que tous les catalogues avec un espace de noms global donné (par exemple, l'espace de noms openshift-marketplace par défaut ou un espace de noms global personnalisé) soient sains. Lorsqu'un catalogue est malsain, toutes les opérations d'installation ou de mise à jour de l'opérateur au sein de son espace de noms global partagé échoueront avec une condition CatalogSourcesUnsanté. Lorsque ces opérations étaient autorisées dans un état malsain, OLM pourrait prendre des décisions de résolution et d'installation qui étaient inattendues pour l'administrateur du cluster.

En tant qu'administrateur de cluster, si vous observez un catalogue malsain et que vous souhaitez considérer le catalogue comme invalide et reprendre les installations de l'opérateur, consultez les sections "Supprimer les catalogues personnalisés" ou "Désactiver les sources de catalogue OperatorHub par défaut" pour obtenir des informations sur la suppression du catalogue malsain.

2.4.1.2.3. Abonnement

L'abonnement, défini par un objet Abonnement, représente une intention d'installer un Opérateur. C'est la ressource personnalisée qui relie un opérateur à une source de catalogue.

Les abonnements décrivent le canal d'un package Opérateur auquel s'abonner et s'il y a lieu d'effectuer des mises à jour automatiquement ou manuellement. En cas de configuration automatique, l'abonnement garantit que le gestionnaire de cycle de vie de l'opérateur (OLM) gère et met à niveau l'opérateur pour s'assurer que la dernière version est toujours en cours d'exécution dans le cluster.

Exemple d'objet d'abonnement

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-namespace
spec:
  channel: stable
  name: example-operator
  source: example-catalog
  sourceNamespace: openshift-marketplace
```

Cet objet d'abonnement définit le nom et l'espace de noms de l'opérateur, ainsi que le catalogue à partir duquel les données de l'opérateur peuvent être trouvées. Le canal, tel que alpha, bêta ou stable, aide à déterminer quel flux d'opérateur doit être installé à partir de la source du catalogue.

Les noms des canaux dans un abonnement peuvent différer entre les opérateurs, mais le schéma de dénomination doit suivre une convention commune au sein d'un opérateur donné. À titre d'exemple, les noms de canaux peuvent suivre un flux de mise à jour de version mineure pour l'application fournie par l'opérateur (1.2, 1.3) ou une fréquence de libération (stable, rapide).

En plus d'être facilement visible à partir de la console Web dédiée OpenShift, il est possible d'identifier quand il y a une version plus récente d'un opérateur disponible en inspectant l'état de l'abonnement connexe. La valeur associée au champ CurrentCSV est la version la plus récente connue de OLM, et installéeCSV est la version installée sur le cluster.

Ressources supplémentaires

- Affichage du statut d'abonnement de l'opérateur en utilisant le CLI

2.4.1.2.4. Installer le plan

Le plan d'installation, défini par un objet InstallPlan, décrit un ensemble de ressources créées par Operator Lifecycle Manager (OLM) pour installer ou mettre à niveau une version spécifique d'un opérateur. La version est définie par une version de service cluster (CSV).

L'installation d'un opérateur, d'un administrateur de cluster ou d'un utilisateur qui a obtenu les autorisations d'installation de l'opérateur doit d'abord créer un objet d'abonnement. L'abonnement représente l'intention de s'abonner à un flux de versions disponibles d'un opérateur à partir d'une source de catalogue. L'abonnement crée ensuite un objet InstallPlan pour faciliter l'installation des ressources pour l'opérateur.

Le plan d'installation doit ensuite être approuvé selon l'une des stratégies d'approbation suivantes:

- Lorsque le champ spec.installPlanApproval de l'abonnement est défini sur Automatique, le plan d'installation est approuvé automatiquement.
- Lorsque le champ spec.installPlanApproval de l'abonnement est défini sur Manuel, le plan d'installation doit être approuvé manuellement par un administrateur de cluster ou un utilisateur avec les autorisations appropriées.

Après l'approbation du plan d'installation, OLM crée les ressources spécifiées et installe l'opérateur dans l'espace de noms spécifié par l'abonnement.

Exemple 2.12. Exemple d'objet InstallPlan

```
apiVersion: operators.coreos.com/v1alpha1
kind: InstallPlan
metadata:
  name: install-abcde
  namespace: operators
spec:
  approval: Automatic
  approved: true
  clusterServiceVersionNames:
    - my-operator.v1.0.1
  generation: 1
status:
  ...
  catalogSources: []
  conditions:
    - lastTransitionTime: '2021-01-01T20:17:27Z'
      lastUpdateTime: '2021-01-01T20:17:27Z'
      status: 'True'
      type: Installed
  phase: Complete
  plan:
    - resolving: my-operator.v1.0.1
      resource:
        group: operators.coreos.com
        kind: ClusterServiceVersion
        manifest: >-
        ...
        name: my-operator.v1.0.1
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1alpha1
      status: Created
    - resolving: my-operator.v1.0.1
      resource:
        group: apiextensions.k8s.io
        kind: CustomResourceDefinition
        manifest: >-
        ...
        name: webservers.web.servers.org
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1beta1
      status: Created
    - resolving: my-operator.v1.0.1
      resource:
        group: ''
        kind: ServiceAccount
        manifest: >-
        ...
        name: my-operator
        sourceName: redhat-operators
        sourceNamespace: openshift-marketplace
        version: v1
      status: Created
```

```
  - resolving: my-operator.v1.0.1
    resource:
      group: rbac.authorization.k8s.io
      kind: Role
      manifest: >-
      ...
      name: my-operator.v1.0.1-my-operator-6d7cbc6f57
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1
    status: Created
  - resolving: my-operator.v1.0.1
    resource:
      group: rbac.authorization.k8s.io
      kind: RoleBinding
      manifest: >-
      ...
      name: my-operator.v1.0.1-my-operator-6d7cbc6f57
      sourceName: redhat-operators
      sourceNamespace: openshift-marketplace
      version: v1
    status: Created
    ...
```

2.4.1.2.5. Groupes d'opérateurs

Le groupe d'opérateurs, défini par la ressource OperatorGroup, fournit une configuration multilocataires aux Opérateurs installés par OLM. Le groupe d'opérateur sélectionne les espaces de noms cibles dans lesquels générer l'accès RBAC requis pour ses opérateurs membres.

L'ensemble des espaces de noms cibles est fourni par une chaîne délimitée par virgule stockée dans l'annotation olm.targetNamespaces d'une version de service cluster (CSV). Cette annotation est appliquée aux instances CSV des opérateurs membres et est projetée dans leurs déploiements.

Ressources supplémentaires

- Groupes d'opérateurs

2.4.1.2.6. Conditions de l'opérateur

Dans le cadre de son rôle dans la gestion du cycle de vie d'un opérateur, Operator Lifecycle Manager (OLM) infère l'état d'un opérateur à partir de l'état des ressources Kubernetes qui définissent l'opérateur. Bien que cette approche fournisse un certain niveau d'assurance qu'un exploitant est dans un état donné, il existe de nombreux cas où un opérateur pourrait avoir besoin de communiquer des informations à OLM qui ne pourraient pas être déduits autrement. Ces informations peuvent ensuite être utilisées par OLM pour mieux gérer le cycle de vie de l'opérateur.

L'ODM fournit une définition de ressource personnalisée (CRD) appelée OperatorCondition qui permet aux opérateurs de communiquer les conditions à OLM. Il existe un ensemble de conditions supportées qui influencent la gestion de l'opérateur par OLM lorsqu'elles sont présentes dans la gamme Spec.Conditions d'une ressource OperatorCondition.

NOTE

Le tableau Spec.Conditions n'est pas présent dans un objet OperatorCondition tant qu'il n'est pas ajouté par un utilisateur ou à la suite de la logique personnalisée de l'opérateur.

Ressources supplémentaires

- Conditions de l'opérateur

2.4.2. Architecture du gestionnaire de cycle de vie de l'opérateur

Ce guide décrit l'architecture des composants du gestionnaire de cycle de vie de l'opérateur (OLM) dans OpenShift Dedicated.

2.4.2.1. Les responsabilités des composantes

Le gestionnaire du cycle de vie de l'opérateur (OLM) est composé de deux opérateurs : l'opérateur OLM et l'opérateur de catalogue.

Les OLM et les opérateurs de catalogue sont responsables de la gestion des définitions de ressources personnalisées (DRC) qui constituent la base du cadre OLM:

Tableau 2.3. CRDS géré par OLM et les opérateurs de catalogue

A) Ressources	Court nom	Le propriétaire	Description
ClusterService Version (CSV)	**CSV**	LES OLM	Les métadonnées de l'application: nom, version, icône, ressources requises, installation, etc.
InstallPlan	**IP**	Catalogue	Liste calculée des ressources à créer pour installer ou mettre à jour automatiquement un CSV.
CatalogueSource	**catsrc**	Catalogue	Dépôt de CSV, CRD et packages qui définissent une application.
Abonnement	**a) Souss**	Catalogue	Il est utilisé pour garder les CSV à jour en suivant un canal dans un paquet.
Groupe d'opérateurs	**à propos de OG**	LES OLM	Configure tous les opérateurs déployés dans le même espace de noms que l'objet OperatorGroup pour surveiller leur ressource personnalisée (CR) dans une liste d'espaces de noms ou à l'échelle du cluster.

Chacun de ces opérateurs est également responsable de la création des ressources suivantes:

Tableau 2.4. Les ressources créées par OLM et Catalog Operators

A) Ressources	Le propriétaire
Déploiements	LES OLM
Comptes de service	
(Cluster)Roles	
(Cluster)RoleBindings	
CustomResourceDefinitions (CRD)	Catalogue
ClusterServiceVersions	

2.4.2.2. Opérateur OLM

L'opérateur OLM est responsable du déploiement des applications définies par les ressources CSV une fois que les ressources requises spécifiées dans le CSV sont présentes dans le cluster.

L'opérateur OLM n'est pas concerné par la création des ressources requises; vous pouvez choisir de créer manuellement ces ressources à l'aide du CLI ou à l'aide de l'opérateur de catalogue. Cette séparation des préoccupations permet aux utilisateurs d'adhérer progressivement en fonction de la quantité du cadre OLM qu'ils choisissent d'utiliser pour leur application.

L'opérateur OLM utilise le flux de travail suivant:

1. Consultez les versions de service cluster (CSV) dans un espace de noms et vérifiez que les exigences sont satisfaites.

2. Lorsque les exigences sont remplies, exécutez la stratégie d'installation pour le CSV.

NOTE

Le CSV doit être un membre actif d'un groupe d'opérateurs pour que la stratégie d'installation s'exécute.

2.4.2.3. Opérateur de catalogue

L'opérateur de catalogue est responsable de la résolution et de l'installation des versions de services de cluster (CSV) et des ressources requises qu'ils spécifient. Il est également responsable de regarder les sources de catalogue pour les mises à jour des paquets dans les canaux et de les mettre à niveau, automatiquement si désiré, vers les dernières versions disponibles.

Afin de suivre un paquet dans un canal, vous pouvez créer un objet d'abonnement configurant le paquet, le canal et l'objet CatalogSource que vous souhaitez utiliser pour tirer les mises à jour. Lorsque des mises à jour sont trouvées, un objet InstallPlan approprié est écrit dans l'espace de noms pour le compte de l'utilisateur.

L'opérateur de catalogue utilise le flux de travail suivant:

1. Connectez-vous à chaque source de catalogue dans le cluster.

2. Attention aux plans d'installation non résolus créés par un utilisateur, et s'il est trouvé:

    a. Cherchez le CSV correspondant au nom demandé et ajoutez le CSV en tant que ressource résolue.

    b. Ajoutez le CRD en tant que ressource résolue pour chaque CRD géré ou requis.

    c. Dans chaque CRD requis, trouvez le CSV qui le gère.

3. Consultez les plans d'installation résolus et créez toutes les ressources découvertes pour elle, si elles sont approuvées par un utilisateur ou automatiquement.

4. Consultez les sources de catalogue et les abonnements et créez des plans d'installation basés sur elles.

2.4.2.4. Registre du catalogue

Le registre de catalogue stocke les CSV et les CRD pour la création dans un cluster et stocke les métadonnées sur les paquets et les canaux.

Le manifeste de paquet est une entrée dans le Registre de catalogue qui associe une identité de paquet à des ensembles de CSV. Dans un paquet, les canaux pointent vers un CSV particulier. Comme les CSV se réfèrent explicitement au CSV qu'ils remplacent, un manifeste de paquets fournit à l'opérateur de catalogue toutes les informations nécessaires pour mettre à jour un CSV vers la dernière version d'un canal, passant par chaque version intermédiaire.

2.4.3. Flux de travail du gestionnaire de cycle de vie de l'opérateur

Ce guide décrit le flux de travail du gestionnaire de cycle de vie de l'opérateur (OLM) dans OpenShift Dedicated.

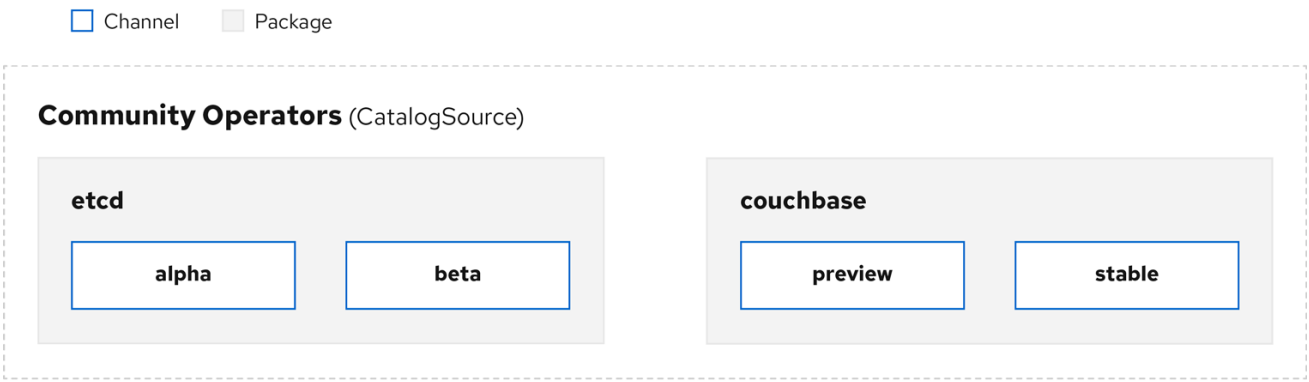
2.4.3.1. Installation de l'opérateur et mise à niveau du flux de travail dans OLM

Dans l'écosystème Gestionnaire du cycle de vie de l'opérateur (GLO), les ressources suivantes sont utilisées pour résoudre les installations et les mises à niveau de l'opérateur:

- ClusterServiceVersion (CSV)
- **CatalogueSource**
- **Abonnement**

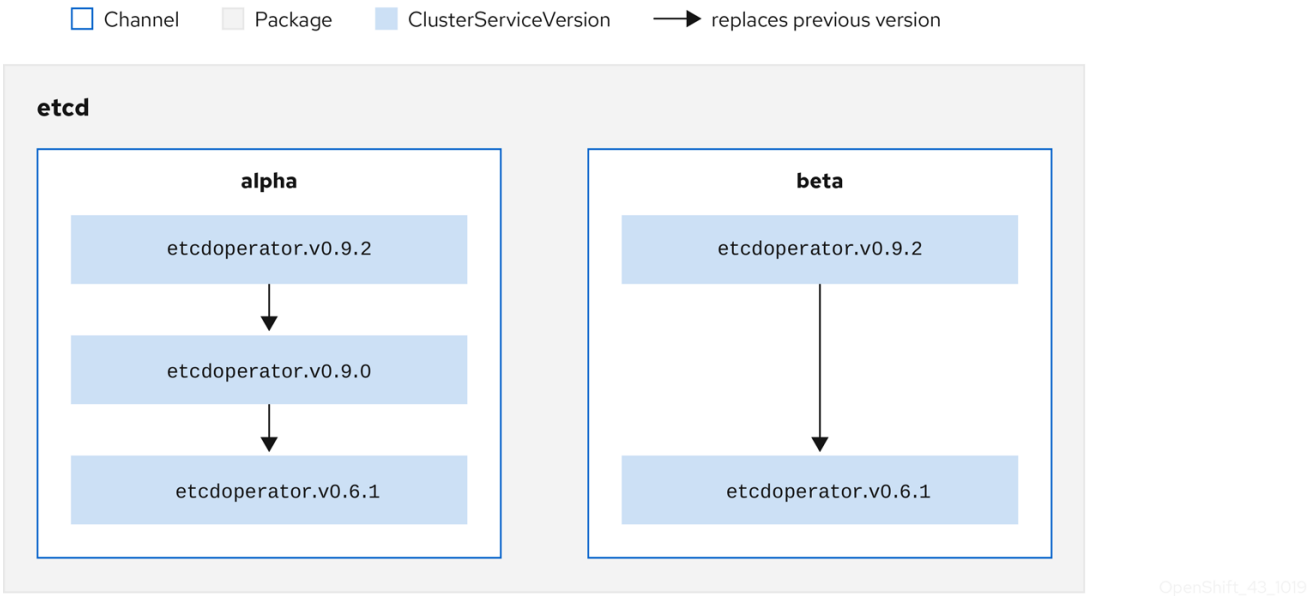
Les métadonnées de l'opérateur, définies dans les CSV, peuvent être stockées dans une collection appelée source de catalogue. L'OLM utilise des sources de catalogue, qui utilisent l'API de registre de l'opérateur, pour interroger les opérateurs disponibles ainsi que des mises à niveau pour les opérateurs installés.

Figure 2.3. Aperçu de la source du catalogue



Au sein d'une source de catalogue, les opérateurs sont organisés en paquets et en flux de mises à jour appelées canaux, ce qui devrait être un modèle de mise à jour familier à partir d'OpenShift Dedicated ou d'autres logiciels sur un cycle de libération continu comme les navigateurs Web.

Figure 2.4. Forfaits et canaux dans une source de catalogue



L'utilisateur indique un paquet et un canal particuliers dans une source de catalogue particulière dans un abonnement, par exemple un paquet etcd et son canal alpha. Lorsqu'un abonnement est effectué à un paquet qui n'a pas encore été installé dans l'espace de noms, le dernier opérateur de ce paquet est installé.

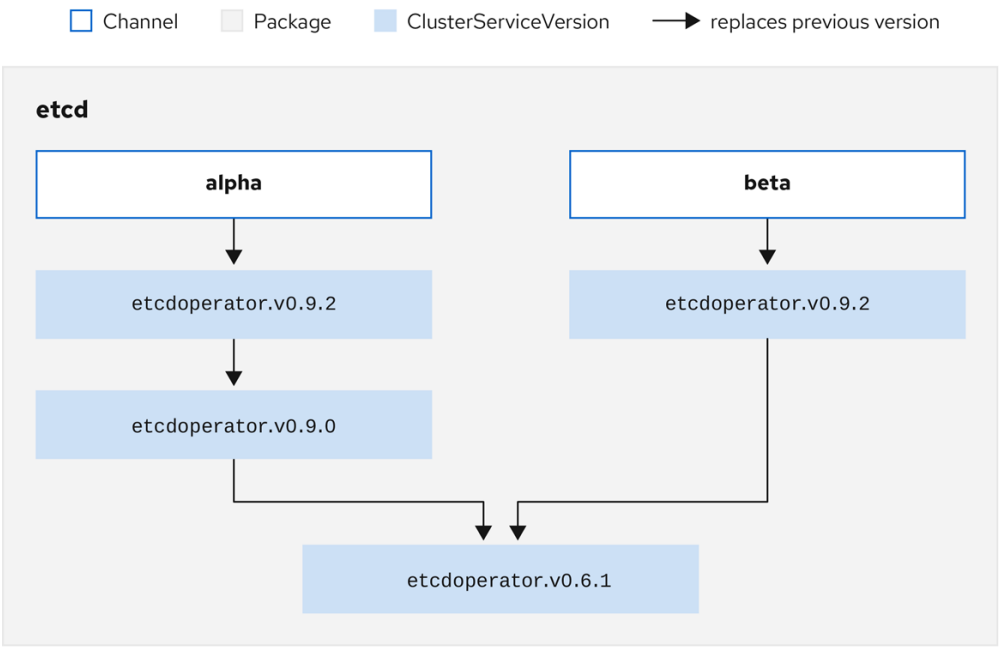
NOTE

L'OLM évite délibérément les comparaisons de versions, de sorte que le "dernier" ou "plus récent" Opérateur disponible à partir d'un catalogue donné → canal → chemin de paquet n'a pas nécessairement besoin d'être le numéro de version le plus élevé. Il devrait être plus considéré comme la référence en tête d'un canal, semblable à un dépôt Git.

Chaque CSV a un paramètre de remplacement qui indique quel opérateur il remplace. Cela crée un graphique de CSV qui peut être interrogé par OLM, et les mises à jour peuvent être partagées entre les canaux. Les canaux peuvent être considérés comme des points d'entrée dans le graphique des mises à

jour:

Figure 2.5. Graphique OLM des mises à jour des canaux disponibles



Exemples de canaux dans un paquet

```
packageName: example
channels:
- name: alpha
  currentCSV: example.v0.1.2
- name: beta
  currentCSV: example.v0.1.3
defaultChannel: alpha
```

Afin que OLM puisse interroger avec succès les mises à jour, compte tenu d'une source de catalogue, d'un paquet, d'un canal et d'un CSV, un catalogue doit pouvoir retourner, sans ambiguïté et de façon déterministe, un seul CSV qui remplace le CSV d'entrée.

2.4.3.1.1. Exemple de chemin de mise à niveau

Dans un exemple de scénario de mise à niveau, envisagez un opérateur installé correspondant à la version 0.1.1 de CSV. L'OLM interroge la source du catalogue et détecte une mise à niveau dans le canal souscrit avec la nouvelle version CSV 0.1.3 qui remplace une version CSV ancienne mais non installée 0.1.2, qui remplace à son tour la version CSV plus ancienne et installée 0.1.1.

L'OLM retourne de la tête du canal aux versions précédentes via le champ de remplacement spécifié dans les CSV pour déterminer le chemin de mise à niveau 0.1.3 → 0.1.2 → 0.1.1; la direction de la flèche indique que le premier remplace la seconde. L'OLM met à niveau l'opérateur une version à l'époque jusqu'à ce qu'il atteigne la tête du canal.

Dans ce scénario donné, OLM installe la version 0.1.2 de l'opérateur pour remplacer la version 0.1.1 de l'opérateur existant. Ensuite, il installe la version 0.1.3 de l'opérateur pour remplacer la version 0.1.2 de l'opérateur précédemment installé. À ce stade, la version 0.1.3 de l'opérateur installé correspond à la tête du canal et la mise à niveau est terminée.

2.4.3.1.2. Des mises à niveau de saut

Le chemin de base pour les mises à niveau dans OLM est:

- La source du catalogue est mise à jour avec une ou plusieurs mises à jour d'un opérateur.
- L'OLM traverse toutes les versions de l'Opérateur jusqu'à atteindre la dernière version que contient la source du catalogue.

Cependant, parfois, ce n'est pas une opération sûre à effectuer. Il y aura des cas où une version publiée d'un opérateur ne devrait jamais être installée sur un cluster s'il ne l'a pas déjà fait, par exemple parce qu'une version introduit une vulnérabilité grave.

Dans ces cas, OLM doit tenir compte de deux états de cluster et fournir un graphique de mise à jour qui prend en charge les deux:

- Le "mauvais" opérateur intermédiaire a été vu par le cluster et installé.
- Le "mauvais" opérateur intermédiaire n'a pas encore été installé sur le cluster.

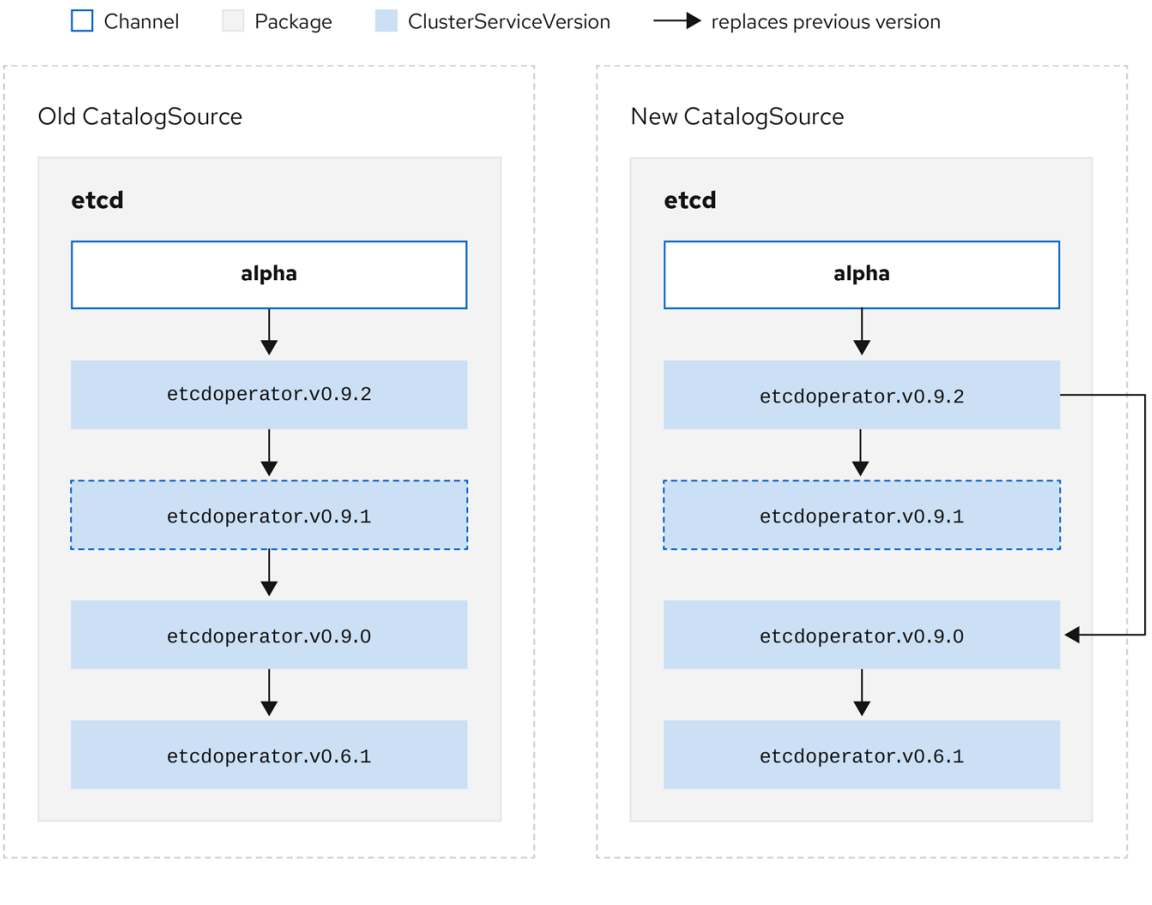
En expédiant un nouveau catalogue et en ajoutant une version sautée, OLM est assuré qu'il peut toujours obtenir une seule mise à jour unique quel que soit l'état du cluster et s'il a vu la mauvaise mise à jour encore.

Exemple CSV avec version sautée

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: etcdoperator.v0.9.2
  namespace: placeholder
  annotations:
spec:
    displayName: etcd
    description: Etcd Operator
    replaces: etcdoperator.v0.9.0
    skips:
    - etcdoperator.v0.9.1
```

Considérez l'exemple suivant de Old CatalogSource et New CatalogSource.

Figure 2.6. Sauter les mises à jour



Ce graphique maintient que:

- Chaque opérateur trouvé dans Old CatalogSource a un seul remplacement dans New CatalogSource.
- Chaque opérateur trouvé dans New CatalogSource a un seul remplacement dans New CatalogSource.
- Lorsque la mauvaise mise à jour n'a pas encore été installée, elle ne le sera jamais.

2.4.3.1.3. Le remplacement de plusieurs opérateurs

Créer un nouveau catalogueSource comme décrit nécessite la publication de CSV qui remplacent un opérateur, mais peuvent en sauter plusieurs. Cela peut être accompli en utilisant l'annotation de skipRange:

```
olm.skipRange: <semver_range>
```

lorsque <semver_range> a le format de plage de version pris en charge par la bibliothèque semver.

Lors de la recherche de catalogues pour les mises à jour, si la tête d'un canal a une annotation skipRange et que l'opérateur actuellement installé a un champ de version qui tombe dans la plage, OLM met à jour la dernière entrée dans le canal.

L'ordre de préséance est:

1. Tête de canal dans la source spécifiée par sourceName sur l'abonnement, si les autres critères de saut sont remplis.

2. L'opérateur suivant qui remplace le courant, dans la source spécifiée par sourceName.

3. Canal tête dans une autre source qui est visible à l'abonnement, si les autres critères de saut sont remplis.

4. L'opérateur suivant qui remplace le courant dans n'importe quelle source visible à l'abonnement.

Exemple CSV avec skipRange

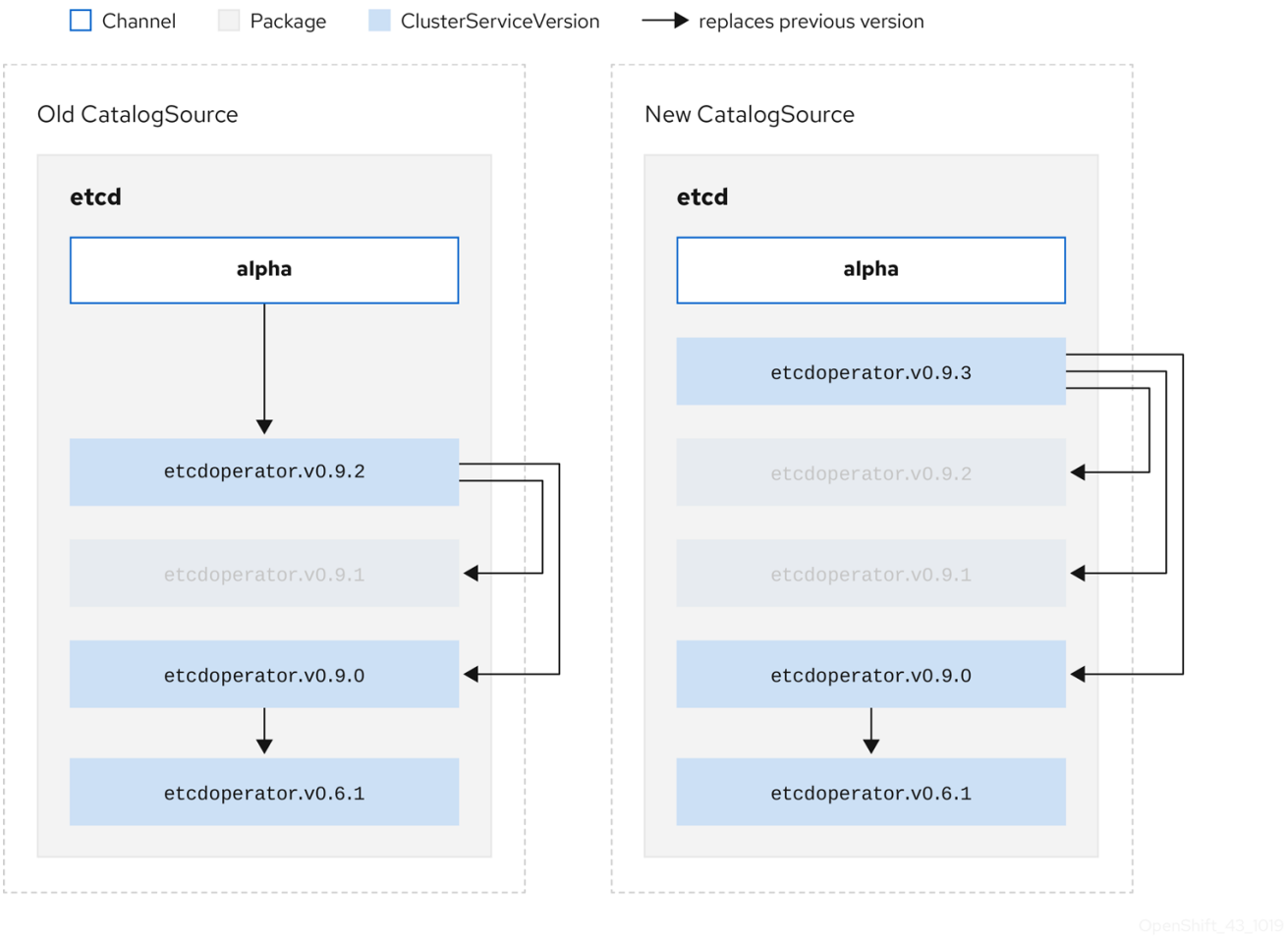
```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: elasticsearch-operator.v4.1.2
  namespace: <namespace>
  annotations:
    olm.skipRange: '>=4.1.0 <4.1.2'
```

2.4.3.1.4. Le support Z-stream

La version z-stream ou patch doit remplacer toutes les versions précédentes de z-stream pour la même version mineure. L'OLM ne considère pas les versions majeures, mineures ou patchées, il suffit de construire le graphique correct dans un catalogue.

En d'autres termes, OLM doit pouvoir prendre un graphique comme dans Old CatalogSource et, comme auparavant, générer un graphique comme dans New CatalogSource:

Figure 2.7. Le remplacement de plusieurs opérateurs



Ce graphique maintient que:

- Chaque opérateur trouvé dans Old CatalogSource a un seul remplacement dans New CatalogSource.
- Chaque opérateur trouvé dans New CatalogSource a un seul remplacement dans New CatalogSource.
- Chaque version z-stream dans Old CatalogSource sera mise à jour vers la dernière version z-stream dans New CatalogSource.
- Les versions indisponibles peuvent être considérées comme des nœuds graphiques « virtuels » ; leur contenu n'a pas besoin d'exister, le registre doit simplement répondre comme si le graphique ressemblait à cela.

2.4.4. La résolution de dépendance du gestionnaire de cycle de vie de l'opérateur

Ce guide décrit la résolution des dépendances et la définition de ressources personnalisées (CRD) mises à niveau des cycles de vie avec Operator Lifecycle Manager (OLM) dans OpenShift Dedicated.

2.4.4.1. À propos de la résolution de dépendance

Le gestionnaire de cycle de vie de l'opérateur (OLM) gère la résolution de dépendance et la mise à niveau du cycle de vie des opérateurs en cours d'exécution. À bien des égards, les problèmes auxquels OLM est confronté sont similaires à d'autres gestionnaires de systèmes ou de paquets linguistiques, tels

que yum et rpm.

Cependant, il y a une contrainte que les systèmes similaires n'ont généralement pas que OLM: parce que les opérateurs sont toujours en cours d'exécution, OLM tente de s'assurer que vous ne restez jamais avec un ensemble d'opérateurs qui ne fonctionnent pas les uns avec les autres.

En conséquence, OLM ne doit jamais créer les scénarios suivants:

- Installer un ensemble d'opérateurs nécessitant des API qui ne peuvent pas être fournies
- Actualisez un opérateur d'une manière qui en casse un autre qui en dépend

Cela est rendu possible avec deux types de données:

Les propriétés	Dactylographié métadonnées sur l'opérateur qui constitue l'interface publique pour lui dans le résolveur de dépendance. Les exemples incluent le groupe/version/type (GVK) des API fournies par l'opérateur et la version sémantique (semver) de l'opérateur.
Contraintes ou dépendances	Les exigences d'un opérateur qui devraient être satisfaites par d'autres opérateurs qui auraient pu ou non déjà été installés sur le cluster cible. Ceux-ci agissent comme des requêtes ou des filtres sur tous les opérateurs disponibles et limitent la sélection lors de la résolution et de l'installation de dépendance. Les exemples incluent le fait d'exiger qu'une API spécifique soit disponible sur le cluster ou de s'attendre à ce qu'un opérateur particulier avec une version particulière soit installé.

L'OLM convertit ces propriétés et contraintes en un système de formules booléennes et les transmet à un solveur SAT, un programme qui établit la satisfaction booléenne, qui fait le travail de déterminer ce que les opérateurs devraient être installés.

2.4.4.2. Les propriétés de l'opérateur

Les opérateurs d'un catalogue ont les propriétés suivantes:

emballage OLM.

Comprend le nom du paquet et la version de l'opérateur

à propos de OLM.gvk

D'une seule propriété pour chaque API fournie à partir de la version de service cluster (CSV)

Des propriétés supplémentaires peuvent également être déclarées directement par un auteur de l'opérateur en incluant un fichier properties.yaml dans les métadonnées/annuaire du paquet Opérateur.

Exemple de propriété arbitraire

```
properties:
- type: olm.kubeversion
  value:
    version: "1.16.0"
```

2.4.4.2.1. Des propriétés arbitraires

Les auteurs d'opérateurs peuvent déclarer des propriétés arbitraires dans un fichier properties.yaml dans le répertoire métadonnées du paquet Opérateur. Ces propriétés sont traduites en une structure de

données cartographiques qui est utilisée comme entrée pour le résolveur de cycle de vie de l'opérateur (OLM) au moment de l'exécution.

Ces propriétés sont opaques pour le résolveur car il ne comprend pas les propriétés, mais il peut évaluer les contraintes génériques par rapport à ces propriétés pour déterminer si les contraintes peuvent être satisfaites compte tenu de la liste des propriétés.

Exemple de propriétés arbitraires

```
properties:
  - property:
      type: color
      value: red
  - property:
      type: shape
      value: square
  - property:
      type: olm.gvk
      value:
        group: olm.coreos.io
        version: v1alpha1
        kind: myresource
```

Cette structure peut être utilisée pour construire une expression du langage d'expression commune (CEL) pour des contraintes génériques.

Ressources supplémentaires

- Les contraintes du langage d'expression commune (CEL)

2.4.4.3. Dépendances des opérateurs

Les dépendances d'un opérateur sont listées dans un fichier de dépendances.yaml dans le dossier métadonnées d'un bundle. Ce fichier est facultatif et actuellement utilisé uniquement pour spécifier des dépendances explicites de version d'opérateur.

La liste de dépendances contient un champ de type pour chaque élément pour spécifier quel type de dépendance il s'agit. Les types de dépendances des opérateurs suivants sont pris en charge:

emballage OLM.

Ce type indique une dépendance pour une version spécifique de l'opérateur. Les informations de dépendance doivent inclure le nom du paquet et la version du paquet au format semver. À titre d'exemple, vous pouvez spécifier une version exacte telle que 0.5.2 ou une gamme de versions telles que >0.5.1.

à propos de OLM.gvk

Avec ce type, l'auteur peut spécifier une dépendance avec des informations de groupe/version/type (GVK), similaire à l'utilisation existante CRD et API dans un CSV. Il s'agit d'un chemin permettant aux auteurs de l'opérateur de consolider toutes les dépendances, API ou versions explicites, pour être au même endroit.

limite OLM.

Ce type déclare des contraintes génériques sur les propriétés arbitraires de l'opérateur.

Dans l'exemple suivant, les dépendances sont spécifiées pour un opérateur Prometheus et etcd CRD:

Exemple de fichier dépendances.yaml

```
dependencies:
  - type: olm.package
    value:
      packageName: prometheus
      version: ">0.27.0"
  - type: olm.gvk
    value:
      group: etcd.database.coreos.com
      kind: EtcdCluster
      version: v1beta2
```

2.4.4.4. Contraintes génériques

La propriété olm.constraint déclare une contrainte de dépendance d'un type particulier, différenciant les propriétés non contraintes et les propriétés de contrainte. Le champ de valeur est un objet contenant un champ failMessage tenant une chaîne-représentation du message de contrainte. Ce message est présenté comme un commentaire informatif aux utilisateurs si la contrainte n'est pas satisfaisante au moment de l'exécution.

Les touches suivantes indiquent les types de contrainte disponibles:

GVK

Le type dont la valeur et l'interprétation sont identiques au type olm.gvk

forfait

Le type dont la valeur et l'interprétation sont identiques au type olm.package

CEL

Expression du langage d'expression commune (CEL) évaluée au cours de l'exécution par le résolveur du gestionnaire du cycle de vie de l'opérateur (OLM) sur les propriétés arbitraires des faisceaux et les informations de cluster

de tout, n'importe quoi, pas

Contraintes de conjonction, de disjonction et de négation, respectivement, contenant une ou plusieurs contraintes concrètes, telles que gvk ou une contrainte composée imbriquée

2.4.4.4.1. Les contraintes du langage d'expression commune (CEL)

Le type de contrainte Cel prend en charge le langage d'expression commun (CEL) comme langage d'expression. La structure de cel a un champ de règle qui contient la chaîne d'expression CEL qui est évaluée par rapport aux propriétés de l'opérateur à l'exécution pour déterminer si l'opérateur satisfait à la contrainte.

Exemple de contrainte cel

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified"'
  cel:
    rule: 'properties.exists(p, p.type == "certified")'
```

La syntaxe CEL prend en charge un large éventail d'opérateurs logiques, tels que AND et OR. En conséquence, une seule expression CEL peut avoir plusieurs règles pour de multiples conditions qui sont

liées entre elles par ces opérateurs logiques. Ces règles sont évaluées par rapport à un ensemble de données de plusieurs propriétés différentes à partir d'un faisceau ou d'une source donnée, et la sortie est résolue en un seul faisceau ou opérateur qui satisfait toutes ces règles à l'intérieur d'une seule contrainte.

Exemple de contrainte cel avec plusieurs règles

```
type: olm.constraint
value:
  failureMessage: 'require to have "certified" and "stable" properties'
  cel:
    rule: 'properties.exists(p, p.type == "certified") && properties.exists(p, p.type == "stable")'
```

2.4.4.4.2. Contraintes composées (toutes, toutes, pas)

Les types de contrainte composée sont évalués en fonction de leurs définitions logiques.

Ce qui suit est un exemple de contrainte conjonctive (tous) de deux paquets et d'un GVK. C'est-à-dire qu'ils doivent tous être satisfaits par des paquets installés:

Exemple toutes les contraintes

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: All are required for Red because...
    all:
      constraints:
      - failureMessage: Package blue is needed for...
        package:
          name: blue
          versionRange: '>=1.0.0'
      - failureMessage: GVK Green/v1 is needed for...
        gvk:
          group: greens.example.com
          version: v1
          kind: Green
```

Ce qui suit est un exemple d'une contrainte disjonctive (toute) de trois versions du même GVK. C'est-à-dire qu'au moins un doit être satisfait par des paquets installés:

Exemple de toute contrainte

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Any are required for Red because...
    any:
      constraints:
      - gvk:
```

```
      group: blues.example.com
      version: v1beta1
      kind: Blue
    - gvk:
      group: blues.example.com
      version: v1beta2
      kind: Blue
    - gvk:
      group: blues.example.com
      version: v1
      kind: Blue
```

Ce qui suit est un exemple de contrainte de négation (non) d'une version d'un GVK. C'est-à-dire que ce GVK ne peut être fourni par aucun paquet dans l'ensemble de résultats:

Exemple non contrainte

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
  all:
    constraints:
    - failureMessage: Package blue is needed for...
      package:
        name: blue
        versionRange: '>=1.0.0'
    - failureMessage: Cannot be required for Red because...
      not:
        constraints:
        - gvk:
            group: greens.example.com
            version: v1alpha1
            kind: greens
```

La sémantique de négation peut sembler floue dans le contexte non de contrainte. Afin de clarifier, la négation est vraiment d'ordonner au résolveur de supprimer toute solution possible qui inclut un GVK particulier, un paquet dans une version, ou satisfait une contrainte composée d'enfants à partir de l'ensemble de résultats.

En tant que corollaire, la contrainte non composée ne doit être utilisée que dans toutes les contraintes, car la négation sans d'abord sélectionner un ensemble possible de dépendances n'a pas de sens.

2.4.4.4.3. Contraintes des composés imbriqués

La contrainte composée imbriquée, qui contient au moins une contrainte composée enfantine ainsi que zéro ou plus contraintes simples, est évaluée à partir du bas vers le haut selon les procédures pour chaque type de contrainte décrit précédemment.

Ce qui suit est un exemple de disjonction de conjonctions, où l'une, l'autre, ou les deux peuvent satisfaire la contrainte:

Exemple de contrainte composée imbriquée

```
schema: olm.bundle
name: red.v1.0.0
properties:
- type: olm.constraint
  value:
    failureMessage: Required for Red because...
    any:
      constraints:
      - all:
          constraints:
          - package:
              name: blue
              versionRange: '>=1.0.0'
          - gvk:
              group: blues.example.com
              version: v1
              kind: Blue
      - all:
          constraints:
          - package:
              name: blue
              versionRange: '<1.0.0'
          - gvk:
              group: blues.example.com
              version: v1beta1
              kind: Blue
```

NOTE

La taille brute maximale d'un type olm.constraint est de 64KB pour limiter les attaques d'épuisement des ressources.

2.4.4.5. Les préférences de dépendance

Il peut y avoir de nombreuses options qui répondent également à une dépendance d'un opérateur. Le résolveur de dépendance dans Operator Lifecycle Manager (OLM) détermine quelle option correspond le mieux aux exigences de l'opérateur demandé. En tant qu'auteur ou utilisateur de l'opérateur, il peut être important de comprendre comment ces choix sont faits afin que la résolution de dépendance soit claire.

2.4.4.5.1. La priorité du catalogue

Dans OpenShift Dedicated cluster, OLM lit les sources du catalogue pour savoir quels opérateurs sont disponibles pour l'installation.

Exemple d'objet CatalogSource

```
apiVersion: "operators.coreos.com/v1alpha1"
kind: "CatalogSource"
metadata:
  name: "my-operators"
  namespace: "operators"
spec:
  sourceType: grpc
```

```
grpcPodConfig:
  securityContextConfig: <security_mode> 1
image: example.com/my/operator-index:v1
displayName: "My Operators"
priority: 100
```

1 Indiquez la valeur de l'héritage ou de la restriction. Lorsque le champ n'est pas défini, la valeur par défaut est héritée. Dans une version ultérieure d'OpenShift Dedicated, il est prévu que la valeur par défaut soit limitée. Dans le cas où votre catalogue ne peut pas fonctionner avec des autorisations restreintes, il est recommandé de définir manuellement ce champ sur l'héritage.

L'objet CatalogSource a un champ de priorité, qui est utilisé par le résolveur pour savoir comment préférer les options pour une dépendance.

Il y a deux règles qui régissent la préférence du catalogue:

- Les options dans les catalogues à priorité élevée sont préférées aux options dans les catalogues à priorité inférieure.
- Les options dans le même catalogue que la personne dépendante sont préférées à tout autre catalogue.

2.4.4.5.2. Commande de canaux

Le package Opérateur dans un catalogue est une collection de canaux de mise à jour auxquels un utilisateur peut s'abonner dans un cluster dédié OpenShift. Les canaux peuvent être utilisés pour fournir un flux particulier de mises à jour pour une libération mineure (1.2, 1.3) ou une fréquence de libération (stable, rapide).

Il est probable qu'une dépendance pourrait être satisfaite par les opérateurs dans le même paquet, mais différents canaux. À titre d'exemple, la version 1.2 d'un opérateur peut exister dans les canaux stables et rapides.

Chaque paquet a un canal par défaut, qui est toujours préféré aux canaux non par défaut. Dans le cas où aucune option dans le canal par défaut ne peut satisfaire une dépendance, des options sont envisagées à partir des canaux restants dans l'ordre lexicographique du nom du canal.

2.4.4.5.3. Commandez à l'intérieur d'un canal

Il y a presque toujours plusieurs options pour satisfaire une dépendance au sein d'un seul canal. À titre d'exemple, les opérateurs d'un seul paquet et d'un canal fournissent le même ensemble d'API.

Lorsqu'un utilisateur crée un abonnement, il indique quel canal recevoir des mises à jour. Cela réduit immédiatement la recherche à ce seul canal. Cependant, à l'intérieur du canal, il est probable que de nombreux opérateurs répondent à une dépendance.

Au sein d'un canal, les opérateurs plus récents qui sont plus haut dans le graphique de mise à jour sont préférés. Lorsque la tête d'un canal satisfait à une dépendance, elle sera d'abord essayée.

2.4.4.5.4. Autres contraintes

En plus des contraintes fournies par les dépendances de paquets, OLM inclut des contraintes supplémentaires pour représenter l'état utilisateur souhaité et imposer des invariants de résolution.

2.4.4.5.4.1. Contrainte d'abonnement

La contrainte d'abonnement filtre l'ensemble des opérateurs qui peuvent satisfaire un abonnement. Les abonnements sont des contraintes fournies par l'utilisateur pour le résolveur de dépendance. Ils déclarent avoir l'intention soit d'installer un nouvel opérateur s'il n'est pas déjà sur le cluster, soit de tenir à jour un opérateur existant.

2.4.4.5.4.2. Contrainte du paquet

Dans un espace de noms, aucun opérateur ne peut provenir du même paquet.

2.4.4.5.5. Ressources supplémentaires

- Exigences en matière de santé du catalogue

2.4.4.6. Les mises à niveau de CRD

L'ODM met immédiatement à niveau une définition de ressource personnalisée (CRD) s'il appartient à une version de service de cluster singulier (CSV). Lorsqu'un CRD est détenu par plusieurs CSV, le CRD est mis à niveau lorsqu'il a satisfait à toutes les conditions suivantes:

- Dans le nouveau CRD, toutes les versions de service existantes sont présentes dans le nouveau CRD.
- Les instances existantes, ou ressources personnalisées, associées aux versions de service du CRD sont valides lorsqu'elles sont validées par rapport au schéma de validation du nouveau CRD.

Ressources supplémentaires

- Ajout d'une nouvelle version CRD
- Dépréciation ou suppression d'une version CRD

2.4.4.7. Les meilleures pratiques en matière de dépendance

Lorsque vous spécifiez les dépendances, il existe des pratiques exemplaires que vous devriez considérer.

Dépendez des API ou d'une gamme de versions spécifique d'opérateurs

Les opérateurs peuvent ajouter ou supprimer des API à tout moment; toujours spécifier une dépendance olm.gvk sur toutes les API requises par vos opérateurs. L'exception à cela est si vous spécifiez les contraintes olm.package à la place.

Définir une version minimale

La documentation Kubernetes sur les modifications d'API décrit les changements autorisés pour les opérateurs de style Kubernetes. Ces conventions de version permettent à un opérateur de mettre à jour une API sans heurter la version API, tant que l'API est rétrocompatible.
En ce qui concerne les dépendances des opérateurs, cela signifie que connaître la version API d'une dépendance peut ne pas suffire à s'assurer que l'opérateur dépendant fonctionne comme prévu.

À titre d'exemple:

- Le TestOperator v1.0.0 fournit la version API v1alpha1 de la ressource MyObject.

- Le TestOperator v1.0.1 ajoute un nouveau champ spec.newfield à MyObject, mais toujours à v1alpha1.

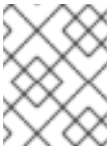
Il se peut que votre opérateur ait besoin de la possibilité d'écrire spec.newfield dans la ressource MyObject. À elle seule, une contrainte olm.gvk ne suffit pas à OLM pour déterminer que vous avez besoin de TestOperator v1.0.1 et non TestOperator v1.0.0.

Dans la mesure du possible, si un opérateur spécifique qui fournit une API est connu à l'avance, spécifiez une contrainte supplémentaire olm.package pour définir un minimum.

Omettre une version maximale ou permettre une très large gamme

Étant donné que les opérateurs fournissent des ressources en grappes telles que les services API et les CRD, un opérateur qui spécifie une petite fenêtre pour une dépendance peut entraver inutilement les mises à jour pour les autres consommateurs de cette dépendance.
Dans la mesure du possible, ne définissez pas une version maximale. Alternativement, définissez une gamme sémantique très large pour éviter les conflits avec d'autres opérateurs. À titre d'exemple, >1.0.0 <2.0.0.

Contrairement aux gestionnaires de paquets conventionnels, les auteurs d'opérateurs codent explicitement que les mises à jour sont sûres via les canaux dans OLM. Lorsqu'une mise à jour est disponible pour un abonnement existant, il est supposé que l'auteur de l'opérateur indique qu'il peut mettre à jour à partir de la version précédente. La définition d'une version maximale pour une dépendance remplace le flux de mise à jour de l'auteur en la tronquant inutilement à une limite supérieure particulière.



NOTE

Les administrateurs de cluster ne peuvent pas remplacer les dépendances définies par un auteur de l'opérateur.

Cependant, les versions maximales peuvent et doivent être définies s'il y a des incompatibilités connues qui doivent être évitées. Les versions spécifiques peuvent être omises avec la syntaxe de la plage de versions, par exemple > 1.0.0 !1.2.1.

Ressources supplémentaires

- Documentation Kubernetes : Changer l'API

2.4.4.8. Avertissements de dépendance

Lorsque vous spécifiez des dépendances, il y a des mises en garde que vous devriez considérer.

Aucune contrainte composée (AND)

Il n'existe actuellement aucune méthode pour spécifier une relation ET entre les contraintes. En d'autres termes, il n'y a aucun moyen de spécifier qu'un opérateur dépend d'un autre opérateur qui fournit à la fois une API donnée et une version >1.1.0.
Cela signifie que lorsque vous spécifiez une dépendance telle que:

```
dependencies:
- type: olm.package
  value:
    packageName: etcd
    version: ">3.1.0"
```

```
- type: olm.gvk
  value:
    group: etcd.database.coreos.com
    kind: EtcdCluster
    version: v1beta2
```

Il serait possible pour OLM de satisfaire cela avec deux opérateurs : l'un qui fournit EtcdCluster et l'autre qui a la version >3.1.0. La question de savoir si cela se produit, ou si un opérateur est sélectionné qui satisfait aux deux contraintes, dépend de la commande que les options potentielles sont visitées. Les préférences de dépendance et les options de commande sont bien définies et peuvent être motivées, mais pour faire preuve de prudence, les opérateurs devraient s'en tenir à un mécanisme ou à l'autre.

Compatibilité cross-namespace

L'ODM effectue une résolution de dépendance à la portée de l'espace de noms. Il est possible d'entrer dans une impasse de mise à jour si la mise à jour d'un opérateur dans un espace de noms serait un problème pour un opérateur dans un autre espace de noms, et vice-versa.

2.4.4.9. Exemples de scénarios de résolution de dépendance

Dans les exemples suivants, un fournisseur est un opérateur qui « possède » un service CRD ou API.

Exemple : Dépréciation des API dépendantes
A et B sont des API (CRD):

- Le fournisseur de A dépend de B.
- Le fournisseur de B a un abonnement.
- Le fournisseur de mises à jour B pour fournir C mais déprécie B.

Il en résulte:

- B n'a plus de fournisseur.
- A ne fonctionne plus.

C'est un cas que OLM empêche avec sa stratégie de mise à niveau.

Exemple : dans l'impasse de version
A et B sont des API:

- Le fournisseur de A nécessite B.
- Le fournisseur de B nécessite A.
- Le fournisseur de mises à jour A (fournir A2, nécessite B2) et déprécater A.
- Le fournisseur de mises à jour B (fournir B2, nécessite A2) et déprécier B.

Lorsque OLM tente de mettre à jour A sans mettre à jour simultanément B, ou vice-versa, il est impossible de passer à de nouvelles versions des Opérateurs, même si un nouvel ensemble compatible peut être trouvé.

C'est un autre cas que OLM empêche avec sa stratégie de mise à niveau.

2.4.5. Groupes d'opérateurs

Ce guide décrit l'utilisation des groupes d'opérateurs avec le gestionnaire de cycle de vie de l'opérateur (OLM) dans OpenShift Dedicated.

2.4.5.1. À propos des groupes d'opérateurs

Le groupe d'opérateurs, défini par la ressource OperatorGroup, fournit une configuration multilocataires aux Opérateurs installés par OLM. Le groupe d'opérateur sélectionne les espaces de noms cibles dans lesquels générer l'accès RBAC requis pour ses opérateurs membres.

L'ensemble des espaces de noms cibles est fourni par une chaîne délimitée par virgule stockée dans l'annotation olm.targetNamespaces d'une version de service cluster (CSV). Cette annotation est appliquée aux instances CSV des opérateurs membres et est projetée dans leurs déploiements.

2.4.5.2. Adhésion au groupe d'opérateurs

L'opérateur est considéré comme un membre d'un groupe d'opérateurs si les conditions suivantes sont vraies:

- Le CSV de l'Opérateur existe dans le même espace de noms que le groupe Opérateur.
- Les modes d'installation dans le CSV de l'Opérateur prennent en charge l'ensemble d'espaces de noms ciblés par le groupe Opérateur.

Le mode d'installation dans un CSV se compose d'un champ InstallModeType et d'un champ supporté booléen. La spécification d'un CSV peut contenir un ensemble de modes d'installation de quatre InstallModeTypes distincts:

Tableau 2.5. Installer des modes et des groupes d'opérateurs pris en charge

InstallerModeType	Description
À propos de OwnNamespace	L'opérateur peut être membre d'un groupe d'opérateurs qui sélectionne son propre espace de noms.
Espace de SingleNam	L'opérateur peut être membre d'un groupe d'opérateurs qui sélectionne un espace de noms.
Espace MultiNam	L'opérateur peut être membre d'un groupe d'opérateurs qui sélectionne plus d'un espace de noms.
AllNamespaces	L'opérateur peut être membre d'un groupe d'opérateurs qui sélectionne tous les espaces de noms (le jeu d'espace de noms cible est la chaîne vide "").

NOTE

Lorsque la spécification d'un CSV omet une entrée de InstallModeType, alors ce type est considéré comme non pris en charge, sauf si le support peut être déduit par une entrée existante qui le supporte implicitement.

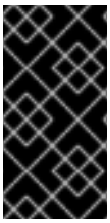
2.4.5.3. Choix de l'espace de noms cible

Il est possible de nommer explicitement l'espace de noms cible pour un groupe d'opérateurs en utilisant le paramètre spec.targetNamespaces:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  targetNamespaces:
  - my-namespace
```

Alternativement, vous pouvez spécifier un espace de noms à l'aide d'un sélecteur d'étiquettes avec le paramètre spec.selector:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
spec:
  selector:
    cool.io/prod: "true"
```



IMPORTANT

L'inscription de plusieurs espaces de noms via spec.targetNamespaces ou l'utilisation d'un sélecteur d'étiquette via spec.selector n'est pas recommandée, car la prise en charge de plus d'un espace de noms cible dans un groupe d'opérateurs sera probablement supprimée dans une version ultérieure.

Lorsque spec.targetNamespaces et spec.selector sont définis, spec.selector est ignoré. Alternativement, vous pouvez omettre à la fois spec.selector et spec.targetNamespaces pour spécifier un groupe d'opérateur global, qui sélectionne tous les espaces de noms:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: my-group
  namespace: my-namespace
```

L'ensemble résolu d'espaces de noms sélectionnés est affiché dans le paramètre status.namespaces d'un groupe Opeator. Le status.namespace d'un groupe d'opérateur mondial contient la chaîne vide (""), qui signale à un opérateur consommant qu'il devrait regarder tous les espaces de noms.

2.4.5.4. Annotations du groupe d'opérateurs CSV

Les CSV membres d'un groupe d'opérateurs ont les annotations suivantes:

Annotation	Description
groupe OLM.operator=<group_name>	Contient le nom du groupe Opérateur.

Annotation	Description
espace OLM.operatorNamespace= <group_namespace>	Contient l'espace de noms du groupe Opérateur.
espaces OLM.targetNamespaces=<target_namespaces>	Contient une chaîne délimitée par virgule qui répertorie la sélection de l'espace de noms cible du groupe Opérateur.

NOTE

Les annotations sauf olm.targetNamespaces sont incluses avec des CSV copiés. L'omission de l'annotation olm.targetNamespaces sur les CSV copiés empêche la duplication des espaces de noms cibles entre les locataires.

2.4.5.5. Annotation d'API fournie

Groupe/version/type (GVK) est un identifiant unique pour une API Kubernetes. Les informations sur ce que les GVK sont fournis par un groupe d'opérateurs sont affichées dans une annotation des API olm. La valeur de l'annotation est une chaîne composée de <kind>.<version>.<group> délimitée avec des virgules. Les GVK de CRD et les services API fournis par tous les CSV membres actifs d'un groupe d'opérateurs sont inclus.

Examinez l'exemple suivant d'un objet OperatorGroup avec un seul membre actif CSV qui fournit la ressource PackageManifest:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: PackageManifest.v1alpha1.packages.apps.redhat.com
  name: olm-operators
  namespace: local
  ...
spec:
  selector: {}
  serviceAccountName:
    metadata:
      creationTimestamp: null
  targetNamespaces:
  - local
status:
  lastUpdated: 2019-02-19T16:18:28Z
  namespaces:
  - local
```

2.4.5.6. Contrôle d'accès basé sur le rôle

Lorsqu'un groupe d'opérateurs est créé, trois rôles de cluster sont générés. Chacune contient une seule règle d'agrégation avec un sélecteur de rôle de cluster défini pour correspondre à une étiquette, comme indiqué ci-dessous:

Le rôle des clusters	Étiquette à assortir
<operatorgroup_name>-admin-<hash_value>	**accueil OLM.opgroup.permissions/aggrégate-to-admin: <operatorgroup_name>**
<operatorgroup_name>-edit-<hash_value>	**accueil OLM.opgroup.permissions/aggrégate-to-edit: <operatorgroup_name>**
> OLM.og.<operatorgroup_name>-view-<hash_value>	**accueil OLM.opgroup.permissions/aggrégate-to-view: <operatorgroup_name>**

Les ressources RBAC suivantes sont générées lorsqu'un CSV devient un membre actif d'un groupe d'opérateurs, tant que le CSV surveille tous les espaces de noms avec le mode d'installation AllNamespaces et n'est pas dans un état défaillant avec la raison InterOperatorGroupOwnerConflict:

- Les rôles de cluster pour chaque ressource API à partir d'un CRD
- Les rôles de cluster pour chaque ressource API à partir d'un service API
- Autres rôles et liens de rôle

Tableau 2.6. Les rôles de cluster générés pour chaque ressource API à partir d'un CRD

Le rôle des clusters	Les paramètres
<kind>.<group>-<version>-admin	Les verbes sur <kind>: • ***** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-admin: true** • **accueil OLM.opgroup.permissions/aggrégate-to-admin: <operatorgroup_name>**

Le rôle des clusters	Les paramètres
<kind>.<group>-<version>-edit	Les verbes sur <kind>: • **créer** • **la mise à jour** • **le patch** • **effacer** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-edit: true** • **accueil OLM.opgroup.permissions/aggrégate -to-edit: <operatorgroup_name>**
<kind>.<group>-<version>-view	Les verbes sur <kind>: • **J'obtiens** • **liste** • **la montre** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-view: true** • **accueil OLM.opgroup.permissions/aggrégate -to-view: <operatorgroup_name>**
<kind>.<group>-<version>-view-crdview	Les verbes sur apiextensions.k8s.io customresourcedefinitions <crd-name>: • **J'obtiens** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-view: true** • **accueil OLM.opgroup.permissions/aggrégate -to-view: <operatorgroup_name>**

Tableau 2.7. Les rôles de cluster générés pour chaque ressource API à partir d'un service API

Le rôle des clusters	Les paramètres
<kind>.<group>-<version>-admin	Les verbes sur <kind>: • ***** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-admin: true** • **accueil OLM.opgroup.permissions/aggrégate-to-admin: <operatorgroup_name>**
<kind>.<group>-<version>-edit	Les verbes sur <kind>: • **créer** • **la mise à jour** • **le patch** • **effacer** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-edit: true** • **accueil OLM.opgroup.permissions/aggrégate-to-edit: <operatorgroup_name>**
<kind>.<group>-<version>-view	Les verbes sur <kind>: • **J'obtiens** • **liste** • **la montre** Étiquettes d'agrégation: • **le RBAC. Authorization.k8s.io/agrégate-to-view: true** • **accueil OLM.opgroup.permissions/aggrégate-to-view: <operatorgroup_name>**

Autres rôles et liens de rôle

- Lorsque le CSV définit exactement un espace de noms cible qui contient *, un rôle de cluster et une liaison de rôle de cluster correspondante sont générés pour chaque autorisation définie dans le champ permissions du CSV. L'ensemble des ressources générées sont les étiquettes olm.owner: <csv_name> et olm.owner.namespace: <csv_namespace>.

- Lorsque le CSV ne définit pas exactement un espace de noms cible qui contient *, alors toutes les liaisons de rôles et de rôles dans l'espace de noms de l'opérateur avec les étiquettes olm.owner: <csv_name> et olm.owner.namespace: <csv_namespace> les étiquettes sont copiées dans l'espace de noms cible.

2.4.5.7. CSV copiés

L'OLM crée des copies de tous les CSV membres actifs d'un groupe d'opérateurs dans chacun des espaces de noms cibles de ce groupe d'opérateurs. Le but d'un CSV copié est de dire aux utilisateurs d'un espace de noms cible qu'un opérateur spécifique est configuré pour regarder les ressources créées là-bas.

Les CSV copiés ont une raison de statut Copié et sont mis à jour pour correspondre à l'état de leur CSV source. L'annotation olm.targetNamespaces est dépouillée des CSV copiés avant qu'ils ne soient créés sur le cluster. L'omission de la sélection de l'espace de noms cible évite la duplication des espaces de noms cibles entre les locataires.

Les CSV copiés sont supprimés lorsque leur CSV source n'existe plus ou que le groupe Opérateur que leur source CSV appartient à ne plus cibler l'espace de noms du CSV copié.

NOTE

Le champ de désactivationCopiedCSVs est désactivé par défaut. Après avoir activé un champ de désactivationCopiedCSVs, l'OMM supprime les CSV existants copiés sur un cluster. Lorsqu'un champ de désactivationCopiedCSVs est désactivé, l'OMM ajoute à nouveau des CSV copiés.

- Désactiver le champ de désactivationCopiedCSVs:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: false
EOF
```

- Activer le champ de désactivationCopiedCSVs:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OLMConfig
metadata:
  name: cluster
spec:
  features:
    disableCopiedCSVs: true
EOF
```

2.4.5.8. Groupes d'opérateurs statiques

Le groupe opérateur est statique si son champ spec.staticProvidedAPIs est défini sur true. En conséquence, OLM ne modifie pas l'annotation olm.suppldAPIs d'un groupe d'opérateurs, ce qui signifie qu'il peut être défini à l'avance. Ceci est utile lorsqu'un utilisateur souhaite utiliser un groupe d'opérateurs pour empêcher la contention des ressources dans un ensemble d'espaces de noms, mais n'a pas de CSV membre actif qui fournissent les API pour ces ressources.

Ci-dessous est un exemple d'un groupe d'opérateurs qui protège les ressources Prometheus dans tous les espaces de noms avec le something.cool.io/cluster-monitoring: "vrai" annotation:

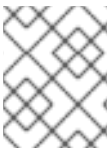
```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-monitoring
  namespace: cluster-monitoring
  annotations:
    olm.providedAPIs:
Alertmanager.v1.monitoring.coreos.com,Prometheus.v1.monitoring.coreos.com,PrometheusRule.v1.mo
nitoring.coreos.com,ServiceMonitor.v1.monitoring.coreos.com
spec:
  staticProvidedAPIs: true
```

```
selector:
  matchLabels:
    something.cool.io/cluster-monitoring: "true"
```

2.4.5.9. Intersection du groupe d'opérateurs

Deux groupes d'opérateurs auraient des API fournies si l'intersection de leurs jeux d'espace de noms n'est pas vide et que l'intersection des jeux d'API fournis, définies par les annotations olm.providedAPIs, n'est pas un ensemble vide.

Le problème potentiel réside dans le fait que les groupes d'opérateurs dotés d'API intersectées peuvent rivaliser pour les mêmes ressources dans l'ensemble des espaces de noms croisés.



NOTE

Lors de la vérification des règles d'intersection, un espace de noms de groupe d'opérateur est toujours inclus dans les espaces de noms cibles sélectionnés.

Les règles pour l'intersection
Chaque fois qu'un membre actif se synchronise, OLM interroge le cluster pour l'ensemble d'intersecting fourni des API entre le groupe Opérateur du CSV et tous les autres. Ensuite, OLM vérifie si cet ensemble est un jeu vide:

- Lorsque true et les API fournies par le CSV sont un sous-ensemble du groupe d'opérateurs:
  - Continuez la transition.
- Lorsque true et les API fournies par le CSV ne sont pas un sous-ensemble du groupe d'opérateurs:
  - Lorsque le groupe d'opérateurs est statique:
    - Nettoyer tous les déploiements appartenant au CSV.
    - La transition du CSV vers un état défaillant avec la raison de statut CannotModifierStaticOperatorGroupProvidedAPIs.
  - B) Si le groupe d'opérateurs n'est pas statique:
    - De remplacer l'annotation olm.fourdAPIs du groupe d'opérateurs par l'union de lui-même et les API fournies par le CSV.
- En cas de faux et les API fournies par le CSV ne sont pas un sous-ensemble du groupe d'opérateurs:
  - Nettoyer tous les déploiements appartenant au CSV.
  - Faites passer le CSV à un état défaillant avec raison de statut InterOperatorGroupOwnerConflict.
- En cas de faux et les API fournies par le CSV sont un sous-ensemble du groupe d'opérateurs:
  - Lorsque le groupe d'opérateurs est statique:
    - Nettoyer tous les déploiements appartenant au CSV.

- La transition du CSV vers un état défaillant avec la raison de statut CannotModifierStaticOperatorGroupProvidedAPIs.

- B) Si le groupe d'opérateurs n'est pas statique:
  - De remplacer l'annotation olm.fourdAPIs du groupe d'opérateurs par la différence entre lui-même et les API fournies par le CSV.

NOTE

Les états de défaillance causés par les groupes d'opérateurs sont non-terminaux.

Les actions suivantes sont effectuées chaque fois qu'un groupe d'opérateurs synchronise:

- L'ensemble d'API fournies à partir des CSV membres actifs est calculé à partir du cluster. Il est à noter que les CSV copiés sont ignorés.
- L'ensemble de clusters est comparé à olm.providedAPIs, et si olm.providedAPIs contient des API supplémentaires, ces API sont taillées.
- Les CSV qui fournissent les mêmes API dans tous les espaces de noms sont requeued. Cela informe les CSV conflictuels des groupes croisés que leur conflit a pu être résolu, soit par le redimensionnement ou par la suppression du CSV conflictuel.

2.4.5.10. Limites pour la gestion des opérateurs multilocataires

Le logiciel OpenShift Dedicated fournit une prise en charge limitée pour l'installation simultanée de différentes versions d'un opérateur sur le même cluster. Le gestionnaire de cycle de vie de l'opérateur (OLM) installe des opérateurs plusieurs fois dans différents espaces de noms. L'une des contraintes est que les versions API de l'opérateur doivent être les mêmes.

Les opérateurs sont des extensions de plan de contrôle en raison de leur utilisation des objets CustomResourceDefinition (CRD), qui sont des ressources globales dans Kubernetes. Différentes versions majeures d'un opérateur ont souvent des CRD incompatibles. Cela les rend incompatibles à installer simultanément dans différents espaces de noms sur un cluster.

Les locataires, ou espaces de noms, partagent le même plan de contrôle d'un cluster. Ainsi, les locataires d'un cluster multilocataires partagent également des CRD mondiaux, ce qui limite les scénarios dans lesquels différents cas d'un même opérateur peuvent être utilisés en parallèle sur le même cluster.

Les scénarios pris en charge comprennent les éléments suivants:

- Les opérateurs de différentes versions qui envoient exactement la même définition CRD (dans le cas des CRD versionnés, le même ensemble de versions)
- Les opérateurs de différentes versions qui n'expédient pas de CRD, et ont plutôt leur CRD disponible dans un paquet séparé sur OperatorHub

Les autres scénarios ne sont pas pris en charge, car l'intégrité des données du cluster ne peut pas être garantie s'il existe plusieurs CRD concurrents ou qui se chevauchent de différentes versions de l'opérateur à concilier sur le même cluster.

Ressources supplémentaires

- Opérateurs en clusters multilocataires

2.4.5.11. Dépannage Groupes d'opérateurs

Adhésion

- L'espace de noms d'un plan d'installation ne doit contenir qu'un seul groupe d'opérateurs. Lors de la tentative de générer une version de service cluster (CSV) dans un espace de noms, un plan d'installation considère un groupe opérateur invalide dans les scénarios suivants:
  - Aucun groupe d'opérateur n'existe dans l'espace de noms du plan d'installation.
  - Des groupes d'opérateurs multiples existent dans l'espace de noms du plan d'installation.
  - Le nom de compte de service incorrect ou inexistant est spécifié dans le groupe Opérateur.

  Lorsqu'un plan d'installation rencontre un groupe d'opérateur invalide, le CSV n'est pas généré et la ressource InstallPlan continue d'installer avec un message pertinent. À titre d'exemple, le message suivant est fourni si plus d'un groupe d'opérateurs existe dans le même espace de noms:

  > attenuated service account query failed - more than one operator group(s) are managing this namespace count=2

  lorsque Count= spécifie le nombre de groupes d'opérateurs dans l'espace de noms.

- Lorsque les modes d'installation d'un CSV ne prennent pas en charge la sélection de l'espace de noms cible du groupe Opérateur dans son espace de noms, le CSV passe à un état de défaillance avec la raison pour laquelle UnsupportedOperatorGroup. Les CSV dans un état défaillant pour cette raison passent en attente après que la sélection de l'espace de noms cible du groupe d'opérateurs modifie une configuration prise en charge, soit les modes d'installation du CSV ont été modifiés pour prendre en charge la sélection de l'espace de noms cible.

2.4.6. Colocation multitenance et opérateur

Ce guide décrit la multitenance et la colocalisation de l'opérateur dans Operator Lifecycle Manager (OLM).

2.4.6.1. Colocation d'opérateurs dans un espace de noms

Le gestionnaire de cycle de vie de l'opérateur (OLM) gère les opérateurs gérés par OLM qui sont installés dans le même espace de noms, ce qui signifie que leurs ressources d'abonnement sont situées dans le même espace de noms, que les opérateurs associés. Bien qu'ils ne soient pas réellement liés, OLM considère leurs états, tels que leur version et leur politique de mise à jour, lorsque l'un d'entre eux est mis à jour.

Ce comportement par défaut se manifeste de deux façons:

- Les ressources InstallPlan des mises à jour en attente incluent les ressources ClusterServiceVersion (CSV) de tous les autres opérateurs qui se trouvent dans le même espace de noms.
- Dans le même espace de noms, tous les opérateurs partagent la même stratégie de mise à jour. À titre d'exemple, si un opérateur est configuré sur des mises à jour manuelles, toutes les politiques de mise à jour des autres opérateurs sont également définies sur manuel.

Ces scénarios peuvent entraîner les problèmes suivants:

- Il devient difficile de raisonner au sujet des plans d'installation pour les mises à jour de l'opérateur, car il y a beaucoup plus de ressources définies en eux que l'opérateur mis à jour.

- Il devient impossible d'avoir certains opérateurs dans une mise à jour de l'espace de noms automatiquement tandis que d'autres sont mis à jour manuellement, ce qui est un désir commun pour les administrateurs de clusters.

Ces problèmes apparaissent généralement parce que, lors de l'installation d'opérateurs avec la console Web dédiée OpenShift, le comportement par défaut installe des opérateurs qui prennent en charge le mode Tous les espaces de noms dans l'espace de noms global openshift-operators par défaut.

En tant qu'administrateur avec le rôle admin dédié, vous pouvez contourner ce comportement par défaut manuellement en utilisant le flux de travail suivant:

1. Créer un projet pour l'installation de l'opérateur.

2. Créez un groupe d'opérateurs mondial personnalisé, qui est un groupe d'opérateurs qui surveille tous les espaces de noms. En associant ce groupe d'opérateurs à l'espace de noms que vous venez de créer, il fait de l'espace de noms d'installation un espace de noms global, ce qui rend les opérateurs installés là-bas disponibles dans tous les espaces de noms.

3. Installez l'opérateur souhaité dans l'espace de noms d'installation.

Lorsque l'opérateur a des dépendances, les dépendances sont automatiquement installées dans l'espace de noms pré-créé. En conséquence, il est alors valable pour les opérateurs de dépendance d'avoir la même stratégie de mise à jour et des plans d'installation partagés. Dans le cadre d'une procédure détaillée, voir « Installation des opérateurs mondiaux dans des espaces de noms personnalisés ».

Ressources supplémentaires

- Installation d'opérateurs globaux dans des espaces de noms personnalisés
- Opérateurs en clusters multilocataires

2.4.7. Conditions de l'opérateur

Ce guide décrit comment le gestionnaire de cycle de vie de l'opérateur (OLM) utilise les conditions de l'opérateur.

2.4.7.1. À propos des conditions de l'opérateur

Dans le cadre de son rôle dans la gestion du cycle de vie d'un opérateur, Operator Lifecycle Manager (OLM) infère l'état d'un opérateur à partir de l'état des ressources Kubernetes qui définissent l'opérateur. Bien que cette approche fournisse un certain niveau d'assurance qu'un exploitant est dans un état donné, il existe de nombreux cas où un opérateur pourrait avoir besoin de communiquer des informations à OLM qui ne pourraient pas être déduits autrement. Ces informations peuvent ensuite être utilisées par OLM pour mieux gérer le cycle de vie de l'opérateur.

L'ODM fournit une définition de ressource personnalisée (CRD) appelée OperatorCondition qui permet aux opérateurs de communiquer les conditions à OLM. Il existe un ensemble de conditions supportées qui influencent la gestion de l'opérateur par OLM lorsqu'elles sont présentes dans la gamme Spec.Conditions d'une ressource OperatorCondition.

NOTE

Le tableau Spec.Conditions n'est pas présent dans un objet OperatorCondition tant qu'il n'est pas ajouté par un utilisateur ou à la suite de la logique personnalisée de l'opérateur.

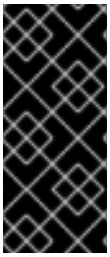
2.4.7.2. Conditions prises en charge

Gestionnaire de cycle de vie de l'opérateur (OLM) prend en charge les conditions suivantes de l'opérateur.

2.4.7.2.1. Condition de mise à niveau

La condition de l'opérateur mis à niveau empêche une version existante de service de cluster (CSV) d'être remplacée par une version plus récente du CSV. Cette condition est utile lorsque:

- L'opérateur est sur le point de démarrer un processus critique et ne doit pas être mis à niveau tant que le processus n'est pas terminé.
- L'opérateur effectue une migration de ressources personnalisées (CR) qui doit être complétée avant que l'opérateur ne soit prêt à être mis à niveau.



IMPORTANT

La mise à niveau de la condition de l'opérateur sur la valeur False n'évite pas la perturbation de la pod. Dans le cas où vous devez vous assurer que vos pods ne sont pas perturbés, consultez « Utiliser les budgets de perturbation de pods pour spécifier le nombre de gousses qui doivent être en hausse » et « Termination Grace » dans la section « Ressources supplémentaires ».

Exemple État de l'opérateur mis à niveau

```
apiVersion: operators.coreos.com/v1
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
  conditions:
  - type: Upgradeable (1)
    status: "False" (2)
    reason: "migration"
    message: "The Operator is performing a migration."
    lastTransitionTime: "2020-08-24T23:15:55Z"
```

(1) Le nom de la condition.

(2) La valeur fausse indique que l'opérateur n'est pas prêt à être mis à niveau. L'OLM empêche un CSV qui remplace le CSV existant de l'opérateur de quitter la phase en attente. La valeur fausse ne bloque pas les mises à niveau de cluster.

2.4.7.3. Ressources supplémentaires

- Gestion des conditions de l'opérateur

- Conditions d'activation de l'opérateur

2.4.8. Gestion du cycle de vie de l'opérateur

2.4.8.1. Les métriques exposées

Le gestionnaire du cycle de vie de l'opérateur (OLM) expose certaines ressources spécifiques aux OLM destinées à être utilisées par la pile de surveillance du cluster OpenShift Dedicated basée sur Prometheus.

Tableau 2.8. Les métriques exposées par OLM

Le nom	Description
catalogue_source_count	Le nombre de sources de catalogue.
cataloguesource_ready	État d'une source de catalogue. La valeur 1 indique que la source du catalogue est dans un état READY. La valeur de 0 indique que la source du catalogue n'est pas dans un état READY.
csv_abnormal	Lors de la réconciliation d'une version de service cluster (CSV), présentez chaque fois qu'une version CSV est dans un état autre que Succeed, par exemple lorsqu'elle n'est pas installée. Inclut le nom, l'espace de noms, la phase, la raison et les étiquettes de version. Lorsque cette métrique est présente, une alerte Prometheus est créée.
csv_count	Le nombre de CSV enregistrés avec succès.
csv_succeed	Lors de la réconciliation d'un CSV, représente si une version CSV est dans un état Succeed (valeur 1) ou non (valeur 0). Inclut le nom, l'espace de noms et les étiquettes de version.
csv_upgrade_count	Comptage monotonique des mises à niveau CSV.
installer_plan_count	Le nombre de plans d'installation.
installplan_avertissements_total	Compte monotone des avertissements générés par les ressources, telles que les ressources dépréciées, inclus dans un plan d'installation.
accueil > olm_resolution_duration_seconds	La durée d'une tentative de résolution de dépendance.
abonnement_Compte	Le nombre d'abonnements.

Le nom	Description
abonnement_sync_total	Comptage monotonique des synchronisations d'abonnement. Inclut le canal, les étiquettes CSV installées et les étiquettes de nom d'abonnement.

2.4.9. Gestion Webhook dans Operator Lifecycle Manager

Les webhooks permettent aux auteurs de l'opérateur d'intercepter, de modifier et d'accepter ou de rejeter des ressources avant qu'elles ne soient enregistrées dans le magasin d'objets et traitées par le contrôleur de l'opérateur. Le gestionnaire de cycle de vie de l'opérateur (OLM) peut gérer le cycle de vie de ces webhooks lorsqu'ils sont expédiés aux côtés de votre opérateur.

Consultez Définir les versions de services de cluster (CSV) pour plus de détails sur la façon dont un développeur d'opérateur peut définir des webhooks pour son opérateur, ainsi que des considérations lors de l'exécution sur OLM.

2.4.9.1. Ressources supplémentaires

- Documentation de Kubernetes:
  - Validation des webhooks d'admission
  - La mutation des webhooks d'admission
  - Conversion webhooks

2.5. COMPRENDRE L'OPÉRATEURHUB

2.5.1. À propos de OperatorHub

OperatorHub est l'interface de console Web d'OpenShift Dédicée que les administrateurs de cluster utilisent pour découvrir et installer des Opérateurs. En un clic, un opérateur peut être tiré de sa source hors-cluster, installé et abonné sur le cluster, et prêt pour les équipes d'ingénierie à gérer en libre-service le produit dans les environnements de déploiement en utilisant Operator Lifecycle Manager (OLM).

Les administrateurs de clusters peuvent choisir parmi les catalogues regroupés dans les catégories suivantes:

Catégorie	Description
Les opérateurs de chapeaux rouges	Les produits Red Hat emballés et expédiés par Red Hat. Avec le soutien de Red Hat.
Des opérateurs certifiés	Les produits des principaux fournisseurs de logiciels indépendants (ISV). Le Red Hat s'associe à des ISV pour emballer et expédier. Avec le soutien de l'ISV.
La place de marché Red Hat	Logiciel certifié qui peut être acheté chez Red Hat Marketplace.

Catégorie	Description
Les opérateurs communautaires	Logiciel éventuellement visible par les représentants concernés dans le référentiel GitHub de redhat-openshift-ecosystem/community-operators-prod/operators. Aucun soutien officiel.
Des opérateurs personnalisés	Les opérateurs que vous ajoutez au cluster vous-même. Dans le cas où vous n'avez pas ajouté d'opérateurs personnalisés, la catégorie Custom n'apparaît pas dans la console Web de votre OperatorHub.

Les opérateurs sur OperatorHub sont emballés pour fonctionner sur OLM. Cela inclut un fichier YAML appelé une version de service de cluster (CSV) contenant tous les CRD, les règles RBAC, les déploiements et les images de conteneur nécessaires pour installer et exécuter en toute sécurité l'opérateur. Il contient également des informations visibles par l'utilisateur comme une description de ses fonctionnalités et les versions prises en charge de Kubernetes.

Le SDK de l'opérateur peut être utilisé pour aider les développeurs à emballer leurs opérateurs pour une utilisation sur OLM et OperatorHub. Lorsque vous avez une application commerciale que vous souhaitez rendre accessible à vos clients, faites-le inclure en utilisant le flux de travail de certification fourni sur le portail Red Hat Partner Connect sur connect.redhat.com.

2.5.2. Architecture OperatorHub

Le composant OperatorHub UI est piloté par l'opérateur Marketplace par défaut sur OpenShift Dedicated dans l'espace de noms openshift-marketplace.

2.5.2.1. OperatorHub ressource personnalisée

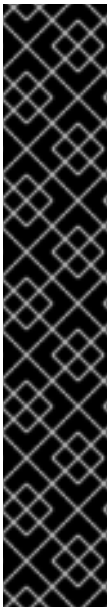
L'opérateur Marketplace gère une ressource personnalisée OperatorHub (CR) nommée cluster qui gère les objets CatalogSource par défaut fournis avec OperatorHub.

2.5.3. Ressources supplémentaires

- Catalogue source
- À propos de l'opérateur SDK
- Définition des versions de service cluster (CSV)
- Installation de l'opérateur et mise à niveau du flux de travail dans OLM
- Connexion de Red Hat Partner
- La place de marché Red Hat

2.6. CATALOGUES D'OPÉRATEURS RED HAT

Le Red Hat fournit plusieurs catalogues d'opérateurs qui sont inclus avec OpenShift Dedicated par défaut.



IMPORTANT

À partir d'OpenShift Dedicated 4.11, le catalogue de l'opérateur par défaut Red Hat est publié dans le format de catalogue basé sur des fichiers. Les catalogues d'opérateurs Red Hat fournis par défaut pour OpenShift Dedicated 4.6 à 4.10 publiés dans le format de base de données SQLite obsolète.

Les sous-commandes opm, les drapeaux et les fonctionnalités liés au format de base de données SQLite sont également obsolètes et seront supprimés dans une version ultérieure. Les fonctionnalités sont toujours prises en charge et doivent être utilisées pour les catalogues utilisant le format de base de données SQLite obsolète.

La plupart des sous-commandes et des drapeaux opm pour travailler avec le format de base de données SQLite, tels que le prune de l'index opm, ne fonctionnent pas avec le format de catalogue basé sur des fichiers. Consultez Gérer les catalogues personnalisés et le format d'emballage du cadre d'exploitation pour plus d'informations sur le travail avec les catalogues basés sur les fichiers.

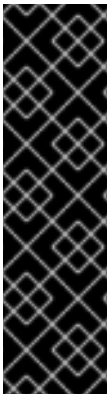
2.6.1. À propos des catalogues d'opérateurs

Le catalogue d'opérateurs est un référentiel de métadonnées que Operator Lifecycle Manager (OLM) peut interroger pour découvrir et installer des opérateurs et leurs dépendances sur un cluster. L'OMM installe toujours des opérateurs à partir de la dernière version d'un catalogue.

L'image d'index, basée sur le format du paquet Opérateur, est un instantané conteneurisé d'un catalogue. Il s'agit d'un artefact immuable qui contient la base de données des pointeurs vers un ensemble de contenus manifestes de l'opérateur. Le catalogue peut référencer une image d'index pour trouver son contenu pour OLM sur le cluster.

Au fur et à mesure que les catalogues sont mis à jour, les dernières versions des Opérateurs changent et les anciennes versions peuvent être supprimées ou modifiées. En outre, lorsque OLM s'exécute sur un cluster dédié OpenShift dans un environnement réseau restreint, il est incapable d'accéder aux catalogues directement à partir d'Internet pour tirer les derniers contenus.

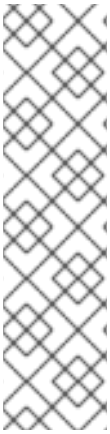
En tant qu'administrateur de cluster, vous pouvez créer votre propre image d'index personnalisée, soit sur la base d'un catalogue fourni par Red Hat, soit à partir de zéro, qui peut être utilisée pour approvisionner le contenu du catalogue sur le cluster. La création et la mise à jour de votre propre image d'index fournit une méthode pour personnaliser l'ensemble des opérateurs disponibles sur le cluster, tout en évitant les problèmes d'environnement réseau restreint susmentionnés.



IMPORTANT

Kubernetes déprécie périodiquement certaines API qui sont supprimées dans les versions ultérieures. En conséquence, les opérateurs ne peuvent pas utiliser les API supprimées à partir de la version d'OpenShift Dedicated qui utilise la version Kubernetes qui a supprimé l'API.

Lorsque votre cluster utilise des catalogues personnalisés, consultez la compatibilité de Controlling Operator avec les versions dédiées d'OpenShift pour plus de détails sur la façon dont les auteurs de l'opérateur peuvent mettre à jour leurs projets afin d'éviter les problèmes de charge de travail et d'éviter les mises à niveau incompatibles.



NOTE

La prise en charge du format de manifeste du paquet hérité pour les opérateurs, y compris les catalogues personnalisés qui utilisaient le format hérité, est supprimée dans OpenShift Dedicated 4.8 et ultérieure.

Lors de la création d'images de catalogue personnalisées, les versions précédentes d'OpenShift Dedicated 4 requises à l'aide de la commande oc adm catalog build, qui a été obsolète pour plusieurs versions et qui est maintenant supprimée. Avec la disponibilité des images d'index fournies par Red Hat à partir de OpenShift Dedicated 4.6, les constructeurs de catalogues doivent utiliser la commande d'index opm pour gérer les images d'index.

Ressources supplémentaires

- Gestion des catalogues personnalisés
- Format d'emballage

2.6.2. À propos des catalogues d'opérateurs Red Hat

Les sources de catalogue fournies par Red Hat sont installées par défaut dans l'espace de noms openshift-marketplace, ce qui rend les catalogues disponibles à l'échelle du cluster dans tous les espaces de noms.

Les catalogues d'opérateurs suivants sont distribués par Red Hat:

Catalogue	Image de l'index	Description
les opérateurs RedHat	**Registry.redhat.io/redhat/redhat-operator-index:v4**	Les produits Red Hat emballés et expédiés par Red Hat. Avec le soutien de Red Hat.
des opérateurs certifiés	**Registry.redhat.io/redhat/certified-operator-index:v4**	Les produits des principaux fournisseurs de logiciels indépendants (ISV). Le Red Hat s'associe à des ISV pour emballer et expédier. Avec le soutien de l'ISV.
lieu de marché RedHat	**Registry.redhat.io/redhat/redhat-marketplace-index:v4**	Logiciel certifié qui peut être acheté chez Red Hat Marketplace.

Catalogue	Image de l'index	Description
les opérateurs communautaires	**Registry.redhat.io/redhat/community-operator-index:v4**	Logiciel géré par les représentants compétents dans le référentiel GitHub de redhat-openshift-ecosystem/community-operators-prod/operators. Aucun soutien officiel.

Lors d'une mise à niveau de cluster, la balise d'image d'index pour les sources de catalogue par défaut Red Hat est mise à jour automatiquement par l'opérateur de versions de cluster (CVO) de sorte que le gestionnaire de cycle de vie de l'opérateur (OLM) tire la version mise à jour du catalogue. Lors d'une mise à jour de OpenShift Dedicated 4.8 à 4.9, le champ spec.image dans l'objet CatalogSource pour le catalogue redhat-operators est mis à jour à partir de:

```
registry.redhat.io/redhat/redhat-operator-index:v4.8
```

à:

```
registry.redhat.io/redhat/redhat-operator-index:v4.9
```

2.7. OPÉRATEURS EN CLUSTERS MULTILOCATAIRES

Le comportement par défaut pour Operator Lifecycle Manager (OLM) vise à fournir une simplicité lors de l'installation de l'opérateur. Cependant, ce comportement peut manquer de flexibilité, en particulier dans les clusters multilocataires. Afin que plusieurs locataires sur un cluster dédié OpenShift utilisent un opérateur, le comportement par défaut de OLM exige que les administrateurs installent l'opérateur en mode Tous les espaces de noms, ce qui peut être considéré comme violant le principe du moindre privilège.

Considérez les scénarios suivants pour déterminer quel flux de travail d'installation de l'opérateur fonctionne le mieux pour votre environnement et vos exigences.

Ressources supplémentaires

- Conditions communes: Multitenant
- Limites pour la gestion des opérateurs multilocataires

2.7.1. L'opérateur par défaut installe les modes et le comportement

Lors de l'installation d'opérateurs avec la console Web en tant qu'administrateur, vous avez généralement deux choix pour le mode d'installation, en fonction des capacités de l'opérateur:

Espace de noms unique

Installe l'Opérateur dans l'espace de noms unique choisi et met à disposition toutes les autorisations que l'Opérateur demande dans cet espace de noms.

L'ensemble des espaces de noms

Installe l'opérateur dans l'espace de noms openshift-operators par défaut pour regarder et être mis à la disposition de tous les espaces de noms du cluster. Fournit toutes les autorisations demandées par l'Opérateur dans tous les espaces de noms. Dans certains cas, un auteur de l'opérateur peut définir des métadonnées pour donner à l'utilisateur une deuxième option pour l'espace de noms suggéré par cet opérateur.

Ce choix signifie également que les utilisateurs des espaces de noms concernés ont accès aux API Opérateurs, qui peuvent tirer parti des ressources personnalisées (CR) qu'ils possèdent, en fonction de leur rôle dans l'espace de noms:

- Les rôles namespace-admin et namespace-edit peuvent lire/écrire dans les API de l'opérateur, ce qui signifie qu'ils peuvent les utiliser.
- Le rôle d'affichage de l'espace de noms peut lire les objets CR de cet opérateur.

En mode espace de noms unique, parce que l'opérateur lui-même installe dans l'espace de noms choisi, son compte pod et service y sont également situés. Dans tous les modes d'espaces de noms, les privilèges de l'opérateur sont tous automatiquement élevés à des rôles de cluster, ce qui signifie que l'opérateur a ces autorisations dans tous les espaces de noms.

Ressources supplémentaires

- Ajout d'opérateurs à un cluster
- Installer les types de modes
- Définition d'un espace de noms suggéré

2.7.2. La solution recommandée pour les clusters multilocataires

Bien qu'un mode d'installation Multinamespace existe, il est pris en charge par très peu d'opérateurs. En tant que solution intermédiaire entre tous les espaces de noms standard et les modes d'installation de l'espace de noms unique, vous pouvez installer plusieurs instances du même opérateur, une pour chaque locataire, en utilisant le flux de travail suivant:

1. Créez un espace de noms pour l'opérateur locataire qui est séparé de l'espace de noms du locataire. C'est ce que vous pouvez faire en créant un projet.
2. Créer un groupe d'opérateurs pour le locataire Opérateur étendu uniquement à l'espace de noms du locataire.
3. Installez l'opérateur dans l'espace de noms de l'opérateur locataire.

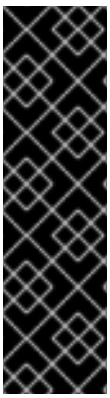
En conséquence, l'Opérateur réside dans l'espace de noms de l'opérateur locataire et surveille l'espace de noms du locataire, mais ni la pod de l'Opérateur ni son compte de service ne sont visibles ou utilisables par le locataire.

Cette solution offre une meilleure séparation des locataires, moins un principe de privilège au coût de l'utilisation des ressources, et une orchestration supplémentaire pour s'assurer que les contraintes sont respectées. Dans le cas d'une procédure détaillée, voir « Préparation de multiples instances d'un opérateur pour les clusters multilocataires ».

Limites et considérations

Cette solution ne fonctionne que lorsque les contraintes suivantes sont remplies:

- Chaque instance d'un même opérateur doit être la même version.
- L'opérateur ne peut pas avoir de dépendances à l'égard d'autres opérateurs.
- L'opérateur ne peut pas expédier un webhook de conversion CRD.



IMPORTANT

Il est impossible d'utiliser différentes versions du même opérateur sur le même cluster. Finalement, l'installation d'une autre instance de l'opérateur serait bloquée lorsqu'elle remplit les conditions suivantes:

- L'instance n'est pas la version la plus récente de l'opérateur.
- L'instance expédie une révision plus ancienne des CRD qui manquent d'informations ou de versions que les révisions les plus récentes ont déjà utilisées sur le cluster.

Ressources supplémentaires

- La préparation de plusieurs instances d'un opérateur pour les clusters multilocataires

2.7.3. Colocation d'opérateurs et groupes d'opérateurs

Le gestionnaire de cycle de vie de l'opérateur (OLM) gère les opérateurs gérés par OLM qui sont installés dans le même espace de noms, ce qui signifie que leurs ressources d'abonnement sont situées dans le même espace de noms, que les opérateurs associés. Bien qu'ils ne soient pas réellement liés, OLM considère leurs états, tels que leur version et leur politique de mise à jour, lorsque l'un d'entre eux est mis à jour.

Afin d'obtenir de plus amples informations sur la colocation de l'opérateur et l'utilisation efficace des groupes d'opérateurs, consultez Operator Lifecycle Manager (OLM) → Multitenancy and Operator colocation.

2.8. CRDS

2.8.1. Gestion des ressources à partir de définitions de ressources personnalisées

Ce guide décrit comment les développeurs peuvent gérer les ressources personnalisées (CR) qui proviennent de définitions de ressources personnalisées (CRD).

2.8.1.1. Définitions de ressources personnalisées

Dans l'API Kubernetes, une ressource est un point de terminaison qui stocke une collection d'objets API d'un certain type. À titre d'exemple, la ressource Pods intégrée contient une collection d'objets Pod.

L'objet de définition de ressource personnalisée (CRD) définit un nouveau type d'objet unique, appelé type, dans le cluster et permet au serveur API Kubernetes de gérer tout son cycle de vie.

Les objets de ressources personnalisées (CR) sont créés à partir de CRD qui ont été ajoutés au cluster par un administrateur de cluster, permettant à tous les utilisateurs de cluster d'ajouter le nouveau type de ressource dans des projets.

Les opérateurs utilisent notamment les CRD en les emballant avec toute politique RBAC requise et d'autres logiques spécifiques au logiciel.

2.8.1.2. Créer des ressources personnalisées à partir d'un fichier

Après qu'une définition de ressource personnalisée (CRD) a été ajoutée au cluster, des ressources personnalisées (CRs) peuvent être créées avec le CLI à partir d'un fichier en utilisant la spécification CR.

Procédure

1. Créez un fichier YAML pour le CR. Dans la définition d'exemple suivante, les champs personnalisés cronSpec et image sont définis dans un CR de Kind: CronTab. Le genre provient du champ spec.kind de l'objet CRD:

   Exemple de fichier YAML pour un CR

   ```
   apiVersion: "stable.example.com/v1" 1
   kind: CronTab 2
   metadata:
     name: my-new-cron-object 3
     finalizers: 4
     - finalizer.stable.example.com
   spec: 5
     cronSpec: "* * * * /5"
     image: my-awesome-cron-image
   ```

   1. Indiquez le nom du groupe et la version API (nom/version) à partir du CRD.
   2. Indiquez le type dans le CRD.
   3. Indiquez un nom pour l'objet.
   4. Indiquez les finalisateurs de l'objet, le cas échéant. Les finalisateurs permettent aux contrôleurs de mettre en œuvre des conditions qui doivent être remplies avant que l'objet puisse être supprimé.
   5. Indiquez des conditions spécifiques au type d'objet.

2. Après avoir créé le fichier, créez l'objet:

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

2.8.1.3. Inspecter les ressources personnalisées

Il est possible d'inspecter des objets de ressource personnalisée (CR) qui existent dans votre cluster à l'aide du CLI.

Conditions préalables

- Il existe un objet CR dans un espace de noms auquel vous avez accès.

Procédure

1. Afin d'obtenir des informations sur un type spécifique de CR, exécutez:

```
$ oc get <kind>
```

À titre d'exemple:

```
$ oc get crontab
```

Exemple de sortie

```
NAME                 KIND
my-new-cron-object   CronTab.v1.stable.example.com
```

Les noms de ressource ne sont pas sensibles à la casse, et vous pouvez utiliser les formes singulières ou plurielles définies dans le CRD, ainsi que tout nom court. À titre d'exemple:

```
$ oc get crontabs
```

```
$ oc get crontab
```

```
$ oc get ct
```

2. Il est également possible de visualiser les données brutes YAML pour un CR:

```
$ oc get <kind> -o yaml
```

À titre d'exemple:

```
$ oc get ct -o yaml
```

Exemple de sortie

```
apiVersion: v1
items:
- apiVersion: stable.example.com/v1
  kind: CronTab
  metadata:
    clusterName: ""
    creationTimestamp: 2017-05-31T12:56:35Z
    deletionGracePeriodSeconds: null
    deletionTimestamp: null
    name: my-new-cron-object
    namespace: default
    resourceVersion: "285"
    selfLink: /apis/stable.example.com/v1/namespaces/default/crontabs/my-new-cron-object
    uid: 9423255b-4600-11e7-af6a-28d2447dc82b
  spec:
    cronSpec: '* * * * /5' 1
    image: my-awesome-cron-image 2
```

1 2 Données personnalisées à partir du YAML que vous avez utilisé pour créer l'objet s'affiche.

CHAPITRE 3. LES TÂCHES DE L'UTILISATEUR

3.1. CRÉATION D'APPLICATIONS À PARTIR D'OPÉRATEURS INSTALLÉS

Ce guide guide les développeurs à travers un exemple de création d'applications à partir d'un opérateur installé à l'aide de la console Web dédiée OpenShift.

3.1.1. Création d'un cluster etcd à l'aide d'un opérateur

Cette procédure passe par la création d'un nouveau cluster etcd à l'aide de l'opérateur etcd, géré par Operator Lifecycle Manager (OLM).

Conditions préalables

- Accès à un cluster dédié OpenShift.
- L'opérateur etcd déjà installé à l'échelle du cluster par un administrateur.

Procédure

1. Créez un nouveau projet dans la console Web dédiée OpenShift pour cette procédure. Cet exemple utilise un projet appelé my-etcd.
2. Accédez à la page Opérateurs installés → Opérateurs installés. Les opérateurs qui ont été installés sur le cluster par l'administrateur dédié et qui sont disponibles pour utilisation sont présentés ici sous la forme d'une liste de versions de services de cluster (CSV). Les CSV sont utilisés pour lancer et gérer le logiciel fourni par l'opérateur.

   ASTUCE

   Cette liste peut être obtenue à partir du CLI en utilisant:

   ```
   $ oc get csv
   ```

3. Dans la page Opérateurs installés, cliquez sur l'opérateur etcd pour voir plus de détails et les actions disponibles.
   Comme indiqué dans les API fournies, cet opérateur met à disposition trois nouveaux types de ressources, dont un pour un cluster etcd (la ressource EtcdCluster). Ces objets fonctionnent comme les Kubernetes natifs intégrés, tels que Déploiement ou ReplicaSet, mais contiennent une logique spécifique à la gestion etcd.
4. Créer un nouveau cluster etcd:
   a. Dans la zone API de cluster etcd, cliquez sur Créer une instance.
   b. La page suivante vous permet d'apporter des modifications au modèle de démarrage minimal d'un objet EtcdCluster, comme la taille du cluster. Cliquez pour l'instant sur Créer pour finaliser. Cela déclenche l'opérateur pour démarrer les pods, services et autres composants du nouveau cluster etcd.

5. Cliquez sur le cluster exemple etcd, puis cliquez sur l'onglet Ressources pour voir que votre projet contient maintenant un certain nombre de ressources créées et configurées automatiquement par l'opérateur.
   Assurez-vous qu'un service Kubernetes a été créé qui vous permet d'accéder à la base de données à partir d'autres pods de votre projet.

6. L'ensemble des utilisateurs ayant le rôle d'édition dans un projet donné peuvent créer, gérer et supprimer des instances d'application (un cluster etcd, dans cet exemple) gérées par des opérateurs qui ont déjà été créés dans le projet, de manière en libre-service, tout comme un service cloud. Lorsque vous souhaitez activer d'autres utilisateurs avec cette capacité, les administrateurs de projet peuvent ajouter le rôle à l'aide de la commande suivante:

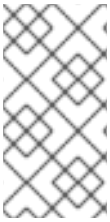
   ```
   $ oc policy add-role-to-user edit <user> -n <target_project>
   ```

Il y a maintenant un cluster etcd qui va réagir aux défaillances et rééquilibrer les données à mesure que les pods deviennent malsains ou sont migrés entre les nœuds du cluster. Le plus important, les administrateurs dédiés ou les développeurs avec un accès approprié peuvent désormais facilement utiliser la base de données avec leurs applications.

CHAPITRE 4. LES TÂCHES D'ADMINISTRATEUR

4.1. AJOUT D'OPÉRATEURS À UN CLUSTER

En utilisant le gestionnaire de cycle de vie de l'opérateur (OLM), les administrateurs ayant le rôle d'administrateur dédié peuvent installer des opérateurs basés sur OLM dans un cluster dédié OpenShift.



NOTE

Des informations sur la façon dont OLM gère les mises à jour pour les opérateurs installés situés dans le même espace de noms, ainsi qu'une méthode alternative pour installer des Opérateurs avec des groupes d'opérateurs mondiaux personnalisés, voir Multitenancy and Operator colocation.

4.1.1. Installation de l'opérateur avec OperatorHub

OperatorHub est une interface utilisateur pour découvrir les Opérateurs ; elle fonctionne en collaboration avec Operator Lifecycle Manager (OLM), qui installe et gère les Opérateurs sur un cluster.

En tant qu'administrateur dédié, vous pouvez installer un opérateur depuis OperatorHub en utilisant la console Web OpenShift dédiée ou CLI. L'abonnement d'un opérateur à un ou plusieurs espaces de noms met l'opérateur à la disposition des développeurs de votre cluster.

Lors de l'installation, vous devez déterminer les paramètres initiaux suivants pour l'opérateur:

Le mode d'installation

Choisissez Tous les espaces de noms du cluster (par défaut) pour que l'Opérateur soit installé sur tous les espaces de noms ou choisissez des espaces de noms individuels, le cas échéant, pour installer uniquement l'Opérateur sur les espaces de noms sélectionnés. Cet exemple choisit tous les espaces de noms... pour mettre l'opérateur à la disposition de tous les utilisateurs et projets.

Canal de mise à jour

Lorsqu'un Opérateur est disponible via plusieurs canaux, vous pouvez choisir le canal auquel vous souhaitez vous abonner. À titre d'exemple, pour déployer à partir du canal stable, s'il est disponible, sélectionnez-le dans la liste.

La stratégie d'approbation

Choisissez des mises à jour automatiques ou manuelles.
Lorsque vous choisissez des mises à jour automatiques pour un opérateur installé, lorsqu'une nouvelle version de cet opérateur est disponible dans le canal sélectionné, Operator Lifecycle Manager (OLM) met automatiquement à niveau l'instance en cours d'exécution de votre opérateur sans intervention humaine.

Lorsque vous sélectionnez des mises à jour manuelles, lorsqu'une version plus récente d'un opérateur est disponible, OLM crée une demande de mise à jour. En tant qu'administrateur dédié, vous devez ensuite approuver manuellement cette demande de mise à jour pour que l'opérateur soit mis à jour vers la nouvelle version.

Ressources supplémentaires

- Comprendre l'opérateurHub

4.1.2. Installation depuis OperatorHub à l'aide de la console Web

Il est possible d'installer et de s'abonner à un opérateur depuis OperatorHub à l'aide de la console Web dédiée OpenShift.

Conditions préalables

- Accès à un cluster dédié OpenShift à l'aide d'un compte avec le rôle d'administrateur dédié.

Procédure

1. Accédez à la console Web vers la page Opérateurs → OperatorHub.
2. Faites défiler ou tapez un mot clé dans la zone Filtrer par mot-clé pour trouver l'opérateur que vous souhaitez. À titre d'exemple, tapez avancé pour trouver la gestion avancée des clusters pour Kubernetes Operator.
   Il est également possible de filtrer les options par Infrastructure Features. À titre d'exemple, sélectionnez Déconnecté si vous souhaitez voir les opérateurs qui fonctionnent dans des environnements déconnectés, également connus sous le nom d'environnements réseau restreints.
3. Choisissez l'opérateur pour afficher des informations supplémentaires.

NOTE

Choisir un opérateur communautaire avertit que Red Hat ne certifie pas les opérateurs communautaires; vous devez en tenir compte avant de continuer.

4. Lisez les informations sur l'opérateur et cliquez sur Installer.
5. Dans la page Installer l'opérateur, configurez l'installation de votre opérateur:
   a. Dans le cas où vous souhaitez installer une version spécifique d'un opérateur, sélectionnez un canal de mise à jour et une version dans les listes. Il est possible de parcourir les différentes versions d'un opérateur sur tous les canaux qu'il peut avoir, d'afficher les métadonnées de ce canal et de cette version, et de sélectionner la version exacte que vous souhaitez installer.

NOTE

La sélection de la version par défaut à la dernière version pour le canal sélectionné. Lorsque la dernière version du canal est sélectionnée, la stratégie d'approbation automatique est activée par défaut. Dans le cas contraire, l'approbation manuelle est requise lorsque vous n'installez pas la dernière version du canal sélectionné.

L'installation d'un opérateur avec l'approbation manuelle fait que tous les opérateurs installés dans l'espace de noms fonctionnent avec la stratégie d'approbation manuelle et tous les opérateurs sont mis à jour ensemble. Dans le cas où vous souhaitez mettre à jour les Opérateurs de manière indépendante, installez les Opérateurs dans des espaces de noms distincts.

   b. Confirmez le mode d'installation de l'opérateur:
      - L'ensemble des espaces de noms du cluster (par défaut) installe l'opérateur dans l'espace de noms openshift-operators par défaut pour regarder et être mis à la disposition de tous les espaces de noms du cluster. Cette option n'est pas toujours

disponible.

- L'espace de noms spécifique sur le cluster vous permet de choisir un espace de noms unique spécifique dans lequel installer l'opérateur. L'opérateur ne regardera et sera mis à disposition que dans cet espace de noms unique.

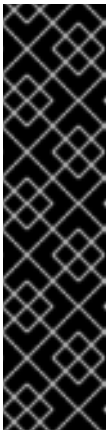
c. Dans le cas des clusters sur les fournisseurs de cloud avec authentification de jetons activé:

- Dans le cas où le cluster utilise AWS Security Token Service (mode STS dans la console Web), entrez le nom de ressource Amazon (ARN) du rôle AWS IAM de votre compte de service dans le champ ARN des rôles. Afin de créer l'ARN du rôle, suivez la procédure décrite dans Préparer le compte AWS.

- Lorsque le cluster utilise Microsoft Entra Workload ID (Workload Identity / Federated Identity Mode dans la console Web), ajoutez l'ID client, l'ID du locataire et l'ID d'abonnement dans les champs appropriés.

- Lorsque le cluster utilise Google Cloud Platform Workload Identity (GCP Workload Identity / Federated Identity) dans la console Web, ajoutez le numéro de projet, l'ID de pool, l'ID du fournisseur et l'e-mail de compte de service dans les champs appropriés.

d. Dans le cas de l'approbation de mise à jour, sélectionnez soit la stratégie d'approbation automatique ou la stratégie d'approbation manuelle.



IMPORTANT

Lorsque la console Web montre que le cluster utilise AWS STS, Microsoft Entra Workload ID ou GCP Workload Identity, vous devez définir l'approbation de mise à jour sur Manuel.

Les abonnements avec approbation automatique pour les mises à jour ne sont pas recommandés car il peut y avoir des modifications d'autorisation à apporter avant la mise à jour. Les abonnements avec approbation manuelle pour les mises à jour garantissent que les administrateurs ont la possibilité de vérifier les autorisations de la version ultérieure, de prendre toutes les mesures nécessaires, puis de mettre à jour.

6. Cliquez sur Installer pour mettre l'opérateur à la disposition des espaces de noms sélectionnés sur ce cluster dédié OpenShift:

a. Lorsque vous avez sélectionné une stratégie d'approbation manuelle, l'état de mise à niveau de l'abonnement reste mis à jour jusqu'à ce que vous révisiez et approuvez le plan d'installation.
Après avoir approuvé sur la page Plan d'installation, l'état de mise à jour de l'abonnement passe à jour.

b. Lorsque vous avez sélectionné une stratégie d'approbation automatique, l'état de mise à jour doit être résolu à jour sans intervention.

La vérification

- Après que l'état de mise à jour de l'abonnement soit mis à jour, sélectionnez Opérateurs installés → Opérateurs installés pour vérifier que la version de service de cluster (CSV) de l'opérateur installé s'affiche finalement. Le Statut devrait éventuellement résoudre à Succeed dans l'espace de noms pertinent.

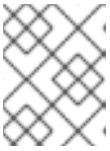
NOTE

Dans le mode d'installation de Tous les espaces de noms, l'état se résout à Succeed dans l'espace de noms openshift-operators, mais l'état est copié si vous vérifiez d'autres espaces de noms.

Dans le cas contraire:

- Consultez les journaux de tous les pods du projet openshift-operators (ou d'un autre espace de noms pertinent si un espace de noms spécifique... mode d'installation a été sélectionné) sur la page Charges de travail → Pods qui signalent des problèmes à résoudre plus loin.

- Lorsque l'opérateur est installé, les métadonnées indiquent quel canal et quelle version sont installées.



NOTE

Les menus déroulants Channel et Version sont toujours disponibles pour visualiser d'autres métadonnées de version dans ce contexte de catalogue.

Ressources supplémentaires

- Approbation manuelle d'une mise à jour de l'opérateur en attente

4.1.3. Installation depuis OperatorHub en utilisant le CLI

Au lieu d'utiliser la console Web dédiée OpenShift, vous pouvez installer un opérateur depuis OperatorHub en utilisant le CLI. La commande oc permet de créer ou de mettre à jour un objet d'abonnement.

Dans le cas du mode d'installation de SingleNamespace, vous devez également vous assurer qu'un groupe opérateur approprié existe dans l'espace de noms associé. Le groupe Opérateur, défini par un objet OperatorGroup, sélectionne les espaces de noms cibles dans lesquels générer l'accès RBAC requis pour tous les Opérateurs dans le même espace de noms que le groupe Opérateurs.

ASTUCE

Dans la plupart des cas, la méthode de la console web de cette procédure est préférée car elle automatise les tâches en arrière-plan, telles que la gestion automatique de la création d'objets OperatorGroup et Abonnement lors du choix du mode SingleNamespace.

Conditions préalables

- Accès à un cluster dédié OpenShift à l'aide d'un compte avec le rôle d'administrateur dédié.
- L'OpenShift CLI (oc) a été installé.

Procédure

1. Consultez la liste des opérateurs disponibles pour le cluster de OperatorHub:

```
$ oc get packagemanifests -n openshift-marketplace
```

Exemple 4.1. Exemple de sortie

```
NAME                              CATALOG               AGE
3scale-operator                   Red Hat Operators     91m
advanced-cluster-management       Red Hat Operators     91m
amq7-cert-manager                 Red Hat Operators     91m
# ...
couchbase-enterprise-certified    Certified Operators   91m
crunchy-postgres-operator         Certified Operators   91m
mongodb-enterprise                Certified Operators   91m
# ...
etcd                              Community Operators   91m
jaeger                            Community Operators   91m
kubefed                           Community Operators   91m
# ...
```

Indiquez le catalogue de votre opérateur souhaité.

2. Inspectez l'opérateur souhaité pour vérifier les modes d'installation pris en charge et les canaux disponibles:

```
$ oc describe packagemanifests <operator_name> -n openshift-marketplace
```

Exemple 4.2. Exemple de sortie

```
# ...
Kind:         PackageManifest
# ...
      Install Modes: (1)
        Supported:  true
        Type:       OwnNamespace
        Supported:  true
        Type:       SingleNamespace
        Supported:  false
        Type:       MultiNamespace
        Supported:  true
        Type:       AllNamespaces
# ...
    Entries:
      Name:       example-operator.v3.7.11
      Version:    3.7.11
      Name:       example-operator.v3.7.10
      Version:    3.7.10
    Name:         stable-3.7 (2)
# ...
   Entries:
      Name:       example-operator.v3.8.5
      Version:    3.8.5
      Name:       example-operator.v3.8.4
      Version:    3.8.4
    Name:         stable-3.8 (3)
  Default Channel:  stable-3.8 (4)
```

1 Indique quels modes d'installation sont pris en charge.

2 3 Exemple de noms de canaux.

4 Le canal sélectionné par défaut si un canal n'est pas spécifié.

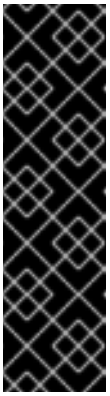
ASTUCE

Il est possible d'imprimer la version d'un opérateur et de canaliser les informations au format YAML en exécutant la commande suivante:

```
$ oc get packagemanifests <operator_name> -n <catalog_namespace> -o yaml
```

- Lorsque plus d'un catalogue est installé dans un espace de noms, exécutez la commande suivante pour rechercher les versions et canaux disponibles d'un opérateur à partir d'un catalogue spécifique:

```
$ oc get packagemanifest \
  --selector=catalog=<catalogsource_name> \
  --field-selector metadata.name=<operator_name> \
  -n <catalog_namespace> -o yaml
```

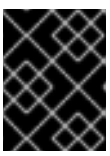


IMPORTANT

Dans le cas où vous ne spécifiez pas le catalogue de l'opérateur, l'exécution de l'oc get packagemanifest et la description des commandes de packagemanifest peuvent renvoyer un paquet à partir d'un catalogue inattendu si les conditions suivantes sont remplies:

- Des catalogues multiples sont installés dans le même espace de noms.
- Les catalogues contiennent les mêmes Opérateurs ou Opérateurs avec le même nom.

3. Lorsque l'opérateur que vous avez l'intention d'installer prend en charge le mode d'installation d'AllNamespaces, et que vous choisissez d'utiliser ce mode, passez cette étape, car l'espace de noms openshift-operators dispose déjà d'un groupe d'opérateurs approprié en place par défaut, appelé global-operators.
   Lorsque l'opérateur que vous avez l'intention d'installer prend en charge le mode d'installation de SingleNamespace et que vous choisissez d'utiliser ce mode, vous devez vous assurer qu'un groupe d'opérateurs approprié existe dans l'espace de noms correspondant. Dans le cas contraire, vous pouvez en créer une en suivant les étapes suivantes:



IMPORTANT

Il n'y a qu'un seul groupe d'opérateurs par espace de noms. Consultez « Groupes d'opérateurs » pour plus d'informations.

   a. Créer un fichier OperatorGroup objet YAML, par exemple operatorgroup.yaml, pour le mode d'installation SingleNamespace:

Exemple d'objet OperatorGroup pour le mode d'installation de SingleNamespace

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace> 1
spec:
  targetNamespaces:
  - <namespace> 2
```

1 2 Dans le mode d'installation de SingleNamespace, utilisez la même valeur <namespace> pour les champs métadonnées.namespace et spec.targetNamespaces.

b. Créer l'objet OperatorGroup:

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

4. Créer un objet d'abonnement pour abonner un espace de noms à un opérateur:

a. Créer un fichier YAML pour l'objet Abonnement, par exemple subscription.yaml:

NOTE

Lorsque vous souhaitez vous abonner à une version spécifique d'un opérateur, définissez le champ StartCSV sur la version souhaitée et définissez le champ installPlanApproval sur Manuel pour empêcher l'opérateur de mettre à niveau automatiquement si une version ultérieure existe dans le catalogue. Afin de plus de détails, voir « Exemple Subscription objet avec une version de départ spécifique de l'opérateur ».

Exemple 4.3. Exemple d'objet d'abonnement

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: <subscription_name>
  namespace: <namespace_per_install_mode> 1
spec:
  channel: <channel_name> 2
  name: <operator_name> 3
  source: <catalog_name> 4
  sourceNamespace: <catalog_source_namespace> 5
  config:
    env: 6
    - name: ARGS
      value: "-v=10"
    envFrom: 7
    - secretRef:
        name: license-secret
```

```
  volumes: 8
  - name: <volume_name>
    configMap:
      name: <configmap_name>
  volumeMounts: 9
  - mountPath: <directory_name>
    name: <volume_name>
  tolerations: 10
  - operator: "Exists"
  resources: 11
    requests:
      memory: "64Mi"
      cpu: "250m"
    limits:
      memory: "128Mi"
      cpu: "500m"
  nodeSelector: 12
    foo: bar
```

1. Dans le cas de l'utilisation par défaut du mode d'installation d'AllNamespaces, spécifiez l'espace de noms openshift-operators. Alternativement, vous pouvez spécifier un espace de noms global personnalisé, si vous en avez créé un. Dans le cas de l'utilisation du mode d'installation de SingleNamespace, spécifiez l'espace de noms unique pertinent.
2. Le nom du canal auquel s'abonner.
3. Le nom de l'opérateur auquel s'abonner.
4. Le nom de la source du catalogue qui fournit l'opérateur.
5. Espace de noms de la source du catalogue. Utilisez openshift-marketplace pour les sources de catalogue OperatorHub par défaut.
6. Le paramètre env définit une liste de variables d'environnement qui doivent exister dans tous les conteneurs dans la pod créée par OLM.
7. Le paramètre envFrom définit une liste de sources pour peupler les variables d'environnement dans le conteneur.
8. Le paramètre volumes définit une liste de volumes qui doivent exister sur le pod créé par OLM.
9. Le paramètre VolumeMounts définit une liste de montages de volume qui doivent exister dans tous les conteneurs dans la pod créée par OLM. Lorsqu'un volumeMount fait référence à un volume qui n'existe pas, OLM ne parvient pas à déployer l'opérateur.
10. Le paramètre de tolérance définit une liste de tolérances pour le pod créé par OLM.
11. Le paramètre des ressources définit les contraintes de ressources pour tous les conteneurs dans la pod créée par OLM.
12. Le paramètre nodeSelector définit un NodeSelector pour le pod créé par OLM.

Exemple 4.4. Exemple Objet d'abonnement avec une version de départ spécifique de l'opérateur

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: example-operator
  namespace: example-operator
spec:
  channel: stable-3.7
  installPlanApproval: Manual (1)
  name: example-operator
  source: custom-operators
  sourceNamespace: openshift-marketplace
  startingCSV: example-operator.v3.7.10 (2)
```

(1) Définissez la stratégie d'approbation sur Manuel dans le cas où votre version spécifiée est remplacée par une version ultérieure dans le catalogue. Ce plan empêche une mise à niveau automatique vers une version ultérieure et nécessite une approbation manuelle avant que le CSV de démarrage puisse terminer l'installation.

(2) Définissez une version spécifique d'un opérateur CSV.

b. Dans le cas des clusters sur les fournisseurs de cloud avec authentification de jetons activés, tels que Amazon Web Services (AWS) Security Token Service (STS), Microsoft Entra Workload ID, ou Google Cloud Platform Workload Identity, configurez votre objet Abonnement en suivant ces étapes:

i. Assurez-vous que l'objet Abonnement est configuré pour les approbations manuelles de mise à jour:

Exemple 4.5. Exemple Objet d'abonnement avec approbations manuelles de mise à jour

```
kind: Subscription
# ...
spec:
  installPlanApproval: Manual (1)
```

(1) Les abonnements avec approbation automatique pour les mises à jour ne sont pas recommandés car il peut y avoir des modifications d'autorisation à apporter avant la mise à jour. Les abonnements avec approbation manuelle pour les mises à jour garantissent que les administrateurs ont la possibilité de vérifier les autorisations de la version ultérieure, de prendre toutes les mesures nécessaires, puis de mettre à jour.

ii. Inclure les champs spécifiques aux fournisseurs de cloud pertinents dans la section Configuration de l'objet Abonnement:

- Lorsque le cluster est en mode AWS STS, incluez les champs suivants:

Exemple 4.6. Exemple d'objet d'abonnement avec des variables AWS STS

```
kind: Subscription
# ...
spec:
  config:
    env:
    - name: ROLEARN
      value: "<role_arn>" (1)
```

(1) Inclure le rôle ARN détails.

- Lorsque le cluster est en mode ID de charge de travail, incluez les champs suivants:

Exemple 4.7. Exemple Objet d'abonnement avec des variables ID de charge de travail

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: CLIENTID
     value: "<client_id>" (1)
   - name: TENANTID
     value: "<tenant_id>" (2)
   - name: SUBSCRIPTIONID
     value: "<subscription_id>" (3)
```

(1) Inclure l'identifiant du client.

(2) Inclure l'identifiant du locataire.

(3) Inclure l'identifiant d'abonnement.

- Lorsque le cluster est en mode Identité de charge de travail GCP, incluez les champs suivants:

Exemple 4.8. Exemple d'objet d'abonnement avec des variables d'identité de charge de travail GCP

```
kind: Subscription
# ...
spec:
 config:
   env:
   - name: AUDIENCE
     value: "<audience_url>" (1)
   - name: SERVICE_ACCOUNT_EMAIL
     value: "<service_account_email>" (2)
```

là où:

<audience>

Créée en GCP par l'administrateur lors de la configuration de GCP Workload Identity, la valeur AUDIENCE doit être une URL préformatée dans le format suivant:

```
//iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>
```

<service_account_email>

La valeur SERVICE_ACCOUNT_EMAIL est un e-mail de compte de service GCP qui est usurpé lors de l'exploitation de l'opérateur, par exemple:

```
<service_account_name>@<project_id>.iam.gserviceaccount.com
```

c. Créez l'objet Abonnement en exécutant la commande suivante:

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

5. Lorsque vous définissez le champ installPlanApproval sur Manuel, approuver manuellement le plan d'installation en attente pour terminer l'installation de l'opérateur. En savoir plus, voir « Approuver manuellement une mise à jour de l'opérateur en attente ».

À ce stade, OLM est maintenant au courant de l'opérateur sélectionné. La version de service de cluster (CSV) pour l'opérateur devrait apparaître dans l'espace de noms cible, et les API fournies par l'opérateur devraient être disponibles pour la création.

La vérification

1. Consultez l'état de l'objet Abonnement pour votre opérateur installé en exécutant la commande suivante:

```
$ oc describe subscription <subscription_name> -n <namespace>
```

2. Lorsque vous avez créé un groupe d'opérateurs pour le mode d'installation de SingleNamespace, vérifiez l'état de l'objet OperatorGroup en exécutant la commande suivante:

```
$ oc describe operatorgroup <operatorgroup_name> -n <namespace>
```

Ressources supplémentaires

- À propos des groupes d'opérateurs
- Installation d'opérateurs globaux dans des espaces de noms personnalisés
- Approbation manuelle d'une mise à jour de l'opérateur en attente

4.1.4. La préparation de plusieurs instances d'un opérateur pour les clusters multilocataires

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez ajouter plusieurs instances d'un opérateur pour une utilisation dans des clusters multilocataires. Il s'agit d'une solution alternative à l'utilisation du mode d'installation standard Tous les espaces de noms, qui peut être considéré comme violant le principe du moindre privilège, ou le mode Multinamespace, qui n'est pas largement adopté. En savoir plus, voir « Opérations en grappes multilocataires ».

Dans la procédure suivante, le locataire est un utilisateur ou un groupe d'utilisateurs qui partagent un accès et des privilèges communs pour un ensemble de charges de travail déployées. L'opérateur locataire est l'instance d'un opérateur qui est destiné à être utilisé uniquement par ce locataire.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- Chaque instance de l'opérateur que vous souhaitez installer doit être la même version sur un cluster donné.

IMPORTANT

Afin d'obtenir de plus amples renseignements sur cette question et sur d'autres limites, voir « Opérations en grappes multilocataires ».

Procédure

1. Avant d'installer l'opérateur, créez un espace de noms pour l'opérateur locataire qui est séparé de l'espace de noms du locataire. C'est ce que vous pouvez faire en créant un projet. Ainsi, si l'espace de noms du locataire est team1, vous pouvez créer un projet team1-operator:

```
$ oc new-project team1-operator
```

2. Créer un groupe d'opérateurs pour le locataire Opérateur étendu à l'espace de noms du locataire, avec seulement une entrée d'espace de noms dans la liste spec.targetNamespaces:

   a. Définissez une ressource OperatorGroup et enregistrez le fichier YAML, par exemple team1-operatorgroup.yaml:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: team1-operatorgroup
  namespace: team1-operator
spec:
  targetNamespaces:
  - team1 1
```

   1 1 Définissez uniquement l'espace de noms du locataire dans la liste spec.targetNamespaces.

   b. Créez le groupe Opérateur en exécutant la commande suivante:

```
$ oc create -f team1-operatorgroup.yaml
```

Les prochaines étapes

- Installez l'opérateur dans l'espace de noms de l'opérateur locataire. Cette tâche est plus facile à effectuer en utilisant le OperatorHub dans la console Web au lieu du CLI; pour une procédure détaillée, voir Installing from OperatorHub à l'aide de la console Web.

NOTE

Après avoir terminé l'installation de l'Opérateur, l'Opérateur réside dans l'espace de noms de l'opérateur locataire et surveille l'espace de noms du locataire, mais ni la pod de l'Opérateur ni son compte de service ne sont visibles ou utilisables par le locataire.

Ressources supplémentaires

- Opérateurs en clusters multilocataires

4.1.5. Installation d'opérateurs globaux dans des espaces de noms personnalisés

Lors de l'installation des Opérateurs avec la console Web dédiée OpenShift, le comportement par défaut installe les Opérateurs qui prennent en charge le mode d'installation de Tous les espaces de noms dans l'espace de noms global openshift-operators par défaut. Cela peut causer des problèmes liés aux plans d'installation partagés et aux stratégies de mise à jour entre tous les opérateurs dans l'espace de noms. En savoir plus sur ces limitations, voir « Multiculturalité et colocation de l'opérateur ».

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez contourner manuellement ce comportement par défaut en créant un espace de noms global personnalisé et en utilisant cet espace de noms pour installer votre ensemble individuel ou étendu d'opérateurs et leurs dépendances.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

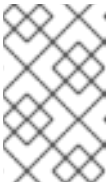
1. Avant d'installer l'opérateur, créez un espace de noms pour l'installation de votre opérateur souhaité. C'est ce que vous pouvez faire en créant un projet. L'espace de noms de ce projet deviendra l'espace de noms global personnalisé:

```
$ oc new-project global-operators
```

2. Créer un groupe d'opérateurs mondial personnalisé, qui est un groupe d'opérateurs qui surveille tous les espaces de noms:

   a. Définissez une ressource OperatorGroup et enregistrez le fichier YAML, par exemple global-operatorgroup.yaml. Évitez les champs spec.selector et spec.targetNamespaces pour en faire un groupe d'opérateur global, qui sélectionne tous les espaces de noms:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: global-operatorgroup
  namespace: global-operators
```



NOTE

Le status.namespaces d'un groupe d'opérateur mondial créé contient la chaîne vide (""), qui signale à un opérateur consommant qu'il devrait regarder tous les espaces de noms.

b. Créez le groupe Opérateur en exécutant la commande suivante:

```
$ oc create -f global-operatorgroup.yaml
```

Les prochaines étapes

- Installez l'opérateur souhaité dans votre espace de noms global personnalisé. Étant donné que la console Web ne remplit pas le menu Namespace installé pendant l'installation de l'opérateur avec des espaces de noms globaux personnalisés, cette tâche ne peut être effectuée qu'avec l'OpenShift CLI (oc). Dans le cas d'une procédure détaillée, voir Installing from OperatorHub à l'aide du CLI.

NOTE

Lorsque vous lancez l'installation de l'opérateur, si l'opérateur a des dépendances, les dépendances sont également installées automatiquement dans l'espace de noms global personnalisé. En conséquence, il est alors valable pour les opérateurs de dépendance d'avoir la même stratégie de mise à jour et des plans d'installation partagés.

Ressources supplémentaires

- Colocation multitenance et opérateur

4.1.6. Emplacement de la pod des charges de travail de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) place des pods sur des nœuds de travail arbitraires lors de l'installation d'un opérateur ou du déploiement de charges de travail Operand. En tant qu'administrateur, vous pouvez utiliser des projets avec une combinaison de sélecteurs de nœuds, de taintes et de tolérances pour contrôler le placement des Opérateurs et Operands sur des nœuds spécifiques.

Le contrôle du placement des pods des charges de travail de l'opérateur et de l'exploitation comporte les conditions préalables suivantes:

1. Déterminez un nœud ou un ensemble de nœuds à cibler pour les gousses selon vos besoins. Le cas échéant, notez une étiquette existante, telle que node-role.kubernetes.io/app, qui identifie le nœud ou les nœuds. Dans le cas contraire, ajoutez une étiquette, telle que myoperator, en utilisant un ensemble de machines de calcul ou en éditant directement le nœud. Dans une étape ultérieure, vous utiliserez cette étiquette comme sélecteur de nœud sur votre projet.

2. Lorsque vous voulez vous assurer que seuls les pods avec une certaine étiquette sont autorisés à fonctionner sur les nœuds, tout en dirigeant des charges de travail non liées à d'autres nœuds, ajoutez un taint au nœud ou aux nœuds en utilisant un ensemble de machines de calcul ou en éditant le nœud directement. Faites appel à un effet qui garantit que les nouveaux pods qui ne correspondent pas à la tainte ne peuvent pas être programmés sur les nœuds. À titre d'exemple,

une tainte myoperator:NoSchedule garantit que les nouveaux pods qui ne correspondent pas à la tainte ne sont pas programmés sur ce nœud, mais les gousses existantes sur le nœud sont autorisées à rester.

3. Créez un projet configuré avec un sélecteur de nœud par défaut et, si vous avez ajouté une tainte, une tolérance correspondante.

À ce stade, le projet que vous avez créé peut être utilisé pour orienter les pods vers les nœuds spécifiés dans les scénarios suivants:

Les pods d'opérateur

Les administrateurs peuvent créer un objet d'abonnement dans le projet comme décrit dans la section suivante. En conséquence, les pods d'opérateur sont placés sur les nœuds spécifiés.

Les gousses d'opérand

À l'aide d'un opérateur installé, les utilisateurs peuvent créer une application dans le projet, qui place la ressource personnalisée (CR) détenue par l'opérateur dans le projet. En conséquence, les pods Operand sont placés sur les nœuds spécifiés, sauf si l'Opérateur déploie des objets ou des ressources à l'échelle du cluster dans d'autres espaces de noms, auquel cas ce placement de pod personnalisé ne s'applique pas.

Ressources supplémentaires

- Création de sélecteurs de nœuds à l'échelle du projet

4.1.7. Contrôler l'endroit où un opérateur est installé

Lorsque vous installez un opérateur, OpenShift Dedicated installe par défaut la pod de l'opérateur sur l'un de vos nœuds de travail au hasard. Cependant, il peut y avoir des situations où vous voulez que ce pod soit programmé sur un nœud spécifique ou un ensemble de nœuds.

Les exemples suivants décrivent des situations où vous voudrez peut-être planifier un pod d'opérateur à un nœud spécifique ou à un ensemble de nœuds:

- Les opérateurs qui travaillent ensemble sur le même hôte ou sur des hôtes situés sur le même rack
- · si vous voulez que les opérateurs soient dispersés dans toute l'infrastructure pour éviter les temps d'arrêt en raison de problèmes de réseau ou de matériel

Il est possible de contrôler l'installation d'une pod d'opérateur en ajoutant des contraintes d'affinité des nœuds, d'affinité de pod ou de pod anti-affinité à l'objet Abonnement de l'opérateur. L'affinité des nœuds est un ensemble de règles utilisées par le planificateur pour déterminer où un pod peut être placé. L'affinité de pod vous permet de vous assurer que les gousses associées sont programmées au même nœud. La pod anti-affinité vous permet d'empêcher une gousse d'être programmée sur un nœud.

Les exemples suivants montrent comment utiliser l'affinité des nœuds ou la pod anti-affinité pour installer une instance du Custom Metrics Autoscaler Operator à un nœud spécifique dans le cluster:

Exemple d'affinité des nœuds qui place la pod de l'opérateur sur un nœud spécifique

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
```

```
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: (1)
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - ip-10-0-163-94.us-west-2.compute.internal
#...
```

(1) Affinité du nœud qui exige que la gousse de l'opérateur soit programmée sur un nœud nommé ip-10-0-163-94.us-west-2.compute.internal.

Exemple d'affinité des nœuds qui place le pod de l'opérateur sur un nœud avec une plate-forme spécifique

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      nodeAffinity: (1)
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/arch
              operator: In
              values:
              - arm64
            - key: kubernetes.io/os
              operator: In
              values:
              - linux
#...
```

(1) Affinité du nœud qui exige que la gousse de l'opérateur soit programmée sur un nœud avec les étiquettes kubernetes.io/arch=arm64 et kubernetes.io/os=linux.

Exemple d'affinité de pod qui place la pod de l'opérateur sur un ou plusieurs nœuds spécifiques

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: app
              operator: In
              values:
              - test
          topologyKey: kubernetes.io/hostname
#...
```

1 Affinité de pod qui place la gousse de l'opérateur sur un nœud qui a des pods avec l'étiquette app=test.

Exemple de pod anti-affinité qui empêche la gousse d'opérateur d'un ou de plusieurs nœuds spécifiques

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity:
      podAntiAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: cpu
              operator: In
              values:
              - high
          topologyKey: kubernetes.io/hostname
#...
```

1 La pod anti-affinité qui empêche la gousse de l'opérateur d'être programmée sur un nœud qui a des pods avec l'étiquette cpu=haute.

Procédure

Afin de contrôler le placement d'une gousse d'opérateur, remplissez les étapes suivantes:

1. Installez l'opérateur comme d'habitude.
2. Au besoin, assurez-vous que vos nœuds sont étiquetés pour répondre correctement à l'affinité.
3. Éditer l'objet d'abonnement opérateur pour ajouter une affinité:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-custom-metrics-autoscaler-operator
  namespace: openshift-keda
spec:
  name: my-package
  source: my-operators
  sourceNamespace: operator-registries
  config:
    affinity: 1
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - ip-10-0-185-229.ec2.internal
#...
```

1 Ajouter un nodeAffinity, podAffinity ou podAntiAffinity. Consultez la section Ressources supplémentaires qui suit pour obtenir de l'information sur la création de l'affinité.

La vérification

- Afin de s'assurer que le pod est déployé sur le nœud spécifique, exécutez la commande suivante:

```
$ oc get pods -o wide
```

Exemple de sortie

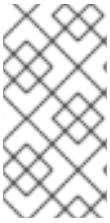
```
NAME                                  READY   STATUS    RESTARTS   AGE   IP
NODE                          NOMINATED NODE   READINESS GATES
custom-metrics-autoscaler-operator-5dcc45d656-bhshg   1/1     Running   0          50s
10.131.0.20   ip-10-0-185-229.ec2.internal   <none>           <none>
```

Ressources supplémentaires

- Comprendre l'affinité des gousses
- Comprendre l'affinité des nœuds

4.2. LA MISE À JOUR DES OPÉRATEURS INSTALLÉS

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez mettre à jour les opérateurs qui ont déjà été installés à l'aide du gestionnaire de cycle de vie de l'opérateur (OLM) sur votre cluster dédié OpenShift.



NOTE

Des informations sur la façon dont OLM gère les mises à jour pour les opérateurs installés situés dans le même espace de noms, ainsi qu'une méthode alternative pour installer des Opérateurs avec des groupes d'opérateurs mondiaux personnalisés, voir Multitenancy and Operator colocation.

4.2.1. La préparation d'une mise à jour de l'opérateur

L'abonnement d'un opérateur installé spécifie un canal de mise à jour qui suit et reçoit des mises à jour pour l'opérateur. Le canal de mise à jour peut être modifié pour commencer à suivre et recevoir des mises à jour à partir d'un canal plus récent.

Les noms des canaux de mise à jour dans un abonnement peuvent différer entre les opérateurs, mais le schéma de dénomination suit généralement une convention commune au sein d'un opérateur donné. À titre d'exemple, les noms de canaux peuvent suivre un flux de mise à jour de version mineure pour l'application fournie par l'opérateur (1.2, 1.3) ou une fréquence de libération (stable, rapide).

NOTE

Les opérateurs installés ne peuvent pas changer de canal plus ancien que le canal actuel.

Les laboratoires de portail client Red Hat incluent l'application suivante qui aide les administrateurs à se préparer à mettre à jour leurs opérateurs:

- Contrôle d'information de l'opérateur de mise à jour de la plate-forme de conteneur Red Hat OpenShift

L'application permet de rechercher des opérateurs basés sur le gestionnaire de cycle de vie de l'opérateur et de vérifier la version de l'opérateur disponible par canal de mise à jour sur différentes versions d'OpenShift Dedicated. Les opérateurs basés sur les opérateurs ne sont pas inclus.

4.2.2. Changer le canal de mise à jour pour un opérateur

Il est possible de modifier le canal de mise à jour d'un opérateur à l'aide de la console Web OpenShift Dedicated.

ASTUCE

Lorsque la stratégie d'approbation de l'abonnement est définie sur Automatique, le processus de mise à jour démarre dès qu'une nouvelle version de l'opérateur est disponible dans le canal sélectionné. Lorsque la stratégie d'approbation est définie sur Manuel, vous devez approuver manuellement les mises à jour en attente.

Conditions préalables

- Exploitant précédemment installé à l'aide du gestionnaire de cycle de vie de l'opérateur (OLM).

Procédure

1. Dans la perspective de l'administrateur de la console Web, accédez aux opérateurs → Opérateurs installés.
2. Cliquez sur le nom de l'opérateur pour lequel vous souhaitez modifier le canal de mise à jour.
3. Cliquez sur l'onglet **Subscription**.
4. Cliquez sur le nom du canal de mise à jour sous le canal Mise à jour.
5. Cliquez sur le nouveau canal de mise à jour que vous souhaitez modifier, puis cliquez sur Enregistrer.
6. Dans le cas des abonnements avec une stratégie d'approbation automatique, la mise à jour commence automatiquement. Accédez à la page Opérateurs installés → Opérateurs installés pour suivre l'avancement de la mise à jour. Lorsque vous avez terminé, le statut passe à **Succeeded** et **Up to date**.
   En ce qui concerne les abonnements avec une stratégie d'approbation manuelle, vous pouvez approuver manuellement la mise à jour à partir de l'onglet Abonnement.

4.2.3. Approbation manuelle d'une mise à jour de l'opérateur en attente

Lorsqu'un opérateur installé a la stratégie d'approbation dans son abonnement à Manuel, lorsque de nouvelles mises à jour sont publiées dans son canal de mise à jour actuel, la mise à jour doit être approuvée manuellement avant le début de l'installation.

Conditions préalables

- Exploitant précédemment installé à l'aide du gestionnaire de cycle de vie de l'opérateur (OLM).

Procédure

1. Dans la perspective de l'administrateur de la console Web dédiée OpenShift, accédez aux opérateurs → Opérateurs installés.
2. Les opérateurs qui ont une mise à jour en attente affichent un statut avec Mise à niveau disponible. Cliquez sur le nom de l'opérateur que vous souhaitez mettre à jour.
3. Cliquez sur l'onglet **Subscription**. Les mises à jour nécessitant l'approbation sont affichées à côté de l'état de mise à niveau. Par exemple, **1 requires approval** peut être affiché.
4. Cliquez sur **1 requires approval**, puis sur **Preview Install Plan**.
5. Examinez les ressources qui sont énumérées comme disponibles pour mise à jour. Lorsque vous êtes satisfait, cliquez sur Approuver.
6. Accédez à la page Opérateurs installés → Opérateurs installés pour suivre l'avancement de la mise à jour. Lorsque vous avez terminé, le statut passe à **Succeeded** et **Up to date**.

4.3. LA SUPPRESSION DES OPÉRATEURS D'UN CLUSTER

Ce qui suit décrit comment supprimer, ou désinstaller, les opérateurs qui ont déjà été installés à l'aide de Operator Lifecycle Manager (OLM) sur votre cluster OpenShift Dedicated.

IMPORTANT

Il faut désinstaller avec succès et complètement un Opérateur avant de tenter de réinstaller le même Opérateur. Le défaut de désinstaller correctement l'opérateur peut laisser des ressources, telles qu'un projet ou un espace de noms, bloquées dans un état "Terminating" et provoquer l'observation des messages de « résolution d'erreur » lors de la réinstallation de l'opérateur.

4.3.1. Suppression des opérateurs d'un cluster à l'aide de la console Web

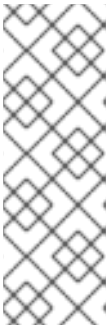
Les administrateurs de clusters peuvent supprimer les opérateurs installés d'un espace de noms sélectionné à l'aide de la console Web.

Conditions préalables

- Accès à une console Web OpenShift Dedicated cluster à l'aide d'un compte doté d'autorisations d'administration dédiées.

Procédure

1. Accédez à la page Opérateurs installés → Opérateurs installés.
2. Faites défiler ou entrez un mot clé dans le champ Filtrer par nom pour trouver l'opérateur que vous souhaitez supprimer. Ensuite, cliquez dessus.
3. À droite de la page Détails de l'opérateur, sélectionnez Désinstaller l'opérateur dans la liste Actions.
   La boîte de dialogue Un opérateur de désinstallation? s'affiche.
4. Cliquez sur Désinstaller pour supprimer les déploiements de l'opérateur, de l'opérateur et des pods. Après cette action, l'opérateur cesse de fonctionner et ne reçoit plus de mises à jour.



NOTE

Cette action ne supprime pas les ressources gérées par l'opérateur, y compris les définitions de ressources personnalisées (CRD) et les ressources personnalisées (CRs). Les tableaux de bord et les éléments de navigation activés par la console Web et les ressources hors groupe qui continuent de fonctionner peuvent nécessiter un nettoyage manuel. Afin de les supprimer après la désinstallation de l'opérateur, vous devrez peut-être supprimer manuellement les CRD de l'opérateur.

4.3.2. La suppression des opérateurs d'un cluster à l'aide du CLI

Les administrateurs de clusters peuvent supprimer les opérateurs installés d'un espace de noms sélectionné en utilisant le CLI.

Conditions préalables

- Accès à un cluster dédié OpenShift à l'aide d'un compte doté d'autorisations d'administration dédiées.
- L'OpenShift CLI (oc) est installé sur votre poste de travail.

Procédure

1. Assurez-vous que la dernière version de l'opérateur souscrit (par exemple, l'opérateur sans serveur) est identifiée dans le champ CSV actuel.

```
$ oc get subscription.operators.coreos.com serverless-operator -n openshift-serverless -o
yaml | grep currentCSV
```

Exemple de sortie

```
  currentCSV: serverless-operator.v1.28.0
```

2. Effacer l'abonnement (par exemple, l'opérateur sans serveur):

```
$ oc delete subscription.operators.coreos.com serverless-operator -n openshift-serverless
```

Exemple de sortie

```
subscription.operators.coreos.com "serverless-operator" deleted
```

3. Effacer le CSV pour l'opérateur dans l'espace de noms cible en utilisant la valeur actuelleCSV de l'étape précédente:

```
$ oc delete clusterserviceversion serverless-operator.v1.28.0 -n openshift-serverless
```

Exemple de sortie

```
clusterserviceversion.operators.coreos.com "serverless-operator.v1.28.0" deleted
```

4.3.3. Abonnements défaillants rafraîchissants

Dans Operator Lifecycle Manager (OLM), si vous vous abonnez à un opérateur qui fait référence à des images qui ne sont pas accessibles sur votre réseau, vous pouvez trouver des emplois dans l'espace de noms openshift-marketplace qui échouent avec les erreurs suivantes:

Exemple de sortie

```
ImagePullBackOff for
Back-off pulling image "example.com/openshift4/ose-elasticsearch-operator-
bundle@sha256:6d2587129c846ec28d384540322b40b05833e7e00b25cca584e004af9a1d292e"
```

Exemple de sortie

```
rpc error: code = Unknown desc = error pinging docker registry example.com: Get
"https://example.com/v2/": dial tcp: lookup example.com on 10.0.0.1:53: no such host
```

En conséquence, l'abonnement est bloqué dans cet état défaillant et l'Opérateur est incapable d'installer ou de mettre à niveau.

Il est possible d'actualiser un abonnement défaillant en supprimant l'abonnement, la version du service cluster (CSV) et d'autres objets connexes. Après avoir recréé l'abonnement, OLM réinstalle ensuite la version correcte de l'opérateur.

Conditions préalables

- Il y a un abonnement défaillant qui est incapable de tirer une image de paquet inaccessible.
- « vous avez confirmé que l'image de paquet correcte est accessible.

Procédure

1. Obtenez les noms des objets Abonnement et ClusterServiceVersion à partir de l'espace de noms où l'opérateur est installé:

```
$ oc get sub,csv -n <namespace>
```

Exemple de sortie

```
NAME                                         PACKAGE                  SOURCE             CHANNEL
subscription.operators.coreos.com/elasticsearch-operator   elasticsearch-operator   redhat-operators   5.0

NAME                                                  DISPLAY                            VERSION
REPLACES   PHASE
clusterserviceversion.operators.coreos.com/elasticsearch-operator.5.0.0-65   OpenShift Elasticsearch Operator   5.0.0-65
Succeeded
```

2. Effacer l'abonnement:

```
$ oc delete subscription <subscription_name> -n <namespace>
```

3. Effacer la version du service cluster:

```
$ oc delete csv <csv_name> -n <namespace>
```

4. Bénéficiez des noms de tous les emplois défaillants et des cartes de configuration connexes dans l'espace de noms openshift-marketplace:

```
$ oc get job,configmap -n openshift-marketplace
```

Exemple de sortie

```
NAME                                                COMPLETIONS   DURATION   AGE
job.batch/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   1/1
26s        9m30s

NAME                                                DATA   AGE
configmap/1de9443b6324e629ddf31fed0a853a121275806170e34c926d69e53a7fcbccb   3
9m30s
```

5. Effacer la tâche:

```
$ oc delete job <job_name> -n openshift-marketplace
```

Cela garantit que les pods qui tentent de tirer l'image inaccessible ne sont pas recréés.

6. Effacer la carte de configuration:

```
$ oc delete configmap <configmap_name> -n openshift-marketplace
```

7. Installez l'opérateur en utilisant OperatorHub dans la console Web.

La vérification

- Assurez-vous que l'opérateur a été réinstallé avec succès:

```
$ oc get sub,csv,installplan -n <namespace>
```

4.4. CONFIGURATION DU SUPPORT PROXY DANS OPERATOR LIFECYCLE MANAGER

Lorsqu'un proxy global est configuré sur le cluster dédié OpenShift, Operator Lifecycle Manager (OLM) configure automatiquement les opérateurs qu'il gère avec le proxy à l'échelle du cluster. Cependant, vous pouvez également configurer les opérateurs installés pour remplacer le proxy global ou injecter un certificat CA personnalisé.

Ressources supplémentaires

- Configuration d'un proxy à l'échelle du cluster
- Développer des opérateurs qui prennent en charge les paramètres proxy pour Go, Ansible et Helm

4.4.1. Dépassement des paramètres proxy d'un opérateur

En cas de configuration d'un proxy de sortie à l'échelle du cluster, les opérateurs s'exécutant avec Operator Lifecycle Manager (OLM) héritent des paramètres proxy à l'échelle du cluster sur leurs déploiements. Les administrateurs avec le rôle dédié-admin peuvent également remplacer ces paramètres proxy en configurant l'abonnement d'un opérateur.

IMPORTANT

Les opérateurs doivent gérer les variables d'environnement de réglage pour les paramètres proxy dans les pods pour tous les Operands gérés.

Conditions préalables

- Accès à un cluster OpenShift dédié en tant qu'utilisateur avec le rôle d'administrateur dédié.

Procédure

1. Accédez à la console Web vers la page Opérateurs → OperatorHub.
2. Choisissez l'opérateur et cliquez sur Installer.
3. Dans la page Installer l'opérateur, modifier l'objet Abonnement pour inclure une ou plusieurs des variables d'environnement suivantes dans la section Spécifications:
   - **HTTP_PROXY**

- **HTTPS_PROXY**
- **AUCUN_PROXY**

À titre d'exemple:

L'objet d'abonnement avec le paramètre proxy remplace

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: etcd-config-test
  namespace: openshift-operators
spec:
  config:
    env:
    - name: HTTP_PROXY
      value: test_http
    - name: HTTPS_PROXY
      value: test_https
    - name: NO_PROXY
      value: test
  channel: clusterwide-alpha
  installPlanApproval: Automatic
  name: etcd
  source: community-operators
  sourceNamespace: openshift-marketplace
  startingCSV: etcdoperator.v0.9.4-clusterwide
```

NOTE

Ces variables d'environnement peuvent également être dédéfinies à l'aide d'une valeur vide pour supprimer tous les paramètres de proxy définis précédemment à l'échelle du cluster ou sur mesure.

L'ODM gère ces variables d'environnement en tant qu'unité; si au moins une d'entre elles est définie, les trois sont considérés comme dépassés et les valeurs par défaut à l'échelle du cluster ne sont pas utilisées pour les déploiements de l'opérateur souscrit.

4. Cliquez sur Installer pour rendre l'opérateur disponible dans les espaces de noms sélectionnés.
5. Après que le CSV pour l'opérateur s'affiche dans l'espace de noms pertinent, vous pouvez vérifier que des variables d'environnement proxy personnalisées sont définies dans le déploiement. À titre d'exemple, l'utilisation du CLI:

```
$ oc get deployment -n openshift-operators \
    etcd-operator -o yaml \
    | grep -i "PROXY" -A 2
```

Exemple de sortie

```
        - name: HTTP_PROXY
          value: test_http
        - name: HTTPS_PROXY
```

```
          value: test_https
        - name: NO_PROXY
          value: test
        image: quay.io/coreos/etcd-
operator@sha256:66a37fd61a06a43969854ee6d3e21088a98b93838e284a6086b13917f96b0
d9c
...
```

4.4.2. Injection d'un certificat CA personnalisé

Lorsqu'un administrateur avec le rôle d'administrateur dédié ajoute un certificat CA personnalisé à un cluster à l'aide d'une carte de configuration, l'opérateur réseau de cluster fusionne les certificats fournis par l'utilisateur et les certificats CA système en un seul paquet. Il est possible d'injecter ce paquet fusionné dans votre opérateur fonctionnant sur Operator Lifecycle Manager (OLM), ce qui est utile si vous avez un proxy HTTPS man-in-the-middle.

Conditions préalables

- Accès à un cluster OpenShift dédié en tant qu'utilisateur avec le rôle d'administrateur dédié.
- Certificat CA personnalisé ajouté au cluster à l'aide d'une carte de configuration.
- L'opérateur désiré a installé et exécuté sur OLM.

Procédure

1. Créez une carte de configuration vide dans l'espace de noms où l'abonnement pour votre opérateur existe et incluez l'étiquette suivante:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: trusted-ca 1
  labels:
    config.openshift.io/inject-trusted-cabundle: "true" 2
```

1. Le nom de la carte de configuration.
2. Demande au Cluster Network Operator d'injecter le paquet fusionné.

   Après avoir créé cette carte de configuration, il est immédiatement rempli avec le contenu du certificat du paquet fusionné.

2. Actualisez l'objet Abonnement pour inclure une section spec.config qui monte la carte de configuration de confiance en tant que volume à chaque conteneur dans un pod qui nécessite une CA personnalisée:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: my-operator
spec:
  package: etcd
  channel: alpha
```

```
config: 1
  selector:
    matchLabels:
      <labels_for_pods> 2
  volumes: 3
  - name: trusted-ca
    configMap:
      name: trusted-ca
      items:
        - key: ca-bundle.crt 4
          path: tls-ca-bundle.pem 5
  volumeMounts: 6
  - name: trusted-ca
    mountPath: /etc/pki/ca-trust/extracted/pem
    readOnly: true
```

1. Ajoutez une section de configuration si elle n'existe pas.
2. Indiquez les étiquettes pour correspondre aux pods appartenant à l'opérateur.
3. Créez un volume de confiance.
4. ca-bundle.crt est nécessaire en tant que clé map de configuration.
5. le TLS-ca-bundle.pem est requis en tant que chemin de configuration de la carte.
6. Créez une monture de volume de confiance.

NOTE

Les déploiements d'un opérateur peuvent ne pas valider l'autorité et afficher un certificat x509 signé par erreur d'autorité inconnue. Cette erreur peut se produire même après l'injection d'un CA personnalisé lors de l'utilisation de l'abonnement d'un opérateur. Dans ce cas, vous pouvez définir le MountPath comme /etc/ssl/certs pour confiance-ca en utilisant l'abonnement d'un opérateur.

4.5. STATUT DE L'OPÉRATEUR

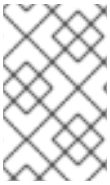
Comprendre l'état du système dans Operator Lifecycle Manager (OLM) est important pour prendre des décisions concernant et débogage des problèmes avec les opérateurs installés. L'OLM fournit un aperçu des abonnements et des sources de catalogue connexes concernant leur état et les actions effectuées. Cela aide les utilisateurs à mieux comprendre la santé de leurs opérateurs.

4.5.1. Conditions d'abonnement à l'opérateur

Les abonnements peuvent signaler les types d'état suivants:

Tableau 4.1. Conditions d'abonnement

État de l'état	Description
CatalogueSourcesUnsanté	Certaines ou toutes les sources de catalogue à utiliser en résolution sont malsaines.
InstallPlanMissing	Il manque un plan d'installation pour un abonnement.
InstallPlanPending	Le plan d'installation d'un abonnement est en attente d'installation.
InstallPlanFailed	Le plan d'installation d'un abonnement a échoué.
La RésolutionFailed	La résolution de dépendance pour un abonnement a échoué.



NOTE

Les opérateurs de cluster dédiés par défaut sont gérés par l'opérateur de versions de cluster (CVO) et ils n'ont pas d'objet d'abonnement. Les opérateurs d'applications sont gérés par Operator Lifecycle Manager (OLM) et ils ont un objet d'abonnement.

Ressources supplémentaires

- Abonnements défaillants rafraîchissants

4.5.2. Affichage du statut d'abonnement de l'opérateur en utilisant le CLI

En utilisant le CLI, vous pouvez voir l'état de l'abonnement à l'opérateur.

Conditions préalables

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

Procédure

1. Abonnements à l'opérateur de liste:

```
$ oc get subs -n <operator_namespace>
```

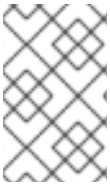
2. Consultez la commande de description d'oc pour inspecter une ressource d'abonnement:

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. Dans la sortie de commande, trouvez la section Conditions pour l'état des types de condition d'abonnement Opérateur. Dans l'exemple suivant, le type de condition de CatalogSourcesUnsanté a un statut de faux parce que toutes les sources de catalogue disponibles sont saines:

Exemple de sortie

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```



NOTE

Les opérateurs de cluster dédiés par défaut sont gérés par l'opérateur de versions de cluster (CVO) et ils n'ont pas d'objet d'abonnement. Les opérateurs d'applications sont gérés par Operator Lifecycle Manager (OLM) et ils ont un objet d'abonnement.

4.5.3. Affichage de l'état de la source du catalogue de l'opérateur en utilisant le CLI

Il est possible d'afficher l'état d'une source de catalogue de l'opérateur à l'aide du CLI.

Conditions préalables

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

Procédure

1. Énumérez les sources du catalogue dans un espace de noms. À titre d'exemple, vous pouvez vérifier l'espace de noms openshift-marketplace, qui est utilisé pour les sources de catalogue à l'échelle du cluster:

```
$ oc get catalogsources -n openshift-marketplace
```

Exemple de sortie

```
NAME                  DISPLAY               TYPE   PUBLISHER   AGE
certified-operators   Certified Operators   grpc   Red Hat     55m
community-operators   Community Operators   grpc   Red Hat     55m
example-catalog       Example Catalog       grpc   Example Org 2m25s
redhat-marketplace    Red Hat Marketplace   grpc   Red Hat     55m
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. La commande oc described permet d'obtenir plus de détails et d'état sur une source de catalogue:

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

Exemple de sortie

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
    Protocol:           grpc
    Service Name:       example-catalog
    Service Namespace:  openshift-marketplace
# ...
```

Dans l'exemple précédent, le dernier état observé est TRANSIENT_FAILURE. Cet état indique qu'il y a un problème à établir une connexion pour la source du catalogue.

3. Énumérez les pods dans l'espace de noms où votre source de catalogue a été créée:

```
$ oc get pods -n openshift-marketplace
```

Exemple de sortie

```
NAME                                  READY   STATUS             RESTARTS   AGE
certified-operators-cv9nn             1/1     Running            0          36m
community-operators-6v8lp             1/1     Running            0          36m
marketplace-operator-86bfc75f9b-jkgbc 1/1     Running            0          42m
example-catalog-bwt8z                 0/1     ImagePullBackOff   0          3m55s
redhat-marketplace-57p8c              1/1     Running            0          36m
redhat-operators-smxx8                1/1     Running            0          36m
```

Lorsqu'une source de catalogue est créée dans un espace de noms, un pod pour la source du catalogue est créé dans cet espace de noms. Dans l'exemple précédent, l'état de la pod example-catalog-bwt8z est ImagePullBackOff. Ce statut indique qu'il y a un problème à tirer l'image de l'index de la source du catalogue.

4. Consultez la commande de description d'oc pour inspecter un pod pour obtenir des informations plus détaillées:

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

Exemple de sortie

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
```

```
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age                From               Message
  ----     ------          ----               ----               -------
  Normal   Scheduled       48s                default-scheduler  Successfully assigned openshift-marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s                multus             Add eth0 [10.131.0.40/23] from openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet            Back-off pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet            Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet            Pulling image "quay.io/example-org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet            Failed to pull image "quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet            Error: ErrImagePull
```

Dans l'exemple précédent, les messages d'erreur indiquent que l'image d'index de la source du catalogue ne parvient pas à tirer avec succès en raison d'un problème d'autorisation. À titre d'exemple, l'image d'index peut être stockée dans un registre qui nécessite des identifiants de connexion.

Ressources supplémentaires

- Concepts et ressources du gestionnaire de cycle de vie de l'opérateur → Source du catalogue
- documentation du GRPC : États de connectivité

4.6. GESTION DES CONDITIONS DE L'OPÉRATEUR

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez gérer les conditions de l'opérateur en utilisant Operator Lifecycle Manager (OLM).

4.6.1. Conditions primordiales de l'opérateur

En tant qu'administrateur avec le rôle d'administrateur dédié, vous voudrez peut-être ignorer une condition d'opérateur prise en charge rapportée par un opérateur. Lorsqu'ils sont présents, les conditions de l'opérateur dans le tableau Spec.Overrides remplacent les conditions du tableau Spec.Conditions, permettant aux administrateurs dédiés de traiter les situations où un opérateur signale incorrectement un état à Operator Lifecycle Manager (OLM).

NOTE

Le tableau Spec.Overrides n'est pas présent dans un objet OperatorCondition jusqu'à ce qu'il soit ajouté par un administrateur avec le rôle dédié-admin . Le tableau Spec.Conditions n'est pas non plus présent tant qu'il n'est pas ajouté par un utilisateur ou à la suite de la logique personnalisée de l'opérateur.

À titre d'exemple, considérez une version connue d'un opérateur qui communique toujours qu'elle n'est pas modifiable. Dans ce cas, vous voudrez peut-être mettre à niveau l'opérateur malgré la

communication de l'opérateur qu'il n'est pas mis à niveau. Cela pourrait être accompli en dépassant la condition de l'opérateur en ajoutant le type et l'état de condition au tableau Spec.Overrides dans l'objet OperatorCondition.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- Exploitant avec un objet OperatorCondition, installé à l'aide de OLM.

Procédure

1. Éditer l'objet OperatorCondition pour l'Opérateur:

```
$ oc edit operatorcondition <name>
```

2. Ajouter un tableau Spec.Overrides à l'objet:

Exemple de condition de l'opérateur

```
apiVersion: operators.coreos.com/v2
kind: OperatorCondition
metadata:
  name: my-operator
  namespace: operators
spec:
  overrides:
  - type: Upgradeable (1)
    status: "True"
    reason: "upgradeIsSafe"
    message: "This is a known issue with the Operator where it always reports that it cannot
be upgraded."
  conditions:
  - type: Upgradeable
    status: "False"
    reason: "migration"
    message: "The operator is performing a migration."
    lastTransitionTime: "2020-08-24T23:15:55Z"
```

(1) Permet à l'utilisateur dédié-admin de changer la disponibilité de mise à jour vers True.

4.6.2. La mise à jour de votre opérateur pour utiliser les conditions de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) crée automatiquement une ressource OperatorCondition pour chaque ressource ClusterServiceVersion qu'elle concilie. L'ensemble des comptes de service du CSV se voient accorder au RBAC d'interagir avec la Condition de l'Opérateur détenue par l'Opérateur.

L'auteur d'un opérateur peut développer son opérateur pour utiliser la bibliothèque de l'opérateur de sorte qu'une fois que l'opérateur a été déployé par OLM, il peut définir ses propres conditions. Consultez la page Conditions de l'opérateur pour plus de ressources sur la configuration des conditions de l'opérateur en tant qu'auteur de l'opérateur.

4.6.2.1. Définir les valeurs par défaut

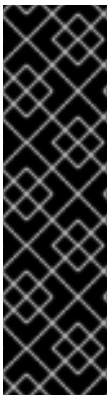
Dans un effort pour rester rétrocompatible, OLM traite l'absence d'une ressource OperatorCondition comme se désignant de la condition. En conséquence, un opérateur qui opte pour l'utilisation des conditions de l'opérateur doit définir des conditions par défaut avant que la sonde prête pour le pod ne soit définie à true. Cela fournit à l'opérateur un délai de grâce pour mettre à jour la condition à l'état correct.

4.6.3. Ressources supplémentaires

- Conditions de l'opérateur

4.7. GESTION DES CATALOGUES PERSONNALISÉS

Les administrateurs ayant le rôle d'administrateur dédié et les responsables du catalogue de l'opérateur peuvent créer et gérer des catalogues personnalisés emballés à l'aide du format bundle sur Operator Lifecycle Manager (OLM) dans OpenShift Dedicated.



IMPORTANT

Kubernetes déprécie périodiquement certaines API qui sont supprimées dans les versions ultérieures. En conséquence, les opérateurs ne peuvent pas utiliser les API supprimées à partir de la version d'OpenShift Dedicated qui utilise la version Kubernetes qui a supprimé l'API.

Lorsque votre cluster utilise des catalogues personnalisés, consultez la compatibilité de Controlling Operator avec les versions dédiées d'OpenShift pour plus de détails sur la façon dont les auteurs de l'opérateur peuvent mettre à jour leurs projets afin d'éviter les problèmes de charge de travail et d'éviter les mises à niveau incompatibles.

Ressources supplémentaires

- Catalogues d'opérateurs Red Hat

4.7.1. Conditions préalables

- C'est vous qui avez installé l'opm CLI.

4.7.2. Catalogues basés sur des fichiers

Les catalogues basés sur des fichiers sont la dernière itération du format de catalogue dans Operator Lifecycle Manager (OLM). Il s'agit d'un texte simple (JSON ou YAML) et d'une évolution de configuration déclarative du format de base de données SQLite antérieur, et il est entièrement rétrocompatible.

IMPORTANT

À partir d'OpenShift Dedicated 4.11, le catalogue de l'opérateur par défaut Red Hat est publié dans le format de catalogue basé sur des fichiers. Les catalogues d'opérateurs Red Hat fournis par défaut pour OpenShift Dedicated 4.6 à 4.10 publiés dans le format de base de données SQLite obsolète.

Les sous-commandes opm, les drapeaux et les fonctionnalités liés au format de base de données SQLite sont également obsolètes et seront supprimés dans une version ultérieure. Les fonctionnalités sont toujours prises en charge et doivent être utilisées pour les catalogues utilisant le format de base de données SQLite obsolète.

La plupart des sous-commandes et des drapeaux opm pour travailler avec le format de base de données SQLite, tels que le prune de l'index opm, ne fonctionnent pas avec le format de catalogue basé sur des fichiers. Consultez le format d'emballage Operator Framework pour plus d'informations sur le travail avec les catalogues basés sur des fichiers.

4.7.2.1. Création d'une image de catalogue basée sur des fichiers

Il est possible d'utiliser l'opm CLI pour créer une image de catalogue qui utilise le format de catalogue basé sur un fichier en texte brut (JSON ou YAML), qui remplace le format de base de données SQLite obsolète.

Conditions préalables

- C'est vous qui avez installé l'opm CLI.
- Il y a la version 1.9.3+ de podman.
- L'image groupée est construite et poussée vers un registre qui prend en charge Docker v2-2.

Procédure

1. Initialiser le catalogue:
   a. Créer un répertoire pour le catalogue en exécutant la commande suivante:

```
$ mkdir <catalog_dir>
```

   b. Générer un Dockerfile qui peut construire une image de catalogue en exécutant la commande opm générer dockerfile:

```
$ opm generate dockerfile <catalog_dir> \
    -i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4 1
```

   1 Indiquez l'image de base officielle Red Hat en utilisant le drapeau -i, sinon le Dockerfile utilise l'image par défaut en amont.

   Le Dockerfile doit être dans le même répertoire parent que le répertoire de catalogue que vous avez créé à l'étape précédente:

   Exemple de structure de répertoire

```
. 1
├── <catalog_dir> 2
└── <catalog_dir>.Dockerfile 3
```

1. Annuaire parent
2. Annuaire du catalogue
3. Dockerfile généré par l'opm générer la commande dockerfile

c. Remplissez le catalogue avec la définition du paquet pour votre opérateur en exécutant la commande opm init:

```
$ opm init <operator_name> \ 1
    --default-channel=preview \ 2
    --description=./README.md \ 3
    --icon=./operator-icon.svg \ 4
    --output yaml \ 5
    > <catalog_dir>/index.yaml 6
```

1. Exploitant, ou paquet, nom
2. Canal auquel les abonnements par défaut s'ils ne sont pas spécifiés
3. Chemin vers la documentation README.md de l'opérateur ou autre documentation
4. Chemin vers l'icône de l'opérateur
5. Format de sortie: JSON ou YAML
6. Chemin de création du fichier de configuration du catalogue

Cette commande génère un blob de configuration déclaratif olm.package dans le fichier de configuration du catalogue spécifié.

2. Ajouter un paquet au catalogue en exécutant la commande de rendu opm:

```
$ opm render <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
    --output=yaml \
    >> <catalog_dir>/index.yaml 2
```

1. Dessinez les spécifications pour l'image de paquet
2. Chemin d'accès au fichier de configuration du catalogue

NOTE

Les canaux doivent contenir au moins un paquet.

3. Ajoutez une entrée de canal pour le paquet. À titre d'exemple, modifiez l'exemple suivant à vos spécifications et ajoutez-le à votre fichier <catalog_dir>/index.yaml:

Exemple d'entrée de canal

```
---
schema: olm.channel
package: <operator_name>
name: preview
entries:
  - name: <operator_name>.v0.1.0 (1)
```

(1) Assurez-vous d'inclure la période (.) après <operator_name> mais avant le v dans la version. Dans le cas contraire, l'entrée ne parvient pas à passer la commande de validation opm.

4. De valider le catalogue basé sur les fichiers:

   a. Exécutez la commande opm valider par rapport au répertoire du catalogue:

   ```
   $ opm validate <catalog_dir>
   ```

   b. Assurez-vous que le code d'erreur est 0:

   ```
   $ echo $?
   ```

   Exemple de sortie

   ```
   0
   ```

5. Créez l'image du catalogue en exécutant la commande podman build:

```
$ podman build . \
    -f <catalog_dir>.Dockerfile \
    -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. Appuyez sur l'image du catalogue vers un registre:

   a. Au besoin, authentifier avec votre registre cible en exécutant la commande de connexion podman:

   ```
   $ podman login <registry>
   ```

   b. Appuyez sur l'image du catalogue en exécutant la commande podman push:

   ```
   $ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
   ```

Ressources supplémentaires

- CLI de référence OPM

4.7.2.2. La mise à jour ou le filtrage d'une image de catalogue basée sur des fichiers

Il est possible d'utiliser l'opm CLI pour mettre à jour ou filtrer une image de catalogue qui utilise le format de catalogue basé sur des fichiers. En extrayant le contenu d'une image de catalogue existante, vous pouvez modifier le catalogue au besoin, par exemple:

- Ajout de paquets
- En supprimant les paquets
- La mise à jour des entrées de paquet existantes
- Détail des messages de déprécation par paquet, canal et paquet

Ensuite, vous pouvez reconstruire l'image comme une version mise à jour du catalogue.

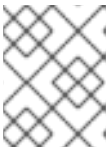
Conditions préalables

- Il y a ce qui suit sur votre poste de travail:
  - L'opm CLI.
  - la version 1.9.3+ de Podman.
  - Image de catalogue basée sur des fichiers.
  - La structure d'un répertoire de catalogue a récemment initialisé sur votre poste de travail lié à ce catalogue.
    Lorsque vous n'avez pas de répertoire de catalogue initialisé, créez le répertoire et générez le Dockerfile. Consultez l'étape "Initialiser le catalogue" à partir de la procédure "Créer une image de catalogue basée sur un fichier".

Procédure

1. Extraire le contenu de l'image du catalogue au format YAML dans un fichier index.yaml dans votre répertoire de catalogue:

```
$ opm render <registry>/<namespace>/<catalog_image_name>:<tag> \
    -o yaml > <catalog_dir>/index.yaml
```



NOTE

Alternativement, vous pouvez utiliser le drapeau -o json pour afficher au format JSON.

2. Modifiez le contenu du fichier index.yaml résultant à vos spécifications:

IMPORTANT

Après la publication d'un paquet dans un catalogue, supposez que l'un de vos utilisateurs l'a installé. Assurez-vous que tous les paquets publiés précédemment dans un catalogue disposent d'un chemin de mise à jour vers la tête de canal actuelle ou plus récente pour éviter d'échouer les utilisateurs qui ont cette version installée.

- Afin d'ajouter un opérateur, suivez les étapes de création de paquets, de paquets et de canaux dans la procédure "Créer une image de catalogue basée sur un fichier".

- Afin de supprimer un opérateur, supprimez les blobs olm.package, olm.channel et olm.bundle qui se rapportent au paquet. L'exemple suivant montre un ensemble qui doit être supprimé pour supprimer le package example-operator du catalogue:

Exemple 4.9. Exemple d'entrées supprimées

```
---
defaultChannel: release-2.7
icon:
  base64data: <base64_string>
  mediatype: image/svg+xml
name: example-operator
schema: olm.package
---
entries:
- name: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.0'
- name: example-operator.v2.7.1
  replaces: example-operator.v2.7.0
  skipRange: '>=2.6.0 <2.7.1'
- name: example-operator.v2.7.2
  replaces: example-operator.v2.7.1
  skipRange: '>=2.6.0 <2.7.2'
- name: example-operator.v2.7.3
  replaces: example-operator.v2.7.2
  skipRange: '>=2.6.0 <2.7.3'
- name: example-operator.v2.7.4
  replaces: example-operator.v2.7.3
  skipRange: '>=2.6.0 <2.7.4'
name: release-2.7
package: example-operator
schema: olm.channel
---
image: example.com/example-inc/example-operator-bundle@sha256:<digest>
name: example-operator.v2.7.0
package: example-operator
properties:
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyObject
    version: v1alpha1
- type: olm.gvk
  value:
    group: example-group.example.io
    kind: MyOtherObject
    version: v1beta1
- type: olm.package
  value:
    packageName: example-operator
    version: 2.7.0
- type: olm.bundle.object
  value:
    data: <base64_string>
```

```
- type: olm.bundle.object
  value:
    data: <base64_string>
relatedImages:
- image: example.com/example-inc/example-related-image@sha256:<digest>
  name: example-related-image
schema: olm.bundle
---
```

- Afin d'ajouter ou de mettre à jour des messages de déprécation pour un opérateur, assurez-vous qu'il y a un fichier déprecations.yaml dans le même répertoire que le fichier index.yaml du paquet. Informations sur le format de fichier deprecations.yaml, voir "olm.deprecations schema".

3. Enregistrez vos modifications.

4. De valider le catalogue:

```
$ opm validate <catalog_dir>
```

5. Reconstruire le catalogue:

```
$ podman build . \
    -f <catalog_dir>.Dockerfile \
    -t <registry>/<namespace>/<catalog_image_name>:<tag>
```

6. Appuyez sur l'image du catalogue mise à jour vers un registre:

```
$ podman push <registry>/<namespace>/<catalog_image_name>:<tag>
```

La vérification

1. Dans la console Web, accédez à la ressource de configuration OperatorHub dans la page Administration → Paramètres de cluster → Configuration.

2. Ajoutez la source du catalogue ou mettez à jour la source du catalogue existant pour utiliser la spécification de traction pour votre image de catalogue mise à jour.
   De plus amples informations, voir "Ajouter une source de catalogue à un cluster" dans les "Ressources supplémentaires" de cette section.

3. Après que la source du catalogue est dans un état READY, accédez à la page Opérateurs → OperatorHub et vérifiez que les modifications que vous avez apportées sont reflétées dans la liste des Opérateurs.

4.7.3. Catalogues SQLite

IMPORTANT

Le format de base de données SQLite pour les catalogues Operator est une fonctionnalité obsolète. La fonctionnalité obsolète est toujours incluse dans OpenShift Dedicated et continue d'être prise en charge; cependant, elle sera supprimée dans une version ultérieure de ce produit et n'est pas recommandée pour de nouveaux déploiements.

Dans la liste la plus récente des fonctionnalités majeures qui ont été dépréciées ou supprimées dans OpenShift Dedicated, faites référence à la section Fonctionnalités obsolètes et supprimées des notes de sortie OpenShift Dedicated.

4.7.3.1. Création d'une image d'index SQLite

Il est possible de créer une image d'index basée sur le format de base de données SQLite à l'aide de l'opm CLI.

Conditions préalables

- C'est vous qui avez installé l'opm CLI.
- Il y a la version 1.9.3+ de podman.
- L'image groupée est construite et poussée vers un registre qui prend en charge Docker v2-2.

Procédure

1. Commencez un nouvel index:

```
$ opm index add \
    --bundles <registry>/<namespace>/<bundle_image_name>:<tag> \ 1
    --tag <registry>/<namespace>/<index_image_name>:<tag> \ 2
    [--binary-image <registry_base_image>] 3
```

1. Liste séparée par virgule d'images groupées à ajouter à l'index.
2. La balise d'image que vous voulez que l'image de l'index ait.
3. Facultatif: Une image de base de registre alternative à utiliser pour servir le catalogue.

2. Appuyez sur l'image de l'index vers un registre.
   a. Au besoin, authentifier avec votre registre cible:

```
$ podman login <registry>
```

   b. Appuyez sur l'image de l'index:

```
$ podman push <registry>/<namespace>/<index_image_name>:<tag>
```

4.7.3.2. La mise à jour d'une image d'index SQLite

Après avoir configuré OperatorHub pour utiliser une source de catalogue qui fait référence à une image d'index personnalisée, les administrateurs avec le rôle dédié-admin peuvent garder les opérateurs disponibles sur leur cluster à jour en ajoutant des images groupées à l'image d'index.

Il est possible de mettre à jour une image d'index existante à l'aide de la commande Opm index add.

Conditions préalables

- C'est vous qui avez installé l'opm CLI.
- Il y a la version 1.9.3+ de podman.
- L'image d'index est construite et poussée vers un registre.
- Il existe une source de catalogue existante faisant référence à l'image de l'index.

Procédure

1. Actualisez l'index existant en ajoutant des images groupées:

```
$ opm index add \
   --bundles <registry>/<namespace>/<new_bundle_image>@sha256:<digest> \ 1
   --from-index <registry>/<namespace>/<existing_index_image>:<existing_tag> \ 2
   --tag <registry>/<namespace>/<existing_index_image>:<updated_tag> \ 3
   --pull-tool podman 4
```

1. L'indicateur --bundles spécifie une liste séparée par des virgules d'images de paquet supplémentaires à ajouter à l'index.
2. L'indicateur --from-index spécifie l'index précédemment poussé.
3. L'indicateur --tag spécifie la balise image à appliquer à l'image d'index mise à jour.
4. Le drapeau --pull-outil spécifie l'outil utilisé pour tirer des images de conteneur.

là où:

<registre>
: Indique le nom d'hôte du registre, tel que quay.io ou mirror.example.com.

<namespace>
: Indique l'espace de noms du registre, tel que ocs-dev ou abc.

<new_bundle_image>
: Indique la nouvelle image de paquet à ajouter au registre, comme ocs-operator.

<digest>
: Indique l'ID d'image SHA, ou digérer, de l'image de faisceau, telle que c7f11097a628f092d8bad148406a0e0951094a03445fd4bc0775431ef683a41.

<bx id="1" />
: Indique l'image précédemment poussée, telle que abc-redhat-operator-index.

<bx id="1" />
: Indique une balise d'image précédemment poussée, telle que 4.

<mise à jour_tag>

Indique la balise image à appliquer à l'image d'index mise à jour, telle que 4.1.

Commande d'exemple

```
$ opm index add \
    --bundles quay.io/ocs-dev/ocs-
operator@sha256:c7f11097a628f092d8bad148406aa0e0951094a03445fd4bc0775431ef683a
41 \
    --from-index mirror.example.com/abc/abc-redhat-operator-index:4 \
    --tag mirror.example.com/abc/abc-redhat-operator-index:4.1 \
    --pull-tool podman
```

2. Appuyez sur l'image de l'index mise à jour:

```
$ podman push <registry>/<namespace>/<existing_index_image>:<updated_tag>
```

3. Après que Operator Lifecycle Manager (OLM) sonde automatiquement l'image de l'index référencée dans la source du catalogue à son intervalle régulier, vérifiez que les nouveaux paquets sont ajoutés avec succès:

```
$ oc get packagemanifests -n openshift-marketplace
```

4.7.3.3. Filtrage d'une image d'index SQLite

L'image d'index, basée sur le format du paquet Opérateur, est un instantané conteneurisé d'un catalogue Opérateur. Il est possible de filtrer, ou tailler, un index de tous les paquets sauf une liste spécifiée, qui crée une copie de l'index source contenant uniquement les Opérateurs que vous souhaitez.

Conditions préalables

- Il y a la version 1.9.3+ de podman.
- Il y a grpcurl (outil de ligne de commande tiers).
- C'est vous qui avez installé l'opm CLI.
- Il y a accès à un registre qui prend en charge Docker v2-2.

Procédure

1. Authentifier avec votre registre cible:

```
$ podman login <target_registry>
```

2. Déterminez la liste des paquets que vous souhaitez inclure dans votre index élevé.

   a. Exécutez l'image de l'index source que vous souhaitez tailler dans un conteneur. À titre d'exemple:

```
$ podman run -p50051:50051 \
    -it registry.redhat.io/redhat/redhat-operator-index:v4
```

Exemple de sortie

```
Trying to pull registry.redhat.io/redhat/redhat-operator-index:v4...
Getting image source signatures
Copying blob ae8a0c23f5b1 done
...
INFO[0000] serving registry                 database=/database/index.db port=50051
```

b. Dans une session terminale séparée, utilisez la commande grpcurl pour obtenir une liste des paquets fournis par l'index:

```
$ grpcurl -plaintext localhost:50051 api.Registry/ListPackages > packages.out
```

c. Inspectez le fichier packages.out et identifiez les noms de paquets de cette liste que vous souhaitez conserver dans votre index élevé. À titre d'exemple:

Exemples d'extraits de liste de paquets

```
...
{
  "name": "advanced-cluster-management"
}
...
{
  "name": "jaeger-product"
}
...
{
{
  "name": "quay-operator"
}
...
```

d. Dans la session terminale où vous avez exécuté la commande podman exécuter, appuyez sur Ctrl et C pour arrêter le processus de conteneur.

3. Exécutez la commande suivante pour tailler l'index source de tous les paquets sauf les paquets spécifiés:

```
$ opm index prune \
   -f registry.redhat.io/redhat/redhat-operator-index:v4 \ 1
   -p advanced-cluster-management,jaeger-product,quay-operator \ 2
   [-i registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4] \ 3
   -t <target_registry>:<port>/<namespace>/redhat-operator-index:v4 4
```

1 Index sur prune.

2 Liste des paquets séparés par les virgules à conserver.

3 Requis uniquement pour les images IBM Power® et IBM Z® : image de base du registre de l'opérateur avec l'étiquette correspondant à la version majeure et mineure du cluster dédié à OpenShift.

4 Balise personnalisée pour la nouvelle image d'index en cours de construction.

4. Exécutez la commande suivante pour pousser la nouvelle image d'index vers votre registre cible:

```
$ podman push <target_registry>:<port>/<namespace>/redhat-operator-index:v4
```

lorsque <namespace> est un espace de noms existant sur le registre.

4.7.4. Catalogue sources et admission de sécurité de pod

L'admission à la sécurité des pod a été introduite dans OpenShift Dedicated 4.11 pour garantir les normes de sécurité des pod. Les sources de catalogue construites à l'aide du format de catalogue SQLite et d'une version de l'outil opm CLI publié avant OpenShift Dedicated 4.11 ne peuvent pas fonctionner sous l'application de la sécurité des pods restreints.

Dans OpenShift Dedicated 4, les espaces de noms n'ont pas restreint l'application de la sécurité des pod par défaut et le mode de sécurité source du catalogue par défaut est défini à l'héritage.

L'application restreinte par défaut pour tous les espaces de noms est prévue pour inclusion dans une future version d'OpenShift Dedicated. Lorsque l'exécution restreinte se produit, le contexte de sécurité de la spécification de la pod pour les gousses sources de catalogue doit correspondre à la norme de sécurité de la gousse restreinte. Lorsque l'image source de votre catalogue nécessite une norme de sécurité différente, l'étiquette d'entrées de sécurité de pod pour l'espace de noms doit être explicitement définie.

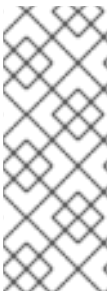
NOTE

Dans le cas où vous ne souhaitez pas exécuter vos pods source de catalogue SQLite comme restreints, vous n'avez pas besoin de mettre à jour votre source de catalogue dans OpenShift Dedicated 4.

Cependant, il est recommandé de prendre des mesures dès maintenant pour s'assurer que vos sources de catalogue s'exécutent sous l'application de la sécurité des pods restreints. Lorsque vous ne prenez pas d'action pour vous assurer que vos sources de catalogue s'exécutent sous l'application de la sécurité des pods restreints, vos sources de catalogue pourraient ne pas fonctionner dans les futures versions d'OpenShift Dedicated.

En tant qu'auteur de catalogue, vous pouvez activer la compatibilité avec l'application de la sécurité des pods restreints en complétant l'une des actions suivantes:

- Faites migrer votre catalogue vers le format de catalogue basé sur des fichiers.
- Actualisez l'image de votre catalogue avec une version de l'outil opm CLI publié avec OpenShift Dedicated 4.11 ou version ultérieure.



NOTE

Le format du catalogue de base de données SQLite est obsolète, mais toujours pris en charge par Red Hat. Dans une version ultérieure, le format de base de données SQLite ne sera pas pris en charge et les catalogues devront migrer vers le format de catalogue basé sur les fichiers. À partir d'OpenShift Dedicated 4.11, le catalogue de l'opérateur par défaut Red Hat est publié dans le format de catalogue basé sur les fichiers. Les catalogues basés sur des fichiers sont compatibles avec l'application de la sécurité des pod restreints.

Lorsque vous ne souhaitez pas mettre à jour votre image de catalogue SQLite ou migrer votre catalogue vers le format de catalogue basé sur des fichiers, vous pouvez configurer votre catalogue pour fonctionner avec des autorisations élevées.

Ressources supplémentaires

- Comprendre et gérer l'admission à la sécurité des pod

4.7.4.1. La migration des catalogues de base de données SQLite vers le format de catalogue basé sur des fichiers

Il est possible de mettre à jour vos catalogues de format de base de données SQLite obsolètes au format de catalogue basé sur des fichiers.

Conditions préalables

- Il y a une source de catalogue de base de données SQLite.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- La dernière version de l'outil Opm CLI est disponible avec OpenShift Dedicated 4 sur votre poste de travail.

Procédure

1. Faites migrer votre catalogue de base de données SQLite vers un catalogue basé sur des fichiers en exécutant la commande suivante:

```
$ opm migrate <registry_image> <fbc_directory>
```

2. Générez un Dockerfile pour votre catalogue basé sur des fichiers en exécutant la commande suivante:

```
$ opm generate dockerfile <fbc_directory> \
  --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4
```

Les prochaines étapes

- Le Dockerfile généré peut être construit, étiqueté et poussé à votre registre.

Ressources supplémentaires

- Ajout d'une source de catalogue à un cluster

4.7.4.2. La reconstruction des images du catalogue SQLite

Il est possible de reconstruire l'image de votre catalogue SQLite avec la dernière version de l'outil Opm CLI qui est publié avec votre version d'OpenShift Dedicated.

Conditions préalables

- Il y a une source de catalogue de base de données SQLite.

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- La dernière version de l'outil Opm CLI est disponible avec OpenShift Dedicated 4 sur votre poste de travail.

Procédure

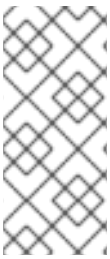
- Exécutez la commande suivante pour reconstruire votre catalogue avec une version plus récente de l'outil Opm CLI:

```
$ opm index add --binary-image \
  registry.redhat.io/openshift4/ose-operator-registry-rhel9:v4 \
  --from-index <your_registry_image> \
  --bundles "" -t \<your_registry_image>
```

4.7.4.3. Configuration des catalogues à exécuter avec des autorisations élevées

Dans le cas où vous ne souhaitez pas mettre à jour votre image de catalogue SQLite ou migrer votre catalogue vers le format de catalogue basé sur des fichiers, vous pouvez effectuer les actions suivantes pour vous assurer que votre source de catalogue s'exécute lorsque l'application de la sécurité de la pod par défaut est limitée:

- Définissez manuellement le mode de sécurité du catalogue à l'héritage dans la définition de votre source de catalogue. Cette action garantit que votre catalogue s'exécute avec des autorisations héritées même si le mode de sécurité du catalogue par défaut change à restreint.
- Étiqueter l'espace de noms source du catalogue pour l'application de la sécurité de base ou de pod privilégié.



NOTE

Le format du catalogue de base de données SQLite est obsolète, mais toujours pris en charge par Red Hat. Dans une version ultérieure, le format de base de données SQLite ne sera pas pris en charge et les catalogues devront migrer vers le format de catalogue basé sur les fichiers. Les catalogues basés sur des fichiers sont compatibles avec l'application de la sécurité des pod restreints.

Conditions préalables

- Il y a une source de catalogue de base de données SQLite.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- Il y a un espace de noms cible qui prend en charge les pods en cours d'exécution avec la norme d'admission de sécurité élevée de base ou privilégiée.

Procédure

1. Editez la définition de CatalogSource en définissant l'étiquette spec.grpcPodConfig.securityContextConfig sur l'héritage, comme indiqué dans l'exemple suivant:

   Exemple de CatalogSource Définition

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-catsrc
  namespace: my-ns
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: legacy
  image: my-image:latest
```

ASTUCE

Dans OpenShift Dedicated 4, le champ spec.grpcPodConfig.securityContextConfig est défini à l'héritage par défaut. Dans une version ultérieure d'OpenShift Dedicated, il est prévu que le paramètre par défaut change pour être limité. Dans le cas où votre catalogue ne peut pas fonctionner sous une application restreinte, il est recommandé de définir manuellement ce champ sur l'héritage.

2. Modifiez votre fichier <namespace>.yaml pour ajouter des normes d'admission de sécurité de pod élevées à votre espace de noms source de catalogue, comme indiqué dans l'exemple suivant:

Exemple <namespace>.yaml fichier

```
apiVersion: v1
kind: Namespace
metadata:
...
  labels:
    security.openshift.io/scc.podSecurityLabelSync: "false" (1)
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: baseline (2)
  name: "<namespace_name>"
```

(1) Désactivez la synchronisation des étiquettes de sécurité de pod en ajoutant l'étiquette security.openshift.io/scc.podSecurityLabelSync=false à l'espace de noms.

(2) Appliquez l'étiquette pod-security.kubernetes.io/enforce. Définissez l'étiquette en ligne de base ou privilégiée. Utilisez le profil de sécurité de la pod de base, sauf si d'autres charges de travail dans l'espace de noms nécessitent un profil privilégié.

4.7.5. Ajout d'une source de catalogue à un cluster

L'ajout d'une source de catalogue à un cluster dédié OpenShift permet la découverte et l'installation des opérateurs pour les utilisateurs. Les administrateurs avec le rôle dédié-admin peuvent créer un objet CatalogSource qui fait référence à une image d'index. OperatorHub utilise des sources de catalogue pour peupler l'interface utilisateur.

ASTUCE

Alternativement, vous pouvez utiliser la console Web pour gérer les sources de catalogue. Dans la page Accueil → Rechercher, sélectionnez un projet, cliquez sur la liste déroulante Ressources et recherchez CatalogSource. Il est possible de créer, mettre à jour, supprimer, désactiver et activer des sources individuelles.

Conditions préalables

- « vous avez construit et poussé une image d'index vers un registre.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

1. Créez un objet CatalogSource qui fait référence à votre image d'index.
   a. Modifiez ce qui suit à vos spécifications et enregistrez-le sous forme de fichier catalogSource.yaml:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: my-operator-catalog
  namespace: openshift-marketplace 1
  annotations:
    olm.catalogImageTemplate: 2
      "<registry>/<namespace>/<index_image_name>:v{kube_major_version}.
{kube_minor_version}.{kube_patch_version}"
spec:
  sourceType: grpc
  grpcPodConfig:
    securityContextConfig: <security_mode> 3
  image: <registry>/<namespace>/<index_image_name>:<tag> 4
  displayName: My Operator Catalog
  publisher: <publisher_name> 5
  updateStrategy:
    registryPoll: 6
      interval: 30m
```

1 Lorsque vous souhaitez que la source du catalogue soit disponible à l'échelle mondiale pour les utilisateurs de tous les espaces de noms, spécifiez l'espace de noms openshift-marketplace. Dans le cas contraire, vous pouvez spécifier un espace de noms différent pour le catalogue à portée et disponible uniquement pour cet espace de noms.

2 Facultatif: Définir l'annotation olm.catalogImageTempler sur votre nom d'image d'index et utiliser une ou plusieurs des variables de version du cluster Kubernetes comme indiqué lors de la construction du modèle pour la balise d'image.

3 Indiquez la valeur de l'héritage ou de la restriction. Lorsque le champ n'est pas défini, la valeur par défaut est héritée. Dans une version ultérieure d'OpenShift Dedicated, il est prévu que la valeur par défaut soit limitée. Dans le cas où votre catalogue ne peut pas fonctionner avec des autorisations restreintes, il est recommandé de définir manuellement ce champ sur l'héritage.

4 Indiquez votre image d'index. Lorsque vous spécifiez une balise après le nom de l'image, par exemple :v4, la source du catalogue utilise une stratégie de traction

5 Indiquez votre nom ou un nom d'organisation qui publie le catalogue.

6 Les sources du catalogue peuvent vérifier automatiquement les nouvelles versions pour se tenir à jour.

b. Créez l'objet CatalogSource:

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

2. Assurez-vous que les ressources suivantes sont créées avec succès.

a. Consultez les gousses:

```
$ oc get pods -n openshift-marketplace
```

Exemple de sortie

```
NAME                                  READY   STATUS    RESTARTS  AGE
my-operator-catalog-6njx6             1/1     Running   0         28s
marketplace-operator-d9f549946-96sgr  1/1     Running   0         26h
```

b. Consultez la source du catalogue:

```
$ oc get catalogsource -n openshift-marketplace
```

Exemple de sortie

```
NAME                  DISPLAY               TYPE PUBLISHER  AGE
my-operator-catalog   My Operator Catalog   grpc           5s
```

c. Consultez le manifeste du paquet:

```
$ oc get packagemanifest -n openshift-marketplace
```

Exemple de sortie

```
NAME                     CATALOG               AGE
jaeger-product           My Operator Catalog   93s
```

Désormais, vous pouvez installer les Opérateurs à partir de la page OperatorHub sur votre console Web dédiée OpenShift.

Ressources supplémentaires

- Concepts et ressources du gestionnaire de cycle de vie de l'opérateur → Source du catalogue

4.7.6. La suppression des catalogues personnalisés

En tant qu'administrateur avec le rôle d'administrateur dédié, vous pouvez supprimer les catalogues d'opérateur personnalisés qui ont déjà été ajoutés à votre cluster en supprimant la source du catalogue connexe.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

1. Dans la perspective de l'administrateur de la console Web, accédez à Home → Rechercher.
2. Choisissez un projet dans la liste Projet:
3. Choisissez CatalogSource dans la liste Ressources.
4. Choisissez le menu Options du catalogue que vous souhaitez supprimer, puis cliquez sur Supprimer CatalogSource.

4.8. CATALOGUE SOURCE DE CALENDRIER DES POD

Lorsqu'un gestionnaire de cycle de vie d'opérateur (OLM) source de type de source grpc définit une image spec.image, l'opérateur de catalogue crée un pod qui sert le contenu de l'image définie. Ce pod définit par défaut ce qui suit dans sa spécification:

- Il n'y a que le sélecteur de nœuds kubernetes.io/os=linux.
- Le nom de classe prioritaire par défaut: system-cluster-critique.
- Aucune tolérance.

En tant qu'administrateur, vous pouvez remplacer ces valeurs en modifiant les champs dans la section optionnelle spec.grpcPodConfig de l'objet CatalogSource.

IMPORTANT

L'opérateur Marketplace, openshift-marketplace, gère la ressource personnalisée (CR) par défaut OperatorHub. Ce CR gère les objets CatalogSource. Lorsque vous tentez de modifier les champs de la section spec.grpcPodConfig de l'objet CatalogSource, l'opérateur Marketplace retourne automatiquement ces modifications.

Afin d'appliquer des modifications persistantes à l'objet CatalogSource, vous devez d'abord désactiver un objet CatalogSource par défaut.

Ressources supplémentaires

- Concepts et ressources OLM → Source du catalogue

4.8.1. Désactivation des objets CatalogSource par défaut au niveau local

Il est possible d'appliquer des modifications persistantes à un objet CatalogSource, comme les pods source de catalogue, au niveau local, en désactivant un objet CatalogSource par défaut. Considérez la configuration par défaut dans les situations où la configuration de l'objet CatalogSource par défaut ne

répond pas aux besoins de votre organisation. Lorsque vous modifiez les champs dans la section spec.grpcPodConfig de l'objet CatalogSource, l'opérateur Marketplace retourne automatiquement ces modifications.

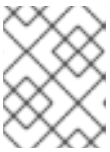
L'opérateur Marketplace, openshift-marketplace, gère les ressources personnalisées (CR) par défaut de l'opérateur OperatorHub. L'opérateurHub gère les objets CatalogSource.

Afin d'appliquer des modifications persistantes à l'objet CatalogSource, vous devez d'abord désactiver un objet CatalogSource par défaut.

Procédure

- Afin de désactiver tous les objets CatalogSource par défaut au niveau local, entrez la commande suivante:

```
$ oc patch operatorhub cluster -p '{"spec": {"disableAllDefaultSources": true}}' --type=merge
```



NOTE

Il est également possible de configurer le OperatorHub CR par défaut pour désactiver tous les objets CatalogSource ou désactiver un objet spécifique.

Ressources supplémentaires

- OperatorHub ressource personnalisée

4.8.2. Écraser le sélecteur de nœud pour les pods sources de catalogue

Conditions préalables

- L'objet CatalogSource de type source grpc avec spec.image est défini.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

- Éditez l'objet CatalogSource et ajoutez ou modifiez la section spec.grpcPodConfig pour inclure ce qui suit:

```
grpcPodConfig:
  nodeSelector:
    custom_label: <label>
```

lorsque <label> est l'étiquette du sélecteur de nœuds que vous souhaitez que les pods source du catalogue utilisent pour la planification.

Ressources supplémentaires

- Placer des pods sur des nœuds spécifiques à l'aide de sélecteurs de nœuds

4.8.3. Dépassement du nom de classe prioritaire pour les pods sources de catalogue

Conditions préalables

- L'objet CatalogSource de type source grpc avec spec.image est défini.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

- Éditez l'objet CatalogSource et ajoutez ou modifiez la section spec.grpcPodConfig pour inclure ce qui suit:

```
grpcPodConfig:
  priorityClassName: <priority_class>
```

lorsque <priority_class> est l'un des éléments suivants:

  - L'une des classes de priorité par défaut fournies par Kubernetes: system-cluster-critique ou system-node-critique
  - Ensemble vide ("") pour attribuer la priorité par défaut
  - Classe de priorité préexistante et définie sur mesure

NOTE

Auparavant, le seul paramètre de planification des pod qui pouvait être dépassé était priorityClassName. Cela a été fait en ajoutant l'annotation de classe de priorité à l'objet CatalogSource. À titre d'exemple:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: example-catalog
  namespace: openshift-marketplace
  annotations:
    operatorframework.io/priorityclass: system-cluster-critical
```

Lorsqu'un objet CatalogSource définit à la fois l'annotation et spec.grpcPodConfig.priorityClassName, l'annotation prime sur le paramètre de configuration.

Ressources supplémentaires

- Classes prioritaires de pod

4.8.4. Les tolérances primordiales pour les pods sources de catalogue

Conditions préalables

- L'objet CatalogSource de type source grpc avec spec.image est défini.
- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.

Procédure

- Editez l'objet CatalogSource et ajoutez ou modifiez la section spec.grpcPodConfig pour inclure ce qui suit:

```
grpcPodConfig:
  tolerations:
    - key: "<key_name>"
      operator: "<operator_type>"
      value: "<value>"
      effect: "<effect>"
```

4.9. DÉPANNAGE DES PROBLÈMES DE L'OPÉRATEUR

Lorsque vous rencontrez des problèmes d'opérateur, vérifiez l'état de l'abonnement de l'opérateur. Consultez la santé de la pod de l'opérateur à travers le cluster et rassemblez les journaux de l'opérateur pour le diagnostic.

4.9.1. Conditions d'abonnement à l'opérateur

Les abonnements peuvent signaler les types d'état suivants:

Tableau 4.2. Conditions d'abonnement

État de l'état	Description
CatalogueSourcesUnsanté	Certaines ou toutes les sources de catalogue à utiliser en résolution sont malsaines.
InstallPlanMissing	Il manque un plan d'installation pour un abonnement.
InstallPlanPending	Le plan d'installation d'un abonnement est en attente d'installation.
InstallPlanFailed	Le plan d'installation d'un abonnement a échoué.
La RésolutionFailed	La résolution de dépendance pour un abonnement a échoué.

NOTE

Les opérateurs de cluster dédiés par défaut sont gérés par l'opérateur de versions de cluster (CVO) et ils n'ont pas d'objet d'abonnement. Les opérateurs d'applications sont gérés par Operator Lifecycle Manager (OLM) et ils ont un objet d'abonnement.

Ressources supplémentaires

- Exigences en matière de santé du catalogue

4.9.2. Affichage du statut d'abonnement de l'opérateur en utilisant le CLI

En utilisant le CLI, vous pouvez voir l'état de l'abonnement à l'opérateur.

Conditions préalables

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

Procédure

1. Abonnements à l'opérateur de liste:

```
$ oc get subs -n <operator_namespace>
```

2. Consultez la commande de description d'oc pour inspecter une ressource d'abonnement:

```
$ oc describe sub <subscription_name> -n <operator_namespace>
```

3. Dans la sortie de commande, trouvez la section Conditions pour l'état des types de condition d'abonnement Opérateur. Dans l'exemple suivant, le type de condition de CatalogSourcesUnsanté a un statut de faux parce que toutes les sources de catalogue disponibles sont saines:

Exemple de sortie

```
Name:         cluster-logging
Namespace:    openshift-logging
Labels:       operators.coreos.com/cluster-logging.openshift-logging=
Annotations:  <none>
API Version:  operators.coreos.com/v1alpha1
Kind:         Subscription
# ...
Conditions:
  Last Transition Time:  2019-07-29T13:42:57Z
  Message:               all available catalogsources are healthy
  Reason:                AllCatalogSourcesHealthy
  Status:                False
  Type:                  CatalogSourcesUnhealthy
# ...
```

NOTE

Les opérateurs de cluster dédiés par défaut sont gérés par l'opérateur de versions de cluster (CVO) et ils n'ont pas d'objet d'abonnement. Les opérateurs d'applications sont gérés par Operator Lifecycle Manager (OLM) et ils ont un objet d'abonnement.

4.9.3. Affichage de l'état de la source du catalogue de l'opérateur en utilisant le CLI

Il est possible d'afficher l'état d'une source de catalogue de l'opérateur à l'aide du CLI.

Conditions préalables

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

Procédure

1. Énumérez les sources du catalogue dans un espace de noms. À titre d'exemple, vous pouvez vérifier l'espace de noms openshift-marketplace, qui est utilisé pour les sources de catalogue à l'échelle du cluster:

```
$ oc get catalogsources -n openshift-marketplace
```

Exemple de sortie

```
NAME                  DISPLAY               TYPE   PUBLISHER   AGE
certified-operators   Certified Operators   grpc   Red Hat     55m
community-operators   Community Operators   grpc   Red Hat     55m
example-catalog       Example Catalog       grpc   Example Org 2m25s
redhat-marketplace    Red Hat Marketplace   grpc   Red Hat     55m
redhat-operators      Red Hat Operators     grpc   Red Hat     55m
```

2. La commande oc described permet d'obtenir plus de détails et d'état sur une source de catalogue:

```
$ oc describe catalogsource example-catalog -n openshift-marketplace
```

Exemple de sortie

```
Name:         example-catalog
Namespace:    openshift-marketplace
Labels:       <none>
Annotations:  operatorframework.io/managed-by: marketplace-operator
              target.workload.openshift.io/management: {"effect": "PreferredDuringScheduling"}
API Version:  operators.coreos.com/v1alpha1
Kind:         CatalogSource
# ...
Status:
  Connection State:
    Address:              example-catalog.openshift-marketplace.svc:50051
    Last Connect:         2021-09-09T17:07:35Z
    Last Observed State:  TRANSIENT_FAILURE
  Registry Service:
    Created At:         2021-09-09T17:05:45Z
    Port:               50051
    Protocol:           grpc
    Service Name:       example-catalog
    Service Namespace:  openshift-marketplace
# ...
```

Dans l'exemple précédent, le dernier état observé est TRANSIENT_FAILURE. Cet état indique qu'il y a un problème à établir une connexion pour la source du catalogue.

3. Énumérez les pods dans l'espace de noms où votre source de catalogue a été créée:

```
$ oc get pods -n openshift-marketplace
```

Exemple de sortie

```
NAME                                    READY   STATUS          RESTARTS   AGE
```

```
certified-operators-cv9nn              1/1     Running           0          36m
community-operators-6v8lp              1/1     Running           0          36m
marketplace-operator-86bfc75f9b-jkgbc  1/1     Running           0          42m
example-catalog-bwt8z                  0/1     ImagePullBackOff  0          3m55s
redhat-marketplace-57p8c               1/1     Running           0          36m
redhat-operators-smxx8                 1/1     Running           0          36m
```

Lorsqu'une source de catalogue est créée dans un espace de noms, un pod pour la source du catalogue est créé dans cet espace de noms. Dans l'exemple précédent, l'état de la pod example-catalog-bwt8z est ImagePullBackOff. Ce statut indique qu'il y a un problème à tirer l'image de l'index de la source du catalogue.

4. Consultez la commande de description d'oc pour inspecter un pod pour obtenir des informations plus détaillées:

```
$ oc describe pod example-catalog-bwt8z -n openshift-marketplace
```

Exemple de sortie

```
Name:         example-catalog-bwt8z
Namespace:    openshift-marketplace
Priority:     0
Node:         ci-ln-jyryyg2-f76d1-ggdbq-worker-b-vsxjd/10.0.128.2
...
Events:
  Type     Reason          Age                From               Message
  ----     ------          ----               ----               -------
  Normal   Scheduled       48s                default-scheduler  Successfully assigned openshift-
marketplace/example-catalog-bwt8z to ci-ln-jyryyf2-f76d1-fgdbq-worker-b-vsxjd
  Normal   AddedInterface  47s                multus             Add eth0 [10.131.0.40/23] from
openshift-sdn
  Normal   BackOff         20s (x2 over 46s)  kubelet            Back-off pulling image
"quay.io/example-org/example-catalog:v1"
  Warning  Failed          20s (x2 over 46s)  kubelet            Error: ImagePullBackOff
  Normal   Pulling         8s (x3 over 47s)   kubelet            Pulling image "quay.io/example-
org/example-catalog:v1"
  Warning  Failed          8s (x3 over 47s)   kubelet            Failed to pull image
"quay.io/example-org/example-catalog:v1": rpc error: code = Unknown desc = reading
manifest v1 in quay.io/example-org/example-catalog: unauthorized: access to the requested
resource is not authorized
  Warning  Failed          8s (x3 over 47s)   kubelet            Error: ErrImagePull
```

Dans l'exemple précédent, les messages d'erreur indiquent que l'image d'index de la source du catalogue ne parvient pas à tirer avec succès en raison d'un problème d'autorisation. À titre d'exemple, l'image d'index peut être stockée dans un registre qui nécessite des identifiants de connexion.

Ressources supplémentaires

- documentation du GRPC : États de connectivité

4.9.4. État de la pod de l'opérateur d'interrogation

Il est possible de répertorier les pods d'opérateur dans un cluster et leur statut. Il est également possible de recueillir un résumé détaillé de la pod de l'opérateur.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- Le service API est toujours fonctionnel.
- L'OpenShift CLI (oc) a été installé.

Procédure

1. Liste Opérateurs fonctionnant dans le cluster. La sortie comprend la version de l'opérateur, la disponibilité et les informations de disponibilité:

```
$ oc get clusteroperators
```

2. Liste Les pods d'opérateur s'exécutant dans l'espace de noms de l'opérateur, plus le statut du pod, le redémarrage et l'âge:

```
$ oc get pod -n <operator_namespace>
```

3. Afficher un résumé détaillé de la pod de l'opérateur:

```
$ oc describe pod <operator_pod_name> -n <operator_namespace>
```

4.9.5. Collecte des journaux de l'opérateur

Lorsque vous rencontrez des problèmes d'opérateur, vous pouvez recueillir des informations de diagnostic détaillées à partir des journaux de pod de l'opérateur.

Conditions préalables

- En tant qu'utilisateur, vous avez accès au cluster avec le rôle d'administrateur dédié.
- Le service API est toujours fonctionnel.
- L'OpenShift CLI (oc) a été installé.
- Les noms de domaine entièrement qualifiés des machines de plan de contrôle ou de contrôle sont disponibles.

Procédure

1. Énumérez les pods d'opérateur qui s'exécutent dans l'espace de noms de l'opérateur, ainsi que le statut, le redémarrage et l'âge du pod:

```
$ oc get pods -n <operator_namespace>
```

2. Journaux d'examen pour une pod d'opérateur:

```
$ oc logs pod/<pod_name> -n <operator_namespace>
```

Dans le cas où une pod d'opérateur a plusieurs conteneurs, la commande précédente produira une erreur qui inclut le nom de chaque conteneur. Journal de requête à partir d'un conteneur individuel:

```
$ oc logs pod/<operator_pod_name> -c <container_name> -n <operator_namespace>
```

3. Lorsque l'API n'est pas fonctionnelle, examinez la pod et le conteneur de l'opérateur sur chaque nœud de plan de contrôle en utilisant SSH à la place. <master-node>.<cluster_name>.<base_domain> par des valeurs appropriées.

   a. Liste des gousses sur chaque nœud de plan de contrôle:

   ```
   $ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl pods
   ```

   b. Dans le cas de n'importe quelle gousse d'opérateur qui ne montre pas d'état prêt, inspectez en détail l'état de la gousse. <operator_pod_id> par l'ID du pod de l'opérateur listé dans la sortie de la commande précédente:

   ```
   $ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspectp
   <operator_pod_id>
   ```

   c. Liste des conteneurs liés à une pod d'opérateur:

   ```
   $ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl ps --pod=
   <operator_pod_id>
   ```

   d. Dans le cas d'un conteneur de l'opérateur qui ne montre pas d'état prêt, inspectez en détail l'état du conteneur. <container_id> par un identifiant de conteneur listé dans la sortie de la commande précédente:

   ```
   $ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl inspect
   <container_id>
   ```

   e. Examinez les journaux pour tous les conteneurs de l'opérateur qui ne montrent pas d'état prêt. <container_id> par un identifiant de conteneur listé dans la sortie de la commande précédente:

   ```
   $ ssh core@<master-node>.<cluster_name>.<base_domain> sudo crictl logs -f
   <container_id>
   ```

NOTE

Les 4 nœuds de cluster de Red Hat Enterprise Linux CoreOS (RHCOS) sont immuables et comptent sur les opérateurs pour appliquer des modifications de cluster. L'accès aux nœuds de cluster en utilisant SSH n'est pas recommandé. Avant d'essayer de recueillir des données diagnostiques sur SSH, vérifiez si les données recueillies en exécutant oc adm doivent recueillir et si d'autres commandes oc sont suffisantes à la place. Cependant, si l'API dédiée OpenShift n'est pas disponible, ou si le kubelet ne fonctionne pas correctement sur le nœud cible, les opérations oc seront affectées. Dans de telles situations, il est possible d'accéder à des nœuds en utilisant ssh core@<node>.<cluster_name>.<base_domain>.

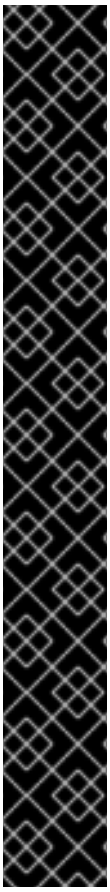
CHAPITRE 5. DÉVELOPPER DES OPÉRATEURS

5.1. À PROPOS DE L'OPÉRATEUR SDK

Le Cadre d'opérateur est une boîte à outils open source pour gérer les applications natives Kubernetes, appelées Opérateurs, de manière efficace, automatisée et évolutive. Les opérateurs profitent de l'extensibilité de Kubernetes pour offrir les avantages d'automatisation des services cloud, tels que le provisionnement, la mise à l'échelle, la sauvegarde et la restauration, tout en étant en mesure d'exécuter n'importe où Kubernetes peut fonctionner.

Les opérateurs facilitent la gestion d'applications complexes et étatiques au-dessus de Kubernetes. Cependant, écrire un opérateur aujourd'hui peut être difficile en raison de défis tels que l'utilisation d'API de bas niveau, l'écriture de plaque de chaudière et un manque de modularité, ce qui conduit à la duplication.

Le SDK de l'opérateur, un composant du Cadre d'opérateur, fournit un outil d'interface de ligne de commande (CLI) que les développeurs d'opérateurs peuvent utiliser pour construire, tester et déployer un opérateur.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Comment utiliser le SDK de l'opérateur?

Le SDK de l'opérateur simplifie ce processus de création d'applications natives Kubernetes, ce qui peut nécessiter des connaissances opérationnelles approfondies et spécifiques à l'application. Le SDK de l'opérateur réduit non seulement cette barrière, mais il aide également à réduire la quantité de code de plaque de chaudière nécessaire pour de nombreuses capacités de gestion communes, telles que le comptage ou la surveillance.

Le SDK de l'opérateur est un framework qui utilise la bibliothèque d'exécution du contrôleur pour faciliter l'écriture des opérateurs en fournissant les fonctionnalités suivantes:

- API de haut niveau et abstractions pour écrire la logique opérationnelle plus intuitivement
- Des outils pour l'échafaudage et la génération de code pour démarrer rapidement un nouveau projet
- Intégration avec Operator Lifecycle Manager (OLM) pour rationaliser l'emballage, l'installation et l'exécution des opérateurs sur un cluster
- Extensions pour couvrir les cas communs d'utilisation de l'opérateur
- Les métriques configurées automatiquement dans n'importe quel opérateur Go-based généré pour une utilisation sur des clusters où l'opérateur Prometheus est déployé

Les auteurs d'opérateurs disposant d'un accès administrateur dédié à OpenShift Dedicated peuvent utiliser l'opérateur SDK CLI pour développer leurs propres opérateurs basés sur Go, Ansible, Java ou Helm. Kubebuilder est intégré dans le SDK de l'opérateur en tant que solution d'échafaudage pour les opérateurs Go, ce qui signifie que les projets Kubebuilder existants peuvent être utilisés comme avec le SDK de l'opérateur et continuer à fonctionner.

NOTE

Le logiciel OpenShift Dedicated 4 prend en charge le SDK 1.38.0 de l'opérateur.

5.1.1. En quoi consistent les opérateurs?

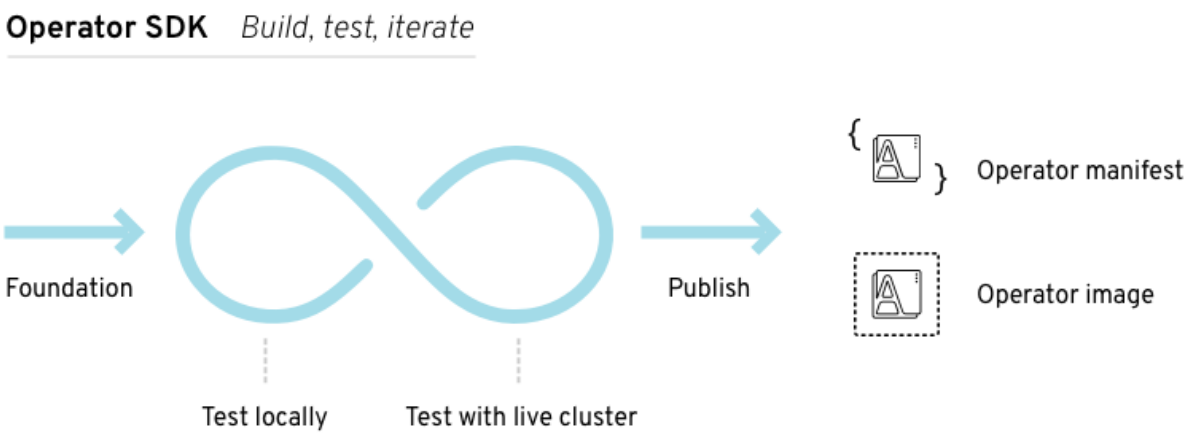
Afin d'obtenir un aperçu des concepts et de la terminologie de base de l'opérateur, voir Comprendre les opérateurs.

5.1.2. Flux de travail de développement

Le SDK de l'opérateur fournit le flux de travail suivant pour développer un nouvel opérateur:

1. Créez un projet d'opérateur à l'aide de l'interface de ligne de commande de l'opérateur SDK (CLI).
2. Définissez de nouvelles API de ressource en ajoutant des définitions de ressources personnalisées (CRD).
3. Indiquez les ressources à surveiller à l'aide de l'API d'opérateur SDK.
4. Définissez la logique de rapprochement de l'opérateur dans un gestionnaire désigné et utilisez l'API d'opérateur SDK pour interagir avec les ressources.
5. Faites appel à l'opérateur SDK CLI pour créer et générer les manifestes de déploiement de l'opérateur.

Figure 5.1. Flux de travail de l'opérateur SDK



À un niveau élevé, un opérateur qui utilise l'opérateur SDK traite les événements pour les ressources surveillées dans un gestionnaire défini par l'opérateur et prend des mesures pour concilier l'état de l'application.

5.1.3. Ressources supplémentaires

- Guide de construction d'opérateur certifié

5.2. INSTALLATION DE L'OPÉRATEUR SDK CLI

Le SDK de l'opérateur fournit un outil d'interface de ligne de commande (CLI) que les développeurs d'opérateurs peuvent utiliser pour construire, tester et déployer un opérateur. Il est possible d'installer le SDK CLI de l'opérateur sur votre poste de travail afin que vous soyez prêt à commencer à créer vos propres opérateurs.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Les auteurs d'opérateurs disposant d'un accès administrateur dédié à OpenShift Dedicated peuvent utiliser l'opérateur SDK CLI pour développer leurs propres opérateurs basés sur Go, Ansible, Java ou Helm. Kubebuilder est intégré dans le SDK de l'opérateur en tant que solution d'échafaudage pour les opérateurs Go, ce qui signifie que les projets Kubebuilder existants peuvent être utilisés comme avec le SDK de l'opérateur et continuer à fonctionner.

NOTE

Le logiciel OpenShift Dedicated 4 prend en charge le SDK 1.38.0 de l'opérateur.

5.2.1. Installation de l'opérateur SDK CLI sur Linux

L'outil OpenShift SDK CLI peut être installé sur Linux.

Conditions préalables

- Aller v1.19+
- Docker v17.03+, podman v1.9.3+, ou buildah v1.7+

Procédure

1. Accédez au site miroir OpenShift.
2. À partir du dernier répertoire 4, téléchargez la dernière version du tarball pour Linux.
3. Décompressez l'archive:

```
$ tar xvf operator-sdk-v1.38.0-ocp-linux-x86_64.tar.gz
```

4. Faire le fichier exécutable:

```
$ chmod +x operator-sdk
```

5. Déplacez le binaire opérateur-sdk extrait vers un répertoire qui se trouve sur votre PATH.

ASTUCE

Afin de vérifier votre PATH:

```
$ echo $PATH
```

```
$ sudo mv ./operator-sdk /usr/local/bin/operator-sdk
```

La vérification

- Après avoir installé l'opérateur SDK CLI, vérifiez qu'il est disponible:

```
$ operator-sdk version
```

Exemple de sortie

```
operator-sdk version: "v1.38.0-ocp", ...
```

5.2.2. Installation de l'opérateur SDK CLI sur macOS

L'outil OpenShift SDK CLI peut être installé sur macOS.

Conditions préalables

- Aller v1.19+
- Docker v17.03+, podman v1.9.3+, ou buildah v1.7+

Procédure

1. En ce qui concerne l'architecture amd64, accédez au site miroir OpenShift pour l'architecture amd64.
2. À partir du dernier répertoire 4, téléchargez la dernière version du tarball pour macOS.
3. Décompressez l'archive de l'opérateur SDK pour l'architecture amd64 en exécutant la commande suivante:

```
$ tar xvf operator-sdk-v1.38.0-ocp-darwin-x86_64.tar.gz
```

4. Faites le fichier exécutable en exécutant la commande suivante:

```
$ chmod +x operator-sdk
```

5. Déplacez le binaire opérateur-sdk extrait vers un répertoire qui est sur votre PATH en exécutant la commande suivante:

   ASTUCE

   Cochez votre PATH en exécutant la commande suivante:

   ```
   $ echo $PATH
   ```

   ```
   $ sudo mv ./operator-sdk /usr/local/bin/operator-sdk
   ```

La vérification

- Après avoir installé l'opérateur SDK CLI, vérifiez qu'il est disponible en exécutant la commande suivante:

  ```
  $ operator-sdk version
  ```

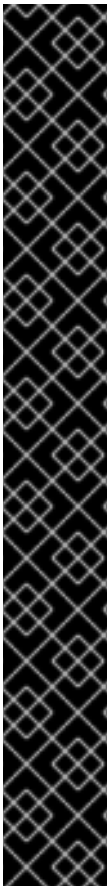
  Exemple de sortie

  ```
  operator-sdk version: "v1.38.0-ocp", ...
  ```

5.3. OPÉRATEURS BASÉS SUR GO

5.3.1. Didacticiel d'opérateur SDK pour les opérateurs Go-based

Les développeurs d'opérateurs peuvent profiter de la prise en charge du langage de programmation Go dans le SDK de l'opérateur pour créer un exemple d'opérateur Go pour Memcached, un magasin à valeur clé distribué, et gérer son cycle de vie.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Ce processus est réalisé à l'aide de deux pièces maîtresses du cadre opérateur:

Le SDK de l'opérateur

L'outil operator-sdk CLI et l'API de bibliothèque de contrôleurs

Gestionnaire du cycle de vie de l'opérateur (OLM)

Installation, mise à niveau et contrôle d'accès basé sur les rôles (RBAC) des opérateurs sur un cluster

NOTE

Ce tutoriel va plus en détail que commencer avec Operator SDK pour les opérateurs Go dans la documentation OpenShift Container Platform.

5.3.1.1. Conditions préalables

- L'opérateur SDK CLI installé
- Installation d'OpenShift CLI (oc) 4+
- Aller 1.21+
- Connecté à un cluster OpenShift dédié avec oc avec un compte qui dispose d'autorisations d'administration dédiées
- Afin de permettre au cluster de tirer l'image, le référentiel où vous poussez votre image doit être défini comme public, ou vous devez configurer une image pull secret

Ressources supplémentaires

- Installation de l'opérateur SDK CLI

- Débuter avec l'OpenShift CLI

5.3.1.2. Créer un projet

Faites appel à l'opérateur SDK CLI pour créer un projet appelé memcached-operator.

Procédure

1. Créer un répertoire pour le projet:

```
$ mkdir -p $HOME/projects/memcached-operator
```

2. Changement dans le répertoire:

```
$ cd $HOME/projects/memcached-operator
```

3. Activer la prise en charge des modules Go:

```
$ export GO111MODULE=on
```

4. Exécutez la commande operator-sdk init pour initialiser le projet:

```
$ operator-sdk init \
    --domain=example.com \
    --repo=github.com/example-inc/memcached-operator
```

NOTE

La commande operator-sdk init utilise le plugin Go par défaut.

La commande operator-sdk init génère un fichier go.mod à utiliser avec les modules Go. L'indicateur --repo est requis lors de la création d'un projet en dehors de $GOPATH/src/, car les fichiers générés nécessitent un chemin de chemin de module valide.

5.3.1.2.1. Fichier PROJET

Il y a parmi les fichiers générés par la commande operator-sdk init un fichier Kubebuilder PROJECT. Les commandes ultérieures de l'opérateur-sdk, ainsi que la sortie d'aide, qui sont exécutées à partir de la racine du projet lisent ce fichier et sont conscientes que le type de projet est Go. À titre d'exemple:

```
domain: example.com
layout:
- go.kubebuilder.io/v3
projectName: memcached-operator
repo: github.com/example-inc/memcached-operator
version: "3"
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
```

5.3.1.2.2. À propos du gestionnaire

Le programme principal pour l'opérateur est le fichier main.go, qui initialise et exécute le gestionnaire. Le gestionnaire enregistre automatiquement le Schéma pour toutes les définitions de l'API de ressources personnalisées (CR) et configure et exécute des contrôleurs et des webhooks.

Le gestionnaire peut restreindre l'espace de noms que tous les contrôleurs surveillent pour les ressources:

```
mgr, err := ctrl.NewManager(cfg, manager.Options{Namespace: namespace})
```

Le gestionnaire surveille par défaut l'espace de noms où l'opérateur s'exécute. Afin de regarder tous les espaces de noms, vous pouvez laisser l'option namespace vide:

```
mgr, err := ctrl.NewManager(cfg, manager.Options{Namespace: ""})
```

Il est également possible d'utiliser la fonction MultiNamespacedCacheBuilder pour regarder un ensemble spécifique d'espaces de noms:

```
var namespaces []string (1)
mgr, err := ctrl.NewManager(cfg, manager.Options{ (2)
  NewCache: cache.MultiNamespacedCacheBuilder(namespaces),
})
```

1. Liste des espaces de noms.
2. Crée une structure Cmd pour fournir des dépendances partagées et des composants de démarrage.

5.3.1.2.3. À propos des API multi-groupes

Avant de créer une API et un contrôleur, examinez si votre opérateur nécessite plusieurs groupes d'API. Ce tutoriel couvre le cas par défaut d'une API de groupe unique, mais pour modifier la mise en page de votre projet pour prendre en charge les API multi-groupes, vous pouvez exécuter la commande suivante:

```
$ operator-sdk edit --multigroup=true
```

Cette commande met à jour le fichier PROJECT, qui devrait ressembler à l'exemple suivant:

```
domain: example.com
layout: go.kubebuilder.io/v3
multigroup: true
...
```

Dans le cas des projets multigroupes, les fichiers de type API Go sont créés dans le répertoire apis/<group>/<version>/ et les contrôleurs sont créés dans le répertoire Controllers/<group>/ /. Le Dockerfile est ensuite mis à jour en conséquence.

B) Ressources supplémentaires

- Consultez la documentation Kubebuilder pour plus de détails sur la migration vers un projet multigroupe.

5.3.1.3. Création d'une API et d'un contrôleur

Faites appel à l'opérateur SDK CLI pour créer une API et un contrôleur de définition de ressources personnalisées (CRD).

Procédure

1. Exécutez la commande suivante pour créer une API avec le cache de groupe, version, v1, et sortez Memcached:

```
$ operator-sdk create api \
    --group=cache \
    --version=v1 \
    --kind=Memcached
```

2. Lorsque vous l'invitez, entrez y pour créer à la fois la ressource et le contrôleur:

```
Create Resource [y/n]
y
Create Controller [y/n]
y
```

Exemple de sortie

```
Writing scaffold for you to edit...
api/v1/memcached_types.go
controllers/memcached_controller.go
...
```

Ce processus génère l'API de ressource Memcached sur api/v1/memcached_types.go et le contrôleur à Controllers/memcached_controller.go.

5.3.1.3.1. Définir l'API

Définissez l'API pour la ressource personnalisée Memcached (CR).

Procédure

1. De modifier les définitions de type Go sur api/v1/memcached_types.go pour avoir les spécifications et le statut suivants:

```
// MemcachedSpec defines the desired state of Memcached
type MemcachedSpec struct {
 // +kubebuilder:validation:Minimum=0
 // Size is the size of the memcached deployment
 Size int32 `json:"size"`
}

// MemcachedStatus defines the observed state of Memcached
type MemcachedStatus struct {
 // Nodes are the names of the memcached pods
 Nodes []string `json:"nodes"`
}
```

2. Actualisez le code généré pour le type de ressource:

```
$ make generate
```

ASTUCE

Après avoir modifié un fichier *_types.go, vous devez exécuter la commande make Generator pour mettre à jour le code généré pour ce type de ressource.

La cible Makefile ci-dessus invoque l'utilitaire Controller-gen pour mettre à jour le fichier api/v1/zz_generated.deepcopy.go. Cela garantit que vos définitions de type API Go implémentent l'interface runtime.Object que tous les types de type doivent implémenter.

5.3.1.3.2. Générer des manifestes CRD

Après que l'API est définie avec des champs de spécification et d'état et des marqueurs de validation de la définition de ressources personnalisées (CRD), vous pouvez générer des manifestes CRD.

Procédure

- Exécutez la commande suivante pour générer et mettre à jour les manifestes CRD:

```
$ make manifests
```

Cette cible Makefile invoque l'utilitaire Controller-gen pour générer les manifestes CRD dans le fichier config/crd/bases/cache.example.com_memcacheds.yaml.

5.3.1.3.2.1. À propos de la validation OpenAPI

Les schémas OpenAPIv3 sont ajoutés aux manifestes CRD dans le bloc spec.validation lorsque les manifestes sont générés. Ce bloc de validation permet à Kubernetes de valider les propriétés dans une ressource personnalisée Memcached (CR) lorsqu'elle est créée ou mise à jour.

Des marqueurs, ou annotations, sont disponibles pour configurer les validations de votre API. Ces marqueurs ont toujours un préfixe +kubebuilder:validation.

Ressources supplémentaires

- Consultez la documentation Kubebuilder suivante pour plus de détails sur l'utilisation des marqueurs dans le code API:
  - Génération de CRD
  - Les marqueurs
  - Liste des marqueurs de validation OpenAPIv3
- Consultez la documentation Kubernetes pour plus de détails sur les schémas de validation OpenAPIv3.

5.3.1.4. Implémentation du contrôleur

Après avoir créé une nouvelle API et un nouveau contrôleur, vous pouvez implémenter la logique du contrôleur.

Procédure

- Dans cet exemple, remplacez les contrôleurs de fichiers de contrôleur générés/memcached_controller.go par une implémentation d'exemple suivante:

Exemple 5.1. Exemple memcached_controller.go

```
/*
Copyright 2020.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
*/

package controllers

import (
	appsv1 "k8s.io/api/apps/v1"
	corev1 "k8s.io/api/core/v1"
	"k8s.io/apimachinery/pkg/api/errors"
	metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
	"k8s.io/apimachinery/pkg/types"
	"reflect"

	"context"

	"github.com/go-logr/logr"
	"k8s.io/apimachinery/pkg/runtime"
	ctrl "sigs.k8s.io/controller-runtime"
	"sigs.k8s.io/controller-runtime/pkg/client"
	ctrllog "sigs.k8s.io/controller-runtime/pkg/log"

	cachev1 "github.com/example-inc/memcached-operator/api/v1"
)

// MemcachedReconciler reconciles a Memcached object
type MemcachedReconciler struct {
	client.Client
	Log    logr.Logger
	Scheme *runtime.Scheme
}

//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds,verbs=get;list;watch
;create;update;patch;delete
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/status,verbs=get;up
date;patch
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/finalizers,verbs=up
```

```
ate
//
+kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;
patch;delete
// +kubebuilder:rbac:groups=core,resources=pods,verbs=get;list;

// Reconcile is part of the main kubernetes reconciliation loop which aims to
// move the current state of the cluster closer to the desired state.
// TODO(user): Modify the Reconcile function to compare the state specified by
// the Memcached object against the actual cluster state, and then
// perform operations to make the cluster state reflect the state specified by
// the user.
//
// For more details, check Reconcile and its Result here:
// - https://pkg.go.dev/sigs.k8s.io/controller-runtime@v0.7.0/pkg/reconcile
func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request)
(ctrl.Result, error) {
	//log := r.Log.WithValues("memcached", req.NamespacedName)
	log := ctrllog.FromContext(ctx)
	// Fetch the Memcached instance
	memcached := &cachev1.Memcached{}
	err := r.Get(ctx, req.NamespacedName, memcached)
	if err != nil {
		if errors.IsNotFound(err) {
			// Request object not found, could have been deleted after reconcile
request.
			// Owned objects are automatically garbage collected. For additional
cleanup logic use finalizers.
			// Return and don't requeue
			log.Info("Memcached resource not found. Ignoring since object must be
deleted")
			return ctrl.Result{}, nil
		}
		// Error reading the object - requeue the request.
		log.Error(err, "Failed to get Memcached")
		return ctrl.Result{}, err
	}

	// Check if the deployment already exists, if not create a new one
	found := &appsv1.Deployment{}
	err = r.Get(ctx, types.NamespacedName{Name: memcached.Name, Namespace:
memcached.Namespace}, found)
	if err != nil && errors.IsNotFound(err) {
		// Define a new deployment
		dep := r.deploymentForMemcached(memcached)
		log.Info("Creating a new Deployment", "Deployment.Namespace",
dep.Namespace, "Deployment.Name", dep.Name)
		err = r.Create(ctx, dep)
		if err != nil {
			log.Error(err, "Failed to create new Deployment",
"Deployment.Namespace", dep.Namespace, "Deployment.Name", dep.Name)
			return ctrl.Result{}, err
		}
		// Deployment created successfully - return and requeue
		return ctrl.Result{Requeue: true}, nil
	} else if err != nil {
```

```
            log.Error(err, "Failed to get Deployment")
            return ctrl.Result{}, err
        }

        // Ensure the deployment size is the same as the spec
        size := memcached.Spec.Size
        if *found.Spec.Replicas != size {
            found.Spec.Replicas = &size
            err = r.Update(ctx, found)
            if err != nil {
                log.Error(err, "Failed to update Deployment", "Deployment.Namespace",
found.Namespace, "Deployment.Name", found.Name)
                return ctrl.Result{}, err
            }
            // Spec updated - return and requeue
            return ctrl.Result{Requeue: true}, nil
        }

        // Update the Memcached status with the pod names
        // List the pods for this memcached's deployment
        podList := &corev1.PodList{}
        listOpts := []client.ListOption{
            client.InNamespace(memcached.Namespace),
            client.MatchingLabels(labelsForMemcached(memcached.Name)),
        }
        if err = r.List(ctx, podList, listOpts...); err != nil {
            log.Error(err, "Failed to list pods", "Memcached.Namespace",
memcached.Namespace, "Memcached.Name", memcached.Name)
            return ctrl.Result{}, err
        }
        podNames := getPodNames(podList.Items)

        // Update status.Nodes if needed
        if !reflect.DeepEqual(podNames, memcached.Status.Nodes) {
            memcached.Status.Nodes = podNames
            err := r.Status().Update(ctx, memcached)
            if err != nil {
                log.Error(err, "Failed to update Memcached status")
                return ctrl.Result{}, err
            }
        }

        return ctrl.Result{}, nil
}

// deploymentForMemcached returns a memcached Deployment object
func (r *MemcachedReconciler) deploymentForMemcached(m *cachev1.Memcached)
*appsv1.Deployment {
        ls := labelsForMemcached(m.Name)
        replicas := m.Spec.Size

        dep := &appsv1.Deployment{
            ObjectMeta: metav1.ObjectMeta{
                Name:      m.Name,
                Namespace: m.Namespace,
            },
```

```
                Spec: appsv1.DeploymentSpec{
                        Replicas: &replicas,
                        Selector: &metav1.LabelSelector{
                                MatchLabels: ls,
                        },
                        Template: corev1.PodTemplateSpec{
                                ObjectMeta: metav1.ObjectMeta{
                                        Labels: ls,
                                },
                                Spec: corev1.PodSpec{
                                        Containers: []corev1.Container{{
                                                Image:   "memcached:1.4.36-alpine",
                                                Name:    "memcached",
                                                Command: []string{"memcached", "-m=64", "-o", "modern",
"-v"},
                                                Ports: []corev1.ContainerPort{{
                                                        ContainerPort: 11211,
                                                        Name:          "memcached",
                                                }},
                                        }},
                                },
                        },
                },
        }
        // Set Memcached instance as the owner and controller
        ctrl.SetControllerReference(m, dep, r.Scheme)
        return dep
}

// labelsForMemcached returns the labels for selecting the resources
// belonging to the given memcached CR name.
func labelsForMemcached(name string) map[string]string {
        return map[string]string{"app": "memcached", "memcached_cr": name}
}

// getPodNames returns the pod names of the array of pods passed in
func getPodNames(pods []corev1.Pod) []string {
        var podNames []string
        for _, pod := range pods {
                podNames = append(podNames, pod.Name)
        }
        return podNames
}

// SetupWithManager sets up the controller with the Manager.
func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
        return ctrl.NewControllerManagedBy(mgr).
                For(&cachev1.Memcached{}).
                Owns(&appsv1.Deployment{}).
                Complete(r)
}
```

Le contrôleur d'exemple exécute la logique de réconciliation suivante pour chaque ressource personnalisée Memcached (CR):

- Créez un déploiement Memcached s'il n'existe pas.
- Assurez-vous que la taille de déploiement est la même que celle spécifiée par la spécification CR Memcached.
- Actualisez le statut de Memcached CR avec les noms des pods memcached.

Les sous-sections suivantes expliquent comment le contrôleur dans l'exemple de mise en œuvre surveille les ressources et comment la boucle de rapprochement est déclenchée. Ces sous-sections peuvent passer directement à Running the Operator.

5.3.1.4.1. Les ressources surveillées par le contrôleur

La fonction SetupWithManager() dans Controllers/memcached_controller.go spécifie comment le contrôleur est conçu pour regarder un CR et d'autres ressources qui sont détenues et gérées par ce contrôleur.

```
import (
 ...
 appsv1 "k8s.io/api/apps/v1"
 ...
)

func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
 return ctrl.NewControllerManagedBy(mgr).
  For(&cachev1.Memcached{}).
  Owns(&appsv1.Deployment{}).
  Complete(r)
}
```

Le NewControllerManagedBy() fournit un constructeur de contrôleur qui permet diverses configurations de contrôleur.

For(&cachev1.Memcached{}) spécifie le type Memcached comme ressource principale à regarder. Dans chaque événement Ajouter, mettre à jour ou Supprimer pour un type Memcached, la boucle de réconciliation reçoit un argument Demande réconciliée, qui se compose d'un espace de noms et d'une clé de nom, pour cet objet Memcached.

Owns(&appsv1.Deployment{}) spécifie le type de déploiement comme ressource secondaire à regarder. Dans chaque type de déploiement Ajouter, mettre à jour ou supprimer un événement, le gestionnaire d'événements cartographie chaque événement à une demande de rapprochement pour le propriétaire du déploiement. Dans ce cas, le propriétaire est l'objet Memcached pour lequel le déploiement a été créé.

5.3.1.4.2. Configurations du contrôleur

Il est possible d'initialiser un contrôleur en utilisant de nombreuses autres configurations utiles. À titre d'exemple:

- Définissez le nombre maximal de rapprochements simultanés pour le contrôleur en utilisant l'option MaxConcurrentReconciles, qui par défaut à 1:

```
func (r *MemcachedReconciler) SetupWithManager(mgr ctrl.Manager) error {
    return ctrl.NewControllerManagedBy(mgr).
        For(&cachev1.Memcached{}).
        Owns(&appsv1.Deployment{}).
```

```
        WithOptions(controller.Options{
            MaxConcurrentReconciles: 2,
        }).
        Complete(r)
}
```

- Filtrez les événements à l'aide de prédicats.
- Choisissez le type d'EventHandler pour changer la façon dont un événement montre se traduit pour concilier les demandes de la boucle de réconciliation. Dans le cas des relations avec les opérateurs qui sont plus complexes que les ressources primaires et secondaires, vous pouvez utiliser le gestionnaire EnqueueRequestsFromMapFunc pour transformer un événement de montre en un ensemble arbitraire de demandes de rapprochement.

Consultez le Builder et le contrôleur GoDocs en amont pour plus de détails sur ces configurations et d'autres.

5.3.1.4.3. Boucle de réconciliation

Chaque contrôleur a un objet réconciliateur avec une méthode Reconcile() qui implémente la boucle de réconciliation. La boucle de réconciliation est passée l'argument Demande, qui est un espace de noms et la clé de nom utilisé pour trouver l'objet de ressource primaire, Memcached, à partir du cache:

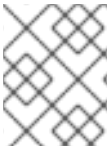
```
import (
 ctrl "sigs.k8s.io/controller-runtime"

 cachev1 "github.com/example-inc/memcached-operator/api/v1"
 ...
)

func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
  // Lookup the Memcached instance for this reconcile request
  memcached := &cachev1.Memcached{}
  err := r.Get(ctx, req.NamespacedName, memcached)
  ...
}
```

Basé sur les valeurs de retour, le résultat et l'erreur, la requête peut être requeued et la boucle de réconciliation peut être déclenchée à nouveau:

```
// Reconcile successful - don't requeue
return ctrl.Result{}, nil
// Reconcile failed due to error - requeue
return ctrl.Result{}, err
// Requeue for any reason other than an error
return ctrl.Result{Requeue: true}, nil
```

Après une période de grâce, vous pouvez définir le résultat.Requeue Après avoir requelé la demande:

```
import "time"

// Reconcile for any reason other than an error after 5 seconds
return ctrl.Result{RequeueAfter: time.Second*5}, nil
```



NOTE

Il est possible de retourner Résultat avec Requeue après ensemble pour réconcilier périodiquement un CR.

En savoir plus sur les réconciliateurs, les clients et les interactions avec les événements de ressources, consultez la documentation de l'API Controller Runtime Client.

5.3.1.4.4. Autorisations et manifestations RBAC

Le contrôleur nécessite certaines autorisations RBAC pour interagir avec les ressources qu'il gère. Ceux-ci sont spécifiés à l'aide de marqueurs RBAC, tels que les suivants:

```
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds,verbs=get;list;watch;create;upd
ate;patch;delete
//
+kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/status,verbs=get;update;patch

// +kubebuilder:rbac:groups=cache.example.com,resources=memcacheds/finalizers,verbs=update
//
+kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;patch;delete

// +kubebuilder:rbac:groups=core,resources=pods,verbs=get;list;

func (r *MemcachedReconciler) Reconcile(ctx context.Context, req ctrl.Request) (ctrl.Result, error) {
  ...
}
```

Le manifeste de l'objet ClusterRole à config/rbac/role.yaml est généré à partir des marqueurs précédents en utilisant l'utilitaire Controller-gen chaque fois que la commande make manifests est exécutée.

5.3.1.5. Activer le support proxy

Les auteurs d'opérateurs peuvent développer des opérateurs qui prennent en charge les proxys réseau. Les administrateurs dotés du rôle d'administrateur dédié configurent la prise en charge du proxy pour les variables d'environnement qui sont gérées par Operator Lifecycle Manager (OLM). Afin de prendre en charge les clusters proxiés, votre opérateur doit inspecter l'environnement pour les variables proxy standard suivantes et transmettre les valeurs à Operands:

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **AUCUN_PROXY**

NOTE

Ce tutoriel utilise HTTP_PROXY comme exemple de variable d'environnement.

Conditions préalables

- C'est un cluster doté d'un proxy de sortie à l'échelle du cluster activé.

Procédure

1. Éditez le fichier Controllers/memcached_controller.go pour inclure ce qui suit:

   a. Importer le paquet proxy à partir de la bibliothèque de l'opérateur-lib:

   ```
   import (
     ...
     "github.com/operator-framework/operator-lib/proxy"
   )
   ```

   b. Ajouter la fonction d'aide proxy.ReadProxyVarsFromEnv à la boucle de réconciliation et ajouter les résultats aux environnements Operand:

   ```
   for i, container := range dep.Spec.Template.Spec.Containers {
     dep.Spec.Template.Spec.Containers[i].Env = append(container.Env,
   proxy.ReadProxyVarsFromEnv()...)
   }
   ...
   ```

2. Définissez la variable d'environnement sur le déploiement de l'opérateur en ajoutant ce qui suit au fichier config/manager/manager.yaml:

   ```
   containers:
    - args:
     - --leader-elect
     - --leader-election-id=ansible-proxy-demo
     image: controller:latest
     name: manager
     env:
       - name: "HTTP_PROXY"
         value: "http_proxy_test"
   ```

5.3.1.6. Exécution de l'opérateur

Afin de construire et d'exécuter votre opérateur, utilisez l'opérateur SDK CLI pour regrouper votre opérateur, puis utilisez le gestionnaire de cycle de vie de l'opérateur (OLM) pour le déploiement sur le cluster.

NOTE

Lorsque vous souhaitez déployer votre opérateur sur un cluster OpenShift Container Platform au lieu d'un cluster dédié OpenShift, deux options de déploiement supplémentaires sont disponibles:

- Exécutez localement en dehors du cluster en tant que programme Go.
- Exécutez comme un déploiement sur le cluster.

NOTE

Avant d'exécuter votre opérateur Go-based en tant que paquet utilisant OLM, assurez-vous que votre projet a été mis à jour pour utiliser les images prises en charge.

Ressources supplémentaires

- Exécution locale en dehors du cluster (document OpenShift Container Platform)
- Exécution en tant que déploiement sur le cluster (document OpenShift Container Platform)

5.3.1.6.1. Groupement d'un opérateur et déploiement avec le gestionnaire du cycle de vie de l'opérateur

5.3.1.6.1.1. Groupement d'un opérateur

Le format de paquet Opérateur est la méthode d'emballage par défaut pour Operator SDK et Operator Lifecycle Manager (OLM). En utilisant le SDK de l'opérateur, vous pouvez préparer votre opérateur à une utilisation sur OLM pour construire et pousser votre projet Opérateur en tant qu'image groupée.

Conditions préalables

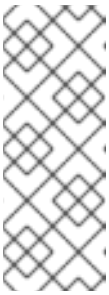
- L'opérateur SDK CLI installé sur un poste de travail de développement
- Installation d'OpenShift CLI (oc) v4+
- Le projet d'opérateur initialisé à l'aide du SDK de l'opérateur
- Dans le cas où votre opérateur est Go-based, votre projet doit être mis à jour pour utiliser les images prises en charge pour s'exécuter sur OpenShift Dedicated

Procédure

1. Exécutez les commandes suivantes dans votre répertoire de projet Opérateur pour construire et pousser l'image de votre opérateur. Modifiez l'argument IMG dans les étapes suivantes pour faire référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

   a. Construire l'image:

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   NOTE

   Le Dockerfile généré par le SDK pour l'opérateur renvoie explicitement GOARCH=amd64 pour la construction de go. Cela peut être modifié à GOARCH=$TARGETARCH pour les architectures non-AMD64. Docker définira automatiquement la variable d'environnement à la valeur spécifiée par -platform. Avec Buildah, le -build-arg devra être utilisé à cet effet. En savoir plus, consultez Multiple Architectures.

   b. Appuyez sur l'image vers un référentiel:

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

2. Créez votre paquet Opérateur manifeste en exécutant la commande make bundle, qui invoque plusieurs commandes, y compris l'opérateur SDK génère des paquets et des sous-commandes validant:

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Les manifestes de paquets pour un opérateur décrivent comment afficher, créer et gérer une application. La commande make bundle crée les fichiers et répertoires suivants dans votre projet Opérateur:

- Le bundle manifeste un répertoire nommé bundle/manifests qui contient un objet ClusterServiceVersion
- Annuaire de métadonnées groupé nommé bundle/metadata
- L'ensemble des définitions de ressources personnalisées (CRD) dans un répertoire config/crd
- Dockerfile bundle.Dockerfile

Ces fichiers sont ensuite automatiquement validés en utilisant le bundle opérateur-sdk valide pour s'assurer que la représentation des faisceaux sur disque est correcte.

3. Créez et poussez votre image de paquet en exécutant les commandes suivantes. L'OLM consomme des faisceaux d'opérateurs à l'aide d'une image d'index, qui référence à une ou plusieurs images groupées.

   a. Construisez l'image du bundle. Définissez BUNDLE_IMG avec les détails du registre, de l'espace de noms d'utilisateur et de la balise d'image où vous avez l'intention de pousser l'image:

   ```
   $ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
   ```

   b. Appuyez sur l'image du paquet:

   ```
   $ docker push <registry>/<user>/<bundle_image_name>:<tag>
   ```

5.3.1.6.1.2. Déploiement d'un opérateur avec le gestionnaire du cycle de vie de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) vous aide à installer, mettre à jour et gérer le cycle de vie des Opérateurs et de leurs services associés sur un cluster Kubernetes. Le système OLM est installé par défaut sur OpenShift Dedicated et s'exécute sous forme d'extension Kubernetes afin que vous puissiez utiliser la console Web et l'OpenShift CLI (oc) pour toutes les fonctions de gestion du cycle de vie de l'opérateur sans outils supplémentaires.

Le format de paquet opérateur est la méthode d'emballage par défaut pour l'opérateur SDK et OLM. Le SDK de l'opérateur permet d'exécuter rapidement une image groupée sur OLM afin de s'assurer qu'elle fonctionne correctement.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement
- Ensemble d'image de l'opérateur construit et poussé à un registre
- Installation OLM sur un cluster basé sur Kubernetes (v1.16.0 ou version ultérieure si vous utilisez apiextensions.k8s.io/v1 CRD, par exemple OpenShift Dedicated 4)

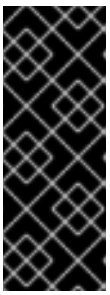
- Connexion au cluster avec oc à l'aide d'un compte avec des autorisations d'administration dédiées
- Dans le cas où votre opérateur est Go-based, votre projet doit être mis à jour pour utiliser les images prises en charge pour s'exécuter sur OpenShift Dedicated

Procédure

- Entrez la commande suivante pour exécuter l'opérateur sur le cluster:

```
$ operator-sdk run bundle \ 1
    -n <namespace> \ 2
    <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. La commande run bundle crée un catalogue basé sur des fichiers valide et installe le paquet Opérateur sur votre cluster en utilisant OLM.
2. Facultatif: Par défaut, la commande installe l'opérateur dans le projet actuellement actif dans votre fichier ~/.kube/config. Il est possible d'ajouter le drapeau -n pour définir un espace de noms différent pour l'installation.
3. Dans le cas où vous ne spécifiez pas une image, la commande utilise quay.io/operator-framework/opm:latest comme image d'index par défaut. Lorsque vous spécifiez une image, la commande utilise l'image du faisceau lui-même comme image d'index.



IMPORTANT

À partir d'OpenShift Dedicated 4.11, la commande run bundle prend en charge le format de catalogue basé sur des fichiers pour les catalogues Opérateur par défaut. Le format de base de données SQLite obsolète pour les catalogues d'opérateurs continue d'être pris en charge; cependant, il sera supprimé dans une version ultérieure. Il est recommandé aux auteurs de l'opérateur de migrer leurs flux de travail vers le format de catalogue basé sur les fichiers.

Cette commande effectue les actions suivantes:

- Créez une image d'index faisant référence à votre image de paquet. L'image de l'index est opaque et éphémère, mais reflète avec précision comment un paquet serait ajouté à un catalogue en production.
- Créez une source de catalogue qui pointe vers votre nouvelle image d'index, ce qui permet à OperatorHub de découvrir votre opérateur.
- Déployez votre opérateur dans votre cluster en créant un groupe d'opérateurs, un abonnement, un plan d'installation et toutes les autres ressources requises, y compris RBAC.

5.3.1.7. Créer une ressource personnalisée

Après l'installation de votre opérateur, vous pouvez le tester en créant une ressource personnalisée (CR) qui est maintenant fournie sur le cluster par l'opérateur.

Conditions préalables

- Exemple Memcached Operator, qui fournit le Memcached CR, installé sur un cluster

Procédure

1. Changer l'espace de noms où votre opérateur est installé. À titre d'exemple, si vous avez déployé l'opérateur à l'aide de la commande make deployment:

```
$ oc project memcached-operator-system
```

2. Éditer l'échantillon Memcached CR manifeste à config/samples/cache_v1_memcached.yaml pour contenir les spécifications suivantes:

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
...
spec:
...
  size: 3
```

3. Créer le CR:

```
$ oc apply -f config/samples/cache_v1_memcached.yaml
```

4. Assurez-vous que l'opérateur Memcached crée le déploiement de l'échantillon CR avec la bonne taille:

```
$ oc get deployments
```

Exemple de sortie

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           8m
memcached-sample                        3/3     3            3           1m
```

5. Consultez le statut des pods et CR pour confirmer que le statut est mis à jour avec les noms de pod de Memcached.

   a. Consultez les gousses:

```
$ oc get pods
```

Exemple de sortie

```
NAME                                READY     STATUS    RESTARTS   AGE
memcached-sample-6fd7c98d8-7dqdr    1/1       Running   0          1m
memcached-sample-6fd7c98d8-g5k7v    1/1       Running   0          1m
memcached-sample-6fd7c98d8-m7vn7    1/1       Running   0          1m
```

   b. Consultez l'état CR:

```
$ oc get memcached/memcached-sample -o yaml
```

Exemple de sortie

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
...
  name: memcached-sample
...
spec:
  size: 3
status:
  nodes:
  - memcached-sample-6fd7c98d8-7dqdr
  - memcached-sample-6fd7c98d8-g5k7v
  - memcached-sample-6fd7c98d8-m7vn7
```

6. Actualisez la taille du déploiement.

   a. Actualisez le fichier config/samples/cache_v1_memcached.yaml pour modifier le champ spec.size dans le CR Memcached de 3 à 5:

```
$ oc patch memcached memcached-sample \
    -p '{"spec":{"size": 5}}' \
    --type=merge
```

   b. Confirmez que l'opérateur modifie la taille du déploiement:

```
$ oc get deployments
```

Exemple de sortie

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           10m
memcached-sample                      5/5     5            5           3m
```

7. Supprimez le CR en exécutant la commande suivante:

```
$ oc delete -f config/samples/cache_v1_memcached.yaml
```

8. Nettoyez les ressources qui ont été créées dans le cadre de ce tutoriel.

   - Lorsque vous avez utilisé la commande make deployment pour tester l'opérateur, exécutez la commande suivante:

```
$ make undeploy
```

   - Lorsque vous avez utilisé la commande operator-sdk run bundle pour tester l'opérateur, exécutez la commande suivante:

```
$ operator-sdk cleanup <project_name>
```

5.3.1.8. Ressources supplémentaires

- Consultez la mise en page du projet pour les opérateurs basés sur Go pour en savoir plus sur les structures d'annuaire créées par le SDK de l'opérateur.
- Lorsqu'un proxy de sortie à l'échelle du cluster est configuré, les administrateurs ayant le rôle d'administrateur dédié peuvent outrepasser les paramètres proxy ou injecter un certificat CA personnalisé pour des opérateurs spécifiques fonctionnant sur Operator Lifecycle Manager (OLM).

5.3.2. Aménagement du projet pour les opérateurs Go-based

L'opérateur-sdk CLI peut générer, ou échafauder, un certain nombre de paquets et de fichiers pour chaque projet d'opérateur.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.3.2.1. Configuration du projet Go-based

Les projets Go-based Operator, le type par défaut, générés à l'aide de la commande operator-sdk init contiennent les fichiers et répertoires suivants:

Fichier ou répertoire	But
à propos de Main.go	Le programme principal de l'opérateur. Ceci instancie un nouveau gestionnaire qui enregistre toutes les définitions de ressources personnalisées (CRD) dans le répertoire apis/ et démarre tous les contrôleurs dans le répertoire contrôleurs / répertoire.

Fichier ou répertoire	But
APIs/	Arbre de répertoire qui définit les API des CRDs. Il faut modifier les fichiers apis/<version>/<kind>_types.go pour définir l'API pour chaque type de ressource et importer ces paquets dans vos contrôleurs pour surveiller ces types de ressources.
contrôleurs/	Implémentations de contrôleur. Éditez les fichiers Controller/<kind>_controller.go pour définir la logique de réconciliation du contrôleur pour gérer un type de ressource du type spécifié.
configuration/	Kubernetes se manifeste utilisé pour déployer votre contrôleur sur un cluster, y compris les CRD, RBAC et les certificats.
À propos de Makefile	Cibles utilisées pour construire et déployer votre contrôleur.
Dockerfile	Instructions utilisées par un moteur de conteneur pour construire votre opérateur.
les manifestes/	Kubernetes se manifeste pour l'enregistrement des CRD, la mise en place de RBAC et le déploiement de l'opérateur en tant que déploiement.

5.3.3. La mise à jour des projets d'opérateur Go-based pour les versions SDK plus récentes de l'opérateur

Le logiciel OpenShift Dedicated 4 prend en charge le SDK 1.38.0 de l'opérateur. Lorsque vous disposez déjà du 1.36.1 CLI installé sur votre poste de travail, vous pouvez mettre à jour le CLI à 1.38.0 en installant la dernière version.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Cependant, pour s'assurer que vos projets d'opérateur existants maintiennent la compatibilité avec le SDK 1.38.0 de l'opérateur, des étapes de mise à jour sont nécessaires pour les modifications de rupture associées introduites depuis 1.36.1. Les étapes de mise à jour doivent être exécutées manuellement dans l'un de vos projets Opérateurs qui ont été précédemment créés ou maintenus avec 1.36.1.

5.3.3.1. La mise à jour des projets d'opérateurs Go-based pour l'opérateur SDK 1.38.0

La procédure suivante met à jour un projet existant d'opérateur basé sur Go pour la compatibilité avec 1.38.0.

Conditions préalables

- L'opérateur SDK 1.38.0 installé
- Création ou maintenance d'un projet opérateur avec l'opérateur SDK 1.36.1

Procédure

1. Éditez le Makefile de votre projet Opérateur pour mettre à jour la version SDK de l'opérateur vers 1.38.0, comme indiqué dans l'exemple suivant:

Exemple de Makefile

```
# Set the Operator SDK version to use. By default, what is installed on the system is used.
# This is useful for CI or a project to utilize a specific version of the operator-sdk toolkit.
OPERATOR_SDK_VERSION ?= v1.38.0 (1)
```

(1) Changer la version de 1.36.1 à 1.38.0.

2. Il faut mettre à niveau les versions Kubernetes de votre projet Opérateur pour utiliser 1.30 et Kubebuilder v4.

ASTUCE

Cette mise à jour comprend des changements complexes d'échafaudage en raison de l'élimination du kube-rbac-proxy. Lorsque ces migrations deviennent difficiles à suivre, échafauder un nouveau projet d'échantillon à des fins de comparaison.

a. Actualisez votre fichier go.mod avec les modifications suivantes pour mettre à jour vos dépendances:

```
go 1.22.0

github.com/onsi/ginkgo/v2 v2.17.1
github.com/onsi/gomega v1.32.0
k8s.io/api v0.30.1
k8s.io/apimachinery v0.30.1
k8s.io/client-go v0.30.1
sigs.k8s.io/controller-runtime v0.18.4
```

b. Les dépendances mises à niveau sont téléchargées en exécutant la commande suivante:

```
$ go mod tidy
```

c. Actualisez votre Makefile avec les modifications suivantes:

```
- ENVTEST_K8S_VERSION = 1.29.0
+ ENVTEST_K8S_VERSION = 1.30.0
```

```
- KUSTOMIZE ?= $(LOCALBIN)/kustomize-$(KUSTOMIZE_VERSION)
- CONTROLLER_GEN ?= $(LOCALBIN)/controller-gen-
$(CONTROLLER_TOOLS_VERSION)
- ENVTEST ?= $(LOCALBIN)/setup-envtest-$(ENVTEST_VERSION)
- GOLANGCI_LINT = $(LOCALBIN)/golangci-lint-$(GOLANGCI_LINT_VERSION)
+ KUSTOMIZE ?= $(LOCALBIN)/kustomize
+ CONTROLLER_GEN ?= $(LOCALBIN)/controller-gen
+ ENVTEST ?= $(LOCALBIN)/setup-envtest
+ GOLANGCI_LINT = $(LOCALBIN)/golangci-lint
```

```
- KUSTOMIZE_VERSION ?= v5.3.0
- CONTROLLER_TOOLS_VERSION ?= v0.14.0
- ENVTEST_VERSION ?= release-0.17
- GOLANGCI_LINT_VERSION ?= v1.57.2
+ KUSTOMIZE_VERSION ?= v5.4.2
+ CONTROLLER_TOOLS_VERSION ?= v0.15.0
+ ENVTEST_VERSION ?= release-0.18
+ GOLANGCI_LINT_VERSION ?= v1.59.1
```

```
- $(call go-install-tool,$(GOLANGCI_LINT),github.com/golangci/golangci-
lint/cmd/golangci-lint,${GOLANGCI_LINT_VERSION})
+ $(call go-install-tool,$(GOLANGCI_LINT),github.com/golangci/golangci-
lint/cmd/golangci-lint,$(GOLANGCI_LINT_VERSION))
```

```
- $(call go-install-tool,$(GOLANGCI_LINT),github.com/golangci/golangci-lint/cmd/golangci-lint,${GOLANGCI_LINT_VERSION})
+ $(call go-install-tool,$(GOLANGCI_LINT),github.com/golangci/golangci-lint/cmd/golangci-lint,$(GOLANGCI_LINT_VERSION))
```

```
- @[ -f $(1) ] || { \
+ @[ -f "$(1)-$(3)" ] || { \
  echo "Downloading $${package}" ;\
+ rm -f $(1) || true ;\
- mv "$$(echo "$(1)" | sed "s/-$(3)$$//")" $(1) ;\
- }
+ mv $(1) $(1)-$(3) ;\
+ } ;\
+ ln -sf $(1)-$(3) $(1)
```

d. Actualisez votre fichier .golangci.yml avec les modifications suivantes:

```
-  exportloopref
+     - ginkgolinter
      - prealloc
+     - revive
+
+ linters-settings:
+   revive:
+     rules:
+       - name: comment-spacings
```

e. Actualisez votre Dockerfile avec les modifications suivantes:

```
- FROM golang:1.21 AS builder
+ FROM golang:1.22 AS builder
```

f. Actualisez votre fichier main.go avec les modifications suivantes:

```
     "sigs.k8s.io/controller-runtime/pkg/log/zap"
+    "sigs.k8s.io/controller-runtime/pkg/metrics/filters"

     var enableHTTP2 bool
-    flag.StringVar(&metricsAddr, "metrics-bind-address", ":8080", "The address the metric endpoint binds to.")
+    var tlsOpts []func(*tls.Config)
+    flag.StringVar(&metricsAddr, "metrics-bind-address", "0", "The address the metrics endpoint binds to. "+
+        "Use :8443 for HTTPS or :8080 for HTTP, or leave as 0 to disable the metrics service.")
     flag.StringVar(&probeAddr, "health-probe-bind-address", ":8081", "The address the probe endpoint binds to.")
     flag.BoolVar(&enableLeaderElection, "leader-elect", false,
         "Enable leader election for controller manager. "+
             "Enabling this will ensure there is only one active controller manager.")
-    flag.BoolVar(&secureMetrics, "metrics-secure", false,
-        "If set the metrics endpoint is served securely")
+    flag.BoolVar(&secureMetrics, "metrics-secure", true,
+        "If set, the metrics endpoint is served securely via HTTPS. Use --metrics-
```

```
secure=false to use HTTP instead.")

-    tlsOpts := []func(*tls.Config){}

+    // Metrics endpoint is enabled in 'config/default/kustomization.yaml'. The Metrics
options configure the server.
+    // More info:
+    // - https://pkg.go.dev/sigs.k8s.io/controller-runtime@v0.18.4/pkg/metrics/server
+    // - https://book.kubebuilder.io/reference/metrics.html
+    metricsServerOptions := metricsserver.Options{
+        BindAddress:   metricsAddr,
+        SecureServing: secureMetrics,
+        // TODO(user): TLSOpts is used to allow configuring the TLS config used for the
server. If certificates are
+        // not provided, self-signed certificates will be generated by default. This option is
not recommended for
+        // production environments as self-signed certificates do not offer the same level of
trust and security
+        // as certificates issued by a trusted Certificate Authority (CA). The primary risk is
potentially allowing
+        // unauthorized access to sensitive metrics data. Consider replacing with CertDir,
CertName, and KeyName
+        // to provide certificates, ensuring the server communicates using trusted and
secure certificates.
+        TLSOpts: tlsOpts,
+    }
+
+    if secureMetrics {
+        // FilterProvider is used to protect the metrics endpoint with authn/authz.
+        // These configurations ensure that only authorized users and service accounts
+        // can access the metrics endpoint. The RBAC are configured in
'config/rbac/kustomization.yaml'. More info:
+        // https://pkg.go.dev/sigs.k8s.io/controller-
runtime@v0.18.4/pkg/metrics/filters#WithAuthenticationAndAuthorization
+        metricsServerOptions.FilterProvider = filters.WithAuthenticationAndAuthorization
+    }
+
     mgr, err := ctrl.NewManager(ctrl.GetConfigOrDie(), ctrl.Options{
-        Scheme: scheme,
-        Metrics: metricsserver.Options{
-            BindAddress:   metricsAddr,
-            SecureServing: secureMetrics,
-            TLSOpts:       tlsOpts,
-        },
+        Scheme:                 scheme,
+        Metrics:                metricsServerOptions,
```

g. Actualisez votre fichier config/default/kustomization.yaml avec les modifications suivantes:

```
  # [PROMETHEUS] To enable prometheus monitor, uncomment all sections with
'PROMETHEUS'.
  #- ../prometheus
+ # [METRICS] Expose the controller manager metrics service.
+ - metrics_service.yaml

+ # Uncomment the patches line if you enable Metrics, and/or are using webhooks and
```

```
cert-manager
  patches:
- # Protect the /metrics endpoint by putting it behind auth.
- # If you want your controller-manager to expose the /metrics
- # endpoint w/o any authn/z, please comment the following line.
- - path: manager_auth_proxy_patch.yaml
+ # [METRICS] The following patch will enable the metrics endpoint using HTTPS and
the port :8443.
+ # More info: https://book.kubebuilder.io/reference/metrics
+ - path: manager_metrics_patch.yaml
+   target:
+     kind: Deployment
```

h. Enlevez les fichiers config/default/manager_auth_proxy_patch.yaml et config/default/manager_config_patch.yaml.

i. Créez un fichier config/default/manager_metrics_patch.yaml avec le contenu suivant:

```
# This patch adds the args to allow exposing the metrics endpoint using HTTPS
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-bind-address=:8443
```

j. Créez un fichier config/default/metrics_service.yaml avec le contenu suivant:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    control-plane: controller-manager
    app.kubernetes.io/name: <operator-name>
    app.kubernetes.io/managed-by: kustomize
  name: controller-manager-metrics-service
  namespace: system
spec:
  ports:
    - name: https
      port: 8443
      protocol: TCP
      targetPort: 8443
  selector:
    control-plane: controller-manager
```

k. Actualisez votre fichier config/manager/manager.yaml avec les modifications suivantes:

```
  - --leader-elect
+ - --health-probe-bind-address=:8081
```

l. Actualisez votre fichier config/prometheus/monitor.yaml avec les modifications suivantes:

```
    - path: /metrics
-     port: https
+     port: https # Ensure this is the name of the port that exposes HTTPS metrics
      tlsConfig:
+       # TODO(user): The option insecureSkipVerify: true is not recommended for
```

```
production since it disables
+       # certificate verification. This poses a significant security risk by making the system
vulnerable to
+       # man-in-the-middle attacks, where an attacker could intercept and manipulate the
communication between
+       # Prometheus and the monitored services. This could lead to unauthorized access
to sensitive metrics data,
+       # compromising the integrity and confidentiality of the information.
+       # Please use the following options for secure configurations:
+       # caFile: /etc/metrics-certs/ca.crt
+       # certFile: /etc/metrics-certs/tls.crt
+       # keyFile: /etc/metrics-certs/tls.key
        insecureSkipVerify: true
```

m. Supprimez les fichiers suivants du répertoire config/rbac/:

- **auth_proxy_client_clusterrole.yaml**
- **auth_proxy_role.yaml**
- **auth_proxy_role_binding.yaml**
- **auth_proxy_service.yaml**

n. Actualisez votre fichier config/rbac/kustomization.yaml avec les modifications suivantes:

```
  - leader_election_role_binding.yaml
- # Comment the following 4 lines if you want to disable
- # the auth proxy (https://github.com/brancz/kube-rbac-proxy)
- # which protects your /metrics endpoint.
- - auth_proxy_service.yaml
- - auth_proxy_role.yaml
- - auth_proxy_role_binding.yaml
- - auth_proxy_client_clusterrole.yaml
+ # The following RBAC configurations are used to protect
+ # the metrics endpoint with authn/authz. These configurations
+ # ensure that only authorized users and service accounts
+ # can access the metrics endpoint. Comment the following
+ # permissions if you want to disable this protection.
+ # More info: https://book.kubebuilder.io/reference/metrics.html
+ - metrics_auth_role.yaml
+ - metrics_auth_role_binding.yaml
+ - metrics_reader_role.yaml
```

o. Créez un fichier config/rbac/metrics_auth_role_binding.yaml avec le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metrics-auth-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metrics-auth-role
subjects:
```

```
- kind: ServiceAccount
  name: controller-manager
  namespace: system
```

p. Créez un fichier config/rbac/metrics_reader_role.yaml avec le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: metrics-reader
rules:
- nonResourceURLs:
  - "/metrics"
  verbs:
  - get
```

5.3.3.2. Ressources supplémentaires

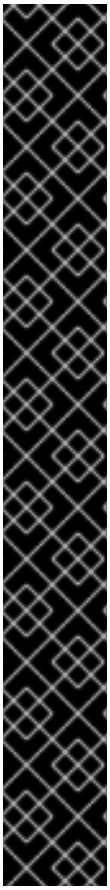
- La mise à jour des projets Go-based pour l'opérateur SDK 1.36.1 (OpenShift dédié 4.17)
- Les projets de manifestation de paquets migratoires au format de paquetage

5.4. OPÉRATEURS BASÉS SUR ANSIBLE

5.4.1. Didacticiel d'opérateur SDK pour les opérateurs basés sur Ansible

Les développeurs d'opérateurs peuvent profiter du support Ansible dans le SDK de l'opérateur pour créer un exemple d'opérateur Ansible pour Memcached, un magasin à valeur clé distribué, et gérer son cycle de vie. Ce tutoriel passe par le processus suivant:

- Créer un déploiement Memcached
- Assurez-vous que la taille de déploiement est la même que celle spécifiée par la spécification Memcached Custom Resource (CR)
- Actualisez le statut de Memcached CR en utilisant l'auteur de statut avec les noms des pods memcached



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Ce processus est réalisé en utilisant deux pièces maîtresses du Cadre opérateur:

Le SDK de l'opérateur

L'outil operator-sdk CLI et l'API de bibliothèque de contrôleurs

Gestionnaire du cycle de vie de l'opérateur (OLM)

Installation, mise à niveau et contrôle d'accès basé sur les rôles (RBAC) des opérateurs sur un cluster

NOTE

Ce tutoriel va plus en détail que commencer avec Operator SDK pour les opérateurs basés sur Ansible dans la documentation OpenShift Container Platform.

5.4.1.1. Conditions préalables

- L'opérateur SDK CLI installé
- Installation d'OpenShift CLI (oc) 4+
- Ansible 2.15.0
- Ansible Runner 2.3.3+
- Ansible Runner HTTP Event Emitter plugin 1.0.0+
- 3,9 + Python
- Client Python Kubernetes

- Connecté à un cluster OpenShift dédié avec oc avec un compte qui dispose d'autorisations d'administration dédiées
- Afin de permettre au cluster de tirer l'image, le référentiel où vous poussez votre image doit être défini comme public, ou vous devez configurer une image pull secret

Ressources supplémentaires

- Installation de l'opérateur SDK CLI
- Débuter avec l'OpenShift CLI

5.4.1.2. Créer un projet

Faites appel à l'opérateur SDK CLI pour créer un projet appelé memcached-operator.

Procédure

1. Créer un répertoire pour le projet:

```
$ mkdir -p $HOME/projects/memcached-operator
```

2. Changement dans le répertoire:

```
$ cd $HOME/projects/memcached-operator
```

3. Exécutez la commande operator-sdk init avec le plugin ansible pour initialiser le projet:

```
$ operator-sdk init \
    --plugins=ansible \
    --domain=example.com
```

5.4.1.2.1. Fichier PROJET

Il y a parmi les fichiers générés par la commande operator-sdk init un fichier Kubebuilder PROJECT. Les commandes ultérieures de l'opérateur-sdk, ainsi que la sortie d'aide, qui sont exécutées à partir de la racine du projet lisent ce fichier et sont conscientes que le type de projet est Ansible. À titre d'exemple:

```
domain: example.com
layout:
- ansible.sdk.operatorframework.io/v1
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
projectName: memcached-operator
version: "3"
```

5.4.1.3. Création d'une API

Faites appel à l'opérateur SDK CLI pour créer une API Memcached.

Procédure

- Exécutez la commande suivante pour créer une API avec le cache de groupe, version, v1, et sortez Memcached:

```
$ operator-sdk create api \
  --group cache \
  --version v1 \
  --kind Memcached \
  --generate-role 1
```

1 Génère un rôle Ansible pour l'API.

Après avoir créé l'API, votre projet Opérateur se met à jour avec la structure suivante:

À propos de Memcached CRD

Inclut un échantillon de ressource Memcached

Gestionnaire

Le programme qui réconcilie l'état du cluster avec l'état souhaité en utilisant:

- Conciliateur, soit un rôle Ansible ou un playbook
- Fichier watch.yaml, qui connecte la ressource Memcached au rôle memcached Ansible

5.4.1.4. La modification du gestionnaire

Actualisez votre projet Opérateur pour fournir la logique de rapprochement, sous la forme d'un rôle Ansible, qui s'exécute chaque fois qu'une ressource Memcached est créée, mise à jour ou supprimée.

Procédure

1. Actualisez le fichier role/memcached/tasks/main.yml avec la structure suivante:

```
---
- name: start memcached
  k8s:
    definition:
      kind: Deployment
      apiVersion: apps/v1
      metadata:
        name: '{{ ansible_operator_meta.name }}-memcached'
        namespace: '{{ ansible_operator_meta.namespace }}'
      spec:
        replicas: "{{size}}"
        selector:
          matchLabels:
            app: memcached
        template:
          metadata:
            labels:
              app: memcached
          spec:
            containers:
            - name: memcached
              command:
```

```
        - memcached
        - -m=64
        - -o
        - modern
        - -v
        image: "docker.io/memcached:1.4.36-alpine"
        ports:
          - containerPort: 11211
```

Ce rôle memcached assure l'existence d'un déploiement mémcached et définit la taille du déploiement.

2. Définissez les valeurs par défaut pour les variables utilisées dans votre rôle Ansible en éditant le fichier role/memcached/defaults/main.yml:

```
---
# defaults file for Memcached
size: 1
```

3. Actualisez la ressource de l'échantillon Memcached dans le fichier config/samples/cache_v1_memcached.yaml avec la structure suivante:

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  labels:
    app.kubernetes.io/name: memcached
    app.kubernetes.io/instance: memcached-sample
    app.kubernetes.io/part-of: memcached-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: memcached-operator
  name: memcached-sample
spec:
  size: 3
```

Les paires clé-valeur dans la spécification de ressource personnalisée (CR) sont transmises à Ansible sous forme de variables supplémentaires.

NOTE

Les noms de toutes les variables dans le champ spec sont convertis en cas de serpent, c'est-à-dire minuscules avec un accent, par l'opérateur avant d'exécuter Ansible. À titre d'exemple, serviceAccount dans la spécification devient service_account dans Ansible.

Dans votre fichier watch.yaml, vous pouvez désactiver cette conversion en paramétrant l'optionnakCaseParameters. Il est recommandé d'effectuer une validation de type dans Ansible sur les variables pour vous assurer que votre application reçoit les entrées attendues.

5.4.1.5. Activer le support proxy

Les auteurs d'opérateurs peuvent développer des opérateurs qui prennent en charge les proxys réseau. Les administrateurs dotés du rôle d'administrateur dédié configurent la prise en charge du proxy pour les variables d'environnement qui sont gérées par Operator Lifecycle Manager (OLM). Afin de prendre

en charge les clusters proxiés, votre opérateur doit inspecter l'environnement pour les variables proxy standard suivantes et transmettre les valeurs à Operands:

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **AUCUN_PROXY**

NOTE

Ce tutoriel utilise HTTP_PROXY comme exemple de variable d'environnement.

Conditions préalables

- C'est un cluster doté d'un proxy de sortie à l'échelle du cluster activé.

Procédure

1. Ajoutez les variables d'environnement au déploiement en mettant à jour le fichier role/memcached/tasks/main.yml avec ce qui suit:

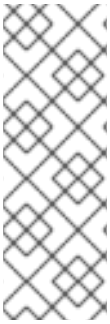
```
...
env:
  - name: HTTP_PROXY
    value: '{{ lookup("env", "HTTP_PROXY") | default("", True) }}'
  - name: http_proxy
    value: '{{ lookup("env", "HTTP_PROXY") | default("", True) }}'
...
```

2. Définissez la variable d'environnement sur le déploiement de l'opérateur en ajoutant ce qui suit au fichier config/manager/manager.yaml:

```
containers:
 - args:
   - --leader-elect
   - --leader-election-id=ansible-proxy-demo
   image: controller:latest
   name: manager
   env:
     - name: "HTTP_PROXY"
       value: "http_proxy_test"
```

5.4.1.6. Exécution de l'opérateur

Afin de construire et d'exécuter votre opérateur, utilisez l'opérateur SDK CLI pour regrouper votre opérateur, puis utilisez le gestionnaire de cycle de vie de l'opérateur (OLM) pour le déploiement sur le cluster.



NOTE

Lorsque vous souhaitez déployer votre opérateur sur un cluster OpenShift Container Platform au lieu d'un cluster dédié OpenShift, deux options de déploiement supplémentaires sont disponibles:

- Exécutez localement en dehors du cluster en tant que programme Go.
- Exécutez comme un déploiement sur le cluster.

Ressources supplémentaires

- Exécution locale en dehors du cluster (document OpenShift Container Platform)
- Exécution en tant que déploiement sur le cluster (document OpenShift Container Platform)

5.4.1.6.1. Groupement d'un opérateur et déploiement avec le gestionnaire du cycle de vie de l'opérateur

5.4.1.6.1.1. Groupement d'un opérateur

Le format de paquet Opérateur est la méthode d'emballage par défaut pour Operator SDK et Operator Lifecycle Manager (OLM). En utilisant le SDK de l'opérateur, vous pouvez préparer votre opérateur à une utilisation sur OLM pour construire et pousser votre projet Opérateur en tant qu'image groupée.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement
- Installation d'OpenShift CLI (oc) v4+
- Le projet d'opérateur initialisé à l'aide du SDK de l'opérateur

Procédure

1. Exécutez les commandes suivantes dans votre répertoire de projet Opérateur pour construire et pousser l'image de votre opérateur. Modifiez l'argument IMG dans les étapes suivantes pour faire référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

   a. Construire l'image:

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   NOTE

   Le Dockerfile généré par le SDK pour l'opérateur renvoie explicitement GOARCH=amd64 pour la construction de go. Cela peut être modifié à GOARCH=$TARGETARCH pour les architectures non-AMD64. Docker définira automatiquement la variable d'environnement à la valeur spécifiée par -platform. Avec Buildah, le -build-arg devra être utilisé à cet effet. En savoir plus, consultez Multiple Architectures.

   b. Appuyez sur l'image vers un référentiel:

```
$ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
```

2. Créez votre paquet Opérateur manifeste en exécutant la commande make bundle, qui invoque plusieurs commandes, y compris l'opérateur SDK génère des paquets et des sous-commandes validant:

```
$ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
```

Les manifestes de paquets pour un opérateur décrivent comment afficher, créer et gérer une application. La commande make bundle crée les fichiers et répertoires suivants dans votre projet Opérateur:

- Le bundle manifeste un répertoire nommé bundle/manifests qui contient un objet ClusterServiceVersion
- Annuaire de métadonnées groupé nommé bundle/metadata
- L'ensemble des définitions de ressources personnalisées (CRD) dans un répertoire config/crd
- Dockerfile bundle.Dockerfile

Ces fichiers sont ensuite automatiquement validés en utilisant le bundle opérateur-sdk valide pour s'assurer que la représentation des faisceaux sur disque est correcte.

3. Créez et poussez votre image de paquet en exécutant les commandes suivantes. L'OLM consomme des faisceaux d'opérateurs à l'aide d'une image d'index, qui référence à une ou plusieurs images groupées.

a. Construisez l'image du bundle. Définissez BUNDLE_IMG avec les détails du registre, de l'espace de noms d'utilisateur et de la balise d'image où vous avez l'intention de pousser l'image:

```
$ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
```

b. Appuyez sur l'image du paquet:

```
$ docker push <registry>/<user>/<bundle_image_name>:<tag>
```

5.4.1.6.1.2. Déploiement d'un opérateur avec le gestionnaire du cycle de vie de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) vous aide à installer, mettre à jour et gérer le cycle de vie des Opérateurs et de leurs services associés sur un cluster Kubernetes. Le système OLM est installé par défaut sur OpenShift Dedicated et s'exécute sous forme d'extension Kubernetes afin que vous puissiez utiliser la console Web et l'OpenShift CLI (oc) pour toutes les fonctions de gestion du cycle de vie de l'opérateur sans outils supplémentaires.

Le format de paquet opérateur est la méthode d'emballage par défaut pour l'opérateur SDK et OLM. Le SDK de l'opérateur permet d'exécuter rapidement une image groupée sur OLM afin de s'assurer qu'elle fonctionne correctement.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement

- Ensemble d'image de l'opérateur construit et poussé à un registre
- Installation OLM sur un cluster basé sur Kubernetes (v1.16.0 ou version ultérieure si vous utilisez apiextensions.k8s.io/v1 CRD, par exemple OpenShift Dedicated 4)
- Connexion au cluster avec oc à l'aide d'un compte avec des autorisations d'administration dédiées

Procédure

- Entrez la commande suivante pour exécuter l'opérateur sur le cluster:

```
$ operator-sdk run bundle \ 1
    -n <namespace> \ 2
    <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. La commande run bundle crée un catalogue basé sur des fichiers valide et installe le paquet Opérateur sur votre cluster en utilisant OLM.
2. Facultatif: Par défaut, la commande installe l'opérateur dans le projet actuellement actif dans votre fichier ~/.kube/config. Il est possible d'ajouter le drapeau -n pour définir un espace de noms différent pour l'installation.
3. Dans le cas où vous ne spécifiez pas une image, la commande utilise quay.io/operator-framework/opm:latest comme image d'index par défaut. Lorsque vous spécifiez une image, la commande utilise l'image du faisceau lui-même comme image d'index.

IMPORTANT

À partir d'OpenShift Dedicated 4.11, la commande run bundle prend en charge le format de catalogue basé sur des fichiers pour les catalogues Opérateur par défaut. Le format de base de données SQLite obsolète pour les catalogues d'opérateurs continue d'être pris en charge; cependant, il sera supprimé dans une version ultérieure. Il est recommandé aux auteurs de l'opérateur de migrer leurs flux de travail vers le format de catalogue basé sur les fichiers.

Cette commande effectue les actions suivantes:

- Créez une image d'index faisant référence à votre image de paquet. L'image de l'index est opaque et éphémère, mais reflète avec précision comment un paquet serait ajouté à un catalogue en production.
- Créez une source de catalogue qui pointe vers votre nouvelle image d'index, ce qui permet à OperatorHub de découvrir votre opérateur.
- Déployez votre opérateur dans votre cluster en créant un groupe d'opérateurs, un abonnement, un plan d'installation et toutes les autres ressources requises, y compris RBAC.

5.4.1.7. Créer une ressource personnalisée

Après l'installation de votre opérateur, vous pouvez le tester en créant une ressource personnalisée (CR) qui est maintenant fournie sur le cluster par l'opérateur.

Conditions préalables

- Exemple Memcached Operator, qui fournit le Memcached CR, installé sur un cluster

Procédure

1. Changer l'espace de noms où votre opérateur est installé. À titre d'exemple, si vous avez déployé l'opérateur à l'aide de la commande make deployment:

```
$ oc project memcached-operator-system
```

2. Éditer l'échantillon Memcached CR manifeste à config/samples/cache_v1_memcached.yaml pour contenir les spécifications suivantes:

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
...
spec:
...
  size: 3
```

3. Créer le CR:

```
$ oc apply -f config/samples/cache_v1_memcached.yaml
```

4. Assurez-vous que l'opérateur Memcached crée le déploiement de l'échantillon CR avec la bonne taille:

```
$ oc get deployments
```

Exemple de sortie

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
memcached-operator-controller-manager   1/1     1            1           8m
memcached-sample                        3/3     3            3           1m
```

5. Consultez le statut des pods et CR pour confirmer que le statut est mis à jour avec les noms de pod de Memcached.

   a. Consultez les gousses:

```
$ oc get pods
```

Exemple de sortie

```
NAME                                READY     STATUS    RESTARTS   AGE
memcached-sample-6fd7c98d8-7dqdr    1/1       Running   0          1m
memcached-sample-6fd7c98d8-g5k7v    1/1       Running   0          1m
memcached-sample-6fd7c98d8-m7vn7    1/1       Running   0          1m
```

   b. Consultez l'état CR:

```
$ oc get memcached/memcached-sample -o yaml
```

Exemple de sortie

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
...
  name: memcached-sample
...
spec:
  size: 3
status:
  nodes:
  - memcached-sample-6fd7c98d8-7dqdr
  - memcached-sample-6fd7c98d8-g5k7v
  - memcached-sample-6fd7c98d8-m7vn7
```

6. Actualisez la taille du déploiement.

   a. Actualisez le fichier config/samples/cache_v1_memcached.yaml pour modifier le champ spec.size dans le CR Memcached de 3 à 5:

   ```
   $ oc patch memcached memcached-sample \
       -p '{"spec":{"size": 5}}' \
       --type=merge
   ```

   b. Confirmez que l'opérateur modifie la taille du déploiement:

   ```
   $ oc get deployments
   ```

   Exemple de sortie

   ```
   NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
   memcached-operator-controller-manager   1/1     1            1           10m
   memcached-sample                        5/5     5            5           3m
   ```

7. Supprimez le CR en exécutant la commande suivante:

   ```
   $ oc delete -f config/samples/cache_v1_memcached.yaml
   ```

8. Nettoyez les ressources qui ont été créées dans le cadre de ce tutoriel.

   - Lorsque vous avez utilisé la commande make deployment pour tester l'opérateur, exécutez la commande suivante:

     ```
     $ make undeploy
     ```

   - Lorsque vous avez utilisé la commande operator-sdk run bundle pour tester l'opérateur, exécutez la commande suivante:

     ```
     $ operator-sdk cleanup <project_name>
     ```

5.4.1.8. Ressources supplémentaires

- Consultez la mise en page du projet pour les opérateurs basés sur Ansible pour en apprendre davantage sur les structures d'annuaire créées par le SDK de l'opérateur.
- Lorsqu'un proxy de sortie à l'échelle du cluster est configuré, les administrateurs ayant le rôle d'administrateur dédié peuvent outrepasser les paramètres proxy ou injecter un certificat CA personnalisé pour des opérateurs spécifiques fonctionnant sur Operator Lifecycle Manager (OLM).

5.4.2. Aménagement du projet pour les opérateurs basés sur Ansible

L'opérateur-sdk CLI peut générer, ou échafauder, un certain nombre de paquets et de fichiers pour chaque projet d'opérateur.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.4.2.1. Disposition de projet basée sur Ansible

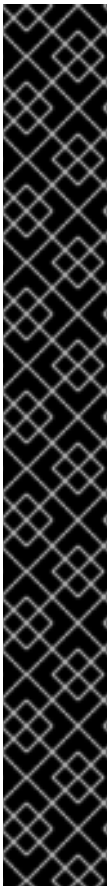
Les projets d'opérateur basés sur Ansible générés à l'aide de la commande operator-sdk init --plugins ansible contiennent les répertoires et fichiers suivants:

Fichier ou répertoire	But
Dockerfile	Dockerfile pour la construction de l'image du conteneur pour l'opérateur.
À propos de Makefile	Cibles pour la construction, la publication, le déploiement de l'image du conteneur qui enveloppe le binaire Opérateur, et les cibles pour l'installation et la désinstallation de la définition de ressources personnalisées (CRD).

Fichier ou répertoire	But
LE PROJET	Fichier YAML contenant des informations de métadonnées pour l'opérateur.
configuration/crd	Les fichiers CRD de base et les paramètres de fichier kustomization.yaml.
configuration/défaut	Collecte tous les manifestes de l'opérateur pour le déploiement. À utiliser par la commande make deployment.
configuration/gestionnaire	Déploiement du gestionnaire de contrôleur.
configuration/prométhée	La ressource ServiceMonitor pour la surveillance de l'opérateur.
config/rbac	Lien de rôle et de rôle pour le proxy d'élection et d'authentification du leader.
configuration/échantillons	Échantillons de ressources créées pour les CRD.
configuration/test	Configurations d'échantillons pour les tests.
livres de lecture/	C'est un sous-répertoire pour que les playbooks s'exécutent.
les rôles/	Le sous-répertoire de l'arborescence des rôles s'exécute.
les montres.yaml	Groupe/version/type (GVK) des ressources à surveiller, et la méthode d'invocation Ansible. De nouvelles entrées sont ajoutées à l'aide de la commande create api.
exigences.yml	Fichier YAML contenant les collections Ansibles et les dépendances de rôles à installer lors d'une construction.
la molécule/	Des scénarios de molécules pour tester de bout en bout votre rôle et votre opérateur.

5.4.3. La mise à jour des projets pour les versions SDK plus récentes de l'opérateur

Le logiciel OpenShift Dedicated 4 prend en charge le SDK 1.38.0 de l'opérateur. Lorsque vous disposez déjà du 1.36.1 CLI installé sur votre poste de travail, vous pouvez mettre à jour le CLI à 1.38.0 en installant la dernière version.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Cependant, pour s'assurer que vos projets d'opérateur existants maintiennent la compatibilité avec le SDK 1.38.0 de l'opérateur, des étapes de mise à jour sont nécessaires pour les modifications de rupture associées introduites depuis 1.36.1. Les étapes de mise à jour doivent être exécutées manuellement dans l'un de vos projets Opérateurs qui ont été précédemment créés ou maintenus avec 1.36.1.

5.4.3.1. La mise à jour des projets d'opérateurs accessibles pour l'opérateur SDK 1.38.0

La procédure suivante met à jour un projet existant d'opérateur basé sur Ansible pour la compatibilité avec 1.38.0.

Conditions préalables

- L'opérateur SDK 1.38.0 installé
- Création ou maintenance d'un projet opérateur avec l'opérateur SDK 1.36.1

Procédure

1. Éditez le Makefile de votre projet Opérateur pour mettre à jour la version SDK de l'opérateur vers 1.38.0, comme indiqué dans l'exemple suivant:

   Exemple de Makefile

   ```
   # Set the Operator SDK version to use. By default, what is installed on the system is used.
   # This is useful for CI or a project to utilize a specific version of the operator-sdk toolkit.
   OPERATOR_SDK_VERSION ?= v1.38.0 (1)
   ```

   (1) Changer la version de 1.36.1 à 1.38.0.

2. Éditez le Dockerfile de votre projet Opérateur pour mettre à jour la balise image ose-ansible-operator à 4, comme indiqué dans l'exemple suivant:

   Exemple Dockerfile

   ```
   FROM registry.redhat.io/openshift4/ose-ansible-operator:v4
   ```

3. Il faut mettre à niveau les versions Kubernetes de votre projet Opérateur pour utiliser 1.30 et Kubebuilder v4.

   ASTUCE

   Cette mise à jour comprend des changements complexes d'échafaudage en raison de l'élimination du kube-rbac-proxy. Lorsque ces migrations deviennent difficiles à suivre, échafauder un nouveau projet d'échantillon à des fins de comparaison.

   a. Actualisez la version Kustomize dans votre Makefile en apportant les modifications suivantes:

   ```
   - curl -sSLo - https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize/v5.3.0/kustomize_v5.3.0_$(OS)_$(ARCH).tar.gz | \
   + curl -sSLo - https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize/v5.4.2/kustomize_v5.4.2_$(OS)_$(ARCH).tar.gz | \
   ```

   b. Actualisez votre fichier config/default/kustomization.yaml avec les modifications suivantes:

   ```
     # [PROMETHEUS] To enable prometheus monitor, uncomment all sections with 'PROMETHEUS'.
     #- ../prometheus
   + # [METRICS] Expose the controller manager metrics service.
   + - metrics_service.yaml

   + # Uncomment the patches line if you enable Metrics, and/or are using webhooks and cert-manager
     patches:
   - # Protect the /metrics endpoint by putting it behind auth.
   - # If you want your controller-manager to expose the /metrics
   - # endpoint w/o any authn/z, please comment the following line.
   - - path: manager_auth_proxy_patch.yaml
   + # [METRICS] The following patch will enable the metrics endpoint using HTTPS and the port :8443.
   + # More info: https://book.kubebuilder.io/reference/metrics
   + - path: manager_metrics_patch.yaml
   +   target:
   +     kind: Deployment
   ```

   c. Enlevez les fichiers config/default/manager_auth_proxy_patch.yaml et config/default/manager_config_patch.yaml.

   d. Créez un fichier config/default/manager_metrics_patch.yaml avec le contenu suivant:

   ```
   # This patch adds the args to allow exposing the metrics endpoint using HTTPS
   ```

```
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-bind-address=:8443
# This patch adds the args to allow securing the metrics endpoint
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-secure
# This patch adds the args to allow RBAC-based authn/authz the metrics endpoint
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-require-rbac
```

e. Créez un fichier config/default/metrics_service.yaml avec le contenu suivant:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    control-plane: controller-manager
    app.kubernetes.io/name: <operator-name>
    app.kubernetes.io/managed-by: kustomize
  name: controller-manager-metrics-service
  namespace: system
spec:
  ports:
    - name: https
      port: 8443
      protocol: TCP
      targetPort: 8443
  selector:
    control-plane: controller-manager
```

f. Actualisez votre fichier config/manager/manager.yaml avec les modifications suivantes:

```
  - --leader-elect
+ - --health-probe-bind-address=:6789
```

g. Actualisez votre fichier config/prometheus/monitor.yaml avec les modifications suivantes:

```
     - path: /metrics
-      port: https
+      port: https # Ensure this is the name of the port that exposes HTTPS metrics
       tlsConfig:
+        # TODO(user): The option insecureSkipVerify: true is not recommended for
production since it disables
+        # certificate verification. This poses a significant security risk by making the system
vulnerable to
+        # man-in-the-middle attacks, where an attacker could intercept and manipulate the
communication between
+        # Prometheus and the monitored services. This could lead to unauthorized access
to sensitive metrics data,
+        # compromising the integrity and confidentiality of the information.
+        # Please use the following options for secure configurations:
+        # caFile: /etc/metrics-certs/ca.crt
```

```
+        # certFile: /etc/metrics-certs/tls.crt
+        # keyFile: /etc/metrics-certs/tls.key
         insecureSkipVerify: true
```

h. Supprimez les fichiers suivants du répertoire config/rbac/:

- **auth_proxy_client_clusterrole.yaml**
- **auth_proxy_role.yaml**
- **auth_proxy_role_binding.yaml**
- **auth_proxy_service.yaml**

i. Actualisez votre fichier config/rbac/kustomization.yaml avec les modifications suivantes:

```
  - leader_election_role_binding.yaml
- # Comment the following 4 lines if you want to disable
- # the auth proxy (https://github.com/brancz/kube-rbac-proxy)
- # which protects your /metrics endpoint.
- - auth_proxy_service.yaml
- - auth_proxy_role.yaml
- - auth_proxy_role_binding.yaml
- - auth_proxy_client_clusterrole.yaml
+ # The following RBAC configurations are used to protect
+ # the metrics endpoint with authn/authz. These configurations
+ # ensure that only authorized users and service accounts
+ # can access the metrics endpoint. Comment the following
+ # permissions if you want to disable this protection.
+ # More info: https://book.kubebuilder.io/reference/metrics.html
+ - metrics_auth_role.yaml
+ - metrics_auth_role_binding.yaml
+ - metrics_reader_role.yaml
```

j. Créez un fichier config/rbac/metrics_auth_role_binding.yaml avec le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metrics-auth-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metrics-auth-role
subjects:
  - kind: ServiceAccount
    name: controller-manager
    namespace: system
```

k. Créez un fichier config/rbac/metrics_reader_role.yaml avec le contenu suivant:

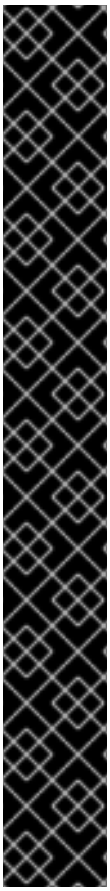
```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: metrics-reader
```

```
rules:
- nonResourceURLs:
  - "/metrics"
  verbs:
  - get
```

5.4.3.2. Ressources supplémentaires

- La mise à jour des projets d'opérateurs accessibles pour l'opérateur SDK 1.36.1 (OpenShift Dédié 4.17)
- Les projets de manifestation de paquets migratoires au format de paquetage

5.4.4. Assistance Ansible dans le SDK de l'opérateur



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.4.4.1. Fichiers de ressources personnalisés

Les opérateurs utilisent le mécanisme d'extension Kubernetes, les définitions de ressources personnalisées (CRD), de sorte que votre ressource personnalisée (CR) ressemble et agit comme les objets Kubernetes natifs intégrés.

Le format de fichier CR est un fichier de ressource Kubernetes. L'objet a des champs obligatoires et facultatifs:

Tableau 5.1. Champs de ressources personnalisés

Le champ	Description
apiVersion	La version du CR à créer.
kind	C'est une sorte de CR à créer.
les métadonnées	Des métadonnées spécifiques à Kubernetes à créer.
caractéristiques (facultatif)	Liste de valeurs clés des variables qui sont transmises à Ansible. Ce champ est vide par défaut.
status	Il résume l'état actuel de l'objet. Dans le cas des opérateurs basés sur Ansible, la sous-ressource d'état est activée pour les CRD et gérée par l'opérateur_sdk.util.k8s_status module Ansible par défaut, qui inclut des informations de condition au statut CR.
annotations	Les annotations spécifiques à Kubernetes doivent être ajoutées au CR.

La liste suivante des annotations CR modifie le comportement de l'Opérateur:

Tableau 5.2. Annotations d'opérateur basées sur Ansible

Annotation	Description
Ansible.operator-sdk/réconcile-période	Indique l'intervalle de rapprochement pour le CR. Cette valeur est analysée en utilisant le temps standard du paquet Golang. En particulier, ParseDuration est utilisé qui applique le suffixe par défaut de s, donnant la valeur en quelques secondes.

Exemple Ansible-basé sur l'annotation de l'opérateur

```
apiVersion: "test1.example.com/v1alpha1"
kind: "Test1"
metadata:
  name: "example"
annotations:
  ansible.operator-sdk/reconcile-period: "30s"
```

5.4.4.2. fichier Watch.yaml

Groupe/version/type (GVK) est un identifiant unique pour une API Kubernetes. Le fichier watch.yaml contient une liste de cartographies à partir de ressources personnalisées (CRs), identifiées par son GVK, à un rôle ou un playbook Ansible. L'opérateur s'attend à ce fichier mappage dans un emplacement prédéfini à /opt/ansible/watches.yaml.

Tableau 5.3. cartographie des fichiers Watch.yaml

Le champ	Description
groupe de travail	Groupe de CR à regarder.
la version	La version de CR à regarder.
kind	Genre de CR à regarder
le rôle (par défaut)	Chemin vers le rôle Ansible ajouté au conteneur. À titre d'exemple, si votre répertoire de rôles est à /opt/ansible/roles/ et que votre rôle est nommé busybox, cette valeur serait /opt/ansible/roles/busybox. Ce champ s'exclut mutuellement avec le champ de playbook.
livre de lecture	Chemin vers le livre de lecture Ansible ajouté au conteneur. Ce livre de lecture devrait être un moyen d'appeler des rôles. Ce champ s'exclut mutuellement avec le domaine des rôles.
concilier la Période (facultatif)	L'intervalle de réconciliation, la fréquence à laquelle le rôle ou le playbook est exécuté, pour un CR donné.
GérerStatus (facultatif)	Lorsqu'il est défini sur true (par défaut), l'opérateur gère le statut du CR génériquement. Lorsqu'il est défini sur false, le statut du CR est géré ailleurs, par le rôle ou le playbook spécifié ou dans un contrôleur séparé.

Exemple watch.yaml fichier

```
- version: v1alpha1 (1)
  group: test1.example.com
  kind: Test1
  role: /opt/ansible/roles/Test1

- version: v1alpha1 (2)
  group: test2.example.com
  kind: Test2
  playbook: /opt/ansible/playbook.yml

- version: v1alpha1 (3)
  group: test3.example.com
  kind: Test3
  playbook: /opt/ansible/test3.yml
  reconcilePeriod: 0
  manageStatus: false
```

(1) Exemple simple de cartographie Test1 au rôle test1.

(2) Exemple simple de mappage Test2 vers un playbook.

(3) Exemple plus complexe pour le type Test3. Désactive la re-requête et la gestion du statut CR dans le livre de lecture.

5.4.4.2.1. Des options avancées

Les fonctionnalités avancées peuvent être activées en les ajoutant à votre fichier watch.yaml par GVK. Ils peuvent aller sous le groupe, la version, le genre et le playbook ou les champs de rôle.

Certaines fonctionnalités peuvent être remplacées par ressource à l'aide d'une annotation sur ce CR. Les options qui peuvent être remplacées ont l'annotation spécifiée ci-dessous.

Tableau 5.4. Advanced watch.yaml options de fichier

Caractéristique	Clé YAML	Description	Annotation pour la surcharge	La valeur par défaut
Concilier la période	**concilier la période**	Le temps entre réconcilier court pour un CR particulier.	**Ansible.operator-sdk/réconcile-période**	**1M**
Gérer le statut	**gérer le statut**	Il permet à l'opérateur de gérer la section conditions de chaque section d'état CR.		**C'est vrai**
Surveiller les ressources dépendantes	**les ressources de WatchDependentResources**	Il permet à l'opérateur de surveiller dynamiquement les ressources créées par Ansible.		**C'est vrai**
Examiner les ressources en grappes	**accueil > WatchClusterScopedResources**	Il permet à l'opérateur de regarder les ressources en grappes créées par Ansible.		**faux**
Artéfacts Max runner	**à propos de maxRunnerArtifacts**	Gère le nombre de répertoires d'artefacts qu'Ansible Runner conserve dans le conteneur de l'opérateur pour chaque ressource individuelle.	**Ansible.operator-sdk/max-runner-artefacts**	**20**

Exemple de fichier watch.yml avec des options avancées

```
- version: v1alpha1
  group: app.example.com
  kind: AppService
  playbook: /opt/ansible/playbook.yml
  maxRunnerArtifacts: 30
```

```
  reconcilePeriod: 5s
  manageStatus: False
  watchDependentResources: False
```

5.4.4.3. Des variables supplémentaires envoyées à Ansible

Des variables supplémentaires peuvent être envoyées à Ansible, qui sont ensuite gérées par l'opérateur. La section Spécification de la ressource personnalisée (CR) passe le long des paires clé-valeur en tant que variables supplémentaires. Cela équivaut à des variables supplémentaires passées à la commande ansible-playbook.

L'opérateur passe également le long de variables supplémentaires sous le champ méta pour le nom du CR et l'espace de noms du CR.

C) pour l'exemple suivant:

```
apiVersion: "app.example.com/v1alpha1"
kind: "Database"
metadata:
  name: "example"
spec:
  message: "Hello world 2"
  newParameter: "newParam"
```

La structure passée à Ansible en tant que variables supplémentaires est:

```
{ "meta": {
        "name": "<cr_name>",
        "namespace": "<cr_namespace>",
  },
  "message": "Hello world 2",
  "new_parameter": "newParam",
  "_app_example_com_database": {
     <full_crd>
   },
}
```

Le message et les champs newParamètre sont définis dans le niveau supérieur en tant que variables supplémentaires, et méta fournit les métadonnées pertinentes pour le CR telles que définies dans l'opérateur. Les champs méta peuvent être consultés à l'aide de la notation par point dans Ansible, par exemple:

```
---
- debug:
    msg: "name: {{ ansible_operator_meta.name }}, {{ ansible_operator_meta.namespace }}"
```

5.4.4.4. Ansible Runner Ansible Ansible

Ansible Runner conserve des informations sur les courses Ansible dans le conteneur. Ceci est situé à /tmp/ansible-operator/runner/<group>/<version>/<kind>/<namespace>/<name>.

Ressources supplémentaires

- En savoir plus sur le répertoire des coureurs, consultez la documentation Ansible Runner.

5.4.5. Collection Kubernetes pour Ansible

Afin de gérer le cycle de vie de votre application sur Kubernetes à l'aide d'Ansible, vous pouvez utiliser la collection Kubernetes pour Ansible. Cette collection de modules Ansible permet à un développeur de tirer parti de ses fichiers de ressources Kubernetes existants écrits dans YAML ou d'exprimer la gestion du cycle de vie dans Ansible natif.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

L'un des plus grands avantages de l'utilisation d'Ansible en conjonction avec les fichiers de ressources Kubernetes existants est la possibilité d'utiliser Jinja tentant afin que vous puissiez personnaliser les ressources avec la simplicité de quelques variables dans Ansible.

Cette section détaille l'utilisation de la collection Kubernetes. Installez la collection sur votre poste de travail local et testez-la à l'aide d'un livre de lecture avant de l'utiliser au sein d'un opérateur.

5.4.5.1. Installation de la collection Kubernetes pour Ansible

Il est possible d'installer la collection Kubernetes pour Ansible sur votre poste de travail local.

Procédure

1. Installer Ansible 2.15+:

```
$ sudo dnf install ansible
```

2. Installez le package client Python Kubernetes:

```
$ pip install kubernetes
```

3. Installez la collection Kubernetes en utilisant l'une des méthodes suivantes:

   - La collection peut être installée directement à partir d'Ansible Galaxy:

     ```
     $ ansible-galaxy collection install community.kubernetes
     ```

   - Lorsque vous avez déjà initialisé votre opérateur, vous pourriez avoir un fichier requirements.yml au niveau supérieur de votre projet. Ce fichier spécifie les dépendances anonymes qui doivent être installées pour que votre opérateur fonctionne. Ce fichier installe par défaut la collection community.kubernetes ainsi que la collection operator_sdk.util, qui fournit des modules et des plugins pour les fonctions spécifiques à l'opérateur.
     Installer les modules dépendants à partir du fichier requirements.yml:

     ```
     $ ansible-galaxy collection install -r requirements.yml
     ```

5.4.5.2. Tester la collection Kubernetes localement

Les développeurs d'opérateurs peuvent exécuter le code Ansible à partir de leur machine locale plutôt que d'exécuter et de reconstruire l'opérateur à chaque fois.

Conditions préalables

- Initialiser un projet d'opérateur basé sur Ansible et créer une API qui a un rôle Ansible généré en utilisant le SDK de l'opérateur
- Installer la collection Kubernetes pour Ansible

Procédure

1. Dans votre répertoire de projet Opérateur Ansible, modifiez le fichier role/<kind>/tasks/main.yml avec la logique Ansible que vous souhaitez. Le répertoire role/<kind>/ est créé lorsque vous utilisez le drapeau --generate-role lors de la création d'une API. Le <kind> remplaçable correspond au type que vous avez spécifié pour l'API. L'exemple suivant crée et supprime une carte de configuration basée sur la valeur d'une variable nommée état:

   ```
   ---
   - name: set ConfigMap example-config to {{ state }}
     community.kubernetes.k8s:
       api_version: v1
       kind: ConfigMap
       name: example-config
       namespace: <operator_namespace> 1
       state: "{{ state }}"
     ignore_errors: true 2
   ```

   1 Indiquez l'espace de noms où vous souhaitez créer la carte de configuration.

   2 La configuration d'ignorer_erreurs: true s'assure que la suppression d'une carte de configuration inexistante ne échoue pas.

2. Changer le fichier role/<kind>/defaults/main.yml pour définir l'état à présenter par défaut:

```
---
state: present
```

3. Créez un playbook Ansible en créant un fichier playbook.yml dans le niveau supérieur de votre répertoire de projet, et incluez votre rôle <kind>:

```
---
- hosts: localhost
  roles:
    - <kind>
```

4. Exécutez le livre de lecture:

```
$ ansible-playbook playbook.yml
```

Exemple de sortie

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit
localhost does not match 'all'

PLAY [localhost] ***************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [memcached : set ConfigMap example-config to present]
*********************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

5. Assurez-vous que la carte de configuration a été créée:

```
$ oc get configmaps
```

Exemple de sortie

```
NAME             DATA   AGE
example-config    0     2m1s
```

6. Exécutez à nouveau l'état de réglage du livre de lecture à l'absence:

```
$ ansible-playbook playbook.yml --extra-vars state=absent
```

Exemple de sortie

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit
localhost does not match 'all'
```

```
PLAY [localhost] ***************************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [localhost]

TASK [memcached : set ConfigMap example-config to absent]
*********************************************************************************
changed: [localhost]

PLAY RECAP *********************************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

7. Assurez-vous que la carte de configuration a été supprimée:

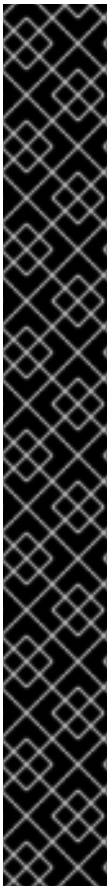
```
$ oc get configmaps
```

5.4.5.3. Les prochaines étapes

- Consultez Utiliser Ansible à l'intérieur d'un opérateur pour plus de détails sur le déclenchement de votre logique personnalisée Ansible à l'intérieur d'un opérateur lorsqu'une ressource personnalisée (CR) change.

5.4.6. En utilisant Ansible à l'intérieur d'un opérateur

Après avoir connu l'utilisation locale de la collection Kubernetes pour Ansible, vous pouvez déclencher la même logique Ansible à l'intérieur d'un opérateur lorsqu'une ressource personnalisée (CR) change. Cet exemple cartographie un rôle Ansible à une ressource spécifique de Kubernetes que l'opérateur regarde. Ce mappage se fait dans le fichier watch.yaml.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.4.6.1. Fichiers de ressources personnalisés

Les opérateurs utilisent le mécanisme d'extension Kubernetes, les définitions de ressources personnalisées (CRD), de sorte que votre ressource personnalisée (CR) ressemble et agit comme les objets Kubernetes natifs intégrés.

Le format de fichier CR est un fichier de ressource Kubernetes. L'objet a des champs obligatoires et facultatifs:

Tableau 5.5. Champs de ressources personnalisés

Le champ	Description
apiVersion	La version du CR à créer.
kind	C'est une sorte de CR à créer.
les métadonnées	Des métadonnées spécifiques à Kubernetes à créer.
caractéristiques (facultatif)	Liste de valeurs clés des variables qui sont transmises à Ansible. Ce champ est vide par défaut.
status	Il résume l'état actuel de l'objet. Dans le cas des opérateurs basés sur Ansible, la sous-ressource d'état est activée pour les CRD et gérée par l'opérateur_sdk.util.k8s_status module Ansible par défaut, qui inclut des informations de condition au statut CR.
annotations	Les annotations spécifiques à Kubernetes doivent être ajoutées au CR.

Le champ	Description

La liste suivante des annotations CR modifie le comportement de l'Opérateur:

Tableau 5.6. Annotations d'opérateur basées sur Ansible

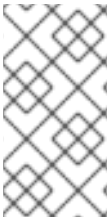
Annotation	Description
Ansible.operator-sdk/réconcile-période	Indique l'intervalle de rapprochement pour le CR. Cette valeur est analysée en utilisant le temps standard du paquet Golang. En particulier, ParseDuration est utilisé qui applique le suffixe par défaut de s, donnant la valeur en quelques secondes.

Exemple Ansible-basé sur l'annotation de l'opérateur

```
apiVersion: "test1.example.com/v1alpha1"
kind: "Test1"
metadata:
  name: "example"
annotations:
  ansible.operator-sdk/reconcile-period: "30s"
```

5.4.6.2. Tester un opérateur basé sur Ansible localement

Il est possible de tester la logique à l'intérieur d'un opérateur basé sur Ansible qui s'exécute localement en utilisant la commande make run à partir du répertoire de haut niveau de votre projet Opérateur. La cible Makefile exécute localement le binaire ansible-operator, qui lit à partir du fichier watch.yaml et utilise votre fichier ~/.kube/config pour communiquer avec un cluster Kubernetes tout comme le font les modules k8s.



NOTE

Il est possible de personnaliser le chemin des rôles en définissant la variable d'environnement ANSIBLE_ROLES_PATH ou en utilisant le drapeau ansible-roles-path. Dans le cas où le rôle n'est pas trouvé dans la valeur ANSIBLE_ROLES_PATH, l'opérateur le recherche dans {{annuaire de courant}}/roles.

Conditions préalables

- Ansible Runner v2.3.3+
- Ansible Runner HTTP Event Emitter plugin v1.0.0+
- A effectué les étapes précédentes pour tester la collection Kubernetes localement

Procédure

1. Installez votre définition de ressources personnalisées (CRD) et les définitions appropriées du contrôle d'accès basé sur les rôles (RBAC) pour votre ressource personnalisée (CR):

```
$ make install
```

Exemple de sortie

```
/usr/bin/kustomize build config/crd | kubectl apply -f -
customresourcedefinition.apiextensions.k8s.io/memcacheds.cache.example.com created
```

2. Exécutez la commande make run:

```
$ make run
```

Exemple de sortie

```
/home/user/memcached-operator/bin/ansible-operator run
{"level":"info","ts":1612739145.2871568,"logger":"cmd","msg":"Version","Go
Version":"go1.15.5","GOOS":"linux","GOARCH":"amd64","ansible-
operator":"v1.10.1","commit":"1abf57985b43bf6a59dcd18147b3c574fa57d3f6"}
...
{"level":"info","ts":1612739148.347306,"logger":"controller-runtime.metrics","msg":"metrics
server is starting to listen","addr":":8080"}
{"level":"info","ts":1612739148.3488882,"logger":"watches","msg":"Environment variable not
set; using default
value","envVar":"ANSIBLE_VERBOSITY_MEMCACHED_CACHE_EXAMPLE_COM","default":
2}
{"level":"info","ts":1612739148.3490262,"logger":"cmd","msg":"Environment variable not set;
using default
value","Namespace":"","envVar":"ANSIBLE_DEBUG_LOGS","ANSIBLE_DEBUG_LOGS":fals
e}
{"level":"info","ts":1612739148.3490646,"logger":"ansible-controller","msg":"Watching
resource","Options.Group":"cache.example.com","Options.Version":"v1","Options.Kind":"Memc
ached"}
{"level":"info","ts":1612739148.350217,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
{"level":"info","ts":1612739148.3506632,"logger":"controller-runtime.manager","msg":"starting
metrics server","path":"/metrics"}
{"level":"info","ts":1612739148.350784,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612739148.5511978,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612739148.5512562,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting workers","worker
count":8}
```

Avec l'opérateur qui surveille maintenant votre CR pour les événements, la création d'un CR déclenchera votre rôle Ansible à exécuter.

NOTE

Considérez un exemple de config/samples/<gvk>.yaml CR manifeste:

```
apiVersion: <group>.example.com/v1alpha1
kind: <kind>
metadata:
  name: "<kind>-sample"
```

Comme le champ spec n'est pas défini, Ansible est invoqué sans variables supplémentaires. Le passage de variables supplémentaires d'un CR à Ansible est couvert dans une autre section. Il est important de fixer des défauts raisonnables pour l'opérateur.

3. Créez une instance de votre CR avec l'état variable par défaut défini pour présenter:

```
$ oc apply -f config/samples/<gvk>.yaml
```

4. Assurez-vous que la carte de configuration example-config a été créée:

```
$ oc get configmaps
```

Exemple de sortie

```
NAME                 STATUS    AGE
example-config         Active    3s
```

5. Modifiez votre fichier config/samples/<gvk>.yaml pour définir le champ d'état à absent. À titre d'exemple:

```
apiVersion: cache.example.com/v1
kind: Memcached
metadata:
  name: memcached-sample
spec:
  state: absent
```

6. Appliquer les modifications:

```
$ oc apply -f config/samples/<gvk>.yaml
```

7. Confirmez que la carte de configuration est supprimée:

```
$ oc get configmap
```

5.4.6.3. Essai d'un opérateur Ansible sur le cluster

Après avoir testé votre logique Ansible personnalisée localement à l'intérieur d'un opérateur, vous pouvez tester l'opérateur à l'intérieur d'un pod sur un cluster dédié OpenShift, qui est préféré pour une utilisation de production.

En tant que déploiement sur votre cluster, vous pouvez exécuter votre projet Opérateur.

Procédure

1. Exécutez les commandes suivantes pour construire et pousser l'image de l'opérateur. Modifiez l'argument IMG dans les étapes suivantes pour faire référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

   a. Construire l'image:

   ```
   $ make docker-build IMG=<registry>/<user>/<image_name>:<tag>
   ```

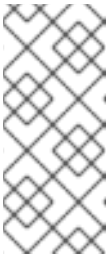
   

   NOTE

   Le Dockerfile généré par le SDK pour l'opérateur renvoie explicitement GOARCH=amd64 pour la construction de go. Cela peut être modifié à GOARCH=$TARGETARCH pour les architectures non-AMD64. Docker définira automatiquement la variable d'environnement à la valeur spécifiée par -platform. Avec Buildah, le -build-arg devra être utilisé à cet effet. En savoir plus, consultez Multiple Architectures.

   b. Appuyez sur l'image vers un référentiel:

   ```
   $ make docker-push IMG=<registry>/<user>/<image_name>:<tag>
   ```

   NOTE

   Le nom et la balise de l'image, par exemple IMG=<registry>/<user>/<image_name>:<tag>, dans les deux commandes peuvent également être définis dans votre Makefile. Changez la valeur IMG ?= Controller:dernière valeur pour définir votre nom d'image par défaut.

2. Exécutez la commande suivante pour déployer l'opérateur:

   ```
   $ make deploy IMG=<registry>/<user>/<image_name>:<tag>
   ```

   Cette commande crée un espace de noms avec le nom de votre projet Opérateur dans le formulaire <project_name>-system et est utilisée pour le déploiement. Cette commande installe également les manifestes RBAC à partir de config/rbac.

3. Exécutez la commande suivante pour vérifier que l'opérateur est en cours d'exécution:

   ```
   $ oc get deployment -n <project_name>-system
   ```

   Exemple de sortie

   ```
   NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
   <project_name>-controller-manager       1/1     1            1           8m
   ```

5.4.6.4. Journaux Ansibles

Les opérateurs basés sur Ansible fournissent des journaux sur l'exécution Ansible, ce qui peut être utile pour déboger vos tâches Ansible. Les journaux peuvent également contenir des informations détaillées sur les internes de l'opérateur et ses interactions avec Kubernetes.

5.4.6.4.1. Affichage des journaux Ansibles

Conditions préalables

- Opérateur basé sur Ansible s'exécutant comme déploiement sur un cluster

Procédure

- Afin d'afficher les journaux d'un opérateur basé sur Ansible, exécutez la commande suivante:

```
$ oc logs deployment/<project_name>-controller-manager \
    -c manager \ 1
    -n <namespace> 2
```

1. Afficher les journaux du conteneur gestionnaire.
2. Lorsque vous avez utilisé la commande make deployment pour exécuter l'opérateur en tant que déploiement, utilisez l'espace de noms <project_name>-system.

Exemple de sortie

```
{"level":"info","ts":1612732105.0579333,"logger":"cmd","msg":"Version","Go
Version":"go1.15.5","GOOS":"linux","GOARCH":"amd64","ansible-
operator":"v1.10.1","commit":"1abf57985b43bf6a59dcd18147b3c574fa57d3f6"}
{"level":"info","ts":1612732105.0587437,"logger":"cmd","msg":"WATCH_NAMESPACE
environment variable not set. Watching all namespaces.","Namespace":""}
I0207 21:08:26.110949       7 request.go:645] Throttling request took 1.035521578s, request:
GET:https://172.30.0.1:443/apis/flowcontrol.apiserver.k8s.io/v1alpha1?timeout=32s
{"level":"info","ts":1612732107.768025,"logger":"controller-runtime.metrics","msg":"metrics
server is starting to listen","addr":"127.0.0.1:8080"}
{"level":"info","ts":1612732107.768796,"logger":"watches","msg":"Environment variable not
set; using default
value","envVar":"ANSIBLE_VERBOSITY_MEMCACHED_CACHE_EXAMPLE_COM","default":
2}
{"level":"info","ts":1612732107.7688773,"logger":"cmd","msg":"Environment variable not set;
using default
value","Namespace":"","envVar":"ANSIBLE_DEBUG_LOGS","ANSIBLE_DEBUG_LOGS":fals
e}
{"level":"info","ts":1612732107.7688901,"logger":"ansible-controller","msg":"Watching
resource","Options.Group":"cache.example.com","Options.Version":"v1","Options.Kind":"Memc
ached"}
{"level":"info","ts":1612732107.770032,"logger":"proxy","msg":"Starting to
serve","Address":"127.0.0.1:8888"}
I0207 21:08:27.770185       7 leaderelection.go:243] attempting to acquire leader lease
memcached-operator-system/memcached-operator...
{"level":"info","ts":1612732107.770202,"logger":"controller-runtime.manager","msg":"starting
metrics server","path":"/metrics"}
I0207 21:08:27.784854       7 leaderelection.go:253] successfully acquired lease
memcached-operator-system/memcached-operator
{"level":"info","ts":1612732107.7850506,"logger":"controller-
```

```
runtime.manager.controller.memcached-controller","msg":"Starting
EventSource","source":"kind source: cache.example.com/v1, Kind=Memcached"}
{"level":"info","ts":1612732107.8853772,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting Controller"}
{"level":"info","ts":1612732107.8854098,"logger":"controller-
runtime.manager.controller.memcached-controller","msg":"Starting workers","worker
count":4}
```

5.4.6.4.2. Activer des résultats complets Ansibles dans les journaux

La variable d'environnement ANSIBLE_DEBUG_LOGS sur True permet de vérifier le résultat complet Ansible dans les journaux, ce qui peut être utile lors du débogage.

Procédure

- Éditer les fichiers config/manager/manager.yaml et config/default/manager_metrics_patch.yaml pour inclure la configuration suivante:

```
containers:
- name: manager
  env:
  - name: ANSIBLE_DEBUG_LOGS
    value: "True"
```

5.4.6.4.3. Activer le débogage verbeux dans les journaux

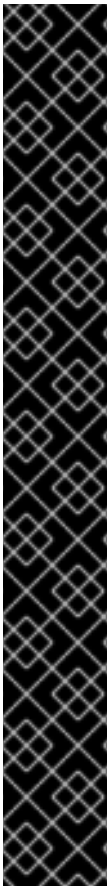
Lors du développement d'un opérateur basé sur Ansible, il peut être utile d'activer un débogage supplémentaire dans les journaux.

Procédure

- Ajoutez l'annotation ansible.sdk.operatorframework.io/verbosity à votre ressource personnalisée pour activer le niveau de verbosité que vous souhaitez. À titre d'exemple:

```
apiVersion: "cache.example.com/v1alpha1"
kind: "Memcached"
metadata:
  name: "example-memcached"
  annotations:
    "ansible.sdk.operatorframework.io/verbosity": "4"
spec:
  size: 4
```

5.4.7. Gestion de l'état des ressources personnalisées



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.4.7.1. À propos de l'état des ressources personnalisées dans Ansible-based Operators

Les opérateurs basés sur Ansible mettent automatiquement à jour les sous-ressources d'état des ressources personnalisées (CR) avec des informations génériques sur l'exécution Ansible précédente. Cela inclut le nombre de tâches réussies et échouées et les messages d'erreur pertinents comme indiqué:

```
status:
  conditions:
  - ansibleResult:
      changed: 3
      completion: 2018-12-03T13:45:57.13329
      failures: 1
      ok: 6
      skipped: 0
    lastTransitionTime: 2018-12-03T13:45:57Z
    message: 'Status code was -1 and not [200]: Request failed: <urlopen error [Errno
      113] No route to host>'
    reason: Failed
    status: "True"
    type: Failure
  - lastTransitionTime: 2018-12-03T13:46:13Z
    message: Running reconciliation
    reason: Running
    status: "True"
    type: Running
```

Les opérateurs basés sur Ansible permettent également aux auteurs de l'opérateur de fournir des valeurs d'état personnalisées avec le module k8s_status Ansible, qui est inclus dans la collection operator_sdk.util. Cela permet à l'auteur de mettre à jour l'état à partir de l'intérieur d'Ansible avec

n'importe quelle paire clé-valeur comme souhaité.

Les opérateurs basés sur Ansible incluent toujours la sortie générique d'exécution Ansible comme indiqué ci-dessus. Lorsque vous préférez que votre application n'ait pas mis à jour l'état avec la sortie Ansible, vous pouvez suivre l'état manuellement à partir de votre application.

5.4.7.2. Le suivi manuel de l'état des ressources personnalisées

La collection operator_sdk.util permet de modifier votre opérateur Ansible afin de suivre manuellement l'état de la ressource personnalisée (CR) depuis votre application.

Conditions préalables

- Le projet d'opérateur basé sur Ansible créé à l'aide du SDK de l'opérateur

Procédure

1. Actualisez le fichier watch.yaml avec un champ ManagStatus défini à false:

```
- version: v1
  group: api.example.com
  kind: <kind>
  role: <role>
  manageStatus: false
```

2. Utilisez le module operator_sdk.util.k8s_status Ansible pour mettre à jour la sous-ressource. À titre d'exemple, pour mettre à jour avec les données clés de test et de valeur, operator_sdk.util peut être utilisé comme indiqué:

```
- operator_sdk.util.k8s_status:
    api_version: app.example.com/v1
    kind: <kind>
    name: "{{ ansible_operator_meta.name }}"
    namespace: "{{ ansible_operator_meta.namespace }}"
    status:
      test: data
```

3. Il est possible de déclarer les collections dans le fichier meta/main.yml pour le rôle, qui est inclus pour les opérateurs échafaudés basés sur Ansible:

```
collections:
  - operator_sdk.util
```

4. Après avoir déclaré les collections dans la méta de rôle, vous pouvez invoquer le module k8s_status directement:

```
k8s_status:
  ...
  status:
    key1: value1
```

5.5. OPÉRATEURS BASÉS SUR LE BARREAU

5.5.1. Didacticiel SDK opérateur pour les opérateurs basés sur Helm

Les développeurs d'opérateurs peuvent profiter du support Helm dans le SDK de l'opérateur pour construire un exemple d'opérateur basé sur Helm pour Nginx et gérer son cycle de vie. Ce tutoriel passe par le processus suivant:

- Créer un déploiement Nginx
- Assurez-vous que la taille de déploiement est la même que celle spécifiée par la spécification de ressources personnalisées Nginx (CR)
- Actualisez le statut Nginx CR en utilisant l'auteur de statut avec les noms des nginx pods

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

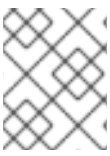
Ce processus est réalisé à l'aide de deux pièces maîtresses du cadre opérateur:

Le SDK de l'opérateur

L'outil operator-sdk CLI et l'API de bibliothèque de contrôleurs

Gestionnaire du cycle de vie de l'opérateur (OLM)

Installation, mise à niveau et contrôle d'accès basé sur les rôles (RBAC) des opérateurs sur un cluster



NOTE

Ce tutoriel va plus en détail que commencer avec Operator SDK pour les opérateurs basés sur Helm dans la documentation OpenShift Container Platform.

5.5.1.1. Conditions préalables

- L'opérateur SDK CLI installé

- Installation d'OpenShift CLI (oc) 4+
- Connecté à un cluster OpenShift dédié avec oc avec un compte qui dispose d'autorisations d'administration dédiées
- Afin de permettre au cluster de tirer l'image, le référentiel où vous poussez votre image doit être défini comme public, ou vous devez configurer une image pull secret

Ressources supplémentaires

- Installation de l'opérateur SDK CLI
- Débuter avec l'OpenShift CLI

5.5.1.2. Créer un projet

Faites appel à l'opérateur SDK CLI pour créer un projet appelé nginx-operator.

Procédure

1. Créer un répertoire pour le projet:

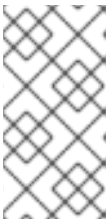
```
$ mkdir -p $HOME/projects/nginx-operator
```

2. Changement dans le répertoire:

```
$ cd $HOME/projects/nginx-operator
```

3. Exécutez la commande operator-sdk init avec le plugin de barre pour initialiser le projet:

```
$ operator-sdk init \
    --plugins=helm \
    --domain=example.com \
    --group=demo \
    --version=v1 \
    --kind=Nginx
```



NOTE

Le plugin de barre initialise par défaut un projet à l'aide d'un graphique Helm en plaque de chaudière. Il est possible d'utiliser des drapeaux supplémentaires, tels que le drapeau --helm-chart, pour initialiser un projet à l'aide d'un graphique Helm existant.

La commande init crée le projet nginx-operator spécifiquement pour regarder une ressource avec la version API example.com/v1 et genre Nginx.

4. Dans le cas des projets Helm, la commande init génère les règles RBAC dans le fichier config/rbac/role.yaml en fonction des ressources qui seraient déployées par le manifeste par défaut pour le graphique. Assurez-vous que les règles générées dans ce fichier répondent aux exigences d'autorisation de l'opérateur.

5.5.1.2.1. Graphiques Helm existants

Au lieu de créer votre projet avec un graphique Helm, vous pouvez utiliser un graphique existant, que ce soit à partir de votre système de fichiers local ou d'un référentiel de graphiques distants, en utilisant les drapeaux suivants:

- **--helm-chart**
- **--helm-chart-repo**
- **--helm-chart-version**

Lorsque le drapeau --helm-chart est spécifié, les drapeaux --group, --version et --kind deviennent facultatifs. Dans le cas où unset est laissé, les valeurs par défaut suivantes sont utilisées:

Drapeau	La valeur
--domaine	**à propos de My.domain**
--groupe	**graphiques**
--version	**à propos de v1**
--le genre	Déduit du graphique spécifié

Lorsque l'indicateur --helm-chart spécifie une archive graphique locale, par exemple-chart-1.2.0.tgz, ou un répertoire, le graphique est validé et non emballé ou copié dans le projet. Dans le cas contraire, le SDK de l'opérateur tente de récupérer le graphique d'un référentiel distant.

Lorsqu'une URL de référentiel personnalisé n'est pas spécifiée par le drapeau --helm-chart-repo, les formats de référence de graphique suivants sont pris en charge:

Format	Description
<repo_name>/<chart_name>	Cherchez le graphique Helm nommé <chart_name> à partir du référentiel du graphique helm nommé <repo_name>, comme spécifié dans le fichier $HELM_HOME/repositories/repositories.yaml. À l'aide de la commande helm repo add pour configurer ce fichier.
<URL>	Chercher l'archive du graphique Helm à l'URL spécifiée.

Lorsqu'une URL de référentiel personnalisé est spécifiée par --helm-chart-repo, le format de référence graphique suivant est pris en charge:

Format	Description
<chart_name>	Cherchez le graphique Helm nommé <chart_name> dans le référentiel du graphique Helm spécifié par la valeur d'URL --helm-chart-repo.

Lorsque le drapeau --helm-chart-version n'est pas défini, le SDK de l'opérateur obtient la dernière version disponible du graphique Helm. Dans le cas contraire, il reprend la version spécifiée. L'option --

helm-chart-version flag n'est pas utilisée lorsque le graphique spécifié avec le drapeau --helm-chart se réfère à une version spécifique, par exemple lorsqu'il s'agit d'un chemin local ou d'une URL.

Afin de plus de détails et d'exemples, exécutez:

```
$ operator-sdk init --plugins helm --help
```

5.5.1.2.2. Fichier PROJET

Il y a parmi les fichiers générés par la commande operator-sdk init un fichier Kubebuilder PROJECT. Les commandes ultérieures de l'opérateur-sdk, ainsi que la sortie d'aide, qui sont exécutées à partir de la racine du projet lisent ce fichier et sont conscientes que le type de projet est Helm. À titre d'exemple:

```
domain: example.com
layout:
- helm.sdk.operatorframework.io/v1
plugins:
  manifests.sdk.operatorframework.io/v2: {}
  scorecard.sdk.operatorframework.io/v2: {}
  sdk.x-openshift.io/v1: {}
projectName: nginx-operator
resources:
- api:
    crdVersion: v1
    namespaced: true
  domain: example.com
  group: demo
  kind: Nginx
  version: v1
version: "3"
```

5.5.1.3. Comprendre la logique de l'opérateur

Dans cet exemple, le projet nginx-operator exécute la logique de réconciliation suivante pour chaque ressource personnalisée Nginx (CR):

- Créez un déploiement Nginx s'il n'existe pas.
- Créez un service Nginx s'il n'existe pas.
- Créez une entrée Nginx si elle est activée et n'existe pas.
- Assurez-vous que le déploiement, le service et l'entrée optionnelle correspondent à la configuration souhaitée comme spécifié par le Nginx CR, par exemple le nombre de répliques, l'image et le type de service.

Le projet nginx-operator regarde par défaut les événements de ressource Nginx comme indiqué dans le fichier watch.yaml et exécute les versions Helm en utilisant le graphique spécifié:

```
# Use the 'create api' subcommand to add watches to this file.
- group: demo
  version: v1
```

```
  kind: Nginx
  chart: helm-charts/nginx
# +kubebuilder:scaffold:watch
```

5.5.1.3.1. Exemple de graphique Helm

Lorsqu'un projet Helm Operator est créé, le SDK de l'opérateur crée un graphique Helm contenant un ensemble de modèles pour une version simple de Nginx.

Dans cet exemple, des modèles sont disponibles pour le déploiement, le service et les ressources d'entrée, ainsi qu'un modèle NOTES.txt, que les développeurs de graphiques Helm utilisent pour transmettre des informations utiles sur une version.

Consultez la documentation du développeur Helm si vous n'êtes pas déjà familier avec les graphiques Helm.

5.5.1.3.2. La modification de la spécification des ressources personnalisées

Helm utilise un concept appelé valeurs pour fournir des personnalisations aux valeurs par défaut d'un graphique Helm, qui sont définis dans le fichier values.yaml.

Il est possible de remplacer ces valeurs par défaut en définissant les valeurs souhaitées dans la spécification de ressource personnalisée (CR). Comme exemple, vous pouvez utiliser le nombre de répliques.

Procédure

1. Le fichier helm-charts/nginx/values.yaml a une valeur appelée replicaCount définie à 1 par défaut. Afin d'avoir deux instances Nginx dans votre déploiement, votre spécification CR doit contenir replicaCount : 2.
   Éditer le fichier config/samples/demo_v1_nginx.yaml pour définir replicaCount: 2:

   ```
   apiVersion: demo.example.com/v1
   kind: Nginx
   metadata:
     name: nginx-sample
   ...
   spec:
   ...
     replicaCount: 2
   ```

2. De même, le port de service par défaut est réglé sur 80. À l'aide de 8080, modifiez le fichier config/samples/demo_v1_nginx.yaml pour définir spec.port: 8080, qui ajoute le port de service de remplacement:

   ```
   apiVersion: demo.example.com/v1
   kind: Nginx
   metadata:
     name: nginx-sample
   spec:
     replicaCount: 2
     service:
       port: 8080
   ```

L'opérateur Helm applique l'ensemble de la spécification comme s'il s'agissait du contenu d'un fichier de valeurs, tout comme la commande helm install -f/overrides.yaml.

5.5.1.4. Activer le support proxy

Les auteurs d'opérateurs peuvent développer des opérateurs qui prennent en charge les proxys réseau. Les administrateurs dotés du rôle d'administrateur dédié configurent la prise en charge du proxy pour les variables d'environnement qui sont gérées par Operator Lifecycle Manager (OLM). Afin de prendre en charge les clusters proxiés, votre opérateur doit inspecter l'environnement pour les variables proxy standard suivantes et transmettre les valeurs à Operands:

- **HTTP_PROXY**
- **HTTPS_PROXY**
- **AUCUN_PROXY**

NOTE

Ce tutoriel utilise HTTP_PROXY comme exemple de variable d'environnement.

Conditions préalables

- C'est un cluster doté d'un proxy de sortie à l'échelle du cluster activé.

Procédure

1. Editez le fichier watch.yaml pour inclure des overrides en fonction d'une variable d'environnement en ajoutant le champ OverrideValues:

```
...
- group: demo.example.com
  version: v1alpha1
  kind: Nginx
  chart: helm-charts/nginx
  overrideValues:
    proxy.http: $HTTP_PROXY
...
```

2. Ajoutez la valeur proxy.http dans le fichier helm-charts/nginx/values.yaml:

```
...
proxy:
  http: ""
  https: ""
  no_proxy: ""
```

3. Afin de s'assurer que le modèle de graphique prend en charge l'utilisation des variables, modifiez le modèle de graphique dans le fichier helm-charts/nginx/templates/deployment.yaml pour contenir ce qui suit:

```
containers:
  - name: {{ .Chart.Name }}
    securityContext:
```

```
    - toYaml {{ .Values.securityContext | nindent 12 }}
  image: "{{ .Values.image.repository }}:{{ .Values.image.tag | default .Chart.AppVersion }}"
  imagePullPolicy: {{ .Values.image.pullPolicy }}
  env:
    - name: http_proxy
      value: "{{ .Values.proxy.http }}"
```

4. Définissez la variable d'environnement sur le déploiement de l'opérateur en ajoutant ce qui suit au fichier config/manager/manager.yaml:

```
containers:
 - args:
   - --leader-elect
   - --leader-election-id=ansible-proxy-demo
   image: controller:latest
   name: manager
   env:
     - name: "HTTP_PROXY"
       value: "http_proxy_test"
```

5.5.1.5. Exécution de l'opérateur

Afin de construire et d'exécuter votre opérateur, utilisez l'opérateur SDK CLI pour regrouper votre opérateur, puis utilisez le gestionnaire de cycle de vie de l'opérateur (OLM) pour le déploiement sur le cluster.

NOTE

Lorsque vous souhaitez déployer votre opérateur sur un cluster OpenShift Container Platform au lieu d'un cluster dédié OpenShift, deux options de déploiement supplémentaires sont disponibles:

- Exécutez localement en dehors du cluster en tant que programme Go.
- Exécutez comme un déploiement sur le cluster.

Ressources supplémentaires

- Exécution locale en dehors du cluster (document OpenShift Container Platform)
- Exécution en tant que déploiement sur le cluster (document OpenShift Container Platform)

5.5.1.5.1. Groupement d'un opérateur et déploiement avec le gestionnaire du cycle de vie de l'opérateur

5.5.1.5.1.1. Groupement d'un opérateur

Le format de paquet Opérateur est la méthode d'emballage par défaut pour Operator SDK et Operator Lifecycle Manager (OLM). En utilisant le SDK de l'opérateur, vous pouvez préparer votre opérateur à une utilisation sur OLM pour construire et pousser votre projet Opérateur en tant qu'image groupée.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement

- Installation d'OpenShift CLI (oc) v4+
- Le projet d'opérateur initialisé à l'aide du SDK de l'opérateur

Procédure

1. Exécutez les commandes suivantes dans votre répertoire de projet Opérateur pour construire et pousser l'image de votre opérateur. Modifiez l'argument IMG dans les étapes suivantes pour faire référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

   a. Construire l'image:

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   NOTE

   Le Dockerfile généré par le SDK pour l'opérateur renvoie explicitement GOARCH=amd64 pour la construction de go. Cela peut être modifié à GOARCH=$TARGETARCH pour les architectures non-AMD64. Docker définira automatiquement la variable d'environnement à la valeur spécifiée par -platform. Avec Buildah, le -build-arg devra être utilisé à cet effet. En savoir plus, consultez Multiple Architectures.

   b. Appuyez sur l'image vers un référentiel:

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

2. Créez votre paquet Opérateur manifeste en exécutant la commande make bundle, qui invoque plusieurs commandes, y compris l'opérateur SDK génère des paquets et des sous-commandes validant:

   ```
   $ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   Les manifestes de paquets pour un opérateur décrivent comment afficher, créer et gérer une application. La commande make bundle crée les fichiers et répertoires suivants dans votre projet Opérateur:

   - Le bundle manifeste un répertoire nommé bundle/manifests qui contient un objet ClusterServiceVersion
   - Annuaire de métadonnées groupé nommé bundle/metadata
   - L'ensemble des définitions de ressources personnalisées (CRD) dans un répertoire config/crd
   - Dockerfile bundle.Dockerfile

   Ces fichiers sont ensuite automatiquement validés en utilisant le bundle opérateur-sdk valide pour s'assurer que la représentation des faisceaux sur disque est correcte.

3. Créez et poussez votre image de paquet en exécutant les commandes suivantes. L'OLM consomme des faisceaux d'opérateurs à l'aide d'une image d'index, qui référence à une ou plusieurs images groupées.

a. Construisez l'image du bundle. Définissez BUNDLE_IMG avec les détails du registre, de l'espace de noms d'utilisateur et de la balise d'image où vous avez l'intention de pousser l'image:

```
$ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
```

b. Appuyez sur l'image du paquet:

```
$ docker push <registry>/<user>/<bundle_image_name>:<tag>
```

5.5.1.5.1.2. Déploiement d'un opérateur avec le gestionnaire du cycle de vie de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) vous aide à installer, mettre à jour et gérer le cycle de vie des Opérateurs et de leurs services associés sur un cluster Kubernetes. Le système OLM est installé par défaut sur OpenShift Dedicated et s'exécute sous forme d'extension Kubernetes afin que vous puissiez utiliser la console Web et l'OpenShift CLI (oc) pour toutes les fonctions de gestion du cycle de vie de l'opérateur sans outils supplémentaires.

Le format de paquet opérateur est la méthode d'emballage par défaut pour l'opérateur SDK et OLM. Le SDK de l'opérateur permet d'exécuter rapidement une image groupée sur OLM afin de s'assurer qu'elle fonctionne correctement.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement
- Ensemble d'image de l'opérateur construit et poussé à un registre
- Installation OLM sur un cluster basé sur Kubernetes (v1.16.0 ou version ultérieure si vous utilisez apiextensions.k8s.io/v1 CRD, par exemple OpenShift Dedicated 4)
- Connexion au cluster avec oc à l'aide d'un compte avec des autorisations d'administration dédiées

Procédure

- Entrez la commande suivante pour exécuter l'opérateur sur le cluster:

```
$ operator-sdk run bundle \ 1
    -n <namespace> \ 2
    <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. La commande run bundle crée un catalogue basé sur des fichiers valide et installe le paquet Opérateur sur votre cluster en utilisant OLM.
2. Facultatif: Par défaut, la commande installe l'opérateur dans le projet actuellement actif dans votre fichier ~/.kube/config. Il est possible d'ajouter le drapeau -n pour définir un espace de noms différent pour l'installation.
3. Dans le cas où vous ne spécifiez pas une image, la commande utilise quay.io/operator-framework/opm:latest comme image d'index par défaut. Lorsque vous spécifiez une image, la commande utilise l'image du faisceau lui-même comme image d'index.

IMPORTANT

À partir d'OpenShift Dedicated 4.11, la commande run bundle prend en charge le format de catalogue basé sur des fichiers pour les catalogues Opérateur par défaut. Le format de base de données SQLite obsolète pour les catalogues d'opérateurs continue d'être pris en charge; cependant, il sera supprimé dans une version ultérieure. Il est recommandé aux auteurs de l'opérateur de migrer leurs flux de travail vers le format de catalogue basé sur les fichiers.

Cette commande effectue les actions suivantes:

- Créez une image d'index faisant référence à votre image de paquet. L'image de l'index est opaque et éphémère, mais reflète avec précision comment un paquet serait ajouté à un catalogue en production.
- Créez une source de catalogue qui pointe vers votre nouvelle image d'index, ce qui permet à OperatorHub de découvrir votre opérateur.
- Déployez votre opérateur dans votre cluster en créant un groupe d'opérateurs, un abonnement, un plan d'installation et toutes les autres ressources requises, y compris RBAC.

5.5.1.6. Créer une ressource personnalisée

Après l'installation de votre opérateur, vous pouvez le tester en créant une ressource personnalisée (CR) qui est maintenant fournie sur le cluster par l'opérateur.

Conditions préalables

- Exemple Nginx Operator, qui fournit le Nginx CR, installé sur un cluster

Procédure

1. Changer l'espace de noms où votre opérateur est installé. À titre d'exemple, si vous avez déployé l'opérateur à l'aide de la commande make deployment:

```
$ oc project nginx-operator-system
```

2. Éditer l'échantillon Nginx CR manifeste à config/samples/demo_v1_nginx.yaml pour contenir les spécifications suivantes:

```
apiVersion: demo.example.com/v1
kind: Nginx
metadata:
  name: nginx-sample
...
spec:
...
  replicaCount: 3
```

3. Le compte de service Nginx nécessite un accès privilégié pour s'exécuter dans OpenShift Dedicated. Ajouter la contrainte de contexte de sécurité suivante (SCC) au compte de service pour l'échantillon nginx:

```
$ oc adm policy add-scc-to-user \
    anyuid system:serviceaccount:nginx-operator-system:nginx-sample
```

4. Créer le CR:

```
$ oc apply -f config/samples/demo_v1_nginx.yaml
```

5. Assurez-vous que l'opérateur Nginx crée le déploiement de l'échantillon CR avec la bonne taille:

```
$ oc get deployments
```

Exemple de sortie

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
nginx-operator-controller-manager     1/1     1            1           8m
nginx-sample                          3/3     3            3           1m
```

6. Consultez le statut des pods et CR pour confirmer que le statut est mis à jour avec les noms de pod Nginx.

   a. Consultez les gousses:

```
$ oc get pods
```

Exemple de sortie

```
NAME                                  READY    STATUS    RESTARTS   AGE
nginx-sample-6fd7c98d8-7dqdr          1/1      Running   0          1m
nginx-sample-6fd7c98d8-g5k7v          1/1      Running   0          1m
nginx-sample-6fd7c98d8-m7vn7          1/1      Running   0          1m
```

   b. Consultez l'état CR:

```
$ oc get nginx/nginx-sample -o yaml
```

Exemple de sortie

```
apiVersion: demo.example.com/v1
kind: Nginx
metadata:
...
  name: nginx-sample
...
spec:
  replicaCount: 3
status:
  nodes:
  - nginx-sample-6fd7c98d8-7dqdr
  - nginx-sample-6fd7c98d8-g5k7v
  - nginx-sample-6fd7c98d8-m7vn7
```

7. Actualisez la taille du déploiement.

a. Actualisez le fichier config/samples/demo_v1_nginx.yaml pour modifier le champ spec.size dans le CR Nginx de 3 à 5:

```
$ oc patch nginx nginx-sample \
    -p '{"spec":{"replicaCount": 5}}' \
    --type=merge
```

b. Confirmez que l'opérateur modifie la taille du déploiement:

```
$ oc get deployments
```

Exemple de sortie

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
nginx-operator-controller-manager     1/1     1            1           10m
nginx-sample                          5/5     5            5           3m
```

8. Supprimez le CR en exécutant la commande suivante:

```
$ oc delete -f config/samples/demo_v1_nginx.yaml
```

9. Nettoyez les ressources qui ont été créées dans le cadre de ce tutoriel.

- Lorsque vous avez utilisé la commande make deployment pour tester l'opérateur, exécutez la commande suivante:

```
$ make undeploy
```

- Lorsque vous avez utilisé la commande operator-sdk run bundle pour tester l'opérateur, exécutez la commande suivante:

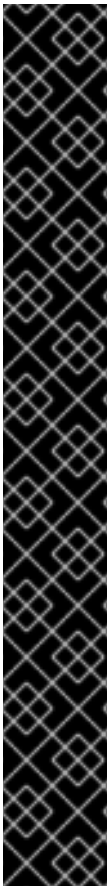
```
$ operator-sdk cleanup <project_name>
```

5.5.1.7. Ressources supplémentaires

- Consultez la mise en page du projet pour les opérateurs basés sur Helm pour en savoir plus sur les structures d'annuaire créées par le SDK de l'opérateur.
- Lorsqu'un proxy de sortie à l'échelle du cluster est configuré, les administrateurs ayant le rôle d'administrateur dédié peuvent outrepasser les paramètres proxy ou injecter un certificat CA personnalisé pour des opérateurs spécifiques fonctionnant sur Operator Lifecycle Manager (OLM).

5.5.2. Aménagement du projet pour les opérateurs basés sur Helm

L'opérateur-sdk CLI peut générer, ou échafauder, un certain nombre de paquets et de fichiers pour chaque projet d'opérateur.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.5.2.1. Disposition de projet basée sur le helm

Les projets d'opérateur basés sur le helm générés à l'aide de la commande operator-sdk init --plugins helm contiennent les répertoires et fichiers suivants:

Fichier/dossiers	But
configuration/	Kustomize se manifeste pour le déploiement de l'opérateur sur un cluster Kubernetes.
fléchettes à barres/	Le diagramme de helm initialisé avec la commande operator-sdk crée api.
Dockerfile	Il est utilisé pour construire l'image de l'opérateur avec la commande docker-build.
les montres.yaml	Groupe/version/type (GVK) et localisation du graphique Helm.
À propos de Makefile	Cibles utilisées pour gérer le projet.
LE PROJET	Fichier YAML contenant des informations de métadonnées pour l'opérateur.

5.5.3. La mise à jour des projets basés sur Helm pour les versions SDK plus récentes de l'opérateur

Le logiciel OpenShift Dedicated 4 prend en charge le SDK 1.38.0 de l'opérateur. Lorsque vous disposez déjà du 1.36.1 CLI installé sur votre poste de travail, vous pouvez mettre à jour le CLI à 1.38.0 en installant la dernière version.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

Cependant, pour s'assurer que vos projets d'opérateur existants maintiennent la compatibilité avec le SDK 1.38.0 de l'opérateur, des étapes de mise à jour sont nécessaires pour les modifications de rupture associées introduites depuis 1.36.1. Les étapes de mise à jour doivent être exécutées manuellement dans l'un de vos projets Opérateurs qui ont été précédemment créés ou maintenus avec 1.36.1.

5.5.3.1. La mise à jour des projets d'opérateur basé sur Helm pour l'opérateur SDK 1.38.0

La procédure suivante met à jour un projet existant d'opérateur basé sur Helm pour la compatibilité avec 1.38.0.

Conditions préalables

- L'opérateur SDK 1.38.0 installé
- Création ou maintenance d'un projet opérateur avec l'opérateur SDK 1.36.1

Procédure

1. Éditez le Makefile de votre projet Opérateur pour mettre à jour la version SDK de l'opérateur vers 1.38.0, comme indiqué dans l'exemple suivant:

   Exemple de Makefile

```
# Set the Operator SDK version to use. By default, what is installed on the system is used.
# This is useful for CI or a project to utilize a specific version of the operator-sdk toolkit.
OPERATOR_SDK_VERSION ?= v1.38.0 (1)
```

(1) Changer la version de 1.36.1 à 1.38.0.

2. Éditez le Makefile de votre projet Opérateur pour mettre à jour la balise image ose-helm-rhel9-operator à 4, comme indiqué dans l'exemple suivant:

Exemple Dockerfile

```
FROM registry.redhat.io/openshift4/ose-helm-rhel9-operator:v4
```

3. Il faut mettre à niveau les versions Kubernetes de votre projet Opérateur pour utiliser 1.30 et Kubebuilder v4.

ASTUCE

Cette mise à jour comprend des changements complexes d'échafaudage en raison de l'élimination du kube-rbac-proxy. Lorsque ces migrations deviennent difficiles à suivre, échafauder un nouveau projet d'échantillon à des fins de comparaison.

a. Actualisez la version Kustomize dans votre Makefile en apportant les modifications suivantes:

```
- curl -sSLo - https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize/v5.3.0/kustomize_v5.3.0_$(OS)_$(ARCH).tar.gz | \
+ curl -sSLo - https://github.com/kubernetes-sigs/kustomize/releases/download/kustomize/v5.4.2/kustomize_v5.4.2_$(OS)_$(ARCH).tar.gz | \
```

b. Actualisez votre fichier config/default/kustomization.yaml avec les modifications suivantes:

```
  # [PROMETHEUS] To enable prometheus monitor, uncomment all sections with 'PROMETHEUS'.
  #- ../prometheus
+ # [METRICS] Expose the controller manager metrics service.
+ - metrics_service.yaml

+ # Uncomment the patches line if you enable Metrics, and/or are using webhooks and cert-manager
  patches:
- # Protect the /metrics endpoint by putting it behind auth.
- # If you want your controller-manager to expose the /metrics
- # endpoint w/o any authn/z, please comment the following line.
- - path: manager_auth_proxy_patch.yaml
+ # [METRICS] The following patch will enable the metrics endpoint using HTTPS and the port :8443.
+ # More info: https://book.kubebuilder.io/reference/metrics
+ - path: manager_metrics_patch.yaml
+   target:
+     kind: Deployment
```

c. Enlevez les fichiers config/default/manager_auth_proxy_patch.yaml et config/default/manager_config_patch.yaml.

d. Créez un fichier config/default/manager_metrics_patch.yaml avec le contenu suivant:

```
# This patch adds the args to allow exposing the metrics endpoint using HTTPS
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-bind-address=:8443
# This patch adds the args to allow securing the metrics endpoint
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-secure
# This patch adds the args to allow RBAC-based authn/authz the metrics endpoint
- op: add
  path: /spec/template/spec/containers/0/args/0
  value: --metrics-require-rbac
```

e. Créez un fichier config/default/metrics_service.yaml avec le contenu suivant:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    control-plane: controller-manager
    app.kubernetes.io/name: <operator-name>
    app.kubernetes.io/managed-by: kustomize
  name: controller-manager-metrics-service
  namespace: system
spec:
  ports:
    - name: https
      port: 8443
      protocol: TCP
      targetPort: 8443
  selector:
    control-plane: controller-manager
```

f. Actualisez votre fichier config/manager/manager.yaml avec les modifications suivantes:

```
  - --leader-elect
+ - --health-probe-bind-address=:8081
```

g. Actualisez votre fichier config/prometheus/monitor.yaml avec les modifications suivantes:

```
     - path: /metrics
-      port: https
+      port: https # Ensure this is the name of the port that exposes HTTPS metrics
       tlsConfig:
+        # TODO(user): The option insecureSkipVerify: true is not recommended for
production since it disables
+        # certificate verification. This poses a significant security risk by making the system
vulnerable to
+        # man-in-the-middle attacks, where an attacker could intercept and manipulate the
```

```
communication between
+       # Prometheus and the monitored services. This could lead to unauthorized access
to sensitive metrics data,
+       # compromising the integrity and confidentiality of the information.
+       # Please use the following options for secure configurations:
+       # caFile: /etc/metrics-certs/ca.crt
+       # certFile: /etc/metrics-certs/tls.crt
+       # keyFile: /etc/metrics-certs/tls.key
        insecureSkipVerify: true
```

h. Supprimez les fichiers suivants du répertoire config/rbac/:

- **auth_proxy_client_clusterrole.yaml**
- **auth_proxy_role.yaml**
- **auth_proxy_role_binding.yaml**
- **auth_proxy_service.yaml**

i. Actualisez votre fichier config/rbac/kustomization.yaml avec les modifications suivantes:

```
  - leader_election_role_binding.yaml
- # Comment the following 4 lines if you want to disable
- # the auth proxy (https://github.com/brancz/kube-rbac-proxy)
- # which protects your /metrics endpoint.
- - auth_proxy_service.yaml
- - auth_proxy_role.yaml
- - auth_proxy_role_binding.yaml
- - auth_proxy_client_clusterrole.yaml
+ # The following RBAC configurations are used to protect
+ # the metrics endpoint with authn/authz. These configurations
+ # ensure that only authorized users and service accounts
+ # can access the metrics endpoint. Comment the following
+ # permissions if you want to disable this protection.
+ # More info: https://book.kubebuilder.io/reference/metrics.html
+ - metrics_auth_role.yaml
+ - metrics_auth_role_binding.yaml
+ - metrics_reader_role.yaml
```

j. Créez un fichier config/rbac/metrics_auth_role_binding.yaml avec le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: metrics-auth-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: metrics-auth-role
subjects:
  - kind: ServiceAccount
    name: controller-manager
    namespace: system
```

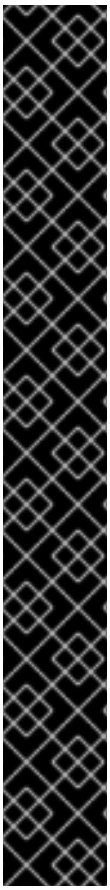
k. Créez un fichier config/rbac/metrics_reader_role.yaml avec le contenu suivant:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: metrics-reader
rules:
- nonResourceURLs:
  - "/metrics"
  verbs:
  - get
```

5.5.3.2. Ressources supplémentaires

- La mise à jour des projets d'opérateur basé sur Helm pour l'opérateur SDK 1.36.1 (OpenShift Dédié 4.17)
- Les projets de manifestation de paquets migratoires au format de paquetage

5.5.4. Appui à la barre dans le SDK de l'opérateur



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.5.4.1. Cartes de barre

L'une des options SDK de l'opérateur pour générer un projet d'opérateur comprend l'exploitation d'un graphique Helm existant pour déployer les ressources Kubernetes en tant qu'application unifiée, sans avoir à écrire de code Go. Ces opérateurs basés sur Helm sont conçus pour exceller dans des applications apatrides qui nécessitent très peu de logique lorsqu'elles sont déployées, car les modifications doivent être appliquées aux objets Kubernetes générés dans le cadre du graphique. Cela peut sembler limitatif, mais peut être suffisant pour une quantité surprenante de cas d'utilisation comme le montre la prolifération des graphiques Helm construits par la communauté Kubernetes.

La fonction principale d'un opérateur est de lire à partir d'un objet personnalisé qui représente votre instance d'application et dont l'état souhaité correspond à ce qui est en cours d'exécution. Dans le cas d'un opérateur basé sur Helm, le champ spec de l'objet est une liste d'options de configuration qui sont généralement décrites dans le fichier Helm values.yaml. Au lieu de définir ces valeurs avec des drapeaux à l'aide du Helm CLI (par exemple, l'installation helm -f valeurs.yaml), vous pouvez les exprimer dans une ressource personnalisée (CR), qui, en tant qu'objet Kubernetes natif, permet les avantages de RBAC appliqué à elle et une piste d'audit.

Exemple d'un simple CR appelé Tomcat:

```
apiVersion: apache.org/v1alpha1
kind: Tomcat
metadata:
  name: example-app
spec:
  replicaCount: 2
```

La valeur replicaCount, 2 dans ce cas, est propagée dans le modèle du graphique où les éléments suivants sont utilisés:

```
{{ .Values.replicaCount }}
```

Après la construction et le déploiement d'un opérateur, vous pouvez déployer une nouvelle instance d'une application en créant une nouvelle instance d'un CR, ou répertorier les différentes instances s'exécutant dans tous les environnements en utilisant la commande oc:

```
$ oc get Tomcats --all-namespaces
```

Il n'y a pas besoin d'utiliser le Helm CLI ou d'installer Tiller; Les opérateurs basés sur Helm importent le code du projet Helm. Il vous suffit d'avoir une instance de l'opérateur qui exécute et enregistre le CR avec une définition de ressource personnalisée (CRD). Comme il obéit à RBAC, vous pouvez plus facilement prévenir les changements de production.

5.6. DÉFINITION DES VERSIONS DE SERVICE CLUSTER (CSV)

La version de service de cluster (CSV), définie par un objet ClusterServiceVersion, est un manifeste YAML créé à partir des métadonnées de l'opérateur qui aide le gestionnaire de cycle de vie de l'opérateur (OLM) à exécuter l'opérateur dans un cluster. Ce sont les métadonnées qui accompagnent une image de conteneur d'opérateur, utilisée pour peupler les interfaces utilisateur avec des informations telles que son logo, sa description et sa version. C'est aussi une source d'informations techniques qui est nécessaire pour exécuter l'opérateur, comme les règles RBAC qu'il exige et de quelles ressources personnalisées (CR) il gère ou dépend.

Le SDK de l'opérateur comprend le générateur CSV pour générer un CSV pour le projet opérateur actuel, personnalisé à l'aide des informations contenues dans les manifestes YAML et les fichiers source de l'opérateur.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

La commande génératrice de CSV supprime la responsabilité des auteurs de l'opérateur ayant des connaissances approfondies en OLM afin que leur opérateur interagisse avec OLM ou publie des métadonnées dans le Registre de catalogue. De plus, étant donné que la spécification CSV changera probablement avec le temps à mesure que de nouvelles fonctionnalités Kubernetes et OLM seront mises en œuvre, le SDK de l'opérateur est équipé pour étendre facilement son système de mise à jour afin de gérer de nouvelles fonctionnalités CSV à l'avenir.

5.6.1. Comment fonctionne la génération de CSV

Les manifestes de paquets d'opérateurs, qui incluent des versions de service cluster (CSV), décrivent comment afficher, créer et gérer une application avec Operator Lifecycle Manager (OLM). Le générateur CSV dans le SDK de l'opérateur, appelé par la sous-commande de groupement génératrice, est la première étape vers la publication de votre opérateur dans un catalogue et le déployer avec OLM. La sous-commande nécessite certains manifestes d'entrée pour construire un manifeste CSV; toutes les entrées sont lues lorsque la commande est invoquée, avec une base CSV, pour générer ou régénérer idempotentement un CSV.

En règle générale, la sous-commande des manifestes de kustomize génératrice serait exécutée en premier pour générer les bases Kustomize d'entrée qui sont consommées par la sous-commande de paquets génératrices. Cependant, le SDK de l'opérateur fournit la commande make bundle, qui automatise plusieurs tâches, y compris l'exécution des sous-commandes suivantes dans l'ordre:

1. **générer des manifestes de kustomize**
2. **générer des paquets**
3. **la validation du paquet**

Ressources supplémentaires

- Consultez Bundling un opérateur pour une procédure complète qui comprend la génération d'un paquet et d'un CSV.

5.6.1.1. Fichiers et ressources générés

La commande make bundle crée les fichiers et répertoires suivants dans votre projet Opérateur:

- Le bundle manifeste un répertoire nommé bundle/manifests qui contient un objet ClusterServiceVersion (CSV)
- Annuaire de métadonnées groupé nommé bundle/metadata
- L'ensemble des définitions de ressources personnalisées (CRD) dans un répertoire config/crd
- Dockerfile bundle.Dockerfile

Les ressources suivantes sont généralement incluses dans un CSV:

Le rôle
: Définit les autorisations de l'opérateur dans un espace de noms.

ClusterRole
: Définit les autorisations de l'opérateur à l'échelle du cluster.

Déploiement
: Définit comment un Operand d'un opérateur est exécuté en pods.

CustomResourceDefinition (CRD)
: Définit des ressources personnalisées que votre opérateur réconcilie.

Exemples de ressources personnalisées
: Exemples de ressources adhérant à la spécification d'un CRD particulier.

5.6.1.2. Gestion des versions

L'indicateur --version pour la sous-commande de paquet génératrice fournit une version sémantique pour votre paquet lors de la création d'un pour la première fois et lors de la mise à niveau d'une version existante.

En définissant la variable VERSION dans votre Makefile, l'indicateur --version est automatiquement invoqué à l'aide de cette valeur lorsque la sous-commande de paquet généré est exécutée par la commande make bundle. La version CSV est la même que la version de l'opérateur, et un nouveau CSV est généré lors de la mise à niveau des versions de l'opérateur.

5.6.2. Champs CSV définis manuellement

De nombreux champs CSV ne peuvent pas être peuplés à l'aide de manifestes génériques générés qui ne sont pas spécifiques à l'opérateur SDK. Ces champs sont principalement des métadonnées écrites par l'homme sur l'opérateur et diverses définitions de ressources personnalisées (CRD).

Les auteurs de l'opérateur doivent modifier directement leur fichier YAML version de service de cluster (CSV) en ajoutant des données personnalisées aux champs requis suivants. Le SDK d'opérateur donne un avertissement lors de la génération de CSV lorsqu'un manque de données dans l'un des champs requis est détecté.

Les tableaux suivants détaillent les champs CSV définis manuellement et qui sont facultatifs.

Tableau 5.7. Champs CSV obligatoires

Le champ	Description
les métadonnées.name	C'est un nom unique pour ce CSV. La version de l'opérateur doit être incluse dans le nom pour assurer l'unicité, par exemple app-operator.v0.1.1.
les métadonnées.capabilitys	Le niveau de capacité selon le modèle de maturité de l'opérateur. Les options incluent l'installation de base, les mises à niveau sans soudure, le cycle de vie complet, les informations approfondies et le pilote automatique.
caractéristiques Spéc.displayName	Le nom public permettant d'identifier l'opérateur.
description Spéc.description	Brève description de la fonctionnalité de l'opérateur.
caractéristiques de Sp.keywords	Les mots clés décrivant l'opérateur.
caractéristiques Sp.maintainers	Entités humaines ou organisationnelles qui maintiennent l'opérateur, avec un nom et un courriel.
fournisseur de Spécification	Le fournisseur de l'opérateur (généralement une organisation), avec un nom.
caractéristiques Sp.labels	Paires clés-valeur à utiliser par les internes de l'opérateur.
caractéristiques de Sp.version	La version sémantique de l'opérateur, par exemple 0.1.1.
définition des ressources de Spéc.custom	Les CRD que l'Opérateur utilise. Ce champ est rempli automatiquement par le SDK de l'opérateur si des fichiers YAML CRD sont présents dans le déploiement/. Cependant, plusieurs champs qui ne figurent pas dans la spécification du manifeste CRD nécessitent l'entrée de l'utilisateur: • description: description de la CRD. • les ressources: toutes les ressources Kubernetes mises à profit par le CRD, par exemple Pod et StatefulSet objets. • description des spécifications: Conseils d'interface utilisateur pour les entrées et les sorties de l'opérateur.

Tableau 5.8. Champs CSV optionnels

Le champ	Description
caractéristiques Sp.replaces	Le nom du CSV étant remplacé par ce CSV.
liens Spéc.links	Les URL (par exemple, sites Web et documentation) relatives à l'opérateur ou à l'application gérée, chacune avec un nom et une url.
caractéristiques Spéc.selector	Les sélecteurs par lesquels l'opérateur peut associer des ressources dans un cluster.
le Spéc.icon	Icône encodée base64 unique à l'opérateur, définie dans un champ de données base64 avec un type média.
caractéristiques Sp.maturity	Le niveau de maturité que le logiciel a atteint à cette version. Les options comprennent la planification, pré-alpha, alpha, bêta, stable, mature, inactive et dépréciée.

De plus amples détails sur les données que chaque champ ci-dessus devrait contenir sont trouvés dans la spécification CSV.

NOTE

À l'heure actuelle, plusieurs champs YAML nécessitant une intervention de l'utilisateur peuvent potentiellement être analysés à partir du code opérateur.

Ressources supplémentaires

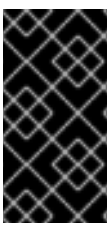
- Le modèle de maturité de l'opérateur

5.6.3. Annotations de métadonnées de l'opérateur

Les développeurs d'opérateurs peuvent définir certaines annotations dans les métadonnées d'une version de service de cluster (CSV) pour activer des fonctionnalités ou mettre en évidence des fonctionnalités dans les interfaces utilisateur (UI), telles que OperatorHub ou Red Hat Ecosystem Catalog. Les annotations de métadonnées de l'opérateur sont définies manuellement en définissant le champ métadonnées.annotations dans le fichier CSV YAML.

5.6.3.1. L'infrastructure comporte des annotations

Les annotations dans le groupe features.operators.openshift.io détaillent les fonctionnalités d'infrastructure qu'un opérateur peut prendre en charge, spécifiées en définissant une valeur "vraie" ou "faux". Les utilisateurs peuvent visualiser et filtrer par ces fonctionnalités lors de la découverte des opérateurs via OperatorHub dans la console Web ou dans le catalogue de l'écosystème Red Hat. Ces annotations sont prises en charge dans OpenShift Dedicated 4.10 et ultérieure.



IMPORTANT

L'infrastructure features.operators.openshift.io comporte des annotations déprécater les annotations.openshift.io/infrastructures utilisées dans les versions antérieures d'OpenShift Dedicated. Consultez « Annotations de fonctionnalités d'infrastructure dépréciée » pour plus d'informations.

Tableau 5.9. L'infrastructure comporte des annotations

Annotation	Description	Les valeurs valides[1]
Features.operators.openshift.io/déconnecté	Indiquez si un opérateur prend en charge la mise en miroir dans des catalogues déconnectés, y compris toutes les dépendances, et ne nécessite pas d'accès à Internet. L'opérateur utilise le champ spec.relatedImages CSV pour se référer à toute image liée par son digeste.	"vrai" ou "faux"
caractéristiques.operators.openshift.io/fips-compliant	Indiquez si un opérateur accepte la configuration FIPS-140 de la plate-forme sous-jacente et fonctionne sur des nœuds qui sont démarrés en mode FIPS. Dans ce mode, l'opérateur et toutes les charges de travail qu'il gère (opérateurs) appellent uniquement la bibliothèque cryptographique Red Hat Enterprise Linux (RHEL) soumise pour validation FIPS-140.	"vrai" ou "faux"
caractéristiques.operators.openshift.io/proxy-aware	Indiquez si un opérateur prend en charge l'exécution d'un cluster derrière un proxy en acceptant les variables d'environnement proxy HTTP_PROXY et HTTPS_PROXY standard. Le cas échéant, l'opérateur transmet ces informations à la charge de travail qu'il gère (opérateurs).	"vrai" ou "faux"
Features.operators.openshift.io/tls-profiles	Indiquez si un opérateur met en œuvre des réglages bien connus pour modifier la suite de chiffrement TLS utilisée par l'opérateur et, le cas échéant, l'une des charges de travail qu'il gère (opérateurs).	"vrai" ou "faux"
caractéristiques.operators.openshift.io/token-auth-aws	Indiquez si un opérateur prend en charge la configuration pour l'authentification tokenized avec les API AWS via AWS Secure Token Service (STS) à l'aide de l'opérateur d'identification en nuage (CCO).	"vrai" ou "faux"
caractéristiques.operators.openshift.io/token-auth-azure	Indiquez si un opérateur prend en charge la configuration pour l'authentification tokenized avec les API Azure via Azure Managed Identity à l'aide de l'opérateur d'identification en nuage (CCO).	"vrai" ou "faux"

Annotation	Description	Les valeurs valides[1]
caractéristiques.operators.openshift.io/token-auth-gcp	Indiquez si un opérateur prend en charge la configuration pour l'authentification tokenized avec les API Google Cloud via GCP Workload Identity Foundation (WIF) à l'aide de l'opérateur d'identification en nuage (CCO).	"vrai" ou "faux"
caractéristiques.operators.openshift.io/cnf	Indiquez si un opérateur fournit un plugin Kubernetes (Native Network Function) Cloud-Native Network Function (CNF).	"vrai" ou "faux"
caractéristiques.operators.openshift.io/cni	Indiquez si un opérateur fournit un plugin Kubernetes (Container Network Interface) (CNI).	"vrai" ou "faux"
caractéristiques.operators.openshift.io/csi	Indiquez si un opérateur fournit un plugin Kubernetes (CSI) Kubernetes.	"vrai" ou "faux"

1. Les valeurs valides sont affichées intentionnellement avec des guillemets doubles, car les annotations Kubernetes doivent être des chaînes.

Exemple CSV avec des annotations de fonctionnalités d'infrastructure

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    features.operators.openshift.io/disconnected: "true"
    features.operators.openshift.io/fips-compliant: "false"
    features.operators.openshift.io/proxy-aware: "false"
    features.operators.openshift.io/tls-profiles: "false"
    features.operators.openshift.io/token-auth-aws: "false"
    features.operators.openshift.io/token-auth-azure: "false"
    features.operators.openshift.io/token-auth-gcp: "false"
```

Ressources supplémentaires

- Activer votre opérateur pour les environnements réseau restreints (mode déconnecté)

5.6.3.2. L'infrastructure dépréciée comporte des annotations

À partir d'OpenShift Dedicated 4.14, le groupe d'annotations operator.openshift.io/infrastructure-features est déprécié par le groupe d'annotations avec l'espace de noms features.operators.openshift.io. Bien que vous soyez encouragé à utiliser les annotations les plus récentes, les deux groupes sont actuellement acceptés lorsqu'ils sont utilisés en parallèle.

Ces annotations détaillent les caractéristiques de l'infrastructure qu'un opérateur prend en charge. Les utilisateurs peuvent visualiser et filtrer par ces fonctionnalités lors de la découverte des opérateurs via OperatorHub dans la console Web ou dans le catalogue de l'écosystème Red Hat.

Tableau 5.10. Les opérateurs désappropriés.openshift.io/infrastructure-fonctionnements annotations

Les valeurs d'annotation valides	Description
déconnecté	L'opérateur prend en charge la mise en miroir dans des catalogues déconnectés, y compris toutes les dépendances, et ne nécessite pas d'accès à Internet. Les images associées requises pour la mise en miroir sont répertoriées par l'opérateur.
CNF	L'opérateur fournit un plugin Kubernetes (CNF) dans le Cloud-Native Network Functions (CNF).
CNI	L'opérateur fournit un plugin Kubernetes de Container Network Interface (CNI).
CSI	L'opérateur fournit un plugin Kubernetes d'interface de stockage de conteneurs (CSI).
FIPS	L'opérateur accepte le mode FIPS de la plate-forme sous-jacente et fonctionne sur des nœuds qui sont démarrés en mode FIPS.  **IMPORTANT** Lors de l'exécution Red Hat Enterprise Linux (RHEL) ou Red Hat Enterprise Linux CoreOS (RHCOS) démarré en mode FIPS, les composants de base dédiés OpenShift utilisent les bibliothèques cryptographiques RHEL qui ont été soumises au NIST pour FIPS 140-2/140-3 Validation sur seulement les architectures x86_64, ppc64le et s390x.
logiciel proxy-aware	L'opérateur prend en charge l'exécution d'un cluster derrière un proxy. L'opérateur accepte les variables d'environnement proxy standard HTTP_PROXY et HTTPS_PROXY, que le gestionnaire de cycle de vie de l'opérateur (OLM) fournit automatiquement à l'opérateur lorsque le cluster est configuré pour utiliser un proxy. Les variables d'environnement requises sont transmises à Operands pour les charges de travail gérées.

Exemple CSV avec support déconnecté et proxy-aware

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    operators.openshift.io/infrastructure-features: '["disconnected", "proxy-aware"]'
```

5.6.3.3. Autres annotations facultatives

Les annotations suivantes de l'opérateur sont facultatives.

Tableau 5.11. Autres annotations facultatives

Annotation	Description
exemples d'ALM	Fournissez des modèles de définition de ressources personnalisées (CRD) avec un ensemble minimal de configuration. Compatible UIs pré-rempli ce modèle pour les utilisateurs à personnaliser davantage.
operatorframework.io/initialisation-ressource	Indiquez une seule ressource personnalisée requise en ajoutant l'annotation de la ressource d'initialisation à la version de service cluster (CSV) pendant l'installation de l'opérateur. L'utilisateur est alors invité à créer la ressource personnalisée à l'aide d'un modèle fourni dans le CSV. Doit inclure un modèle contenant une définition complète de YAML.
operatorframework.io/suggéré-namespace	Définissez un espace de noms suggéré où l'opérateur devrait être déployé.
operatorframework.io/suggéré-namespace-template	Définissez un manifeste pour un objet Namespace avec le sélecteur de nœud par défaut pour l'espace de noms spécifié.
opérateurs.openshift.io/valid-abonnement	Le tableau de forme gratuite pour la liste des abonnements spécifiques qui sont nécessaires pour utiliser l'opérateur. À titre d'exemple, '["3Scale Commercial License", "Red Hat Managed Integration"]
Operator.operatorframework.io/objets internes	Cache les CRD dans l'interface utilisateur qui ne sont pas destinés à la manipulation des utilisateurs.

Exemple CSV avec une exigence de licence OpenShift dédiée

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    operators.openshift.io/valid-subscription: '["OpenShift Container Platform"]'
```

Exemple CSV avec une exigence de licence 3scale

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    operators.openshift.io/valid-subscription: '["3Scale Commercial License", "Red Hat Managed Integration"]'
```

Ressources supplémentaires

- Les modèles CRD
- Initialisation des ressources personnalisées requises
- Définition d'un espace de noms suggéré
- Définir un espace de noms suggéré avec le sélecteur de nœud par défaut
- Cacher des objets internes

5.6.4. Activer votre opérateur pour les environnements réseau restreints

En tant qu'auteur de l'opérateur, votre opérateur doit répondre à des exigences supplémentaires pour fonctionner correctement dans un environnement restreint ou déconnecté.

Exigences de l'opérateur pour soutenir le mode déconnecté

- Le remplacement des références d'images codées en dur par des variables d'environnement.
- Dans la version de service cluster (CSV) de votre opérateur:
  - Énumérez toutes les images connexes, ou d'autres images de conteneurs dont votre opérateur pourrait avoir besoin pour exécuter leurs fonctions.
  - Faites référence à toutes les images spécifiées par un digest (SHA) et non par une balise.
- Les dépendances de votre opérateur doivent également prendre en charge l'exécution en mode déconnecté.
- L'opérateur ne doit pas avoir besoin de ressources hors groupe.

Conditions préalables

- D'un projet d'opérateur avec un CSV. La procédure suivante utilise l'opérateur Memcached comme exemple pour les projets basés sur Go, Ansible et Helm.

Procédure

1. Définissez une variable d'environnement pour les références d'image supplémentaires utilisées par l'opérateur dans le fichier config/manager/manager.yaml:

Exemple 5.2. Exemple de fichier config/manager/manager.yaml

```
...
spec:
  ...
    spec:
      ...
      containers:
      - command:
        - /manager
        ...
        env:
        - name: <related_image_environment_variable> 1
          value: "<related_image_reference_with_tag>" 2
```

1. Définissez la variable d'environnement, telle que RELATED_IMAGE_MEMCACHED.
2. Définissez la référence et la balise d'image associées, telles que docker.io/memcached:1.4.36-alpine.

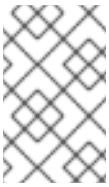
2. Le remplacement des références d'image codées en dur par des variables d'environnement dans le fichier pertinent pour votre type de projet Opérateur:

- Dans le cas des projets Go-based Operator, ajoutez la variable d'environnement au fichier Controllers/memcached_controller.go comme indiqué dans l'exemple suivant:

Exemple 5.3. Exemple de contrôleurs/memcached_controller.go fichier

```
  // deploymentForMemcached returns a memcached Deployment object

...

 Spec: corev1.PodSpec{
        Containers: []corev1.Container{{
-   Image:   "memcached:1.4.36-alpine",  (1)
+   Image:   os.Getenv("<related_image_environment_variable>"),  (2)
    Name:    "memcached",
    Command: []string{"memcached", "-m=64", "-o", "modern", "-v"},
    Ports: []corev1.ContainerPort{{

...
```

1. Effacer la référence et le tag de l'image.
2. La fonction os.Getenv appelle <related_image_environnement_variable>.



NOTE

La fonction os.Getenv renvoie une chaîne vide si une variable n'est pas définie. Définissez le <related_image_environnement_variable> avant de modifier le fichier.

- Dans le cas des projets d'opérateur basé sur Ansible, ajoutez la variable d'environnement au fichier role/memcached/tasks/main.yml comme indiqué dans l'exemple suivant:

Exemple 5.4. Exemple de fichier role/memcached/tasks/main.yml

```
spec:
  containers:
  - name: memcached
    command:
    - memcached
    - -m=64
    - -o
    - modern
    - -v
-   image: "docker.io/memcached:1.4.36-alpine"  (1)
```

```
+   image: "{{ lookup('env', '<related_image_environment_variable>') }}"
    ports:
      - containerPort: 11211

...
```

1. Effacer la référence et le tag de l'image.
2. La fonction de recherche permet d'appeler <related_image_environnement_variable>.

- Dans le cas des projets d'opérateur basé sur Helm, ajoutez le champ OverrideValues au fichier watch.yaml comme indiqué dans l'exemple suivant:

Exemple 5.5. Exemple watch.yaml fichier

```
...
- group: demo.example.com
  version: v1alpha1
  kind: Memcached
  chart: helm-charts/memcached
  overrideValues: (1)
    relatedImage: ${<related_image_environment_variable>} (2)
```

1. Ajoutez le champ OverrideValues.
2. Définissez le champ OverrideValues à l'aide du <related_image_environnement_variable>, tel que RELATED_IMAGE_MEMCACHED.

a. Ajoutez la valeur du champ OverrideValues au fichier helm-charts/memchached/values.yaml comme indiqué dans l'exemple suivant:

Exemple de fichier helm-charts/memchached/values.yaml

```
...
relatedImage: ""
```

b. Éditez le modèle de graphique dans le fichier helm-charts/memcached/templates/deployment.yaml comme indiqué dans l'exemple suivant:

Exemple 5.6. Exemple de fichier helm-charts/memcached/templates/deployment.yaml

```
containers:
  - name: {{ .Chart.Name }}
    securityContext:
      - toYaml {{ .Values.securityContext | nindent 12 }}
    image: "{{ .Values.image.pullPolicy }}
```

```
env: (1)
  - name: related_image (2)
    value: "{{ .Values.relatedImage }}" (3)
```

(1) Ajoutez le champ env.

(2) Indiquez la variable d'environnement.

(3) Définissez la valeur de la variable d'environnement.

3. Ajoutez la définition de variable BUNDLE_GEN_FLAGS à votre Makefile avec les modifications suivantes:

Exemple de Makefile

```
  BUNDLE_GEN_FLAGS ?= -q --overwrite --version $(VERSION)
$(BUNDLE_METADATA_OPTS)

  # USE_IMAGE_DIGESTS defines if images are resolved via tags or digests
  # You can enable this value if you would like to use SHA Based Digests
  # To enable set flag to true
  USE_IMAGE_DIGESTS ?= false
  ifeq ($(USE_IMAGE_DIGESTS), true)
      BUNDLE_GEN_FLAGS += --use-image-digests
  endif

...

-  $(KUSTOMIZE) build config/manifests | operator-sdk generate bundle -q --overwrite --
version $(VERSION) $(BUNDLE_METADATA_OPTS) (1)
+  $(KUSTOMIZE) build config/manifests | operator-sdk generate bundle
$(BUNDLE_GEN_FLAGS) (2)

...
```

(1) Effacer cette ligne dans le Makefile.

(2) C) Remplacer la ligne ci-dessus par cette ligne.

4. Afin de mettre à jour votre image de l'opérateur pour utiliser un digest (SHA) et non une balise, exécutez la commande make bundle et définissez USE_IMAGE_DIGESTS sur true:

```
$ make bundle USE_IMAGE_DIGESTS=true
```

5. Ajouter l'annotation déconnectée, qui indique que l'opérateur fonctionne dans un environnement déconnecté:

```
metadata:
  annotations:
    operators.openshift.io/infrastructure-features: '["disconnected"]'
```

Les opérateurs peuvent être filtrés dans OperatorHub par cette fonctionnalité d'infrastructure.

5.6.5. Activer votre opérateur pour plusieurs architectures et systèmes d'exploitation

Le gestionnaire de cycle de vie de l'opérateur (OLM) suppose que tous les opérateurs s'exécutent sur des hôtes Linux. Cependant, en tant qu'auteur de l'opérateur, vous pouvez spécifier si votre opérateur prend en charge la gestion des charges de travail sur d'autres architectures, si les nœuds de travail sont disponibles dans le cluster dédié OpenShift.

Dans le cas où votre opérateur prend en charge des variantes autres que AMD64 et Linux, vous pouvez ajouter des étiquettes à la version de service de cluster (CSV) qui fournit à l'opérateur la liste des variantes prises en charge. Les étiquettes indiquant les architectures et les systèmes d'exploitation pris en charge sont définies par ce qui suit:

```
labels:
  operatorframework.io/arch.<arch>: supported (1)
  operatorframework.io/os.<os>: supported (2)
```

1. Définissez <arch> sur une chaîne prise en charge.
2. Définissez <os> sur une chaîne prise en charge.

NOTE

Il n'y a que les étiquettes sur la tête du canal par défaut pour filtrer les manifestes de paquets par étiquette. Cela signifie, par exemple, qu'il est possible de fournir une architecture supplémentaire à un opérateur dans le canal non par défaut, mais que cette architecture n'est pas disponible pour le filtrage dans l'API PackageManifest.

Dans le cas où un CSV n'inclut pas d'étiquette os, il est traité comme s'il avait l'étiquette de support Linux suivante par défaut:

```
labels:
  operatorframework.io/os.linux: supported
```

Lorsqu'un CSV n'inclut pas d'étiquette d'arche, il est traité comme s'il avait l'étiquette de support AMD64 suivante par défaut:

```
labels:
  operatorframework.io/arch.amd64: supported
```

Lorsqu'un opérateur prend en charge plusieurs architectures de nœuds ou systèmes d'exploitation, vous pouvez également ajouter plusieurs étiquettes.

Conditions préalables

- D'un projet d'opérateur avec un CSV.
- Afin de prendre en charge la liste de plusieurs architectures et systèmes d'exploitation, l'image de votre opérateur référencée dans le CSV doit être une image de liste manifeste.

- Afin que l'opérateur fonctionne correctement dans un réseau restreint, ou déconnecté, l'image référencée doit également être spécifiée à l'aide d'un digeste (SHA) et non par une balise.

Procédure

- Ajoutez une étiquette dans les métadonnées.labels de votre CSV pour chaque architecture et système d'exploitation pris en charge par votre opérateur:

```
labels:
  operatorframework.io/arch.s390x: supported
  operatorframework.io/os.zos: supported
  operatorframework.io/os.linux: supported 1
  operatorframework.io/arch.amd64: supported 2
```

1 2 Après avoir ajouté une nouvelle architecture ou un nouveau système d'exploitation, vous devez maintenant inclure explicitement les variantes os.linux et arch.amd64 par défaut.

Ressources supplémentaires

- Consultez la spécification Image Manifest V 2, Schema 2 pour plus d'informations sur les listes de manifestes.

5.6.5.1. Architecture et support du système d'exploitation pour les opérateurs

Les chaînes suivantes sont prises en charge dans Operator Lifecycle Manager (OLM) sur OpenShift Dedicated lors de l'étiquetage ou du filtrage des opérateurs qui prennent en charge plusieurs architectures et systèmes d'exploitation:

Tableau 5.12. Architectures prises en charge sur OpenShift Dedicated

Architecture	Chaîne de caractères
AMD64	**amd64**
ARM64	**bras64**
IBM Power®	**à propos de ppc64le**
IBM Z®	**à propos de S390x**

Tableau 5.13. Les systèmes d'exploitation pris en charge sur OpenShift Dedicated

Le système d'exploitation	Chaîne de caractères
Linux	**Linux**
à propos de Z/OS	**à propos de ZOS**

NOTE

Différentes versions d'OpenShift Dedicated et d'autres distributions basées sur Kubernetes peuvent prendre en charge un ensemble différent d'architectures et de systèmes d'exploitation.

5.6.6. Définition d'un espace de noms suggéré

Certains opérateurs doivent être déployés dans un espace de noms spécifique, ou avec des ressources accessoires dans des espaces de noms spécifiques, pour fonctionner correctement. En cas de résolution d'un abonnement, Operator Lifecycle Manager (OLM) par défaut des ressources d'un opérateur par défaut sur l'espace de noms de son abonnement.

En tant qu'auteur de l'opérateur, vous pouvez plutôt exprimer un espace de noms cible souhaité dans le cadre de votre version de service de cluster (CSV) afin de maintenir le contrôle sur les espaces de noms finaux des ressources installées pour leurs opérateurs. Lors de l'ajout de l'opérateur à un cluster en utilisant OperatorHub, cela permet à la console Web de remplir automatiquement l'espace de noms suggéré pour l'installateur pendant le processus d'installation.

Procédure

- Dans votre CSV, définissez l'annotation operatorframework.io/suggéré-namespace à votre espace de noms suggéré:

```
metadata:
  annotations:
    operatorframework.io/suggested-namespace: <namespace> 1
```

1 Définissez votre espace de noms suggéré.

5.6.7. Définir un espace de noms suggéré avec le sélecteur de nœud par défaut

Certains opérateurs s'attendent à fonctionner uniquement sur les nœuds de plan de contrôle, ce qui peut être fait en définissant un nodeSelector dans la spécification Pod par l'opérateur lui-même.

Afin d'éviter d'être dupliqués et potentiellement contradictoires, vous pouvez définir un sélecteur de nœuds par défaut sur l'espace de noms où l'opérateur s'exécute. Le sélecteur de nœud par défaut aura préséance sur le cluster par défaut, de sorte que le cluster par défaut ne sera pas appliqué aux pods dans l'espace de noms des opérateurs.

Lors de l'ajout de l'opérateur à un cluster en utilisant OperatorHub, la console Web peuple automatiquement l'espace de noms suggéré pour l'installateur pendant le processus d'installation. L'espace de noms suggéré est créé à l'aide du manifeste de l'espace de noms dans YAML qui est inclus dans la version de service cluster (CSV).

Procédure

- Dans votre CSV, définissez l'opérateurframework.io/suggéré-namespace-template avec un manifeste pour un objet Namespace. L'échantillon suivant est un manifeste pour un exemple Namespace avec le sélecteur de nœud par défaut de namespace spécifié:

```
metadata:
  annotations:
    operatorframework.io/suggested-namespace-template: 1
```

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "name": "vertical-pod-autoscaler-suggested-template",
    "annotations": {
      "openshift.io/node-selector": ""
    }
  }
}
```

1 Définissez votre espace de noms suggéré.

NOTE

Lorsque les annotations suggérées-namespace et suggéré-namespace-template sont présentes dans le CSV, le modèle suggéré-namespace-template devrait primer.

5.6.8. Conditions d'activation de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) fournit aux opérateurs un canal pour communiquer des états complexes qui influencent le comportement OLM tout en gérant l'opérateur. Par défaut, OLM crée une définition de ressource personnalisée OperatorCondition (CRD) lorsqu'elle installe un opérateur. En fonction des conditions définies dans la ressource personnalisée OperatorCondition (CR), le comportement de OLM change en conséquence.

Afin de prendre en charge les conditions de l'opérateur, un opérateur doit être en mesure de lire la version CR de l'opérateur créé par OLM et d'avoir la capacité d'accomplir les tâches suivantes:

- Ayez la condition spécifique.
- Définissez l'état d'une condition spécifique.

Cela peut être accompli en utilisant la bibliothèque opérateur-lib. L'auteur d'un opérateur peut fournir un client d'exécution du contrôleur dans son opérateur pour que la bibliothèque puisse accéder à la version CR de l'opérateur appartenant à l'opérateur dans le cluster.

La bibliothèque fournit une interface Conditions génériques, qui dispose des méthodes suivantes pour obtenir et définir un type de condition dans la version CR de l'opérateur:

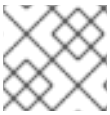
J'obtiens

Afin d'obtenir la condition spécifique, la bibliothèque utilise la fonction client.Get de contrôleur-runtime, qui nécessite une ObjectKey des types de type.NamespacedName présent dans conditionAccessor.

Ensemble

Afin de mettre à jour l'état de la condition spécifique, la bibliothèque utilise la fonction client.Update à partir du contrôleur-runtime. L'erreur se produit si la conditionType n'est pas présente dans le CRD.

L'opérateur est autorisé à modifier uniquement la sous-ressource d'état du CR. Les opérateurs peuvent soit supprimer ou mettre à jour le tableau status.conditions pour inclure la condition. Afin de plus amples détails sur le format et la description des champs présents dans les conditions, voir l'Amont Condition GoDocs.



NOTE

L'opérateur SDK 1.38.0 prend en charge la v0.11.0 de l'opérateur.

Conditions préalables

- Le projet d'opérateur généré à l'aide du SDK de l'opérateur.

Procédure

Activer les conditions de l'opérateur dans votre projet Opérateur:

1. Dans le fichier go.mod de votre projet Opérateur, ajoutez l'opérateur-cadre/opérateur-lib comme bibliothèque requise:

```
module github.com/example-inc/memcached-operator

go 1.19

require (
  k8s.io/apimachinery v0.26.0
  k8s.io/client-go v0.26.0
  sigs.k8s.io/controller-runtime v0.14.1
  operator-framework/operator-lib v0.11.0
)
```

2. Écrivez votre propre constructeur dans votre logique d'opérateur qui se traduira par les résultats suivants:

   - Accepte un client d'exécution de contrôleur.
   - Accepte un type de condition.
   - Il retourne une interface Condition pour mettre à jour ou ajouter des conditions.

   Comme OLM prend actuellement en charge la condition Upgradeable, vous pouvez créer une interface qui dispose de méthodes pour accéder à la condition Upgradeable. À titre d'exemple:

```
import (
  ...
  apiv1 "github.com/operator-framework/api/pkg/operators/v1"
)

func NewUpgradeable(cl client.Client) (Condition, error) {
  return NewCondition(cl, "apiv1.OperatorUpgradeable")
}

cond, err := NewUpgradeable(cl);
```

   Dans cet exemple, le constructeur NewUpgradeable est en outre utilisé pour créer un cond variable de type Condition. La variable cond aurait à son tour des méthodes Get and Set, qui peuvent être utilisées pour gérer la condition OLM Upgradeable.

Ressources supplémentaires

- Conditions de l'opérateur

5.6.9. Définir des webhooks

Les webhooks permettent aux auteurs de l'opérateur d'intercepter, de modifier et d'accepter ou de rejeter des ressources avant qu'elles ne soient enregistrées dans le magasin d'objets et traitées par le contrôleur de l'opérateur. Le gestionnaire de cycle de vie de l'opérateur (OLM) peut gérer le cycle de vie de ces webhooks lorsqu'ils sont expédiés aux côtés de votre opérateur.

La ressource de la version de service de cluster (CSV) d'un opérateur peut inclure une section Webhookdefinitions pour définir les types suivants de webhooks:

- Admission webhooks (validation et mutation)
- Conversion webhooks

Procédure

- Ajoutez une section Webhookdefinitions à la section Spécifications du CSV de votre opérateur et incluez toutes les définitions de webhook à l'aide d'un type de ValidatingAdmissionWebhook, MutatingAdmissionWebhook ou ConversionWebhook. L'exemple suivant contient les trois types de webhooks:

  CSV contenant des webhooks

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: webhook-operator.v0.0.1
spec:
  customresourcedefinitions:
    owned:
    - kind: WebhookTest
      name: webhooktests.webhook.operators.coreos.io 1
      version: v1
  install:
    spec:
      deployments:
      - name: webhook-operator-webhook
        ...
        ...
        ...
    strategy: deployment
  installModes:
  - supported: false
    type: OwnNamespace
  - supported: false
    type: SingleNamespace
  - supported: false
    type: MultiNamespace
  - supported: true
    type: AllNamespaces
  webhookdefinitions:
  - type: ValidatingAdmissionWebhook 2
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
```

```
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    failurePolicy: Fail
    generateName: vwebhooktest.kb.io
    rules:
    - apiGroups:
      - webhook.operators.coreos.io
      apiVersions:
      - v1
      operations:
      - CREATE
      - UPDATE
      resources:
      - webhooktests
    sideEffects: None
    webhookPath: /validate-webhook-operators-coreos-io-v1-webhooktest
  - type: MutatingAdmissionWebhook 3
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    failurePolicy: Fail
    generateName: mwebhooktest.kb.io
    rules:
    - apiGroups:
      - webhook.operators.coreos.io
      apiVersions:
      - v1
      operations:
      - CREATE
      - UPDATE
      resources:
      - webhooktests
    sideEffects: None
    webhookPath: /mutate-webhook-operators-coreos-io-v1-webhooktest
  - type: ConversionWebhook 4
    admissionReviewVersions:
    - v1beta1
    - v1
    containerPort: 443
    targetPort: 4343
    deploymentName: webhook-operator-webhook
    generateName: cwebhooktest.kb.io
    sideEffects: None
    webhookPath: /convert
    conversionCRDs:
    - webhooktests.webhook.operators.coreos.io 5
...
```

1. Les CRD ciblés par le webhook de conversion doivent exister ici.
2. C'est un webhook d'admission valide.

3. C'est un webhook d'admission mutant.
4. C'est un webhook de conversion.
5. La propriété spec.PreserveUnknownFields de chaque CRD doit être définie à faux ou nul.

Ressources supplémentaires

- Documentation de Kubernetes:
  - Validation des webhooks d'admission
  - La mutation des webhooks d'admission
  - Conversion webhooks

5.6.9.1. Considérations Webhook pour OLM

Lorsque vous déployez un opérateur avec des webhooks à l'aide du gestionnaire de cycle de vie de l'opérateur (OLM), vous devez définir ce qui suit:

- Le champ type doit être défini sur ValidatingAdmissionWebhook, MutatingAdmissionWebhook ou ConversionWebhook, ou le CSV sera placé dans une phase défaillante.
- Le CSV doit contenir un déploiement dont le nom est équivalent à la valeur fournie dans le champ deploymentName de la définition webhook.

Lorsque le webhook est créé, OLM s'assure que le webhook agit uniquement sur les espaces de noms correspondant au groupe d'opérateur dans lequel l'opérateur est déployé.

Contraintes de l'autorité de certification
L'OLM est configurée pour fournir à chaque déploiement une seule autorité de certificat (CA). La logique qui génère et monte l'AC dans le déploiement a été utilisée à l'origine par la logique du cycle de vie de l'API. En conséquence:

- Le fichier de certificat TLS est monté au déploiement sur /apiserver.local.config/certificates/apiserver.crt.
- Le fichier clé TLS est monté au déploiement sur /apiserver.local.config/certificates/apiserver.key.

Les règles d'admission sur le webhook
Afin d'empêcher un opérateur de configurer le cluster dans un état non récupérable, OLM place le CSV dans la phase défaillante si les règles définies dans un webhook d'admission interceptent l'une des requêtes suivantes:

- Demandes qui ciblent tous les groupes
- Demande qui cible le groupe operators.coreos.com
- Demandes qui ciblent les ressources ValidatingWebhookConfigurations ou MutatingWebhookConfigurations

Contraintes de conversion webhook
L'OMM place le CSV dans la phase défaillante si une définition de webhook de conversion n'adhère pas aux contraintes suivantes:

- Les CSV dotés d'un webhook de conversion ne peuvent prendre en charge que le mode d'installation d'AllNamespaces.
- Le CRD ciblé par le webhook de conversion doit avoir son champ spec.preserveUnknownFields mis à faux ou nul.
- Le webhook de conversion défini dans le CSV doit cibler un CRD détenu.
- Il ne peut y avoir qu'un seul webhook de conversion sur l'ensemble du cluster pour un CRD donné.

5.6.10. Comprendre vos définitions de ressources personnalisées (CRD)

Il existe deux types de définitions de ressources personnalisées (CRD) que votre opérateur peut utiliser : celles qui lui appartiennent et celles dont elle dépend, qui sont nécessaires.

5.6.10.1. Les CRD possédés

Les définitions de ressources personnalisées (CRD) détenues par votre opérateur sont la partie la plus importante de votre CSV. Cela établit le lien entre votre opérateur et les règles RBAC requises, la gestion des dépendances et d'autres concepts de Kubernetes.

Il est courant pour votre opérateur d'utiliser plusieurs CRD pour relier des concepts, tels que la configuration de base de données de haut niveau dans un objet et une représentation des ensembles de répliques dans un autre. Chacun doit être indiqué dans le fichier CSV.

Tableau 5.14. Champs CRD possédés

Le champ	Description	Requis/facultatif
Le nom	Le nom complet de votre CRD.	A) requis
La version	La version de cette API objet.	A) requis
Je suis gentille.	Le nom lisible par la machine de votre CRD.	A) requis
DisplayName	Il s'agit d'une version lisible humaine de votre nom CRD, par exemple MongoDB Standalone.	A) requis
Description	Brève description de la façon dont ce CRD est utilisé par l'Opérateur ou une description de la fonctionnalité fournie par le CRD.	A) requis
Groupe de travail	Le groupe API auquel appartient ce CRD, par exemple Database.example.com.	Facultatif

Le champ	Description	Requis/facultatif
Ressources	Les CRD possèdent un ou plusieurs types d'objets Kubernetes. Ceux-ci sont listés dans la section ressources pour informer vos utilisateurs des objets dont ils pourraient avoir besoin pour résoudre les problèmes ou comment se connecter à l'application, comme la règle du service ou de l'entrée qui expose une base de données. Il est recommandé de ne énumérer que les objets qui sont importants pour un humain, pas une liste exhaustive de tout ce que vous orchestrez. À titre d'exemple, ne répertoriez pas les cartes de configuration qui stockent l'état interne qui ne sont pas destinés à être modifiés par un utilisateur.	Facultatif
Description des SpecDescripteurs, StatutDescripteurs et ActionDescripteurs	Ces descripteurs sont un moyen d'indiquer les UI avec certaines entrées ou sorties de votre opérateur qui sont les plus importantes pour un utilisateur final. Lorsque votre CRD contient le nom d'une carte secrète ou de configuration que l'utilisateur doit fournir, vous pouvez le spécifier ici. Ces éléments sont liés et mis en évidence dans les UI compatibles. Il existe trois types de descripteurs: ● Description des Spec: Une référence aux champs dans le bloc spec d'un objet. ● Description du statut: Une référence aux champs dans le bloc d'état d'un objet. ● ActionDescriptors: Une référence aux actions qui peuvent être effectuées sur un objet. Les descripteurs acceptent les champs suivants: ● DisplayName: Un nom lisible par l'homme pour le Spec, Status ou Action. ● Description : Une courte description de la Spécification, de l'état ou de l'action et de la façon dont il est utilisé par l'opérateur. ● Chemin : Un chemin délimité par point du champ sur l'objet décrit par ce descripteur. ● Description des x : Utilisé pour déterminer les « capacités » de ce descripteur et quel composant d'interface utilisateur utiliser. Consultez le projet openshift/console pour une liste canonique de React UI X-Descriptors for OpenShift Dedicated. Consultez également le projet openshift/console pour plus d'informations sur Descripteurs en général.	Facultatif

L'exemple suivant représente un CRD autonome MongoDB qui nécessite une entrée de l'utilisateur sous la forme d'une carte secrète et de configuration, et orchestrate les services, les ensembles d'état, les pods et les cartes de configuration:

Exemple appartenant à CRD

```
- displayName: MongoDB Standalone
  group: mongodb.com
  kind: MongoDbStandalone
  name: mongodbstandalones.mongodb.com
  resources:
    - kind: Service
      name: ''
      version: v1
    - kind: StatefulSet
      name: ''
      version: v1beta2
    - kind: Pod
      name: ''
      version: v1
    - kind: ConfigMap
      name: ''
      version: v1
  specDescriptors:
    - description: Credentials for Ops Manager or Cloud Manager.
      displayName: Credentials
      path: credentials
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:selector:core:v1:Secret'
    - description: Project this deployment belongs to.
      displayName: Project
      path: project
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:selector:core:v1:ConfigMap'
    - description: MongoDB version to be installed.
      displayName: Version
      path: version
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:label'
  statusDescriptors:
    - description: The status of each of the pods for the MongoDB cluster.
      displayName: Pod Status
      path: pods
      x-descriptors:
        - 'urn:alm:descriptor:com.tectonic.ui:podStatuses'
  version: v1
  description: >-
    MongoDB Deployment consisting of only one host. No replication of
    data.
```

5.6.10.2. CRD requis

Compter sur d'autres CRD requis est complètement facultatif et n'existe que pour réduire la portée des opérateurs individuels et fournir un moyen de composer plusieurs opérateurs ensemble pour résoudre un cas d'utilisation de bout en bout.

C'est un exemple d'opérateur qui pourrait configurer une application et installer un cluster etcd (à partir d'un opérateur etcd) à utiliser pour le verrouillage distribué et une base de données Postgres (à partir d'un opérateur Postgres) pour le stockage des données.

Le gestionnaire de cycle de vie de l'opérateur (OLM) vérifie les CRD et les opérateurs disponibles dans le cluster pour répondre à ces exigences. Lorsque des versions appropriées sont trouvées, les opérateurs sont lancés dans l'espace de noms souhaité et un compte de service créé pour chaque opérateur pour créer, surveiller et modifier les ressources Kubernetes requises.

Tableau 5.15. Champs CRD requis

Le champ	Description	Requis/facultatif
Le nom	Le nom complet du CRD dont vous avez besoin.	A) requis
La version	La version de cette API objet.	A) requis
Je suis gentille.	Le genre d'objets Kubernetes.	A) requis
DisplayName	C'est une version lisible par l'homme de la CRD.	A) requis
Description	Le résumé de la façon dont le composant s'intègre dans votre architecture plus grande.	A) requis

Exemple requis CRD

```
required:
- name: etcdclusters.etcd.database.coreos.com
  version: v1beta2
  kind: EtcdCluster
  displayName: etcd Cluster
  description: Represents a cluster of etcd nodes.
```

5.6.10.3. Les mises à niveau de CRD

L'ODM met immédiatement à niveau une définition de ressource personnalisée (CRD) s'il appartient à une version de service de cluster singulier (CSV). Lorsqu'un CRD est détenu par plusieurs CSV, le CRD est mis à niveau lorsqu'il a satisfait à toutes les conditions suivantes:

- Dans le nouveau CRD, toutes les versions de service existantes sont présentes dans le nouveau CRD.
- Les instances existantes, ou ressources personnalisées, associées aux versions de service du CRD sont valides lorsqu'elles sont validées par rapport au schéma de validation du nouveau CRD.

5.6.10.3.1. Ajout d'une nouvelle version CRD

Procédure

Ajouter une nouvelle version d'un CRD à votre opérateur:

1. Ajoutez une nouvelle entrée dans la ressource CRD dans la section versions de votre CSV.

Ainsi, si le CRD actuel possède une version v1alpha1 et que vous souhaitez ajouter une nouvelle version v1beta1 et la marquer comme nouvelle version de stockage, ajoutez une nouvelle entrée pour v1beta1:

```
versions:
  - name: v1alpha1
    served: true
    storage: false
  - name: v1beta1 1
    served: true
    storage: true
```

1 C'est une nouvelle entrée.

2. Assurez-vous que la version de référence de la CRD dans la section propre de votre CSV est mise à jour si le CSV a l'intention d'utiliser la nouvelle version:

```
customresourcedefinitions:
  owned:
  - name: cluster.example.com
    version: v1beta1 1
    kind: cluster
    displayName: Cluster
```

1 Actualisez la version.

3. Appuyez sur le CRD et le CSV mis à jour sur votre paquet.

5.6.10.3.2. Dépréciation ou suppression d'une version CRD

Le gestionnaire de cycle de vie de l'opérateur (OLM) ne permet pas de supprimer immédiatement une version de service d'une définition de ressource personnalisée (CRD). Au lieu de cela, une version obsolète du CRD doit d'abord être désactivée en définissant le champ servi dans le CRD à faux. Ensuite, la version non-servante peut être supprimée sur la mise à niveau CRD ultérieure.

Procédure

Déprécier et supprimer une version spécifique d'un CRD:

1. Marquer la version obsolète comme non-servant pour indiquer que cette version n'est plus utilisée et peut être supprimée dans une mise à jour ultérieure. À titre d'exemple:

```
versions:
  - name: v1alpha1
    served: false 1
    storage: true
```

1 J'ai mis sur false.

2. Basculez la version de stockage en version de service si la version à déprécier est actuellement la version de stockage. À titre d'exemple:

```
versions:
  - name: v1alpha1
    served: false
    storage: false 1
  - name: v1beta1
    served: true
    storage: true 2
```

1 2 Actualisez les champs de stockage en conséquence.

NOTE

Afin de supprimer une version spécifique qui est ou était la version de stockage d'un CRD, cette version doit être supprimée de la version stockée dans l'état du CRD. Les OLM tenteront de le faire pour vous s'il détecte qu'une version stockée n'existe plus dans le nouveau CRD.

3. Améliorez le CRD avec les modifications ci-dessus.

4. Dans les cycles de mise à niveau ultérieurs, la version non-servante peut être complètement supprimée du CRD. À titre d'exemple:

```
versions:
  - name: v1beta1
    served: true
    storage: true
```

5. Assurez-vous que la version CRD de référence dans la section détenue de votre CSV est mise à jour en conséquence si cette version est supprimée du CRD.

5.6.10.4. Les modèles CRD

Les utilisateurs de votre opérateur doivent être informés des options requises par rapport aux options facultatives. Il est possible de fournir des modèles pour chacune de vos définitions de ressources personnalisées (CRD) avec un ensemble minimal de configuration sous forme d'annotation nommée alm-exemples. Compatible UIs pré-remplira ce modèle pour les utilisateurs à personnaliser davantage.

L'annotation se compose d'une liste du type, par exemple, le nom CRD et les métadonnées et spécifications correspondantes de l'objet Kubernetes.

L'exemple complet suivant fournit des modèles pour EtcdCluster, EtcdBackup et EtcdRestore:

```
metadata:
  annotations:
    alm-examples: >-
      [{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdCluster","metadata":
{"name":"example","namespace":"<operator_namespace>"},"spec":{"size":3,"version":"3.2.13"}},
{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdRestore","metadata":
{"name":"example-etcd-cluster"},"spec":{"etcdCluster":{"name":"example-etcd-
cluster"},"backupStorageType":"S3","s3":{"path":"<full-s3-path>","awsSecret":"<aws-secret>"}}},
{"apiVersion":"etcd.database.coreos.com/v1beta2","kind":"EtcdBackup","metadata":
{"name":"example-etcd-cluster-backup"},"spec":{"etcdEndpoints":["<etcd-cluster-
endpoints>"],"storageType":"S3","s3":{"path":"<full-s3-path>","awsSecret":"<aws-secret>"}}}]
```

5.6.10.5. Cacher des objets internes

Il est courant pour les opérateurs d'utiliser des définitions de ressources personnalisées (CRD) en interne pour accomplir une tâche. Ces objets ne sont pas destinés aux utilisateurs à manipuler et peuvent être source de confusion pour les utilisateurs de l'opérateur. À titre d'exemple, un opérateur de base de données peut avoir un CRD de réplication qui est créé chaque fois qu'un utilisateur crée un objet de base de données avec réplication: true.

En tant qu'auteur de l'opérateur, vous pouvez cacher tous les CRD dans l'interface utilisateur qui ne sont pas destinés à la manipulation de l'utilisateur en ajoutant l'annotation d'objets internes à la version de service cluster (CSV) de votre opérateur.

Procédure

1. Avant de marquer l'un de vos CRD en interne, assurez-vous que toute information de débogage ou configuration qui pourrait être nécessaire pour gérer l'application est reflétée sur le statut ou le bloc de spécifications de votre CR, le cas échéant à votre opérateur.

2. Ajoutez l'annotation d'un objet interne au CSV de votre opérateur pour spécifier les objets internes à cacher dans l'interface utilisateur:

   Annotation interne d'objet

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  name: my-operator-v1.2.3
  annotations:
    operators.operatorframework.io/internal-objects:
'["my.internal.crd1.io","my.internal.crd2.io"]' 1
...
```

   1 Définissez n'importe quel CRD interne comme un tableau de chaînes.

5.6.10.6. Initialisation des ressources personnalisées requises

L'opérateur peut demander à l'utilisateur d'instancier une ressource personnalisée avant que l'opérateur puisse être pleinement fonctionnel. Cependant, il peut être difficile pour un utilisateur de déterminer ce qui est nécessaire ou comment définir la ressource.

En tant que développeur d'opérateur, vous pouvez spécifier une ressource personnalisée unique requise en ajoutant operatorframework.io/initialisation-ressource à la version de service cluster (CSV) pendant l'installation de l'opérateur. Il vous est alors demandé de créer la ressource personnalisée à l'aide d'un modèle fourni dans le CSV. L'annotation doit inclure un modèle qui contient une définition YAML complète qui est nécessaire pour initialiser la ressource pendant l'installation.

Lorsque cette annotation est définie, après avoir installé l'opérateur à partir de la console Web dédiée OpenShift, l'utilisateur est invité à créer la ressource à l'aide du modèle fourni dans le CSV.

Procédure

- Ajoutez l'annotation operatorframework.io/initialisation-ressource au CSV de votre opérateur pour spécifier une ressource personnalisée requise. À titre d'exemple, l'annotation suivante nécessite la création d'une ressource StorageCluster et fournit une définition YAML complète:

Annotation des ressources d'initialisation

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
 name: my-operator-v1.2.3
 annotations:
  operatorframework.io/initialization-resource: |-
    {
        "apiVersion": "ocs.openshift.io/v1",
        "kind": "StorageCluster",
        "metadata": {
            "name": "example-storagecluster"
        },
        "spec": {
            "manageNodes": false,
            "monPVCTemplate": {
                "spec": {
                    "accessModes": [
                        "ReadWriteOnce"
                    ],
                    "resources": {
                        "requests": {
                            "storage": "10Gi"
                        }
                    },
                    "storageClassName": "gp2"
                }
            },
            "storageDeviceSets": [
                {
                    "count": 3,
                    "dataPVCTemplate": {
                        "spec": {
                            "accessModes": [
                                "ReadWriteOnce"
                            ],
                            "resources": {
                                "requests": {
                                    "storage": "1Ti"
                                }
                            },
                            "storageClassName": "gp2",
                            "volumeMode": "Block"
                        }
                    },
                    "name": "example-deviceset",
                    "placement": {},
                    "portable": true,
                    "resources": {}
                }
            ]
        }
    }
...
```

5.6.11. Comprendre vos services API

Comme pour les CRD, il existe deux types de services d'API que votre opérateur peut utiliser: possédés et requis.

5.6.11.1. Les services API possédés

Lorsqu'un CSV possède un service API, il est responsable de décrire le déploiement de l'extension api-server qui le soutient et du groupe/version/type (GVK) qu'il fournit.

Le service API est identifié de manière unique par le groupe/version qu'il fournit et peut être répertorié plusieurs fois pour désigner les différents types qu'il est censé fournir.

Tableau 5.16. Champs de service API possédés

Le champ	Description	Requis/facultatif
Groupe de travail	Groupe que le service API fournit, par exemple Database.example.com.	A) requis
La version	La version du service API, par exemple v1alpha1.	A) requis
Je suis gentille.	C'est une sorte que le service API devrait fournir.	A) requis
Le nom	Le nom pluriel du service API fourni.	A) requis
DéploiementNam e	Le nom du déploiement défini par votre CSV qui correspond à votre service API (requis pour les services API possédés). Au cours de la phase d'attente CSV, l'opérateur OLM effectue une recherche dans la stratégie d'installation de votre CSV pour trouver une spécification de déploiement avec un nom correspondant, et si elle n'est pas trouvée, ne transitionne pas le CSV vers la phase "Installer Prêt".	A) requis
DisplayName	Il s'agit d'une version lisible humaine du nom de votre service API, par exemple MongoDB Standalone.	A) requis
Description	Brève description de la façon dont ce service API est utilisé par l'opérateur ou une description des fonctionnalités fournies par le service API.	A) requis
Ressources	Les services API possèdent un ou plusieurs types d'objets Kubernetes. Ceux-ci sont listés dans la section ressources pour informer vos utilisateurs des objets dont ils pourraient avoir besoin pour résoudre les problèmes ou comment se connecter à l'application, comme la règle du service ou de l'entrée qui expose une base de données. Il est recommandé de ne énumérer que les objets qui sont importants pour un humain, pas une liste exhaustive de tout ce que vous orchestrez. À titre d'exemple, ne répertoriez pas les cartes de configuration qui stockent l'état interne qui ne sont pas destinés à être modifiés par un utilisateur.	Facultatif

Le champ	Description	Requis/facultatif
Description des SpecDescripteurs, StatutDescripteurs et ActionDescripteurs	Essentiellement les mêmes que pour les CRD possédés.	Facultatif

5.6.11.1.1. Création de ressources de service API

Le gestionnaire de cycle de vie de l'opérateur (OLM) est responsable de la création ou du remplacement des ressources de service et d'API pour chaque service d'API propriétaire unique:

- Les sélecteurs de pod de service sont copiés à partir du déploiement CSV correspondant au champ DeploymentName de la description du service API.
- La nouvelle paire clé/certificat CA est générée pour chaque installation et le paquet CA codé de base64 est intégré dans la ressource de service API respective.

5.6.11.1.2. Certificats de service API

Chaque fois qu'un service d'API est installé, OLM gère la génération d'une paire de clés/certificats de service. Le certificat de service a un nom commun (CN) contenant le nom d'hôte de la ressource Service généré et est signé par la clé privée du paquet CA intégré dans la ressource de service API correspondante.

Le certificat est stocké sous forme de secret de type kubernetes.io/tls dans l'espace de noms de déploiement, et un volume nommé apiservice-cert est automatiquement annexé à la section volumes du déploiement dans le CSV correspondant au champ DeploymentName de la description du service API.

Dans le cas contraire, une monture de volume avec un nom correspondant est également jointe à tous les conteneurs de ce déploiement. Cela permet aux utilisateurs de définir une monture de volume avec le nom attendu pour répondre à toutes les exigences de chemin personnalisé. Le chemin du montage de volume généré par défaut vers /apiserver.local.config/certificates et tous les montages de volume existants avec le même chemin sont remplacés.

5.6.11.2. Les services d'API requis

L'OLM s'assure que tous les CSV requis disposent d'un service API disponible et que tous les GVK attendus sont détectables avant de tenter l'installation. Cela permet à un CSV de s'appuyer sur des types spécifiques fournis par les services API qu'il ne possède pas.

Tableau 5.17. Champs de service API requis

Le champ	Description	Requis/facultatif
Groupe de travail	Groupe que le service API fournit, par exemple Database.example.com.	A) requis
La version	La version du service API, par exemple v1alpha1.	A) requis
Je suis gentille.	C'est une sorte que le service API devrait fournir.	A) requis
DisplayName	Il s'agit d'une version lisible humaine du nom de votre service API, par exemple MongoDB Standalone.	A) requis
Description	Brève description de la façon dont ce service API est utilisé par l'opérateur ou une description des fonctionnalités fournies par le service API.	A) requis

5.7. EN TRAVAILLANT AVEC DES IMAGES GROUPÉES

Il est possible d'utiliser le SDK de l'opérateur pour emballer, déployer et mettre à niveau les Opérateurs dans le format bundle pour les utiliser sur Operator Lifecycle Manager (OLM).

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.7.1. Groupement d'un opérateur

Le format de paquet Opérateur est la méthode d'emballage par défaut pour Operator SDK et Operator Lifecycle Manager (OLM). En utilisant le SDK de l'opérateur, vous pouvez préparer votre opérateur à une utilisation sur OLM pour construire et pousser votre projet Opérateur en tant qu'image groupée.

Conditions préalables

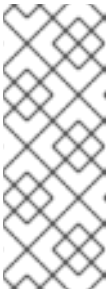
- L'opérateur SDK CLI installé sur un poste de travail de développement
- Installation d'OpenShift CLI (oc) v4+
- Le projet d'opérateur initialisé à l'aide du SDK de l'opérateur
- Dans le cas où votre opérateur est Go-based, votre projet doit être mis à jour pour utiliser les images prises en charge pour s'exécuter sur OpenShift Dedicated

Procédure

1. Exécutez les commandes suivantes dans votre répertoire de projet Opérateur pour construire et pousser l'image de votre opérateur. Modifiez l'argument IMG dans les étapes suivantes pour faire référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

   a. Construire l'image:

   ```
   $ make docker-build IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   NOTE

   Le Dockerfile généré par le SDK pour l'opérateur renvoie explicitement GOARCH=amd64 pour la construction de go. Cela peut être modifié à GOARCH=$TARGETARCH pour les architectures non-AMD64. Docker définira automatiquement la variable d'environnement à la valeur spécifiée par -platform. Avec Buildah, le -build-arg devra être utilisé à cet effet. En savoir plus, consultez Multiple Architectures.

   b. Appuyez sur l'image vers un référentiel:

   ```
   $ make docker-push IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

2. Créez votre paquet Opérateur manifeste en exécutant la commande make bundle, qui invoque plusieurs commandes, y compris l'opérateur SDK génère des paquets et des sous-commandes validant:

   ```
   $ make bundle IMG=<registry>/<user>/<operator_image_name>:<tag>
   ```

   Les manifestes de paquets pour un opérateur décrivent comment afficher, créer et gérer une application. La commande make bundle crée les fichiers et répertoires suivants dans votre projet Opérateur:

   - Le bundle manifeste un répertoire nommé bundle/manifests qui contient un objet ClusterServiceVersion
   - Annuaire de métadonnées groupé nommé bundle/metadata

- L'ensemble des définitions de ressources personnalisées (CRD) dans un répertoire config/crd
- Dockerfile bundle.Dockerfile

Ces fichiers sont ensuite automatiquement validés en utilisant le bundle opérateur-sdk valide pour s'assurer que la représentation des faisceaux sur disque est correcte.

3. Créez et poussez votre image de paquet en exécutant les commandes suivantes. L'OLM consomme des faisceaux d'opérateurs à l'aide d'une image d'index, qui référence à une ou plusieurs images groupées.

   a. Construisez l'image du bundle. Définissez BUNDLE_IMG avec les détails du registre, de l'espace de noms d'utilisateur et de la balise d'image où vous avez l'intention de pousser l'image:

   ```
   $ make bundle-build BUNDLE_IMG=<registry>/<user>/<bundle_image_name>:<tag>
   ```

   b. Appuyez sur l'image du paquet:

   ```
   $ docker push <registry>/<user>/<bundle_image_name>:<tag>
   ```

5.7.2. Déploiement d'un opérateur avec le gestionnaire du cycle de vie de l'opérateur

Le gestionnaire de cycle de vie de l'opérateur (OLM) vous aide à installer, mettre à jour et gérer le cycle de vie des Opérateurs et de leurs services associés sur un cluster Kubernetes. Le système OLM est installé par défaut sur OpenShift Dedicated et s'exécute sous forme d'extension Kubernetes afin que vous puissiez utiliser la console Web et l'OpenShift CLI (oc) pour toutes les fonctions de gestion du cycle de vie de l'opérateur sans outils supplémentaires.

Le format de paquet opérateur est la méthode d'emballage par défaut pour l'opérateur SDK et OLM. Le SDK de l'opérateur permet d'exécuter rapidement une image groupée sur OLM afin de s'assurer qu'elle fonctionne correctement.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement
- Ensemble d'image de l'opérateur construit et poussé à un registre
- Installation OLM sur un cluster basé sur Kubernetes (v1.16.0 ou version ultérieure si vous utilisez apiextensions.k8s.io/v1 CRD, par exemple OpenShift Dedicated 4)
- Connexion au cluster avec oc à l'aide d'un compte avec des autorisations d'administration dédiées
- Dans le cas où votre opérateur est Go-based, votre projet doit être mis à jour pour utiliser les images prises en charge pour s'exécuter sur OpenShift Dedicated

Procédure

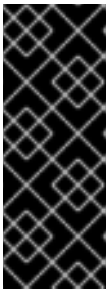
- Entrez la commande suivante pour exécuter l'opérateur sur le cluster:

```
$ operator-sdk run bundle \ 1
    -n <namespace> \ 2
    <registry>/<user>/<bundle_image_name>:<tag> 3
```

1. La commande run bundle crée un catalogue basé sur des fichiers valide et installe le paquet Opérateur sur votre cluster en utilisant OLM.

2. Facultatif: Par défaut, la commande installe l'opérateur dans le projet actuellement actif dans votre fichier ~/.kube/config. Il est possible d'ajouter le drapeau -n pour définir un espace de noms différent pour l'installation.

3. Dans le cas où vous ne spécifiez pas une image, la commande utilise quay.io/operator-framework/opm:latest comme image d'index par défaut. Lorsque vous spécifiez une image, la commande utilise l'image du faisceau lui-même comme image d'index.



IMPORTANT

À partir d'OpenShift Dedicated 4.11, la commande run bundle prend en charge le format de catalogue basé sur des fichiers pour les catalogues Opérateur par défaut. Le format de base de données SQLite obsolète pour les catalogues d'opérateurs continue d'être pris en charge; cependant, il sera supprimé dans une version ultérieure. Il est recommandé aux auteurs de l'opérateur de migrer leurs flux de travail vers le format de catalogue basé sur les fichiers.

Cette commande effectue les actions suivantes:

- Créez une image d'index faisant référence à votre image de paquet. L'image de l'index est opaque et éphémère, mais reflète avec précision comment un paquet serait ajouté à un catalogue en production.
- Créez une source de catalogue qui pointe vers votre nouvelle image d'index, ce qui permet à OperatorHub de découvrir votre opérateur.
- Déployez votre opérateur dans votre cluster en créant un groupe d'opérateurs, un abonnement, un plan d'installation et toutes les autres ressources requises, y compris RBAC.

Ressources supplémentaires

- Catalogues basés sur des fichiers au format d'emballage du cadre opérateur
- Catalogues basés sur des fichiers dans Gérer des catalogues personnalisés
- Format de paquet

5.7.3. La publication d'un catalogue contenant un opérateur groupé

Afin d'installer et de gérer les opérateurs, le gestionnaire de cycle de vie de l'opérateur (OLM) exige que les paquets d'opérateurs soient listés dans une image d'index, qui est référencée par un catalogue sur le cluster. En tant qu'auteur de l'opérateur, vous pouvez utiliser le SDK de l'opérateur pour créer un index contenant le paquet pour votre opérateur et toutes ses dépendances. Ceci est utile pour tester les clusters distants et publier dans des registres de conteneurs.

NOTE

Le SDK d'opérateur utilise l'opm CLI pour faciliter la création d'images d'index. L'expérience avec la commande opm n'est pas requise. Dans les cas avancés d'utilisation, la commande opm peut être utilisée directement au lieu du SDK de l'opérateur.

Conditions préalables

- L'opérateur SDK CLI installé sur un poste de travail de développement
- Ensemble d'image de l'opérateur construit et poussé à un registre
- Installation OLM sur un cluster basé sur Kubernetes (v1.16.0 ou version ultérieure si vous utilisez apiextensions.k8s.io/v1 CRD, par exemple OpenShift Dedicated 4)
- Connexion au cluster avec oc à l'aide d'un compte avec des autorisations d'administration dédiées

Procédure

1. Exécutez la commande make suivante dans votre répertoire de projet Opérateur pour créer une image d'index contenant votre paquet Opérateur:

```
$ make catalog-build CATALOG_IMG=<registry>/<user>/<index_image_name>:<tag>
```

   lorsque l'argument CATALOG_IMG fait référence à un référentiel auquel vous avez accès. Il est possible d'obtenir un compte de stockage des conteneurs sur des sites de dépôt tels que Quay.io.

2. Appuyez sur l'image d'index construite vers un référentiel:

```
$ make catalog-push CATALOG_IMG=<registry>/<user>/<index_image_name>:<tag>
```

   ASTUCE

   Il est possible d'utiliser les commandes de l'opérateur SDK si vous préférez effectuer plusieurs actions en séquence à la fois. Ainsi, si vous n'avez pas encore construit une image groupée pour votre projet Opérateur, vous pouvez construire et pousser à la fois une image de paquet et une image d'index avec la syntaxe suivante:

```
$ make bundle-build bundle-push catalog-build catalog-push \
    BUNDLE_IMG=<bundle_image_pull_spec> \
    CATALOG_IMG=<index_image_pull_spec>
```

   Alternativement, vous pouvez définir le champ IMAGE_TAG_BASE dans votre Makefile sur un référentiel existant:

```
IMAGE_TAG_BASE=quay.io/example/my-operator
```

   Ensuite, vous pouvez utiliser la syntaxe suivante pour créer et pousser des images avec des noms générés automatiquement, tels que quay.io/example/my-operator-bundle:v0.0.1 pour l'image de paquet et quay.io/example/my-operator-catalog:v0.0.1 pour l'image d'index:

```
$ make bundle-build bundle-push catalog-build catalog-push
```

3. Définissez un objet CatalogSource qui fait référence à l'image d'index que vous venez de générer, puis créez l'objet en utilisant la commande oc Apply ou la console web:

   Exemple CatalogSource YAML

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: CatalogSource
   metadata:
     name: cs-memcached
     namespace: <operator_namespace>
   spec:
     displayName: My Test
     publisher: Company
     sourceType: grpc
     grpcPodConfig:
       securityContextConfig: <security_mode> 1
     image: quay.io/example/memcached-catalog:v0.0.1 2
     updateStrategy:
       registryPoll:
         interval: 10m
   ```

   1 Indiquez la valeur de l'héritage ou de la restriction. Lorsque le champ n'est pas défini, la valeur par défaut est héritée. Dans une version ultérieure d'OpenShift Dedicated, il est prévu que la valeur par défaut soit limitée. Dans le cas où votre catalogue ne peut pas fonctionner avec des autorisations restreintes, il est recommandé de définir manuellement ce champ sur l'héritage.

   2 Définissez l'image sur la spécification d'extraction de l'image que vous avez utilisée précédemment avec l'argument CATALOG_IMG.

4. Consultez la source du catalogue:

   ```
   $ oc get catalogsource
   ```

   Exemple de sortie

   ```
   NAME          DISPLAY     TYPE   PUBLISHER   AGE
   cs-memcached  My Test     grpc   Company     4h31m
   ```

La vérification

1. Installez l'opérateur à l'aide de votre catalogue:

   a. Définissez un objet OperatorGroup et créez-le en utilisant la commande oc Apply ou la console web:

      Exemple d'opérateur Groupe YAML

      ```
      apiVersion: operators.coreos.com/v1
      kind: OperatorGroup
      metadata:
        name: my-test
        namespace: <operator_namespace>
      ```

```
spec:
  targetNamespaces:
  - <operator_namespace>
```

b. Définissez un objet d'abonnement et créez-le en utilisant la commande oc Apply ou la console web:

Exemple d'abonnement YAML

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: catalogtest
  namespace: <catalog_namespace>
spec:
  channel: "alpha"
  installPlanApproval: Manual
  name: catalog
  source: cs-memcached
  sourceNamespace: <operator_namespace>
  startingCSV: memcached-operator.v0.0.1
```

2. L'opérateur installé est en cours d'exécution:

a. Consultez le groupe d'opérateurs:

```
$ oc get og
```

Exemple de sortie

```
NAME            AGE
my-test         4h40m
```

b. Consultez la version du service cluster (CSV):

```
$ oc get csv
```

Exemple de sortie

```
NAME                    DISPLAY   VERSION   REPLACES   PHASE
memcached-operator.v0.0.1   Test      0.0.1             Succeeded
```

c. Consultez les pods pour l'opérateur:

```
$ oc get pods
```

Exemple de sortie

```
NAME                                                READY   STATUS      RESTARTS   AGE
9098d908802769fbde8bd45255e69710a9f8420a8f3d814abe88b68f8ervdj6   0/1
Completed   0          4h33m
```

```
catalog-controller-manager-7fd5b7b987-69s4n          2/2   Running   0
4h32m
cs-memcached-7622r                                   1/1   Running   0          4h33m
```

Ressources supplémentaires

- Consultez Gérer les catalogues personnalisés pour plus de détails sur l'utilisation directe de l'opm CLI pour les cas d'utilisation plus avancés.

5.7.4. Tester une mise à niveau de l'opérateur sur le gestionnaire de cycle de vie de l'opérateur

Il est possible de tester rapidement la mise à niveau de votre opérateur en utilisant l'intégration du gestionnaire de cycle de vie de l'opérateur (OLM) dans le SDK de l'opérateur, sans avoir à gérer manuellement les images d'index et les sources de catalogue.

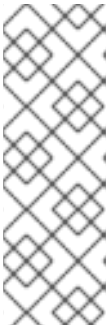
La sous-commande run bundle-upgrade automatise le déclenchement d'un opérateur installé pour passer à une version ultérieure en spécifiant une image de paquet pour la version ultérieure.

Conditions préalables

- L'opérateur installé avec OLM soit à l'aide de la sous-commande run bundle ou avec l'installation OLM traditionnelle
- Image groupée qui représente une version ultérieure de l'opérateur installé

Procédure

1. Dans le cas où votre opérateur n'a pas déjà été installé avec OLM, installez la version précédente soit en utilisant la sous-commande run bundle ou avec l'installation OLM traditionnelle.



NOTE

Lorsque la version antérieure du paquet a été installée traditionnellement à l'aide d'ODM, le nouveau paquet que vous avez l'intention de mettre à niveau ne doit pas exister dans l'image d'index référencée par la source du catalogue. Dans le cas contraire, l'exécution de la sous-commande de mise à niveau de paquets d'exécution entraînera l'échec de la pod de registre parce que le nouveau paquet est déjà référencé par l'index qui fournit la version de service de paquet et de cluster (CSV).

À titre d'exemple, vous pouvez utiliser la sous-commande de paquets d'exécution suivante pour un opérateur Memcached en spécifiant l'image du paquet précédent:

```
$ operator-sdk run bundle <registry>/<user>/memcached-operator:v0.0.1
```

Exemple de sortie

```
INFO[0006] Creating a File-Based Catalog of the bundle "quay.io/demo/memcached-
operator:v0.0.1"
INFO[0008] Generated a valid File-Based Catalog
INFO[0012] Created registry pod: quay-io-demo-memcached-operator-v1-0-1
```

```
INFO[0012] Created CatalogSource: memcached-operator-catalog
INFO[0012] OperatorGroup "operator-sdk-og" created
INFO[0012] Created Subscription: memcached-operator-v0-0-1-sub
INFO[0015] Approved InstallPlan install-h9666 for the Subscription: memcached-operator-v0-0-1-sub
INFO[0015] Waiting for ClusterServiceVersion "my-project/memcached-operator.v0.0.1" to reach 'Succeeded' phase
INFO[0015] Waiting for ClusterServiceVersion ""my-project/memcached-operator.v0.0.1" to appear
INFO[0026] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1" phase: Pending
INFO[0028] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1" phase: Installing
INFO[0059] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.1" phase: Succeeded
INFO[0059] OLM has successfully installed "memcached-operator.v0.0.1"
```

2. Améliorez l'opérateur installé en spécifiant l'image du paquet pour la version ultérieure de l'opérateur:

```
$ operator-sdk run bundle-upgrade <registry>/<user>/memcached-operator:v0.0.2
```

Exemple de sortie

```
INFO[0002] Found existing subscription with name memcached-operator-v0-0-1-sub and namespace my-project
INFO[0002] Found existing catalog source with name memcached-operator-catalog and namespace my-project
INFO[0008] Generated a valid Upgraded File-Based Catalog
INFO[0009] Created registry pod: quay-io-demo-memcached-operator-v0-0-2
INFO[0009] Updated catalog source memcached-operator-catalog with address and annotations
INFO[0010] Deleted previous registry pod with name "quay-io-demo-memcached-operator-v0-0-1"
INFO[0041] Approved InstallPlan install-gvcjh for the Subscription: memcached-operator-v0-0-1-sub
INFO[0042] Waiting for ClusterServiceVersion "my-project/memcached-operator.v0.0.2" to reach 'Succeeded' phase
INFO[0019] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2" phase: Pending
INFO[0042] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2" phase: InstallReady
INFO[0043] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2" phase: Installing
INFO[0044] Found ClusterServiceVersion "my-project/memcached-operator.v0.0.2" phase: Succeeded
INFO[0044] Successfully upgraded to "memcached-operator.v0.0.2"
```

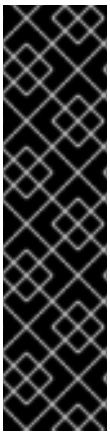
3. Nettoyer les opérateurs installés:

```
$ operator-sdk cleanup memcached-operator
```

Ressources supplémentaires

- Installation traditionnelle d'opérateur avec OLM

5.7.5. Contrôle de la compatibilité de l'opérateur avec les versions dédiées d'OpenShift



IMPORTANT

Kubernetes déprécie périodiquement certaines API qui sont supprimées dans les versions ultérieures. Lorsque votre opérateur utilise une API obsolète, il se peut qu'elle ne fonctionne plus après que le cluster OpenShift Dedicated ait été mis à niveau vers la version Kubernetes où l'API a été supprimée.

En tant qu'auteur de l'opérateur, il est fortement recommandé de consulter le Guide de migration de l'API obsolète dans la documentation Kubernetes et de garder vos projets d'opérateur à jour afin d'éviter d'utiliser des API dépréciées et supprimées. Idéalement, vous devriez mettre à jour votre opérateur avant la sortie d'une future version d'OpenShift Dedicated qui rendrait l'opérateur incompatible.

Lorsqu'une API est supprimée d'une version dédiée à OpenShift, les opérateurs s'exécutant sur cette version de cluster qui utilisent toujours des API supprimées ne fonctionneront plus correctement. En tant qu'auteur de l'opérateur, vous devez prévoir de mettre à jour vos projets d'opérateur afin de tenir compte de la déprécation et de la suppression de l'API afin d'éviter les interruptions pour les utilisateurs de votre opérateur.

ASTUCE

Consultez les alertes d'événements de vos opérateurs pour savoir s'il y a des avertissements sur les API actuellement utilisées. Les alertes suivantes s'allument lorsqu'elles détectent une API utilisée qui sera supprimée dans la prochaine version:

APIRemovedInNextReleaseInUse
: API qui seront supprimées dans la prochaine version OpenShift Dedicated.

APIRemovedInNextEUSReleaseInUse
: API qui seront supprimées dans la prochaine version OpenShift Dedicated Extended Update Support (EUS).

Lorsqu'un administrateur de cluster a installé votre opérateur, avant de passer à la prochaine version d'OpenShift Dedicated, il doit s'assurer qu'une version de votre opérateur est installée et compatible avec la prochaine version du cluster. Bien qu'il soit recommandé de mettre à jour vos projets Opérateur pour ne plus utiliser d'API dépréciée ou supprimée, si vous avez encore besoin de publier vos paquets Opérateur avec des API supprimées pour une utilisation continue sur les versions antérieures d'OpenShift Dedicated, assurez-vous que le paquet est configuré en conséquence.

La procédure suivante empêche les administrateurs d'installer des versions de votre opérateur sur une version incompatible d'OpenShift Dedicated. Ces étapes empêchent également les administrateurs de passer à une nouvelle version d'OpenShift Dedicated incompatible avec la version de votre opérateur qui est actuellement installée sur leur cluster.

Cette procédure est également utile lorsque vous savez que la version actuelle de votre opérateur ne fonctionnera pas bien, pour quelque raison que ce soit, sur une version spécifique d'OpenShift Dedicated. En définissant les versions de cluster où l'opérateur doit être distribué, vous vous assurez que l'opérateur n'apparaît pas dans un catalogue d'une version de cluster qui est en dehors de la plage autorisée.

IMPORTANT

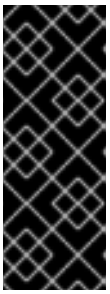
Les opérateurs qui utilisent des API obsolètes peuvent avoir un impact négatif sur les charges de travail critiques lorsque les administrateurs de cluster passent à une future version d'OpenShift Dedicated où l'API n'est plus prise en charge. Lorsque votre opérateur utilise des API obsolètes, vous devez configurer les paramètres suivants dans votre projet Opérateur dès que possible.

Conditions préalables

- D'un projet opérateur existant

Procédure

1. Lorsque vous savez qu'un paquet spécifique de votre opérateur n'est pas pris en charge et ne fonctionne pas correctement sur OpenShift Dedicated plus tard qu'une certaine version de cluster, configurez la version maximale d'OpenShift Dedicated avec laquelle votre opérateur est compatible. Dans la version de service cluster de votre projet Opérateur (CSV), définissez l'annotation olm.maxOpenShiftVersion pour empêcher les administrateurs de mettre à niveau leur cluster avant de mettre à niveau l'opérateur installé vers une version compatible:



IMPORTANT

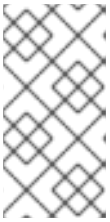
Il faut utiliser l'annotation olm.maxOpenShiftVersion uniquement si la version de votre bundle Opérateur ne peut pas fonctionner dans les versions ultérieures. Gardez à l'esprit que les administrateurs de clusters ne peuvent pas mettre à jour leurs clusters avec votre solution installée. Dans le cas où vous ne fournissez pas une version ultérieure et un chemin de mise à niveau valide, les administrateurs peuvent désinstaller votre opérateur et mettre à niveau la version du cluster.

Exemple CSV avec olm.maxOpenShiftVersion annotation

```
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    "olm.properties": '[{"type": "olm.maxOpenShiftVersion", "value": "<cluster_version>"}]' (1)
```

(1) Indiquez la version de cluster maximale d'OpenShift dédiée à laquelle votre opérateur est compatible. À titre d'exemple, le réglage de la valeur à 4.9 empêche les mises à niveau de cluster vers les versions OpenShift Dédicated plus tard que 4.9 lorsque ce paquet est installé sur un cluster.

2. Dans le cas où votre paquet est destiné à être distribué dans un catalogue d'opérateurs Red Hat, configurez les versions compatibles d'OpenShift Dedicated pour votre opérateur en définissant les propriétés suivantes. Cette configuration garantit que votre opérateur n'est inclus que dans les catalogues qui ciblent les versions compatibles d'OpenShift Dedicated:



NOTE

Cette étape n'est valable que lorsque vous publiez des opérateurs dans des catalogues fournis par Red Hat. Dans le cas où votre paquet n'est destiné qu'à la distribution dans un catalogue personnalisé, vous pouvez sauter cette étape. En savoir plus, voir "Catalogues de l'opérateur fourni par le chapeau rouge".

a. Définissez l'annotation com.redhat.openshift.versions dans le fichier bundle/metadata/annotations.yaml de votre projet:

Exemple de fichier bundle/metadata/annotations.yaml avec des versions compatibles

```
com.redhat.openshift.versions: "v4.7-v4.9" 1
```

1 Défini sur une gamme ou une seule version.

b. Afin d'éviter que votre paquet ne soit transmis à une version incompatible d'OpenShift Dedicated, assurez-vous que l'image de l'index est générée avec l'étiquette com.redhat.openshift.versions appropriée dans l'image du paquet de votre opérateur. À titre d'exemple, si votre projet a été généré à l'aide du SDK de l'opérateur, mettez à jour le fichier bundle.Dockerfile:

Exemple bundle.Dockerfile avec des versions compatibles

```
LABEL com.redhat.openshift.versions="<versions>" 1
```

1 Défini sur une plage ou une seule version, par exemple v4.7-v4.9. Ce paramètre définit les versions de cluster où l'opérateur doit être distribué, et l'opérateur n'apparaît pas dans un catalogue d'une version de cluster qui est en dehors de la plage.

Désormais, vous pouvez regrouper une nouvelle version de votre opérateur et publier la version mise à jour dans un catalogue pour distribution.

Ressources supplémentaires

- Gestion des versions OpenShift dans le guide de construction de l'opérateur certifié
- La mise à jour des opérateurs installés
- Catalogues d'opérateurs Red Hat

5.7.6. Ressources supplémentaires

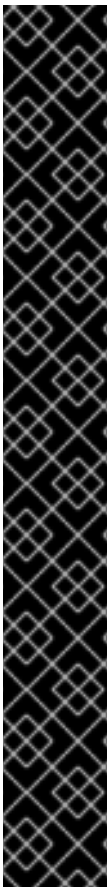
- Consultez le format d'emballage du cadre opérateur pour plus de détails sur le format du paquet.
- Consultez Gérer les catalogues personnalisés pour plus de détails sur l'ajout d'images groupées à l'indexation des images à l'aide de la commande opm.
- Consultez le flux de travail du gestionnaire de cycle de vie de l'opérateur pour plus de détails sur le fonctionnement des mises à niveau pour les opérateurs installés.

5.8. CONFORMITÉ À L'ADMISSION DE SÉCURITÉ DE POD

L'admission à la sécurité de Pod est une mise en œuvre des normes de sécurité des pod Kubernetes. L'admission à la sécurité de Pod limite le comportement des pods. Les pods qui ne sont pas conformes à l'admission de sécurité pod définie globalement ou au niveau de l'espace de noms ne sont pas admis au cluster et ne peuvent pas s'exécuter.

Dans le cas où votre projet Opérateur ne nécessite pas d'autorisations supplémentaires pour s'exécuter, vous pouvez vous assurer que vos charges de travail s'exécutent dans des espaces de noms définis au niveau de sécurité des pod restreint. Dans le cas où votre projet Opérateur nécessite des autorisations accrues pour s'exécuter, vous devez définir les configurations de contexte de sécurité suivantes:

- Le niveau d'admission de sécurité de la pod autorisée pour l'espace de noms de l'opérateur
- Les contraintes de contexte de sécurité autorisées (SCC) pour le compte de service de la charge de travail Pour plus d'informations, voir Comprendre et gérer l'admission à la sécurité des pod.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.8.1. À propos de l'admission à la sécurité de pod

Le programme OpenShift Dedicated inclut l'admission à la sécurité des pod Kubernetes. Les pods qui ne sont pas conformes à l'admission de sécurité pod définie globalement ou au niveau de l'espace de noms ne sont pas admis au cluster et ne peuvent pas s'exécuter.

À l'échelle mondiale, le profil privilégié est appliqué et le profil restreint est utilisé pour les avertissements et les audits.

Il est également possible de configurer les paramètres d'entrée de sécurité pod au niveau de l'espace de noms.

IMPORTANT

Évitez d'exécuter des charges de travail ou de partager l'accès aux projets par défaut. Les projets par défaut sont réservés à l'exécution de composants de cluster de base.

Les projets par défaut suivants sont considérés comme hautement privilégiés: par défaut, kube-public, kube-system, openshift, openshift-infra, openshift-node, et d'autres projets créés par système qui ont l'étiquette openshift.io / run-level définie à 0 ou 1. La fonctionnalité qui repose sur des plugins d'admission, tels que l'admission de sécurité pod, les contraintes de contexte de sécurité, les quotas de ressources de cluster et la résolution de référence d'image, ne fonctionne pas dans des projets hautement privilégiés.

5.8.1.1. Les modes d'admission à la sécurité de Pod

Les modes d'admission de sécurité pod suivants peuvent être configurés pour un espace de noms:

Tableau 5.18. Les modes d'admission à la sécurité de Pod

Le mode	Étiquette	Description
faire respecter	**accueil > Pod-security.kubernetes.io/enforce**	Il rejette un pod d'admission s'il ne se conforme pas au profil défini
audit	**accueil > Pod-security.kubernetes.io/audit**	Enregistre les événements d'audit si un pod ne se conforme pas au profil défini
avertissez	**accueil > Pod-security.kubernetes.io/warn**	Affiche les avertissements si un pod ne se conforme pas au profil défini

5.8.1.2. Les profils d'admission à la sécurité de Pod

Chacun des modes d'admission à la sécurité de pod peut être défini sur l'un des profils suivants:

Tableau 5.19. Les profils d'admission à la sécurité de Pod

Le profil	Description
les privilégiés	La politique la moins restrictive; permet une escalade connue des privilèges
base de données	La politique minimalement restrictive; empêche les escalades de privilèges connues
limité	La politique la plus restrictive; suit les meilleures pratiques actuelles de durcissement de la pod

5.8.1.3. Espaces de noms privilégiés

Les espaces de noms système suivants sont toujours définis sur le profil d'admission de sécurité de pod privilégié:

- **défaut par défaut**
- **Kube-public**
- **Kube-système**

Il est impossible de modifier le profil de sécurité des pod pour ces espaces de noms privilégiés.

5.8.2. À propos de la synchronisation de l'admission de sécurité de pod

En plus de la configuration globale de contrôle d'admission de sécurité de pod, un contrôleur applique les étiquettes de contrôle d'admission de sécurité pod aux espaces de noms selon les autorisations SCC des comptes de service qui se trouvent dans un espace de noms donné.

Le contrôleur examine les autorisations d'objet ServiceAccount pour utiliser les contraintes de contexte de sécurité dans chaque espace de noms. Les contraintes de contexte de sécurité (SCC) sont cartographiées pour pod des profils de sécurité en fonction de leurs valeurs de champ; le contrôleur utilise ces profils traduits. Les étiquettes d'admission de sécurité Pod et les étiquettes d'audit sont définies sur le profil de sécurité de pod le plus privilégié dans l'espace de noms pour empêcher l'affichage des avertissements et les événements d'audit de journalisation lorsque des pods sont créés.

L'étiquetage de l'espace de noms est basé sur la prise en compte des privilèges de compte de service local-nomspace.

Appliquer des pods directement pourrait utiliser les privilèges SCC de l'utilisateur qui exécute le pod. Cependant, les privilèges de l'utilisateur ne sont pas pris en compte lors de l'étiquetage automatique.

5.8.2.1. Exclusions de l'espace de noms de synchronisation de l'entrée de Pod en matière de sécurité

La synchronisation de l'entrée de sécurité Pod est désactivée de façon permanente sur les espaces de noms créés par le système et les espaces de noms préfixés openshift-*.

Les espaces de noms qui sont définis comme faisant partie de la charge utile du cluster ont mis en place une synchronisation d'admission de sécurité désactivée de façon permanente. Les espaces de noms suivants sont désactivés de façon permanente:

- **défaut par défaut**
- **location de Kube-node**
- **Kube-système**
- **Kube-public**
- **à propos de OpenShift**
- L'ensemble des espaces de noms créés par le système qui sont préfixés avec openshift-

5.8.3. Assurer que les charges de travail de l'opérateur s'exécutent dans des espaces de noms définis au niveau de sécurité des pod restreints

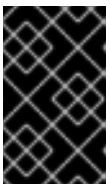
Afin de s'assurer que votre projet Opérateur peut s'exécuter sur une grande variété de déploiements et d'environnements, configurez les charges de travail de l'opérateur pour s'exécuter dans des espaces de noms définis au niveau de sécurité des pod restreint.

AVERTISSEMENT

Il faut laisser le champ runAsUser vide. Lorsque votre image nécessite un utilisateur spécifique, elle ne peut pas être exécutée sous des contraintes de contexte de sécurité restreintes (SCC) et l'application de la sécurité des pod restreints.

Procédure

- Afin de configurer les charges de travail de l'opérateur pour s'exécuter dans des espaces de noms définis au niveau de sécurité des pod restreints, modifiez la définition de l'espace de noms de votre opérateur similaire aux exemples suivants:



IMPORTANT

Il est recommandé de définir le profil seccomp dans la définition de l'espace de noms de votre opérateur. Cependant, la configuration du profil seccomp n'est pas prise en charge dans OpenShift Dedicated 4.10.

  - Dans le cas des projets d'opérateur qui ne doivent être exécutés que dans OpenShift Dedicated 4.11 et plus tard, modifiez la définition de l'espace de noms de votre opérateur semblable à l'exemple suivant:

Exemple de fichier config/manager/manager.yaml

```
...
spec:
 securityContext:
   seccompProfile:
     type: RuntimeDefault (1)
   runAsNonRoot: true
 containers:
   - name: <operator_workload_container>
     securityContext:
       allowPrivilegeEscalation: false
       capabilities:
         drop:
           - ALL
...
```

(1) En définissant le type de profil seccomp sur RuntimeDefault, le SCC par défaut sur le profil de sécurité pod de l'espace de noms.

  - Dans le cas des projets d'opérateur qui doivent également être exécutés dans OpenShift Dedicated 4.10, modifiez la définition de l'espace de noms de votre opérateur semblable à l'exemple suivant:

Exemple de fichier config/manager/manager.yaml

```
...
```

```
spec:
 securityContext: 1
  runAsNonRoot: true
 containers:
  - name: <operator_workload_container>
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
...
```

1 En laissant le type de profil seccomp unset, votre projet Opérateur peut s'exécuter dans OpenShift Dedicated 4.10.

Ressources supplémentaires

- Gestion des contraintes de contexte de sécurité

5.8.4. Gestion de l'admission de sécurité des pod pour les charges de travail de l'opérateur qui nécessitent des autorisations accrues

Dans le cas où votre projet Opérateur nécessite des autorisations accrues pour s'exécuter, vous devez modifier la version du service cluster (CSV) de votre opérateur.

Procédure

1. Définissez la configuration du contexte de sécurité au niveau d'autorisation requis dans le CSV de votre opérateur, semblable à l'exemple suivant:

   Exemple <operator_name>.clusterserviceversion.yaml fichier avec les privilèges d'administrateur réseau

   ```
   ...
   containers:
     - name: my-container
       securityContext:
         allowPrivilegeEscalation: false
         capabilities:
           add:
             - "NET_ADMIN"
   ...
   ```

2. Définissez les privilèges de compte de service qui permettent aux charges de travail de votre opérateur d'utiliser les contraintes de contexte de sécurité (SCC) requises, comme l'exemple suivant:

   Exemple <operator_name>.clusterserviceversion.yaml fichier

   ```
   ...
    install:
      spec:
        clusterPermissions:
   ```

```
      - rules:
        - apiGroups:
          - security.openshift.io
          resourceNames:
          - privileged
          resources:
          - securitycontextconstraints
          verbs:
          - use
        serviceAccountName: default
...
```

3. Éditez la description CSV de votre opérateur pour expliquer pourquoi votre projet Opérateur nécessite des autorisations accrues similaires à l'exemple suivant:

Exemple <operator_name>.clusterserviceversion.yaml fichier

```
...
spec:
  apiservicedefinitions:{}
  ...
description: The <operator_name> requires a privileged pod security admission label set on the Operator's namespace. The Operator's agents require escalated permissions to restart the node if the node needs remediation.
```

5.8.5. Ressources supplémentaires

- Comprendre et gérer l'admission à la sécurité des pod

5.9. LA VALIDATION DES OPÉRATEURS À L'AIDE DE L'OUTIL DE CARTE DE POINTAGE

En tant qu'auteur de l'opérateur, vous pouvez utiliser l'outil de carte de pointage dans le SDK de l'opérateur pour effectuer les tâches suivantes:

- Validez que votre projet Opérateur est exempt d'erreurs de syntaxe et emballé correctement
- Évaluez les suggestions sur les façons d'améliorer votre opérateur

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.9.1. À propos de l'outil de tableau de bord

Alors que la sous-commande de validation du paquet SDK de l'opérateur peut valider des répertoires de faisceaux locaux et des images de faisceau à distance pour le contenu et la structure, vous pouvez utiliser la commande de carte de pointage pour exécuter des tests sur votre opérateur en fonction d'un fichier de configuration et d'images de test. Ces tests sont implémentés dans des images de test qui sont configurées et construites pour être exécutées par le tableau de bord.

Le tableau de bord suppose qu'il est exécuté avec l'accès à un cluster Kubernetes configuré, tel qu'OpenShift Dedicated. Le tableau de bord exécute chaque test dans un pod, à partir duquel les logs de pod sont agrégés et les résultats des tests sont envoyés à la console. Le tableau de bord a intégré les tests de base et de gestion du cycle de vie de l'opérateur (OLM) et fournit également un moyen d'exécuter des définitions de test personnalisées.

Flux de travail de carte de pointage

1. Créer toutes les ressources requises par toutes les ressources personnalisées (CR) et l'opérateur connexes
2. Créer un conteneur proxy dans le déploiement de l'opérateur pour enregistrer les appels au serveur API et exécuter des tests
3. Examiner les paramètres dans les CR

Les tests de tableau de bord ne font pas d'hypothèses quant à l'état de l'opérateur testé. La création d'opérateurs et de CR pour un opérateur dépasse la portée du tableau de bord lui-même. Les tests de tableau de bord peuvent cependant créer les ressources dont ils ont besoin si les tests sont conçus pour la création de ressources.

la syntaxe de commande de carte de pointage

```
$ operator-sdk scorecard <bundle_dir_or_image> [flags]
```

La carte de pointage nécessite un argument positionnel pour le chemin d'accès sur le disque vers votre paquet Opérateur ou le nom d'une image de paquet.

Afin d'obtenir de plus amples informations sur les drapeaux, courez:

```
$ operator-sdk scorecard -h
```

5.9.2. Configuration de la carte de pointage

L'outil de carte de pointage utilise une configuration qui vous permet de configurer des plugins internes, ainsi que plusieurs options de configuration globales. Les tests sont pilotés par un fichier de configuration nommé config.yaml, qui est généré par la commande make bundle, situé dans votre bundle/annuaire:

```
./bundle
...
└── tests
    └── scorecard
        └── config.yaml
```

Exemple de fichier de configuration de carte de pointage

```
kind: Configuration
apiversion: scorecard.operatorframework.io/v1alpha3
metadata:
  name: config
stages:
- parallel: true
  tests:
  - image: quay.io/operator-framework/scorecard-test:v1.38.0
    entrypoint:
    - scorecard-test
    - basic-check-spec
    labels:
      suite: basic
      test: basic-check-spec-test
  - image: quay.io/operator-framework/scorecard-test:v1.38.0
    entrypoint:
    - scorecard-test
    - olm-bundle-validation
    labels:
      suite: olm
      test: olm-bundle-validation-test
```

Le fichier de configuration définit chaque test que la carte de pointage peut exécuter. Les champs suivants du fichier de configuration de la carte de pointage définissent le test comme suit:

Champ de configuration	Description
image	Le nom de l'image du conteneur de test qui implémente un test

Champ de configuration	Description
le point d'entrée	Commande et arguments invoqués dans l'image de test pour exécuter un test
étiquettes	Étiquettes définies par la carte de bord ou personnalisées qui sélectionnent les tests à exécuter

5.9.3. Des tests de carte de pointage intégrés

Le tableau de bord est livré avec des tests prédéfinis qui sont disposés en suites: la suite de test de base et la suite Operator Lifecycle Manager (OLM).

Tableau 5.20. La suite de test de base

Essai	Description	Court nom
Le bloc spec existe	Ce test vérifie la ressource personnalisée (CR) créée dans le cluster pour s'assurer que tous les CR ont un bloc spec.	**Basic-check-spec-test**

Tableau 5.21. La suite de test OLM

Essai	Description	Court nom
La validation des paquets	Ce test valide les manifestes de paquets trouvés dans le paquet qui est transmis dans la carte de pointage. Lorsque le contenu du paquet contient des erreurs, le résultat du test inclut le journal du validateur ainsi que les messages d'erreur de la bibliothèque de validation.	**essai OLM-bundle-validation-test**
Les API fournies ont une validation	Ce test vérifie que les définitions de ressources personnalisées (CRD) pour les CR fournis contiennent une section de validation et qu'il y a validation pour chaque champ de spécification et d'état détecté dans le CR.	**les OLM-crds-ont-validation-test**
Les CRD possédés ont des ressources répertoriées	Ce test permet de s'assurer que les CRD pour chaque CR fournis par l'option du manifeste cr disposent d'une sous-section des ressources de la section CRD de la ClusterServiceVersion (CSV). Lorsque le test détecte les ressources utilisées qui ne sont pas listées dans la section Ressources, il les énumère dans les suggestions à la fin du test. Les utilisateurs sont tenus de remplir la section ressources après la génération de code initiale pour que ce test passe.	**les OLM-crds-ont-ressources-test**
Champs spec avec descripteurs	Ce test vérifie que chaque champ dans les sections spec CRs a un descripteur correspondant listé dans le CSV.	**le test OLM-spec-descriptors-test**

Essai	Description	Court nom
Champs d'état avec descripteurs	Ce test vérifie que chaque champ dans les sections de statut CRs a un descripteur correspondant listé dans le CSV.	**le test OLM-status-descriptors-test**

5.9.4. Exécution de l'outil de tableau de bord

Après l'exécution de la commande init, un ensemble par défaut de fichiers Kustomize est généré par le SDK de l'opérateur. Le fichier bundle/tests/scorecard/config.yaml par défaut qui est généré peut être immédiatement utilisé pour exécuter l'outil de carte de pointage contre votre opérateur, ou vous pouvez modifier ce fichier à vos spécifications de test.

Conditions préalables

- Le projet d'opérateur généré à l'aide du SDK de l'opérateur

Procédure

1. Générer ou régénérer vos manifestes et métadonnées de paquets pour votre opérateur:

   ```
   $ make bundle
   ```

   Cette commande ajoute automatiquement des annotations de carte de pointage à vos métadonnées de paquet, qui est utilisée par la commande de carte de pointage pour exécuter des tests.

2. Exécutez la carte de pointage par rapport au chemin d'accès sur le disque vers votre paquet Opérateur ou le nom d'une image de paquet:

   ```
   $ operator-sdk scorecard <bundle_dir_or_image>
   ```

5.9.5. Sortie du tableau de bord

L'indicateur --output pour la commande de la carte de pointage spécifie le format de sortie des résultats du tableau de bord: texte ou json.

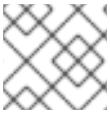
Exemple 5.7. Exemple d'extrait de sortie JSON

```
{
  "apiVersion": "scorecard.operatorframework.io/v1alpha3",
  "kind": "TestList",
  "items": [
    {
      "kind": "Test",
      "apiVersion": "scorecard.operatorframework.io/v1alpha3",
      "spec": {
        "image": "quay.io/operator-framework/scorecard-test:v1.38.0",
        "entrypoint": [
          "scorecard-test",
```

```
          "olm-bundle-validation"
        ],
        "labels": {
          "suite": "olm",
          "test": "olm-bundle-validation-test"
        }
      },
      "status": {
        "results": [
          {
            "name": "olm-bundle-validation",
            "log": "time=\"2020-06-10T19:02:49Z\" level=debug msg=\"Found manifests directory\"
name=bundle-test\ntime=\"2020-06-10T19:02:49Z\" level=debug msg=\"Found metadata
directory\" name=bundle-test\ntime=\"2020-06-10T19:02:49Z\" level=debug msg=\"Getting
mediaType info from manifests directory\" name=bundle-test\ntime=\"2020-06-10T19:02:49Z\"
level=info msg=\"Found annotations file\" name=bundle-test\ntime=\"2020-06-10T19:02:49Z\"
level=info msg=\"Could not find optional dependencies file\" name=bundle-test\n",
            "state": "pass"
          }
        ]
      }
    }
  ]
}
```


Exemple 5.8. Exemple de sortie de texte

```
--------------------------------------------------------------------------------
Image:      quay.io/operator-framework/scorecard-test:v1.38.0
Entrypoint: [scorecard-test olm-bundle-validation]
Labels:
 "suite":"olm"
 "test":"olm-bundle-validation-test"
Results:
 Name: olm-bundle-validation
 State: pass
 Log:
  time="2020-07-15T03:19:02Z" level=debug msg="Found manifests directory" name=bundle-test
  time="2020-07-15T03:19:02Z" level=debug msg="Found metadata directory" name=bundle-test
  time="2020-07-15T03:19:02Z" level=debug msg="Getting mediaType info from manifests
directory" name=bundle-test
  time="2020-07-15T03:19:02Z" level=info msg="Found annotations file" name=bundle-test
  time="2020-07-15T03:19:02Z" level=info msg="Could not find optional dependencies file"
name=bundle-test
```



NOTE

La spécification du format de sortie correspond à la disposition du type de test.

5.9.6. La sélection des tests

Les tests de tableau de bord sont sélectionnés en définissant le drapeau CLI --selector sur un ensemble de chaînes d'étiquettes. En l'absence d'un drapeau sélecteur, tous les tests dans le fichier de configuration de la carte de pointage sont exécutés.

Les tests sont exécutés en série, les résultats des tests étant agrégés par la carte de pointage et écrits en sortie standard, ou stdout.

Procédure

1. Afin de sélectionner un seul test, par exemple Basic-check-spec-test, spécifiez le test en utilisant le drapeau --selector:

```
$ operator-sdk scorecard <bundle_dir_or_image> \
    -o text \
    --selector=test=basic-check-spec-test
```

2. Afin de sélectionner une série de tests, par exemple olm, spécifiez une étiquette utilisée par tous les tests OLM:

```
$ operator-sdk scorecard <bundle_dir_or_image> \
    -o text \
    --selector=suite=olm
```

3. Afin de sélectionner plusieurs tests, spécifiez les noms de test en utilisant l'indicateur sélecteur en utilisant la syntaxe suivante:

```
$ operator-sdk scorecard <bundle_dir_or_image> \
    -o text \
    --selector='test in (basic-check-spec-test,olm-bundle-validation-test)'
```

5.9.7. Activer les tests parallèles

En tant qu'auteur de l'opérateur, vous pouvez définir des étapes distinctes pour vos tests à l'aide du fichier de configuration de la carte de pointage. Les étapes s'exécutent de manière séquentielle dans l'ordre où elles sont définies dans le fichier de configuration. L'étape contient une liste de tests et un réglage parallèle configurable.

Par défaut, ou lorsqu'une étape se définit explicitement en parallèle à false, les tests d'une étape sont exécutés de manière séquentielle dans l'ordre où ils sont définis dans le fichier de configuration. L'exécution de tests un à la fois est utile pour garantir qu'aucun test n'interagisse et n'entre en conflit l'un avec l'autre.

Cependant, si les tests sont conçus pour être complètement isolés, ils peuvent être parallélisés.

Procédure

- Exécuter un ensemble de tests isolés en parallèle, les inclure dans la même étape et définir parallèlement à true:

```
apiVersion: scorecard.operatorframework.io/v1alpha3
kind: Configuration
metadata:
  name: config
stages:
```

```
- parallel: true (1)
  tests:
  - entrypoint:
    - scorecard-test
    - basic-check-spec
    image: quay.io/operator-framework/scorecard-test:v1.38.0
    labels:
      suite: basic
      test: basic-check-spec-test
  - entrypoint:
    - scorecard-test
    - olm-bundle-validation
    image: quay.io/operator-framework/scorecard-test:v1.38.0
    labels:
      suite: olm
      test: olm-bundle-validation-test
```

(1) Active les tests parallèles

L'ensemble des tests dans une phase parallèle sont exécutés simultanément, et le tableau de bord attend que tous les tests soient terminés avant de passer à l'étape suivante. Cela peut rendre vos tests plus rapides.

5.9.8. Des tests personnalisés de carte de pointage

L'outil de tableau de bord peut exécuter des tests personnalisés qui suivent ces conventions prescrites:

- Les tests sont mis en œuvre dans une image de conteneur
- Les tests acceptent un point d'entrée qui inclut une commande et des arguments
- Les tests produisent une sortie de carte de pointage v1alpha3 au format JSON sans enregistrement externe dans la sortie d'essai
- Les tests peuvent obtenir le contenu du paquet à un point de montage partagé de /bundle
- Les tests peuvent accéder à l'API Kubernetes à l'aide d'une connexion client intégrée

L'écriture de tests personnalisés dans d'autres langages de programmation est possible si l'image de test suit les directives ci-dessus.

L'exemple suivant montre une image de test personnalisée écrite dans Go:

Exemple 5.9. Exemple de test de carte de pointage personnalisé

```
// Copyright 2020 The Operator-SDK Authors
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
```

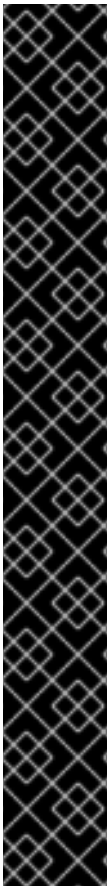
```
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

package main

import (
 "encoding/json"
 "fmt"
 "log"
 "os"

 scapiv1alpha3 "github.com/operator-framework/api/pkg/apis/scorecard/v1alpha3"
 apimanifests "github.com/operator-framework/api/pkg/manifests"
)

// This is the custom scorecard test example binary
// As with the Redhat scorecard test image, the bundle that is under
// test is expected to be mounted so that tests can inspect the
// bundle contents as part of their test implementations.
// The actual test is to be run is named and that name is passed
// as an argument to this binary.  This argument mechanism allows
// this binary to run various tests all from within a single
// test image.

const PodBundleRoot = "/bundle"

func main() {
 entrypoint := os.Args[1:]
 if len(entrypoint) == 0 {
  log.Fatal("Test name argument is required")
 }

 // Read the pod's untar'd bundle from a well-known path.
 cfg, err := apimanifests.GetBundleFromDir(PodBundleRoot)
 if err != nil {
  log.Fatal(err.Error())
 }

 var result scapiv1alpha3.TestStatus

 // Names of the custom tests which would be passed in the
 // `operator-sdk` command.
 switch entrypoint[0] {
 case CustomTest1Name:
  result = CustomTest1(cfg)
 case CustomTest2Name:
  result = CustomTest2(cfg)
 default:
  result = printValidTests()
 }

 // Convert scapiv1alpha3.TestResult to json.
 prettyJSON, err := json.MarshalIndent(result, "", "    ")
 if err != nil {
  log.Fatal("Failed to generate json", err)
```

```
 }
 fmt.Printf("%s\n", string(prettyJSON))

}

// printValidTests will print out full list of test names to give a hint to the end user on what the valid
tests are.
func printValidTests() scapiv1alpha3.TestStatus {
 result := scapiv1alpha3.TestResult{}
 result.State = scapiv1alpha3.FailState
 result.Errors = make([]string, 0)
 result.Suggestions = make([]string, 0)

 str := fmt.Sprintf("Valid tests for this image include: %s %s",
  CustomTest1Name,
  CustomTest2Name)
 result.Errors = append(result.Errors, str)
 return scapiv1alpha3.TestStatus{
  Results: []scapiv1alpha3.TestResult{result},
 }
}

const (
 CustomTest1Name = "customtest1"
 CustomTest2Name = "customtest2"
)

// Define any operator specific custom tests here.
// CustomTest1 and CustomTest2 are example test functions. Relevant operator specific
// test logic is to be implemented in similarly.

func CustomTest1(bundle *apimanifests.Bundle) scapiv1alpha3.TestStatus {
 r := scapiv1alpha3.TestResult{}
 r.Name = CustomTest1Name
 r.State = scapiv1alpha3.PassState
 r.Errors = make([]string, 0)
 r.Suggestions = make([]string, 0)
 almExamples := bundle.CSV.GetAnnotations()["alm-examples"]
 if almExamples == "" {
  fmt.Println("no alm-examples in the bundle CSV")
 }

 return wrapResult(r)
}

func CustomTest2(bundle *apimanifests.Bundle) scapiv1alpha3.TestStatus {
 r := scapiv1alpha3.TestResult{}
 r.Name = CustomTest2Name
 r.State = scapiv1alpha3.PassState
 r.Errors = make([]string, 0)
 r.Suggestions = make([]string, 0)
 almExamples := bundle.CSV.GetAnnotations()["alm-examples"]
 if almExamples == "" {
  fmt.Println("no alm-examples in the bundle CSV")
 }
 return wrapResult(r)
```

```
}

func wrapResult(r scapiv1alpha3.TestResult) scapiv1alpha3.TestStatus {
 return scapiv1alpha3.TestStatus{
  Results: []scapiv1alpha3.TestResult{r},
 }
}
```

5.10. LA VALIDATION DES PAQUETS D'OPÉRATEURS

En tant qu'auteur de l'opérateur, vous pouvez exécuter la commande de validation du paquet dans le SDK de l'opérateur pour valider le contenu et le format d'un paquet Opérateur. Il est possible d'exécuter la commande sur une image de faisceau d'opérateur distant ou un répertoire de paquets Opérateur local.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

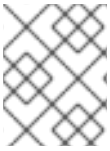
Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.10.1. À propos de la commande de validation du paquet

Alors que la commande Operator SDK scorecard peut exécuter des tests sur votre opérateur en fonction d'un fichier de configuration et d'images de test, la sous-commande valide par paquet peut valider les répertoires de paquets locaux et les images de faisceau à distance pour le contenu et la structure.

bundle valider la syntaxe de commande

```
$ operator-sdk bundle validate <bundle_dir_or_image> <flags>
```



NOTE

La commande de validation du paquet s'exécute automatiquement lorsque vous construisez votre paquet à l'aide de la commande make bundle.

Les images groupées sont tirées d'un registre distant et construites localement avant qu'elles ne soient validées. Les répertoires de paquets locaux doivent contenir des métadonnées et des manifestes de l'opérateur. Les métadonnées et les manifestes du bundle doivent avoir une structure similaire à la disposition de paquet suivante:

Exemple de mise en page de paquets

```
./bundle
├── manifests
│   ├── cache.my.domain_memcacheds.yaml
│   └── memcached-operator.clusterserviceversion.yaml
└── metadata
    └── annotations.yaml
```

Les tests groupés passent la validation et se terminent par un code de sortie de 0 si aucune erreur n'est détectée.

Exemple de sortie

```
INFO[0000] All validation tests have completed successfully
```

Les tests échouent à la validation et se terminent par un code de sortie de 1 si des erreurs sont détectées.

Exemple de sortie

```
ERRO[0000] Error: Value cache.example.com/v1alpha1, Kind=Memcached: CRD
"cache.example.com/v1alpha1, Kind=Memcached" is present in bundle "" but not defined in CSV
```

Les tests groupés qui entraînent des avertissements peuvent encore passer la validation avec un code de sortie de 0 tant qu'aucune erreur n'est détectée. Les tests échouent uniquement sur les erreurs.

Exemple de sortie

```
WARN[0000] Warning: Value : (memcached-operator.v0.0.1) annotations not found
INFO[0000] All validation tests have completed successfully
```

Afin d'obtenir de plus amples informations sur la sous-commande valider le paquet, exécutez:

```
$ operator-sdk bundle validate -h
```

5.10.2. Ensemble intégré valider les tests

Le SDK de l'opérateur est livré avec des validateurs prédéfinis disposés en suites. Lorsque vous exécutez la commande de validation du paquet sans spécifier un validateur, le test par défaut s'exécute. Le test par défaut vérifie qu'un paquet respecte les spécifications définies par la communauté Operator Framework. En savoir plus, voir "Bundle Format".

Il est possible d'exécuter des validateurs optionnels pour tester des problèmes tels que la compatibilité OperatorHub ou les API Kubernetes dépréciées. Les validateurs optionnels s'exécutent toujours en plus du test par défaut.

ensemble valider la syntaxe de commande pour les suites de test optionnelles

```
$ operator-sdk bundle validate <bundle_dir_or_image>
  --select-optional <test_label>
```

Tableau 5.22. Validateurs de validateurs de paquets addtionnels

Le nom	Description	Étiquette
Cadre de l'opérateur	Ce validateur teste un ensemble d'opérateurs par rapport à l'ensemble des validateurs fournis par le Cadre d'opérateur.	**la suite=operatorframework**
L'opérateurHub	Ce validateur teste un ensemble d'opérateurs pour la compatibilité avec OperatorHub.	**le nom=operatorhub**
Bonnes pratiques	Ce validateur vérifie si un groupe d'opérateurs est conforme aux bonnes pratiques définies par le Cadre de l'opérateur. Il vérifie les problèmes, tels qu'une description CRD vide ou des ressources non prises en charge par le gestionnaire de cycle de vie de l'opérateur (OLM).	**le nom = bonnes pratiques**

Ressources supplémentaires

- Format de paquet

5.10.3. Exécution de la commande de validation du paquet

Le validateur par défaut exécute un test chaque fois que vous entrez la commande de validation du paquet. Les validateurs optionnels peuvent être exécutés à l'aide du drapeau --select-optional. Les validateurs optionnels exécutent des tests en plus du test par défaut.

Conditions préalables

- Le projet d'opérateur généré à l'aide du SDK de l'opérateur

Procédure

1. Dans le cas où vous souhaitez exécuter le validateur par défaut par rapport à un répertoire de paquets locaux, entrez la commande suivante depuis votre répertoire de projet Opérateur:

   ```
   $ operator-sdk bundle validate ./bundle
   ```

2. Lorsque vous souhaitez exécuter le validateur par défaut contre une image de faisceau d'opérateur distant, entrez la commande suivante:

   ```
   $ operator-sdk bundle validate \
     <bundle_registry>/<bundle_image_name>:<tag>
   ```

là où:

<bundle_registry>

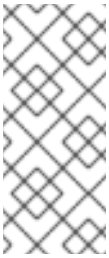
Indique le registre où le paquet est hébergé, tel que quay.io/exemple.

<bundle_image_name>

Indique le nom de l'image du paquet, tel que l'opérateur memcached.

<tag>

Indique la balise de l'image de paquet, telle que v1.38.0.



NOTE

Lorsque vous souhaitez valider une image de groupe Opérateur, vous devez héberger votre image dans un registre à distance. Le SDK de l'opérateur tire l'image et la construit localement avant d'exécuter des tests. La commande de validation du paquet ne prend pas en charge le test d'images de paquets locaux.

3. Lorsque vous souhaitez exécuter un validateur supplémentaire contre un paquet Opérateur, entrez la commande suivante:

```
$ operator-sdk bundle validate \
  <bundle_dir_or_image> \
  --select-optional <test_label>
```

là où:

<bundle_dir_or_image>

Indique le répertoire local de paquets ou l'image de paquet distant, tels que ~/projects/memcached ou quay.io/example/memcached-operator:v1.38.0.

<test_label>

Indique le nom du validateur que vous souhaitez exécuter, tel que name=good-pratiques.

Exemple de sortie

```
ERRO[0000] Error: Value apiextensions.k8s.io/v1, Kind=CustomResource: unsupported
media type registry+v1 for bundle object
WARN[0000] Warning: Value k8sevent.v0.0.1: owned CRD
"k8sevents.k8s.k8sevent.com" has an empty description
```

5.11. DÉTECTION ET SUPPORT DE CLUSTERS À HAUTE DISPONIBILITÉ OU À UN SEUL NŒUD

Afin de s'assurer que votre opérateur fonctionne bien sur les modes haute disponibilité (HA) et non-HA dans les clusters OpenShift Container Platform, vous pouvez utiliser le SDK de l'opérateur pour détecter la topologie de l'infrastructure du cluster et définir les besoins en ressources pour s'adapter à la topologie du cluster.

Le cluster OpenShift Container Platform peut être configuré en mode haute disponibilité (HA), qui utilise plusieurs nœuds, ou en mode non-HA, qui utilise un seul nœud. Il est probable qu'un cluster à un

seul nœud, également connu sous le nom d'OpenShift à un seul nœud, aura des contraintes de ressources plus conservatrices. Il est donc important que les opérateurs installés sur un cluster à nœud unique puissent s'ajuster en conséquence et fonctionner bien.

En accédant à l'API de mode haute disponibilité de cluster fournie dans OpenShift Dedicated, les auteurs de l'opérateur peuvent utiliser le SDK de l'opérateur pour permettre à leur opérateur de détecter la topologie de l'infrastructure d'un cluster, que ce soit en mode HA ou non-HA. La logique de l'opérateur personnalisé peut être développée qui utilise la topologie de cluster détectée pour changer automatiquement les besoins en ressources, à la fois pour l'opérateur et pour tous les Operands ou charges de travail qu'il gère, à un profil qui correspond le mieux à la topologie.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.11.1. À propos de l'API de mode haute disponibilité du cluster

L'OpenShift Dedicated fournit une API de mode à haute disponibilité en cluster qui peut être utilisée par les opérateurs pour aider à détecter la topologie de l'infrastructure. L'API Infrastructure détient des informations à l'échelle du cluster concernant l'infrastructure. Les opérateurs gérés par Operator Lifecycle Manager (OLM) peuvent utiliser l'API Infrastructure s'ils ont besoin de configurer une charge de travail Operand ou gérée différemment en fonction du mode haute disponibilité.

Dans l'API Infrastructure, l'état de l'infrastructureTopology exprime les attentes pour les services d'infrastructure qui ne fonctionnent pas sur les nœuds de plan de contrôle, généralement indiqués par un sélecteur de nœud pour une valeur de rôle autre que master. Le statut controlPlaneTopology exprime les attentes pour les Operands qui fonctionnent normalement sur les nœuds de plan de contrôle.

Le paramètre par défaut pour l'un ou l'autre état est HighlyAvailable, ce qui représente le comportement des opérateurs dans plusieurs clusters de nœuds. Le paramètre SingleReplica est utilisé dans les clusters à nœud unique, également connu sous le nom d'OpenShift à nœud unique, et indique que les opérateurs ne doivent pas configurer leurs opérandes pour une opération à haute disponibilité.

L'installateur dédié OpenShift définit les champs d'état controlPlaneTopology et infrastructure en fonction du nombre de répliques pour le cluster lorsqu'il est créé, selon les règles suivantes:

- Lorsque le nombre de répliques du plan de contrôle est inférieur à 3, le statut controlPlaneTopology est défini sur SingleReplica. Dans le cas contraire, il est réglé sur HighlyDisponible.
- Lorsque le nombre de répliques du travailleur est 0, les nœuds de plan de contrôle sont également configurés en tant que travailleurs. Ainsi, le statut d'infrastructureTopology sera le même que le statut controlPlaneTopology.
- Lorsque le nombre de répliques du travailleur est 1, l'infrastructureTopology est définie sur SingleReplica. Dans le cas contraire, il est réglé sur HighlyDisponible.

5.11.2. Exemple d'utilisation de l'API dans les projets d'opérateur

En tant qu'auteur de l'opérateur, vous pouvez mettre à jour votre projet Opérateur pour accéder à l'API Infrastructure en utilisant les constructions normales de Kubernetes et la bibliothèque contrôleur-exécution, comme indiqué dans les exemples suivants:

exemple de bibliothèque Controller-runtime

```
// Simple query
 nn := types.NamespacedName{
 Name: "cluster",
 }
 infraConfig := &configv1.Infrastructure{}
 err = crClient.Get(context.Background(), nn, infraConfig)
 if err != nil {
 return err
 }
 fmt.Printf("using crclient: %v\n", infraConfig.Status.ControlPlaneTopology)
 fmt.Printf("using crclient: %v\n", infraConfig.Status.InfrastructureTopology)
```

Kubernetes construit l'exemple

```
operatorConfigInformer := configinformer.NewSharedInformerFactoryWithOptions(configClient,
2*time.Second)
 infrastructureLister = operatorConfigInformer.Config().V1().Infrastructures().Lister()
 infraConfig, err := configClient.ConfigV1().Infrastructures().Get(context.Background(), "cluster",
metav1.GetOptions{})
 if err != nil {
 return err
 }
// fmt.Printf("%v\n", infraConfig)
 fmt.Printf("%v\n", infraConfig.Status.ControlPlaneTopology)
 fmt.Printf("%v\n", infraConfig.Status.InfrastructureTopology)
```

5.12. CONFIGURATION DE LA SURVEILLANCE INTÉGRÉE AVEC PROMETHEUS

Le SDK de l'opérateur fournit un support de surveillance intégré à l'aide de l'opérateur Prometheus, que vous pouvez utiliser pour exposer des mesures personnalisées pour votre opérateur.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

AVERTISSEMENT

Par défaut, OpenShift Dedicated fournit un opérateur Prometheus dans le projet openshift-user-workload-monitoring. Cette instance Prometheus doit être utilisée pour surveiller les charges de travail des utilisateurs dans OpenShift Dedicated.

Il ne faut pas utiliser l'opérateur Prometheus dans le projet de surveillance ouverte. Les ingénieurs de fiabilité du site Red Hat (SRE) utilisent cette instance Prometheus pour surveiller les composants du cluster de base.

Ressources supplémentaires

- Exposer des mesures personnalisées pour les opérateurs Go-based (documentation OpenShift Container Platform)
- Exposer des métriques personnalisées pour les opérateurs basés sur Ansible (documentation OpenShift Container Platform)
- Comprendre la pile de surveillance dans OpenShift Dedicated

5.13. CONFIGURATION DE L'ÉLECTION DES DIRIGEANTS

Au cours du cycle de vie d'un opérateur, il est possible qu'il puisse y avoir plus d'une instance en cours d'exécution à un moment donné, par exemple lors du déploiement d'une mise à niveau pour l'opérateur. Dans un tel scénario, il est nécessaire d'éviter toute dispute entre plusieurs instances de l'opérateur

utilisant l'élection de leader. Cela garantit qu'une seule instance leader gère la réconciliation tandis que les autres instances sont inactives mais prêtes à prendre le relais lorsque le leader démissionne.

Il y a deux implémentations électorales différentes à choisir, chacune avec son propre compromis:

Leader pour la vie

La gousse de leader n'abandonne le leadership, en utilisant la collecte des ordures, lorsqu'elle est supprimée. Cette mise en œuvre exclut la possibilité que deux instances courent par erreur en tant que leaders, un état également connu sous le nom de cerveau divisé. Cependant, cette méthode peut être sujette à un retard dans l'élection d'un nouveau leader. Ainsi, lorsque la gousse de leader est sur un nœud sans réponse ou partitionné, vous pouvez spécifier node.kubernetes.io/unreachable et node.kubernetes.io/non-prêt tolérance sur la gousse de leader et utiliser la valeur de toléranceSeconds pour dicter combien de temps il faut pour que la gousse de leader soit supprimée du nœud et démissionne. Ces tolérances sont ajoutées à la pod par défaut lors de l'admission avec une tolérance de 5 minutes. Consultez la documentation Leader-for-life Go pour en savoir plus.

Leader avec location

Le groupe dirigeant renouvelle périodiquement le bail de leader et renonce au leadership lorsqu'il ne peut pas renouveler le bail. Cette implémentation permet une transition plus rapide vers un nouveau leader lorsque le leader existant est isolé, mais il y a une possibilité de diviser le cerveau dans certaines situations. Consultez la documentation Leader-with-lease Go pour plus d'informations.

Le SDK d'opérateur permet par défaut l'implémentation Leader-for-life. Consultez la documentation connexe Go pour les deux approches afin de tenir compte des compromis qui ont du sens pour votre cas d'utilisation.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.13.1. Exemples d'élection du chef d'opérateur

Les exemples suivants illustrent comment utiliser les deux options électorales de leader pour un opérateur, un leader à vie et un leader avec bail.

5.13.1.1. Élection de chef à vie

Avec la mise en œuvre des élections Leader-for-life, un appel à Leader.Become() bloque l'opérateur comme il se retries jusqu'à ce qu'il puisse devenir le leader en créant la carte de configuration nommée memcached-operator-lock:

```
import (
  ...
  "github.com/operator-framework/operator-sdk/pkg/leader"
)

func main() {
  ...
  err = leader.Become(context.TODO(), "memcached-operator-lock")
  if err != nil {
    log.Error(err, "Failed to retry for leader lock")
    os.Exit(1)
  }
  ...
}
```

Lorsque l'opérateur n'est pas en cours d'exécution à l'intérieur d'un cluster, Leader.Become() retourne simplement sans erreur pour sauter l'élection de leader car il ne peut pas détecter le nom de l'opérateur.

5.13.1.2. Élection du chef avec bail

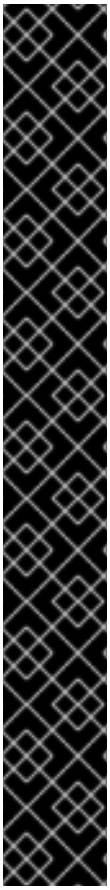
La mise en œuvre de Leader-with-location peut être activée à l'aide des options de gestionnaire pour l'élection des leaders:

```
import (
  ...
  "sigs.k8s.io/controller-runtime/pkg/manager"
)

func main() {
  ...
  opts := manager.Options{
    ...
    LeaderElection: true,
    LeaderElectionID: "memcached-operator-lock"
  }
  mgr, err := manager.New(cfg, opts)
  ...
}
```

Lorsque l'opérateur n'est pas en cours d'exécution dans un cluster, le gestionnaire retourne une erreur au démarrage car il ne peut pas détecter l'espace de noms de l'opérateur pour créer la carte de configuration pour l'élection de leader. L'option LeaderElectionNamespace vous permet de remplacer cet espace de noms en définissant l'option LeaderElectionNamespace pour le gestionnaire.

5.14. UTILITAIRE D'ÉLAGAGE D'OBJETS POUR LES OPÉRATEURS GO-BASED

L'utilitaire d'élagage opérateur-lib permet aux opérateurs Go-based de nettoyer ou tailler les objets lorsqu'ils ne sont plus nécessaires. Les auteurs d'opérateurs peuvent également utiliser l'utilitaire pour créer des crochets et des stratégies personnalisés.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.14.1. À propos de l'utilitaire d'élagage de l'opérateur-lib

Les objets, tels que les emplois ou les pods, sont créés en tant que partie normale du cycle de vie de l'opérateur. Lorsqu'un administrateur ayant le rôle d'administrateur dédié ou l'opérateur ne supprime pas cet objet, il peut rester dans le cluster et consommer des ressources.

Auparavant, les options suivantes étaient disponibles pour l'élagage d'objets inutiles:

- Les auteurs d'opérateurs ont dû créer une solution d'élagage unique pour leurs opérateurs.
- Les administrateurs de clusters devaient nettoyer eux-mêmes les objets.

L'utilitaire d'élagage de l'opérateur-lib supprime les objets d'un cluster Kubernetes pour un espace de noms donné. La bibliothèque a été ajoutée dans la version 0.9.0 de la bibliothèque opérateur-lib dans le cadre de l'opérateur.

5.14.2. Configuration de l'utilitaire d'élagage

L'utilitaire d'élagage opérateur-lib est écrit dans Go et comprend des stratégies d'élagage communes pour les opérateurs basés sur Go.

Configuration d'exemple

```
cfg = Config{
        log:          logf.Log.WithName("prune"),
        DryRun:       false,
```

```
        Clientset:     client,
        LabelSelector: "app=<operator_name>",
        Resources: []schema.GroupVersionKind{
                {Group: "", Version: "", Kind: PodKind},
        },
        Namespaces: []string{"<operator_namespace>"},
        Strategy: StrategyConfig{
                Mode:           MaxCountStrategy,
                MaxCountSetting: 1,
        },
        PreDeleteHook: myhook,
}
```

Le fichier de configuration de l'utilitaire d'élagage définit les actions d'élagage en utilisant les champs suivants:

Champ de configuration	Description
journal de bord	Enregistreur utilisé pour gérer les messages de journal de la bibliothèque.
Dryrun	Booléen qui détermine si les ressources doivent être supprimées. Lorsqu'il est défini à true, l'utilitaire s'exécute mais ne supprime pas les ressources.
Ensemble de clients	Kubernetes ClientSet utilisé pour les appels d'API Kubernetes.
LabelSelector	Expression de sélecteur d'étiquettes Kubernetes utilisée pour trouver des ressources à tailler.
Ressources	Les types de ressources Kubernetes. Actuellement, PodKind et JobKind sont pris en charge.
Espaces de noms	Liste des espaces de noms Kubernetes pour rechercher des ressources.
La stratégie	La stratégie d'élagage à exécuter.
La stratégie.Mode	Actuellement, MaxCountStrategy, MaxAgeStrategy ou CustomStrategy sont pris en charge.
La stratégie.MaxCountSetting	La valeur entière de MaxCountStrategy spécifie combien de ressources devraient rester après l'exécution de l'utilitaire de taille.
La stratégie.MaxAgeSetting	Go time.Duration string valeur, telle que 48h, qui spécifie l'âge des ressources à tailler.
Jeu Stratégie.CustomSettings	Go carte des valeurs qui peuvent être transmises dans une fonction de stratégie personnalisée.
Ajouter au panier PreDeleteHook	Facultatif: Go fonction à appeler avant de tailler une ressource.

Champ de configuration	Description
CustomStrategy	Facultatif: Go fonction qui implémente une stratégie d'élagage personnalisée.

Exécution de l'élagage

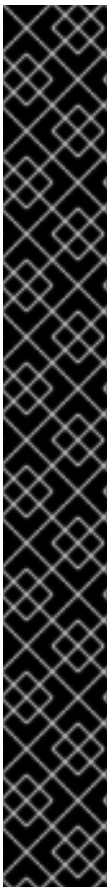
Il est possible d'appeler l'action d'élagage en exécutant la fonction d'exécution sur la configuration de l'élagage.

```
err := cfg.Execute(ctx)
```

Il est également possible d'appeler une action d'élagage en utilisant un paquet cron ou en appelant l'utilitaire d'élagage avec un événement déclencheur.

5.15. LES PROJETS DE MANIFESTATION DE PAQUETS MIGRATOIRES AU FORMAT DE PAQUETAGE

La prise en charge du format de manifeste du paquet hérité pour les opérateurs est supprimée dans OpenShift Dedicated 4.8 et ultérieure. Lorsque vous avez un projet Opérateur qui a été initialement créé à l'aide du format manifeste du paquet, vous pouvez utiliser le SDK de l'opérateur pour migrer le projet vers le format de paquet. Le format de paquet est le format d'emballage préféré pour Operator Lifecycle Manager (OLM) à partir de OpenShift Dedicated 4.6.



IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

5.15.1. À propos de la migration du format d'emballage

La commande Operator SDK pkgman-to-bundle aide à migrer le paquet Operator Lifecycle Manager

(OLM) en paquets. La commande prend un répertoire de manifestes de paquets d'entrée et génère des paquets pour chacune des versions des manifestes présentes dans le répertoire d'entrée. Ensuite, vous pouvez construire des images groupées pour chacun des paquets générés.

À titre d'exemple, considérez les Manifestes de paquets/annuaires suivants pour un projet dans le format manifeste du paquet:

Exemple de mise en page de format de paquet

```
packagemanifests/
└── etcd
    ├── 0.0.1
    │   ├── etcdcluster.crd.yaml
    │   └── etcdoperator.clusterserviceversion.yaml
    ├── 0.0.2
    │   ├── etcdbackup.crd.yaml
    │   ├── etcdcluster.crd.yaml
    │   ├── etcdoperator.v0.0.2.clusterserviceversion.yaml
    │   └── etcdrestore.crd.yaml
    └── etcd.package.yaml
```

Après l'exécution de la migration, les paquets suivants sont générés dans le répertoire bundle /:

Exemple de mise en page de format de paquet

```
bundle/
├── bundle-0.0.1
│   ├── bundle.Dockerfile
│   ├── manifests
│   │   ├── etcdcluster.crd.yaml
│   │   └── etcdoperator.clusterserviceversion.yaml
│   ├── metadata
│   │   └── annotations.yaml
│   └── tests
│       └── scorecard
│           └── config.yaml
└── bundle-0.0.2
    ├── bundle.Dockerfile
    ├── manifests
    │   ├── etcdbackup.crd.yaml
    │   ├── etcdcluster.crd.yaml
    │   ├── etcdoperator.v0.0.2.clusterserviceversion.yaml
    │   └── etcdrestore.crd.yaml
    ├── metadata
    │   └── annotations.yaml
    └── tests
        └── scorecard
            └── config.yaml
```

Basé sur cette mise en page générée, les images groupées pour les deux paquets sont également construites avec les noms suivants:

- **description Quay.io/exemple/etcd:0.0.1**
- **ajouter au panier Quay.io/exemple/etcd:0.0.2**

Ressources supplémentaires

- Format d'emballage du cadre opérateur

5.15.2. La migration d'un projet manifeste de paquet pour regrouper le format

Les auteurs d'opérateurs peuvent utiliser le SDK de l'opérateur pour migrer un projet de format manifeste de paquet vers un projet de format groupé.

Conditions préalables

- L'opérateur SDK CLI installé
- Le projet opérateur initialement généré à l'aide du SDK de l'opérateur au format manifeste du paquet

Procédure

- Employez le SDK de l'opérateur pour migrer votre projet de manifeste de paquets vers le format de paquet et générer des images groupées:

```
$ operator-sdk pkgman-to-bundle <package_manifests_dir> \ 1
  [--output-dir <directory>] \ 2
  --image-tag-base <image_name_base> 3
```

1. Indiquez l'emplacement du répertoire des manifestes de paquets pour le projet, tels que les Manifestes de paquets/ ou les manifestes/.
2. Facultatif: Par défaut, les paquets générés sont écrits localement sur le disque dans le répertoire bundle/. Il est possible d'utiliser le drapeau --output-dir pour spécifier un emplacement alternatif.
3. Définissez le drapeau --image-tag-base pour fournir la base du nom de l'image, tel que quay.io/example/etcd, qui sera utilisé pour les paquets. Fournissez le nom sans balise, car la balise pour les images sera définie en fonction de la version du paquet. À titre d'exemple, les noms complets d'image sont générés dans le format <image_name_base>:<bundle_version>.

La vérification

- Assurez-vous que l'image de paquet générée fonctionne avec succès:

```
$ operator-sdk run bundle <bundle_image_name>:<tag>
```

Exemple de sortie

```
INFO[0025] Successfully created registry pod: quay-io-my-etcd-0-9-4
INFO[0025] Created CatalogSource: etcd-catalog
INFO[0026] OperatorGroup "operator-sdk-og" created
INFO[0026] Created Subscription: etcdoperator-v0-9-4-sub
INFO[0031] Approved InstallPlan install-5t58z for the Subscription: etcdoperator-v0-9-4-sub
INFO[0031] Waiting for ClusterServiceVersion "default/etcdoperator.v0.9.4" to reach
'Succeeded' phase
```

```
INFO[0032]   Waiting for ClusterServiceVersion "default/etcdoperator.v0.9.4" to appear
INFO[0048]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Pending
INFO[0049]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Installing
INFO[0064]   Found ClusterServiceVersion "default/etcdoperator.v0.9.4" phase: Succeeded
INFO[0065] OLM has successfully installed "etcdoperator.v0.9.4"
```

5.16. OPÉRATEUR SDK CLI RÉFÉRENCE

L'interface de ligne de commande de l'opérateur SDK (CLI) est un kit de développement conçu pour faciliter l'écriture des opérateurs.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

L'opérateur SDK CLI syntaxe

```
$ operator-sdk <command> [<subcommand>] [<argument>] [<flags>]
```

Les auteurs d'opérateurs disposant d'un accès administrateur de cluster à un cluster basé sur Kubernetes (comme OpenShift Dedicated) peuvent utiliser l'opérateur SDK CLI pour développer leurs propres opérateurs basés sur Go, Ansible ou Helm. Kubebuilder est intégré dans le SDK de l'opérateur en tant que solution d'échafaudage pour les opérateurs Go, ce qui signifie que les projets Kubebuilder existants peuvent être utilisés comme avec le SDK de l'opérateur et continuer à fonctionner.

5.16.1. le paquet

La commande Operator-sdk bundle gère les métadonnées du bundle Operator.

5.16.1.1. de valider

Le bundle valide la sous-commande valide un bundle d'opérateur.

Tableau 5.23. faisceau valider les drapeaux

Drapeau	Description
-h, --aide	La sortie d'aide pour le paquet valider la sous-commande.
--index-builder (chaîne)	Outil pour tirer et déballer des images groupées. Il n'est utilisé que lors de la validation d'une image de paquet. Les options disponibles sont docker, qui est par défaut, podman, ou aucune.
--liste optionnelle	Liste de tous les validateurs optionnels disponibles. Lorsqu'il est défini, aucun validateur n'est exécuté.
--sélectionner-optionnel (chaîne)	Le sélecteur d'étiquette pour sélectionner les validateurs optionnels à exécuter. Lorsque vous exécutez avec le drapeau --list-optional, liste les validateurs optionnels disponibles.

5.16.2. le nettoyage

La commande de nettoyage de l'opérateur-sdk détruit et supprime les ressources qui ont été créées pour un opérateur qui a été déployé avec la commande run.

Tableau 5.24. drapeaux de nettoyage

Drapeau	Description
-h, --aide	Aide à la sortie pour la sous-commande du paquet d'exécution.
--kubeconfig (chaîne)	Chemin vers le fichier kubeconfig à utiliser pour les requêtes CLI.
-n, --namespace (chaîne)	Le cas échéant, l'espace de noms dans lequel exécuter la demande CLI.
--timeout <duration>	Il est temps d'attendre que la commande soit terminée avant d'échouer. La valeur par défaut est 2m0s.

5.16.3. achèvement des travaux

La commande d'achèvement de l'opérateur-sdk génère des achèvements de shell pour rendre les commandes CLI émettrices plus rapides et plus faciles.

Tableau 5.25. achèvement des sous-commandes

Le sous-commande	Description
bash	Générez des finitions bash.
à propos de Zsh	Générez des finitions zsh.

Tableau 5.26. drapeaux d'achèvement

Drapeau	Description
-h, --aide	Aide à l'utilisation de sortie.

À titre d'exemple:

```
$ operator-sdk completion bash
```

Exemple de sortie

```
# bash completion for operator-sdk                -*- shell-script -*-
...
# ex: ts=4 sw=4 et filetype=sh
```

5.16.4. créer

La commande operator-sdk create est utilisée pour créer, ou échafauder, une API Kubernetes.

5.16.4.1. API

La création de sous-commandes api échafaude une API Kubernetes. La sous-commande doit être exécutée dans un projet initialisé avec la commande init.

Tableau 5.27. créer des drapeaux api

Drapeau	Description
-h, --aide	Aide à la sortie pour la sous-commande du paquet d'exécution.

5.16.5. générer

La commande operator-sdk génère un générateur spécifique pour générer du code ou des manifestes.

5.16.5.1. le paquet

La sous-commande de groupe génératrice génère un ensemble de manifestes de paquets, de métadonnées et d'un fichier bundle.Dockerfile pour votre projet Opérateur.

NOTE

En règle générale, vous exécutez la commande génératrice kustomize manifeste d'abord la sous-commande pour générer les bases Kustomize d'entrée qui sont utilisées par la sous-commande de paquets génératrices. Cependant, vous pouvez utiliser la commande make bundle dans un projet initialisé pour automatiser l'exécution de ces commandes en séquence.

Tableau 5.28. générer des drapeaux de paquets

Drapeau	Description
--canaux (chaîne)	Liste séparée par virgule des canaux auxquels appartient le paquet. La valeur par défaut est alpha.
--CRDS-dir (chaîne)	Le répertoire root pour CustomResoureDefinition se manifeste.
--par défaut-canal (chaîne)	Le canal par défaut pour le paquet.
--déploiement-dir (string)	Le répertoire racine pour les manifestes de l'opérateur, tels que les déploiements et RBAC. Ce répertoire est différent du répertoire passé au drapeau --input-dir.
-h, --aide	Aide pour générer des paquets
--input-dir (chaîne)	Annuaire à partir duquel lire un paquet existant. Ce répertoire est le parent de votre répertoire de manifestes de paquets et est différent du répertoire --deploy-dir.
--kustomize-dir (string)	Répertoire contenant des bases Kustomize et un fichier kustomization.yaml pour les manifestes de paquets. Le chemin par défaut est config/manifestes.
--manifestes	Générez des manifestes de paquets.
--métadonnées	Générez des métadonnées groupées et Dockerfile.
--sortie-dir (chaîne)	Annuaire pour écrire le paquet vers.
--écraser	Écrasez les métadonnées du bundle et Dockerfile s'ils existent. La valeur par défaut est true.
--emballage (chaîne)	Le nom du paquet pour le paquet.
-Q, --quiet	Exécutez en mode silencieux.
----------	Ecrire le paquet manifeste pour standardiser.
--version (chaîne)	La version sémantique de l'opérateur dans le paquet généré. Défini uniquement lors de la création d'un nouveau paquet ou de la mise à niveau de l'opérateur.

Ressources supplémentaires

- Consultez Bundling un opérateur pour une procédure complète qui inclut l'utilisation de la commande make bundle pour appeler la sous-commande du paquet génératrice.

5.16.5.2. kustomize

La sous-commande génératrice kustomize contient des sous-commandes qui génèrent des données Kustomize pour l'opérateur.

5.16.5.2.1. les manifestes

La sous-commande génératrice kustomize manifeste génère ou régénère les bases Kustomize et un fichier kustomization.yaml dans le répertoire config/manifests, qui sont utilisés pour construire des manifestes de paquets par d'autres commandes SDK de l'opérateur. Cette commande demande interactivement des métadonnées UI, un composant important des bases manifestes, par défaut, sauf si une base existe déjà ou si vous définissez l'indicateur --interactive=false.

Tableau 5.29. générer kustomize manifeste des drapeaux

Drapeau	Description
--APIs-dir (chaîne)	Répertoire racine pour les définitions de type API.
-h, --aide	Aide pour générer des manifestes de kustomize.
--input-dir (chaîne)	Répertoire contenant les fichiers Kustomize existants.
--interactif	Lorsqu'il est défini sur false, si aucune base Kustomize n'existe, une invite de commande interactive est présentée pour accepter les métadonnées personnalisées.
--sortie-dir (chaîne)	Annuaire où écrire des fichiers Kustomize.
--emballage (chaîne)	Le nom du paquet.
-Q, --quiet	Exécutez en mode silencieux.

5.16.6. init

La commande operator-sdk init initialise un projet Opérateur et génère, ou des échafaudages, une mise en page de répertoire de projet par défaut pour le plugin donné.

Cette commande écrit les fichiers suivants:

- Fichier de licence de hotplate
- Fichier PROJET avec le domaine et le référentiel
- Makefile pour construire le projet
- fichier Go.mod avec dépendances de projet
- fichier kustomization.yaml pour la personnalisation des manifestes
- Fichier correcteur pour personnaliser les images pour les manifestes du gestionnaire
- Fichier de correction pour activer les métriques Prometheus
- fichier Main.go à exécuter

Tableau 5.30. drapeaux init

Drapeau	Description
--aide, -h	Aide à la sortie pour la commande init.
--plugins (chaîne)	Le nom et la version optionnelle du plugin pour initialiser le projet avec. Les plugins disponibles sont ansible.sdk.operatorframework.io/v1, go.kubebuilder.io/v2, go.kubebuilder.io/v3, et helm.sdk.operatorframework.io/v1.
--version de projet	La version du projet. Les valeurs disponibles sont 2 et 3-alpha, qui est la valeur par défaut.

5.16.7. courir

La commande operator-sdk run fournit des options qui peuvent lancer l'opérateur dans divers environnements.

5.16.7.1. le paquet

La sous-commande de paquet d'exécution déploie un opérateur dans le format de paquet avec Operator Lifecycle Manager (OLM).

Tableau 5.31. lancer des drapeaux de paquets

Drapeau	Description
--index-image (chaîne)	Indexer l'image dans laquelle injecter un paquet. L'image par défaut est quay.io/operator-framework/upstream-opm-builder:latest.
--install-mode <install_mode_value>	Installez le mode supporté par la version de service cluster (CSV) de l'opérateur, par exemple AllNamespaces ou SingleNamespace.
--timeout <duration>	Installez le délai d'attente. La valeur par défaut est 2m0s.
--kubeconfig (chaîne)	Chemin vers le fichier kubeconfig à utiliser pour les requêtes CLI.
-n, --namespace (chaîne)	Le cas échéant, l'espace de noms dans lequel exécuter la demande CLI.
--security-context-config <security_context>	Indique le contexte de sécurité à utiliser pour le pod de catalogue. Les valeurs autorisées incluent des restrictions et des héritages. La valeur par défaut est l'héritage. [1]
-h, --aide	Aide à la sortie pour la sous-commande du paquet d'exécution.

1. Le contexte de sécurité restreint n'est pas compatible avec l'espace de noms par défaut. Afin de configurer l'admission de sécurité de pod de votre opérateur dans votre environnement de

production, voir "Complying with Pod security admission". En savoir plus sur l'admission à la sécurité des pods, voir « Comprendre et gérer l'admission à la sécurité des pod ».

Ressources supplémentaires

- Consultez l'adhésion au groupe Opérateur pour plus de détails sur les modes d'installation possibles.
- Conformité à l'admission de sécurité de pod
- Comprendre et gérer l'admission à la sécurité des pod

5.16.7.2. la mise à niveau de paquet

La sous-commande de mise à niveau de paquets d'exécution met à niveau un opérateur qui a déjà été installé dans le format de paquet avec Operator Lifecycle Manager (OLM).

Tableau 5.32. exécuter des drapeaux de mise à niveau de paquets

Drapeau	Description
--timeout <duration>	Délai de mise à niveau. La valeur par défaut est 2m0s.
--kubeconfig (chaîne)	Chemin vers le fichier kubeconfig à utiliser pour les requêtes CLI.
-n, --namespace (chaîne)	Le cas échéant, l'espace de noms dans lequel exécuter la demande CLI.
--security-context-config <security_context >	Indique le contexte de sécurité à utiliser pour le pod de catalogue. Les valeurs autorisées incluent des restrictions et des héritages. La valeur par défaut est l'héritage. [1]
-h, --aide	Aide à la sortie pour la sous-commande du paquet d'exécution.

1. Le contexte de sécurité restreint n'est pas compatible avec l'espace de noms par défaut. Afin de configurer l'admission de sécurité de pod de votre opérateur dans votre environnement de production, voir "Complying with Pod security admission". En savoir plus sur l'admission à la sécurité des pods, voir « Comprendre et gérer l'admission à la sécurité des pod ».

Ressources supplémentaires

- Conformité à l'admission de sécurité de pod
- Comprendre et gérer l'admission à la sécurité des pod

5.16.8. carte de pointage

La commande operator-sdk scorecard exécute l'outil de carte de pointage pour valider un ensemble d'opérateurs et fournir des suggestions d'améliorations. La commande prend un argument, qu'il s'agisse d'une image groupée ou d'un répertoire contenant des manifestes et des métadonnées. Lorsque

l'argument contient une balise d'image, l'image doit être présente à distance.

Tableau 5.33. drapeaux de carte de pointage

Drapeau	Description
-C, --config (chaîne)	Chemin d'accès au fichier de configuration de la carte de pointage. Le chemin par défaut est bundle/tests/scorecard/config.yaml.
-h, --aide	Aide à la sortie pour la commande de la carte de pointage.
--kubeconfig (chaîne)	Chemin d'accès au fichier kubeconfig.
-L, --liste	Liste des tests disponibles pour exécuter.
-n, --namespace (chaîne)	Espace de noms dans lequel exécuter les images de test.
-O, --sortie (chaîne)	Format de sortie pour les résultats. Les valeurs disponibles sont du texte, qui est la valeur par défaut, et json.
--pod-sécurité <security_context >	L'option d'exécuter une carte de score avec le contexte de sécurité spécifié. Les valeurs autorisées incluent des restrictions et des héritages. La valeur par défaut est l'héritage. [1]
-l, --séléctor (chaîne)	Le sélecteur d'étiquette pour déterminer quels tests sont exécutés.
-s, --service-compte (string)	Compte de service à utiliser pour les tests. La valeur par défaut est par défaut.
-x, --skip-nettoyage	Désactiver le nettoyage des ressources après l'exécution des tests.
-W, --temps d'attente <durée>	Des secondes pour attendre que les tests soient terminés, par exemple 35s. La valeur par défaut est 30s.

1. Le contexte de sécurité restreint n'est pas compatible avec l'espace de noms par défaut. Afin de configurer l'admission de sécurité de pod de votre opérateur dans votre environnement de production, voir "Complying with Pod security admission". En savoir plus sur l'admission à la sécurité des pods, voir « Comprendre et gérer l'admission à la sécurité des pod ».

Ressources supplémentaires

- Consultez les opérateurs de validation à l'aide de l'outil de carte de pointage pour plus de détails sur l'exécution de l'outil de carte de pointage.
- Conformité à l'admission de sécurité de pod
- Comprendre et gérer l'admission à la sécurité des pod

5.17. LA MIGRATION VERS L'OPÉRATEUR SDK V0.1.0

Ce guide décrit comment migrer un projet d'opérateur construit à l'aide de l'opérateur SDK v0.0.x vers la structure du projet requise par l'opérateur SDK v0.1.0.

IMPORTANT

La version prise en charge par Red Hat de l'outil Operator SDK CLI, y compris les outils d'échafaudage et de test connexes pour les projets d'opérateur, est dépréciée et devrait être supprimée dans une version ultérieure d'OpenShift Dedicated. Le Red Hat fournira des corrections de bogues et une prise en charge de cette fonctionnalité pendant le cycle de vie de la version actuelle, mais cette fonctionnalité ne recevra plus d'améliorations et sera supprimée des futures versions d'OpenShift Dedicated.

La version prise en charge par Red Hat du SDK de l'opérateur n'est pas recommandée pour la création de nouveaux projets d'opérateur. Les auteurs d'opérateurs avec des projets d'opérateur existants peuvent utiliser la version de l'outil Operator SDK CLI publié avec OpenShift Dedicated 4 pour maintenir leurs projets et créer des versions d'opérateur ciblant des versions plus récentes d'OpenShift Dedicated.

Les images de base suivantes pour les projets d'opérateur ne sont pas dépréciées. Les fonctionnalités d'exécution et les API de configuration de ces images de base sont toujours prises en charge pour les corrections de bogues et pour l'adressage des CVE.

- L'image de base pour les projets d'opérateurs basés sur Ansible
- L'image de base pour les projets d'opérateur basé sur Helm

Afin d'obtenir de l'information sur la version non prise en charge et gérée par la communauté du SDK de l'opérateur, voir Operator SDK (Operator Framework).

La méthode recommandée pour migrer votre projet est de:

1. Initialisez un nouveau projet v0.1.0.
2. Copiez votre code dans le nouveau projet.
3. Changer le nouveau projet comme décrit pour v0.1.0.

Ce guide utilise l'opérateur memcached, le projet exemple du SDK de l'opérateur, pour illustrer les étapes de migration. Consultez les structures de projet memcached-operator v0.0.7 et v0.1.0 memcached-operator pour les exemples de pré- et post-migration, respectivement.

5.17.1. Création d'un nouveau projet Operator SDK v0.1.0

Donnez un nouveau nom à votre projet Operator SDK v0.0.x et créez un nouveau projet v0.1.0 à sa place.

Conditions préalables

- L'opérateur SDK v0.1.0 CLI installé sur le poste de travail de développement
- le projet Memcached-operator précédemment déployé à l'aide d'une version antérieure de l'opérateur SDK

Procédure

1. Assurez-vous que la version SDK est v0.1.0:

```
$ operator-sdk --version
operator-sdk version 0.1.0
```

2. Créer un nouveau projet:

```
$ mkdir -p $GOPATH/src/github.com/example-inc/
$ cd $GOPATH/src/github.com/example-inc/
$ mv memcached-operator old-memcached-operator
$ operator-sdk new memcached-operator --skip-git-init
$ ls
memcached-operator old-memcached-operator
```

3. Copier .git de l'ancien projet:

```
$ cp -rf old-memcached-operator/.git memcached-operator/.git
```

5.17.2. La migration de types personnalisés à partir de pkg/apis

Faites migrer les types personnalisés de votre projet vers l'utilisation mise à jour du SDK de l'opérateur v0.1.0.

Conditions préalables

- L'opérateur SDK v0.1.0 CLI installé sur le poste de travail de développement
- le projet Memcached-operator précédemment déployé à l'aide d'une version antérieure de l'opérateur SDK
- Création d'un nouveau projet à l'aide de l'opérateur SDK v0.1.0

Procédure

1. **Créez l'API d'échafaudage pour les types personnalisés.**
   a. Créez l'API pour votre ressource personnalisée (CR) dans le nouveau projet avec operator-sdk ajouter api --api-version=<apiversion> --kind=<kind>:

```
$ cd memcached-operator
$ operator-sdk add api --api-version=cache.example.com/v1alpha1 --kind=Memcached

$ tree pkg/apis
pkg/apis/
├── addtoscheme_cache_v1alpha1.go
├── apis.go
└── cache
    └── v1alpha1
        ├── doc.go
        ├── memcached_types.go
        ├── register.go
        └── zz_generated.deepcopy.go
```

b. Répétez la commande précédente pour autant de types personnalisés que vous l'avez défini dans votre ancien projet. Chaque type sera défini dans le fichier pkg/apis/<group>/<version>/<kind>_types.go.

2. **Copiez le contenu du type.**

a. Copiez le contenu Spec and Status du fichier pkg/apis/<group>/<version>/types.go de l'ancien projet vers le fichier<group>pkg/apis/ /<version>/<kind>_types.go du nouveau projet.

b. Chaque fichier <kind>_types.go a une fonction init(). Assurez-vous de ne pas supprimer cela puisque cela enregistre le type avec le schéma du gestionnaire:

```
func init() {
 SchemeBuilder.Register(&Memcached{}, &MemcachedList{})
```

5.17.3. Code de conciliation migratoire

Faites migrer le code de rapprochement de votre projet vers l'utilisation de l'opérateur SDK v0.1.0 de mise à jour.

Conditions préalables

- L'opérateur SDK v0.1.0 CLI installé sur le poste de travail de développement
- le projet Memcached-operator précédemment déployé à l'aide d'une version antérieure de l'opérateur SDK
- Les types personnalisés ont migré à partir de pkg/apis/

Procédure

1. **Ajoutez un contrôleur pour regarder votre CR.**
Dans les projets v0.0.x, les ressources à regarder étaient précédemment définies dans cmd/<operator-name>/main.go:

```
sdk.Watch("cache.example.com/v1alpha1", "Memcached", "default",
time.Duration(5)*time.Second)
```

Dans le cas des projets v0.1.0, vous devez définir un contrôleur pour surveiller les ressources:

a. Ajoutez un contrôleur pour regarder votre type CR avec le contrôleur --api-version=<apiversion> --kind=<kind>.

```
$ operator-sdk add controller --api-version=cache.example.com/v1alpha1 --
kind=Memcached

$ tree pkg/controller
pkg/controller/
├── add_memcached.go
├── controller.go
└── memcached
    └── memcached_controller.go
```

b. Inspectez la fonction add() dans votre fichier pkg/controller/<kind>/<kind>_controller.go:

```
import (
    cachev1alpha1 "github.com/example-inc/memcached-
operator/pkg/apis/cache/v1alpha1"
    ...
)

func add(mgr manager.Manager, r reconcile.Reconciler) error {
    c, err := controller.New("memcached-controller", mgr, controller.Options{Reconciler: r})

    // Watch for changes to the primary resource Memcached
    err = c.Watch(&source.Kind{Type: &cachev1alpha1.Memcached{}},
&handler.EnqueueRequestForObject{})

    // Watch for changes to the secondary resource pods and enqueue reconcile requests
for the owner Memcached
    err = c.Watch(&source.Kind{Type: &corev1.Pod{}},
&handler.EnqueueRequestForOwner{
  IsController: true,
  OwnerType:    &cachev1alpha1.Memcached{},
 })
}
```

Enlevez la deuxième Watch() ou modifiez-la pour regarder un type de ressource secondaire qui appartient à votre CR.

La visualisation de plusieurs ressources vous permet de déclencher la boucle de réconciliation pour plusieurs ressources pertinentes à votre application. Consultez la documentation de surveillance et de traitement des événements et la documentation des conventions du contrôleur Kubernetes pour plus de détails.

Lorsque votre opérateur regarde plus d'un type CR, vous pouvez faire l'un des éléments suivants en fonction de votre demande:

- Lorsque le CR appartient à votre CR primaire, regardez-la comme ressource secondaire dans le même contrôleur pour déclencher la boucle de réconciliation pour la ressource primaire.

```
// Watch for changes to the primary resource Memcached
    err = c.Watch(&source.Kind{Type: &cachev1alpha1.Memcached{}},
&handler.EnqueueRequestForObject{})

    // Watch for changes to the secondary resource AppService and enqueue
reconcile requests for the owner Memcached
    err = c.Watch(&source.Kind{Type: &appv1alpha1.AppService{}},
&handler.EnqueueRequestForOwner{
  IsController: true,
  OwnerType:    &cachev1alpha1.Memcached{},
 })
```

- Ajoutez un nouveau contrôleur pour regarder et réconcilier le CR indépendamment de l'autre CR.

```
$ operator-sdk add controller --api-version=app.example.com/v1alpha1 --
kind=AppService
```

```
  // Watch for changes to the primary resource AppService
    err = c.Watch(&source.Kind{Type: &appv1alpha1.AppService{}},
&handler.EnqueueRequestForObject{})
```

2. **Copiez et modifiez le code de réconciliation de pkg/stub/handler.go.**
Dans un projet v0.1.0, le code de conciliation est défini dans la méthode Reconcile() du Reconciler d'un contrôleur. Ceci est similaire à la fonction Handle() dans l'ancien projet. À noter la différence dans les arguments et les valeurs de retour:

   - Concilier:

```
    func (r *ReconcileMemcached) Reconcile(request reconcile.Request)
(reconcile.Result, error)
```

   - La poignée:

```
    func (h *Handler) Handle(ctx context.Context, event sdk.Event) error
```

   Au lieu de recevoir un événement sdk.Event (avec l'objet), la fonction Reconcile() reçoit une requête (clé Nom/Namespace) pour rechercher l'objet.

   Lorsque la fonction Reconcile() renvoie une erreur, le contrôleur requeue et réessaye la Demande. Dans le cas où aucune erreur n'est retournée, en fonction du résultat, le responsable du traitement ne réessayera pas la demande, ne sera pas immédiatement réessayer, ou réessayera après une durée spécifiée.

   a. Copiez le code de la fonction Handle() de l'ancien projet vers le code existant dans la fonction Reconcile() de votre contrôleur. Assurez-vous de conserver la section initiale dans le code Reconcile() qui recherche l'objet de la demande et vérifie si elle est supprimée.

```
import (
    apierrors "k8s.io/apimachinery/pkg/api/errors"
    cachev1alpha1 "github.com/example-inc/memcached-
operator/pkg/apis/cache/v1alpha1"
    ...
)
func (r *ReconcileMemcached) Reconcile(request reconcile.Request) (reconcile.Result,
error) {
    // Fetch the Memcached instance
 instance := &cachev1alpha1.Memcached{}
    err := r.client.Get(context.TODO()
    request.NamespacedName, instance)
    if err != nil {
        if apierrors.IsNotFound(err) {
            // Request object not found, could have been deleted after reconcile request.
            // Owned objects are automatically garbage collected.
            // Return and don't requeue
            return reconcile.Result{}, nil
        }
        // Error reading the object - requeue the request.
        return reconcile.Result{}, err
```

```
    }

    // Rest of your reconcile code goes here.
    ...
}
```

b. Changez les valeurs de retour dans votre code de rapprochement:

   i. Il faut remplacer l'erreur de retour par le retour réconcilié.Result{}, erreur.

   ii. Il faut remplacer le retour zéro par le retour réconcilié.Result{}, nil.

c. Afin de réconcilier périodiquement un CR dans votre contrôleur, vous pouvez définir le champ Requeue After pour réconcilier.Result. Cela fera que le contrôleur requeue la demande et déclenchera le rapprochement après la durée souhaitée. A noter que la valeur par défaut de 0 signifie aucune requeue.

```
reconcilePeriod := 30 * time.Second
reconcileResult := reconcile.Result{RequeueAfter: reconcilePeriod}
...

// Update the status
err := r.client.Update(context.TODO(), memcached)
if err != nil {
    log.Printf("failed to update memcached status: %v", err)
    return reconcileResult, err
}
return reconcileResult, nil
```

d. En remplacement des appels au client SDK (Créer, mettre à jour, Supprimer, obtenir, liste) par le client du conciliateur.
Consultez les exemples ci-dessous et la documentation API du contrôleur-runtimeclient dans le projet opérateur-sdk pour plus de détails:

```
// Create
dep := &appsv1.Deployment{...}
err := sdk.Create(dep)
// v0.0.1
err := r.client.Create(context.TODO(), dep)

// Update
err := sdk.Update(dep)
// v0.0.1
err := r.client.Update(context.TODO(), dep)

// Delete
err := sdk.Delete(dep)
// v0.0.1
err := r.client.Delete(context.TODO(), dep)

// List
podList := &corev1.PodList{}
labelSelector := labels.SelectorFromSet(labelsForMemcached(memcached.Name))
listOps := &metav1.ListOptions{LabelSelector: labelSelector}
err := sdk.List(memcached.Namespace, podList, sdk.WithListOptions(listOps))
```

```
// v0.1.0
listOps := &client.ListOptions{Namespace: memcached.Namespace, LabelSelector:
labelSelector}
err := r.client.List(context.TODO(), listOps, podList)

// Get
dep := &appsv1.Deployment{APIVersion: "apps/v1", Kind: "Deployment", Name: name,
Namespace: namespace}
err := sdk.Get(dep)
// v0.1.0
dep := &appsv1.Deployment{}
err = r.client.Get(context.TODO(), types.NamespacedName{Name: name, Namespace:
namespace}, dep)
```

e. Copiez et initialisez tous les autres champs de votre structure Handler dans la structure Reconcile<Kind>:

```
// newReconciler returns a new reconcile.Reconciler
func newReconciler(mgr manager.Manager) reconcile.Reconciler {
 return &ReconcileMemcached{client: mgr.GetClient(), scheme: mgr.GetScheme(), foo:
"bar"}
}

// ReconcileMemcached reconciles a Memcached object
type ReconcileMemcached struct {
    client client.Client
    scheme *runtime.Scheme
    // Other fields
    foo string
}
```

3. **Copiez les modifications de main.go.**
   La fonction principale d'un opérateur v0.1.0 dans cmd/manager/main.go configure le gestionnaire, qui enregistre les ressources personnalisées et démarre tous les contrôleurs.

   Il n'y a pas besoin de migrer les fonctions SDK sdk.Watch(), sdk.Handle(), et sdk.Run() à partir de l'ancien main.go puisque cette logique est maintenant définie dans un contrôleur.

   Cependant, s'il existe des drapeaux ou paramètres spécifiques à l'opérateur définis dans l'ancien fichier main.go, copiez-les.

   Lorsque vous avez des types de ressources tiers enregistrés avec le système SDK, consultez les sujets avancés dans le projet opérateur-sdk pour savoir comment les enregistrer avec le schéma du gestionnaire dans le nouveau projet.

4. **Copiez des fichiers définis par l'utilisateur.**
   En cas de pkg, de scripts ou de documentation défini par l'utilisateur dans l'ancien projet, copiez ces fichiers dans le nouveau projet.

5. **Copiez les modifications apportées aux manifestes de déploiement.**
   Dans le cas des mises à jour apportées aux manifestes suivants dans l'ancien projet, copiez les modifications apportées à leurs fichiers correspondants dans le nouveau projet. Faites attention à ne pas écraser directement les fichiers, mais inspectez et apportez les modifications nécessaires:

   - fichier Tmp/Build/Dockerfile à construire/Dockerfile

    - Il n'y a pas de répertoire tmp dans la nouvelle disposition du projet

- Les règles RBAC sont mises à jour depuis le déploiement/rbac.yaml pour déployer/role.yaml et déployer/role_binding.yaml

- déployez/cr.yaml pour déployer/crds/<group>_<version>_<kind>_cr.yaml

- déployez/crd.yaml pour déployer/crds/<group>_<version>_<kind>_crd.yaml

6. **Copiez les dépendances définies par l'utilisateur.**
   Dans le cas des dépendances définies par l'utilisateur ajoutées au Gopkg.toml de l'ancien projet, copiez-les et ajoutez-les au Gopkg.toml du nouveau projet. Exécutez dep assurez-vous de mettre à jour le fournisseur dans le nouveau projet.

7. **Confirmez vos changements.**
   Créez et exécutez votre opérateur pour vérifier qu'il fonctionne.