OpenShift Container Platform 4.12

Nœuds

Configurer et gérer les nœuds dans OpenShift Container Platform

Last Updated: 2023-05-16

OpenShift Container Platform 4.12 Nœuds

Configurer et gérer les nœuds dans OpenShift Container Platform

Notice légale

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Résumé

Ce document fournit des instructions pour configurer et gérer les nœuds, les Pods et les conteneurs dans votre cluster. Il fournit également des informations sur la configuration de la planification et du placement des pods, sur l'utilisation des travaux et des DaemonSets pour automatiser les tâches, ainsi que sur d'autres tâches permettant d'assurer l'efficacité du cluster.

Table des matières

CHAPITRE 1. VUE D'ENSEMBLE DES NŒUDS

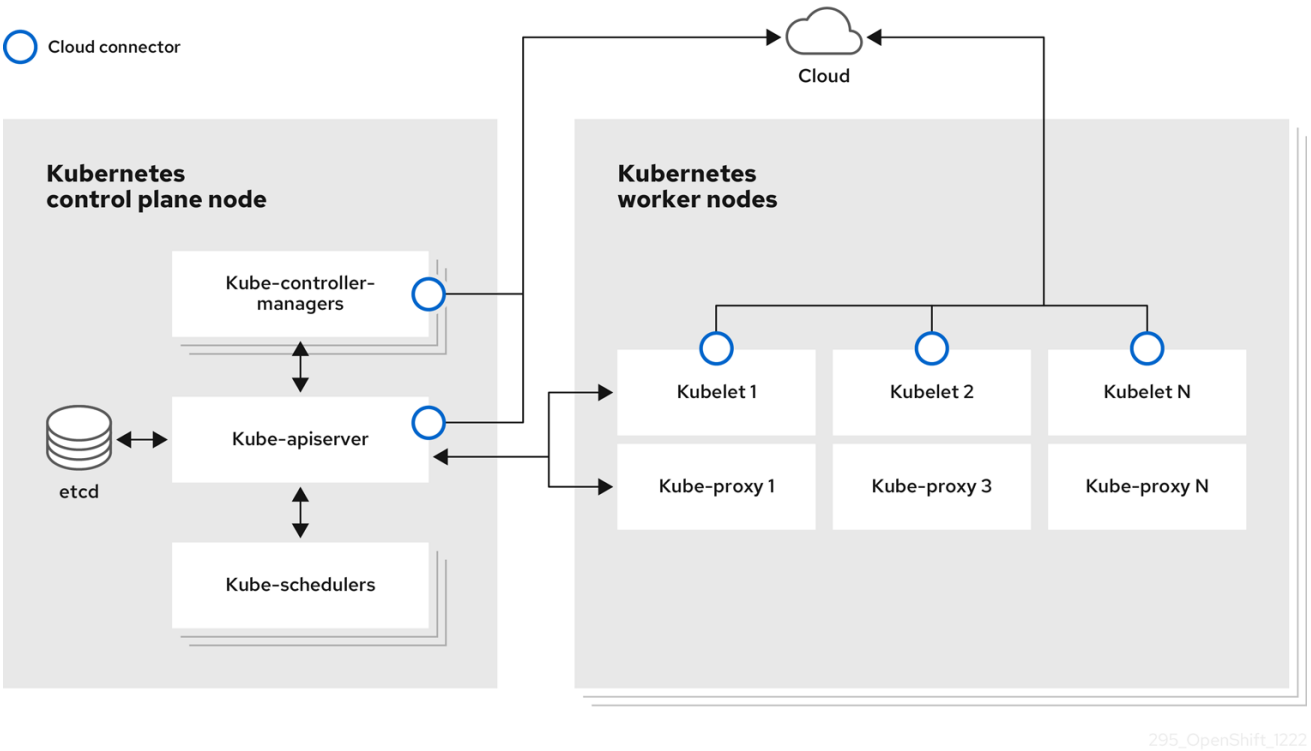
1.1. À PROPOS DES NŒUDS

Un nœud est une machine virtuelle ou bare-metal dans un cluster Kubernetes. Les nœuds de travail hébergent vos conteneurs d'application, regroupés sous forme de pods. Les nœuds du plan de contrôle exécutent les services nécessaires au contrôle du cluster Kubernetes. Dans OpenShift Container Platform, les nœuds du plan de contrôle contiennent plus que les services Kubernetes pour gérer le cluster OpenShift Container Platform.

Disposer de nœuds stables et sains dans un cluster est fondamental pour le bon fonctionnement de votre application hébergée. Dans OpenShift Container Platform, vous pouvez accéder à un nœud, le gérer et le surveiller par le biais de l'objet **Node** représentant le nœud. À l'aide de la CLI OpenShift (**oc**) ou de la console Web, vous pouvez effectuer les opérations suivantes sur un nœud.

Les composants suivants d'un nœud sont chargés de maintenir l'exécution des pods et de fournir l'environnement d'exécution Kubernetes.

- Exécution du conteneur: : Le runtime de conteneur est responsable de l'exécution des conteneurs. Kubernetes propose plusieurs runtimes tels que containerd, cri-o, rktlet et Docker.
- Kubelet: : Kubelet s'exécute sur les nœuds et lit les manifestes des conteneurs. Il s'assure que les conteneurs définis ont démarré et sont en cours d'exécution. Le processus kubelet maintient l'état du travail et du serveur de nœuds. Kubelet gère les règles du réseau et la redirection des ports. Le kubelet gère les conteneurs qui sont créés par Kubernetes uniquement.
- Kube-proxy: : Kube-proxy s'exécute sur chaque nœud du cluster et maintient le trafic réseau entre les ressources Kubernetes. Un Kube-proxy garantit que l'environnement réseau est isolé et accessible.
- DNS: : Cluster DNS est un serveur DNS qui sert les enregistrements DNS pour les services Kubernetes. Les conteneurs démarrés par Kubernetes incluent automatiquement ce serveur DNS dans leurs recherches DNS.



Opérations de lecture

Les opérations de lecture permettent à un administrateur ou à un développeur d'obtenir des informations sur les nœuds d'un cluster OpenShift Container Platform.

- Liste de tous les nœuds d'une grappe .
- Obtenir des informations sur un nœud, telles que l'utilisation de la mémoire et de l'unité centrale, l'état de santé, le statut et l'âge.
- Liste des pods en cours d'exécution sur un nœud .

Opérations de gestion

En tant qu'administrateur, vous pouvez facilement gérer un nœud dans un cluster OpenShift Container Platform grâce à plusieurs tâches :

- Ajouter ou mettre à jour les étiquettes des nœuds . Une étiquette est une paire clé-valeur appliquée à un objet **Node**. Vous pouvez contrôler l'ordonnancement des pods à l'aide des étiquettes.
- Modifier la configuration du nœud à l'aide d'une définition de ressource personnalisée (CRD) ou de l'objet **kubeletConfig**.
- Configurez les nœuds pour qu'ils autorisent ou non la planification des modules. Les nœuds de travail sains ayant un statut **Ready** autorisent le placement de pods par défaut, tandis que les nœuds du plan de contrôle ne le font pas. Vous pouvez modifier ce comportement par défaut en configurant les nœuds de travail pour qu'ils ne soient pas ordonnançables et les nœuds du plan de contrôle pour qu'ils soient ordonnançables.
- Allouez des ressources aux nœuds à l'aide du paramètre **system-reserved**. Vous pouvez permettre à OpenShift Container Platform de déterminer automatiquement les ressources optimales **system-reserved** CPU et mémoire pour vos nœuds, ou vous pouvez déterminer et définir manuellement les meilleures ressources pour vos nœuds.

- Configurez le nombre de pods pouvant être exécutés sur un nœud en fonction du nombre de cœurs de processeur du nœud, d'une limite stricte ou des deux.
- Redémarrer un nœud avec élégance en utilisant l'anti-affinité des pods.
- Supprimez un nœud d'un cluster en réduisant la taille du cluster à l'aide d'un ensemble de machines de calcul. Pour supprimer un nœud d'un cluster bare-metal, vous devez d'abord drainer tous les pods sur le nœud, puis supprimer manuellement le nœud.

Opérations de renforcement

OpenShift Container Platform vous permet de faire plus que d'accéder aux nœuds et de les gérer ; en tant qu'administrateur, vous pouvez effectuer les tâches suivantes sur les nœuds pour rendre le cluster plus efficace, plus convivial pour les applications et pour fournir un meilleur environnement à vos développeurs.

- Gérer les réglages au niveau des nœuds pour les applications à hautes performances qui nécessitent un certain niveau de réglage du noyau en utilisant l'opérateur de réglage des nœuds.
- Activez les profils de sécurité TLS sur le nœud pour protéger la communication entre le kubelet et le serveur API Kubernetes.
- Exécuter automatiquement des tâches d'arrière-plan sur les nœuds à l'aide d'ensembles de démons. Vous pouvez créer et utiliser des ensembles de démons pour créer un stockage partagé, exécuter un pod de journalisation sur chaque nœud ou déployer un agent de surveillance sur tous les nœuds.
- Libérez les ressources des nœuds à l'aide de la collecte des ordures (garbage collection) . Vous pouvez vous assurer que vos nœuds fonctionnent efficacement en supprimant les conteneurs terminés et les images qui ne sont pas référencées par les pods en cours d'exécution.
- Ajouter des arguments de noyau à un ensemble de nœuds .
- Configurez un cluster OpenShift Container Platform pour avoir des nœuds de travailleur à la périphérie du réseau (nœuds de travailleur à distance). Pour plus d'informations sur les défis posés par les nœuds de télétravail dans un cluster OpenShift Container Platform et sur certaines approches recommandées pour gérer les pods sur un nœud de télétravail, voir Utilisation de nœuds de télétravail à la périphérie du réseau .

1.2. À PROPOS DES COSSES

Un pod est un ou plusieurs conteneurs déployés ensemble sur un nœud. En tant qu'administrateur de cluster, vous pouvez définir un module, l'affecter à un nœud sain prêt à être programmé et le gérer. Un module fonctionne tant que les conteneurs sont en cours d'exécution. Vous ne pouvez pas modifier un module une fois qu'il a été défini et qu'il est en cours d'exécution. Voici quelques opérations que vous pouvez effectuer lorsque vous travaillez avec des modules :

Opérations de lecture

En tant qu'administrateur, vous pouvez obtenir des informations sur les pods d'un projet en effectuant les tâches suivantes :

- Liste des pods associés à un projet, avec des informations telles que le nombre de répliques et de redémarrages, l'état actuel et l'âge.
- Afficher les statistiques d'utilisation des pods , telles que la consommation de l'unité centrale, de la mémoire et du stockage.

Opérations de gestion

La liste de tâches suivante donne un aperçu de la façon dont un administrateur peut gérer les pods dans un cluster OpenShift Container Platform.

- Contrôler la planification des pods en utilisant les fonctionnalités avancées de planification disponibles dans OpenShift Container Platform :
  - Règles de liaison entre les nœuds, telles que l'affinité entre les nœuds, l' affinité entre les nœuds et l'anti-affinité.
  - Etiquettes et sélecteurs de nœuds.
  - Taches et tolérances.
  - Contraintes d'étalement de la topologie des pods .
  - Programmation secondaire.
- Configurer l 'ordonnanceur pour qu'il expulse les pod s sur la base de stratégies spécifiques afin que l'ordonnanceur réorganise les pods vers des nœuds plus appropriés.
- Configurer le comportement des pods après un redémarrage à l'aide de contrôleurs de pods et de politiques de redémarrage.
- Limiter le trafic de sortie et d'entrée sur un pod .
- Ajoutez et supprimez des volumes à n'importe quel objet ayant un modèle de pod . Un volume est un système de fichiers monté accessible à tous les conteneurs d'un module. Le stockage des conteneurs est éphémère ; vous pouvez utiliser des volumes pour conserver les données des conteneurs.

Opérations de renforcement

Vous pouvez travailler avec des pods plus facilement et plus efficacement à l'aide de divers outils et fonctionnalités disponibles dans OpenShift Container Platform. Les opérations suivantes impliquent l'utilisation de ces outils et fonctionnalités pour mieux gérer les pods.

Fonctionnement	User	Plus d'informations
Créer et utiliser un pod autoscaler horizontal.	Développeur	Vous pouvez utiliser un pod autoscaler horizontal pour spécifier le nombre minimum et maximum de pods que vous souhaitez exécuter, ainsi que l'utilisation de l'unité centrale ou de la mémoire que vos pods doivent viser. En utilisant un autoscaler de pods horizontal, vous pouvez automatiquement mettre à l'échelle les pods.

Fonctionnement	User	Plus d'informations
Installer et utiliser un pod autoscaler vertical.	Administrateur et développeur	En tant qu'administrateur, utilisez un pod autoscaler vertical pour mieux utiliser les ressources du cluster en surveillant les ressources et les besoins en ressources des charges de travail. En tant que développeur, utilisez un autoscaler vertical de pods pour vous assurer que vos pods restent opérationnels pendant les périodes de forte demande en planifiant les pods sur des nœuds disposant de suffisamment de ressources pour chaque pod.
Permettre l'accès à des ressources externes à l'aide de plugins pour appareils.	Administrateur	Un plugin de périphérique est un service gRPC fonctionnant sur les nœuds (externe au kubelet), qui gère des ressources matérielles spécifiques. Vous pouvez déployer un plugin de périphérique afin de fournir une solution cohérente et portable pour consommer des périphériques matériels à travers les clusters.
Fournir des données sensibles aux pods à l'aide de l'objet **Secret** .	Administrateur	Certaines applications ont besoin d'informations sensibles, telles que les mots de passe et les noms d'utilisateur. Vous pouvez utiliser l'objet **Secret** pour fournir ces informations à un module d'application.

1.3. À PROPOS DES CONTENEURS

Un conteneur est l'unité de base d'une application OpenShift Container Platform, qui comprend le code de l'application emballé avec ses dépendances, bibliothèques et binaires. Les conteneurs assurent la cohérence entre les environnements et les cibles de déploiement multiples : serveurs physiques, machines virtuelles (VM) et nuages privés ou publics.

Les technologies de conteneurs Linux sont des mécanismes légers permettant d'isoler les processus en cours d'exécution et de limiter l'accès aux seules ressources désignées. En tant qu'administrateur, vous pouvez effectuer diverses tâches sur un conteneur Linux, telles que :

- Copier des fichiers depuis et vers un conteneur.
- Permet aux conteneurs de consommer des objets API .
- Exécuter des commandes à distance dans un conteneur .
- Utiliser la redirection de port pour accéder aux applications dans un conteneur .

OpenShift Container Platform propose des conteneurs spécialisés appelés conteneurs Init. Les conteneurs Init s'exécutent avant les conteneurs d'application et peuvent contenir des utilitaires ou des scripts de configuration qui ne sont pas présents dans une image d'application. Vous pouvez utiliser un conteneur Init pour effectuer des tâches avant que le reste d'un pod ne soit déployé.

Outre l'exécution de tâches spécifiques sur les nœuds, les pods et les conteneurs, vous pouvez travailler avec l'ensemble du cluster OpenShift Container Platform pour maintenir l'efficacité du cluster et la haute disponibilité des pods d'application.

1.4. GLOSSAIRE DES TERMES COURANTS POUR LES NŒUDS DE OPENSHIFT CONTAINER PLATFORM

Ce glossaire définit les termes courants utilisés dans le contenu du site *node*.

Conteneur

Il s'agit d'une image légère et exécutable qui comprend un logiciel et toutes ses dépendances. Les conteneurs virtualisent le système d'exploitation, ce qui permet d'exécuter des conteneurs partout, du centre de données au nuage public ou privé, en passant par l'ordinateur portable d'un développeur.

Jeu de démons

Assure qu'une réplique du pod s'exécute sur les nœuds éligibles d'un cluster OpenShift Container Platform.

évacuation

Processus de partage de données vers l'extérieur par le biais du trafic sortant d'un réseau à partir d'un pod.

collecte des ordures

Le processus de nettoyage des ressources du cluster, telles que les conteneurs et les images terminés qui ne sont pas référencés par des pods en cours d'exécution.

Autoscaler horizontal (HPA)

Implémenté comme une ressource API Kubernetes et un contrôleur. Vous pouvez utiliser le HPA pour spécifier le nombre minimum et maximum de pods que vous souhaitez exécuter. Vous pouvez également spécifier l'utilisation du CPU ou de la mémoire que vos pods doivent viser. Le HPA réduit et augmente le nombre de pods lorsqu'un seuil donné de CPU ou de mémoire est franchi.

Entrée

Trafic entrant vers un pod.

Emploi

Un processus qui s'exécute jusqu'à son terme. Un job crée un ou plusieurs objets pods et s'assure que les pods spécifiés sont terminés avec succès.

Étiquettes

Vous pouvez utiliser des étiquettes, qui sont des paires clé-valeur, pour organiser et sélectionner des sous-ensembles d'objets, tels qu'un pod.

Nœud

Une machine de travail dans le cluster OpenShift Container Platform. Un nœud peut être une machine virtuelle (VM) ou une machine physique.

Opérateur de réglage des nœuds

Vous pouvez utiliser l'opérateur d'optimisation des nœuds pour gérer l'optimisation au niveau des nœuds en utilisant le démon TuneD. Il garantit que les spécifications de réglage personnalisées sont transmises à tous les démons TuneD conteneurisés qui s'exécutent dans le cluster dans un format que les démons comprennent. Les démons s'exécutent sur tous les nœuds de la grappe, un par nœud.

Opérateur d'assainissement de nœuds autonomes

L'opérateur s'exécute sur les nœuds de la grappe, identifie et redémarre les nœuds qui ne sont pas en bonne santé.

Cosse

Un ou plusieurs conteneurs avec des ressources partagées, telles que le volume et les adresses IP, fonctionnant dans votre cluster OpenShift Container Platform. Un pod est la plus petite unité de calcul définie, déployée et gérée.

Tolérance

Indique que le module est autorisé (mais pas obligatoire) à être planifié sur des nœuds ou des groupes de nœuds dont les caractéristiques correspondent à celles du module. Vous pouvez utiliser des tolérances pour permettre à l'ordonnanceur de planifier des pods avec des taints correspondants.

Souillure

Un objet central qui comprend une clé, une valeur et un effet. Les taches et les tolérances fonctionnent ensemble pour garantir que les pods ne sont pas programmés sur des nœuds non pertinents.

CHAPITRE 2. TRAVAILLER AVEC DES PODS

2.1. UTILISATION DES DOSETTES

Un *pod* est un ou plusieurs conteneurs déployés ensemble sur un hôte, et la plus petite unité de calcul qui peut être définie, déployée et gérée.

2.1.1. Comprendre les dosettes

Les pods sont l'équivalent approximatif d'une instance de machine (physique ou virtuelle) pour un conteneur. Chaque pod se voit attribuer sa propre adresse IP interne, et possède donc tout son espace portuaire. Les conteneurs au sein des pods peuvent partager leur stockage local et leur réseau.

Les pods ont un cycle de vie ; ils sont définis, puis ils sont affectés à l'exécution sur un nœud, puis ils s'exécutent jusqu'à ce que leur(s) conteneur(s) se termine(nt) ou qu'ils soient supprimés pour une autre raison. Les pods, en fonction de la politique et du code de sortie, peuvent être supprimés après la sortie, ou peuvent être conservés pour permettre l'accès aux journaux de leurs conteneurs.

OpenShift Container Platform considère les pods comme largement immuables ; il n'est pas possible d'apporter des modifications à la définition d'un pod pendant qu'il est en cours d'exécution. OpenShift Container Platform met en œuvre les changements en mettant fin à un pod existant et en le recréant avec une configuration modifiée, une ou plusieurs images de base, ou les deux. Les pods sont également considérés comme consommables et ne conservent pas d'état lorsqu'ils sont recréés. C'est pourquoi les pods doivent généralement être gérés par des contrôleurs de niveau supérieur, plutôt que directement par les utilisateurs.

NOTE

Pour connaître le nombre maximum de pods par hôte de nœud OpenShift Container Platform, voir les limites du cluster.

AVERTISSEMENT

Les pods nus qui ne sont pas gérés par un contrôleur de réplication ne seront pas reprogrammés en cas d'interruption du nœud.

2.1.2. Exemples de configurations de pods

OpenShift Container Platform exploite le concept Kubernetes d'un *pod*, qui est un ou plusieurs conteneurs déployés ensemble sur un hôte, et la plus petite unité de calcul qui peut être définie, déployée et gérée.

Voici un exemple de définition d'un pod à partir d'une application Rails. Il démontre de nombreuses caractéristiques des pods, dont la plupart sont abordées dans d'autres sujets et ne sont donc que brièvement mentionnées ici :

Pod définition de l'objet (YAML)

```
kind: Pod
```

```
apiVersion: v1
metadata:
  name: example
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/example
  uid: 5cc30063-0265780783bc
  resourceVersion: '165032'
  creationTimestamp: '2019-02-13T20:31:37Z'
  labels:
    app: hello-openshift (1)
  annotations:
    openshift.io/scc: anyuid
spec:
  restartPolicy: Always (2)
  serviceAccountName: default
  imagePullSecrets:
    - name: default-dockercfg-5zrhb
  priority: 0
  schedulerName: default-scheduler
  terminationGracePeriodSeconds: 30
  nodeName: ip-10-0-140-16.us-east-2.compute.internal
  securityContext: (3)
    seLinuxOptions:
      level: 's0:c11,c10'
  containers: (4)
    - resources: {}
      terminationMessagePath: /dev/termination-log
      name: hello-openshift
      securityContext:
        capabilities:
          drop:
            - MKNOD
        procMount: Default
      ports:
        - containerPort: 8080
          protocol: TCP
      imagePullPolicy: Always
      volumeMounts: (5)
        - name: default-token-wbqsl
          readOnly: true
          mountPath: /var/run/secrets/kubernetes.io/serviceaccount (6)
      terminationMessagePolicy: File
      image: registry.redhat.io/openshift4/ose-ogging-eventrouter:v4.3 (7)
  serviceAccount: default (8)
  volumes: (9)
    - name: default-token-wbqsl
      secret:
        secretName: default-token-wbqsl
        defaultMode: 420
  dnsPolicy: ClusterFirst
status:
  phase: Pending
  conditions:
    - type: Initialized
      status: 'True'
```

```
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
  - type: Ready
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: ContainersReady
    status: 'False'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
    reason: ContainersNotReady
    message: 'containers with unready status: [hello-openshift]'
  - type: PodScheduled
    status: 'True'
    lastProbeTime: null
    lastTransitionTime: '2019-02-13T20:31:37Z'
hostIP: 10.0.140.16
startTime: '2019-02-13T20:31:37Z'
containerStatuses:
  - name: hello-openshift
    state:
      waiting:
        reason: ContainerCreating
    lastState: {}
    ready: false
    restartCount: 0
    image: openshift/hello-openshift
    imageID: ''
qosClass: BestEffort
```

1. Les pods peuvent être "étiquetés" avec une ou plusieurs étiquettes, qui peuvent ensuite être utilisées pour sélectionner et gérer des groupes de pods en une seule opération. Les étiquettes sont stockées au format clé/valeur dans le hachage **metadata**.

2. La politique de redémarrage des pods avec les valeurs possibles **Always**, **OnFailure**, et **Never**. La valeur par défaut est **Always**.

3. OpenShift Container Platform définit un contexte de sécurité pour les conteneurs qui spécifie s'ils sont autorisés à s'exécuter en tant que conteneurs privilégiés, à s'exécuter en tant qu'utilisateur de leur choix, et plus encore. Le contexte par défaut est très restrictif, mais les administrateurs peuvent le modifier si nécessaire.

4. **containers** spécifie un tableau d'une ou plusieurs définitions de conteneurs.

5. Le conteneur spécifie où les volumes de stockage externes sont montés dans le conteneur. Dans ce cas, il y a un volume pour stocker l'accès aux informations d'identification dont le registre a besoin pour faire des demandes à l'API OpenShift Container Platform.

6. Spécifiez les volumes à fournir pour le module. Les volumes sont montés au chemin spécifié. Ne les montez pas à la racine du conteneur, /, ni à un chemin identique dans l'hôte et le conteneur. Cela peut corrompre votre système hôte si le conteneur est suffisamment privilégié, comme l'hôte **/dev/pts** files. Vous pouvez monter l'hôte en toute sécurité en utilisant **/host**.

7. Chaque conteneur du pod est instancié à partir de sa propre image de conteneur.

8 Les pods qui font des requêtes à l'API de OpenShift Container Platform sont assez courants pour qu'il y ait un champ **serviceAccount** pour spécifier l'utilisateur du compte de service auquel le pod

9 Le module définit les volumes de stockage que son ou ses conteneurs peuvent utiliser. Dans ce cas, il fournit un volume éphémère pour un volume **secret** contenant les jetons du compte de service par défaut.

Si vous attachez des volumes persistants qui ont un nombre élevé de fichiers à des pods, ces pods peuvent échouer ou prendre beaucoup de temps à démarrer. Pour plus d'informations, voir Lors de l'utilisation de volumes persistants avec un nombre élevé de fichiers dans OpenShift, pourquoi les pods ne démarrent-ils pas ou prennent-ils un temps excessif pour atteindre l'état "Ready" ?

NOTE

Cette définition de pod ne comprend pas les attributs qui sont remplis automatiquement par OpenShift Container Platform après la création du pod et le début de son cycle de vie. La documentation sur les pods Kubernetes contient des détails sur la fonctionnalité et le but des pods.

2.1.3. Ressources supplémentaires

- Pour plus d'informations sur les pods et le stockage, voir Comprendre le stockage persistant et Comprendre le stockage éphémère.

2.2. VISUALISATION DES NACELLES

En tant qu'administrateur, vous pouvez visualiser les pods de votre cluster et déterminer l'état de santé de ces pods et du cluster dans son ensemble.

2.2.1. À propos des cosses

OpenShift Container Platform exploite le concept Kubernetes d'un *pod*, qui est un ou plusieurs conteneurs déployés ensemble sur un hôte, et la plus petite unité de calcul qui peut être définie, déployée et gérée. Les pods sont l'équivalent approximatif d'une instance de machine (physique ou virtuelle) pour un conteneur.

Vous pouvez afficher une liste des pods associés à un projet spécifique ou consulter les statistiques d'utilisation des pods.

2.2.2. Visualiser les pods d'un projet

Vous pouvez afficher une liste des pods associés au projet en cours, y compris le nombre de répliques, l'état actuel, le nombre de redémarrages et l'âge du pod.

Procédure

Pour visualiser les pods d'un projet :

1. Modification du projet :

```
$ oc project <nom du projet>
```

2. Exécutez la commande suivante :

```
$ oc get pods
```

Par exemple :

```
$ oc get pods -n openshift-console
```

Exemple de sortie

```
NAME                  READY   STATUS    RESTARTS   AGE
console-698d866b78-bnshf   1/1     Running   2          165m
console-698d866b78-m87pm   1/1     Running   2          165m
```

Ajoutez les drapeaux **-o wide** pour afficher l'adresse IP du pod et le nœud où il se trouve.

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

Exemple de sortie

```
NAME                  READY   STATUS    RESTARTS   AGE    IP           NODE
NOMINATED NODE
console-698d866b78-bnshf   1/1     Running   2          166m   10.128.0.24   ip-10-0-152-
71.ec2.internal    <none>
console-698d866b78-m87pm   1/1     Running   2          166m   10.129.0.23   ip-10-0-173-
237.ec2.internal   <none>
```

2.2.3. Visualisation des statistiques d'utilisation des pods

Vous pouvez afficher les statistiques d'utilisation des pods, qui fournissent les environnements d'exécution des conteneurs. Ces statistiques d'utilisation comprennent la consommation de CPU, de mémoire et de stockage.

Conditions préalables

- Vous devez avoir l'autorisation **cluster-reader** pour voir les statistiques d'utilisation.
- Metrics doit être installé pour afficher les statistiques d'utilisation.

Procédure

Pour consulter les statistiques d'utilisation :

1. Exécutez la commande suivante :

```
$ oc adm top pods
```

Par exemple :

```
$ oc adm top pods -n openshift-console
```

Exemple de sortie

```
NAME                  CPU(cores)   MEMORY(bytes)
```

```
console-7f58c69899-q8c8k     0m          22Mi
console-7f58c69899-xhbgg     0m          25Mi
downloads-594fcccf94-bcxk8   3m          18Mi
downloads-594fcccf94-kv4p6   2m          15Mi
```

2. Exécutez la commande suivante pour afficher les statistiques d'utilisation des pods avec étiquettes :

```
$ oc adm top pod --selector=''
```

Vous devez choisir le sélecteur (requête d'étiquette) sur lequel filtrer. Prend en charge **=**, **==**, et **!=**.

2.2.4. Visualisation des journaux de ressources

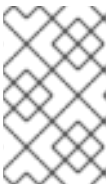
Vous pouvez consulter le journal de diverses ressources dans l'interface de commande OpenShift (oc) et dans la console Web. Les journaux se lisent à partir de la queue, ou de la fin, du journal.

Conditions préalables

- Accès à la CLI d'OpenShift (oc).

Procédure (UI)

1. Dans la console OpenShift Container Platform, naviguez vers **Workloads → Pods** ou naviguez vers le pod via la ressource que vous souhaitez étudier.



NOTE

Certaines ressources, telles que les constructions, n'ont pas de pods à interroger directement. Dans ce cas, vous pouvez trouver le lien **Logs** sur la page **Details** de la ressource.

2. Sélectionnez un projet dans le menu déroulant.
3. Cliquez sur le nom du module que vous souhaitez examiner.
4. Cliquez sur **Logs**.

Procédure (CLI)

- Visualiser le journal d'un pod spécifique :

```
oc logs -f <nom_du_pod> -c <nom_du_conteneur>
```

où :

-f

Facultatif : Spécifie que la sortie suit ce qui est écrit dans les journaux.

<pod_name>

Spécifie le nom du module.

<container_name>

Facultatif : Spécifie le nom d'un conteneur. Lorsqu'un module a plus d'un conteneur, vous devez spécifier le nom du conteneur.

Par exemple :

```
$ oc logs ruby-58cd97df55-mww7r
```

```
$ oc logs -f ruby-57f7f4855b-znl92 -c ruby
```

Le contenu des fichiers journaux est imprimé.

- Consulter le journal d'une ressource spécifique :

```
oc logs <object_type>/<resource_name> 1
```

1 Spécifie le type et le nom de la ressource.

Par exemple :

```
$ oc logs deployment/ruby
```

Le contenu des fichiers journaux est imprimé.

2.3. CONFIGURER UN CLUSTER OPENSHIFT CONTAINER PLATFORM POUR LES PODS

En tant qu'administrateur, vous pouvez créer et maintenir un cluster efficace pour les pods.

En veillant à l'efficacité de votre cluster, vous pouvez fournir un meilleur environnement à vos développeurs en utilisant des outils tels que ce que fait un pod lorsqu'il quitte, en veillant à ce que le nombre requis de pods soit toujours en cours d'exécution, quand redémarrer les pods conçus pour ne fonctionner qu'une seule fois, limiter la bande passante disponible pour les pods et comment maintenir les pods en cours d'exécution pendant les perturbations.

2.3.1. Configurer le comportement des pods après le redémarrage

Une politique de redémarrage de pod détermine comment OpenShift Container Platform réagit lorsque les conteneurs de ce pod quittent le pod. La politique s'applique à tous les conteneurs de ce pod.

Les valeurs possibles sont les suivantes :

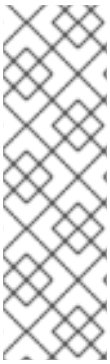
- **Always** - Tente de redémarrer un conteneur quitté avec succès sur le pod en continu, avec un délai exponentiel (10s, 20s, 40s) plafonné à 5 minutes. La valeur par défaut est **Always**.
- **OnFailure** - Tente de redémarrer un conteneur défaillant sur le pod avec un délai exponentiel (10s, 20s, 40s) plafonné à 5 minutes.
- **Never** - N'essaie pas de redémarrer les conteneurs qui sont sortis ou qui ont échoué sur le pod. Les pods échouent immédiatement et quittent le système.

Une fois que le module est lié à un nœud, il ne sera jamais lié à un autre nœud. Cela signifie qu'un contrôleur est nécessaire pour qu'un module puisse survivre à la défaillance d'un nœud :

Condition	Type de contrôleur	Politique de redémarrage
Les pods qui sont censés se terminer (tels que les calculs par lots)	Emploi	**OnFailure** ou **Never**
Les pods qui sont censés ne pas se terminer (tels que les serveurs web)	Contrôleur de réplication	**Always**.
Pods qui doivent fonctionner une fois par machine	Jeu de démons	Tous

Si un conteneur sur un pod échoue et que la politique de redémarrage est définie sur **OnFailure**, le pod reste sur le nœud et le conteneur est redémarré. Si vous ne souhaitez pas que le conteneur redémarre, utilisez la stratégie de redémarrage **Never**.

Si un pod entier tombe en panne, OpenShift Container Platform démarre un nouveau pod. Les développeurs doivent prendre en compte la possibilité que les applications soient redémarrées dans un nouveau pod. En particulier, les applications doivent gérer les fichiers temporaires, les verrous, les résultats incomplets, etc. causés par les exécutions précédentes.



NOTE

L'architecture Kubernetes attend des points d'extrémité fiables de la part des fournisseurs de cloud. Lorsqu'un fournisseur de cloud est en panne, le kubelet empêche OpenShift Container Platform de redémarrer.

Si les points d'extrémité du fournisseur de cloud sous-jacent ne sont pas fiables, n'installez pas un cluster en utilisant l'intégration du fournisseur de cloud. Installez le cluster comme s'il était dans un environnement sans nuage. Il n'est pas recommandé d'activer ou de désactiver l'intégration des fournisseurs de cloud dans un cluster installé.

Pour plus de détails sur la façon dont OpenShift Container Platform utilise la politique de redémarrage avec les conteneurs défaillants, voir les États d'exemple dans la documentation Kubernetes.

2.3.2. Limiter la bande passante disponible pour les pods

Vous pouvez appliquer la mise en forme du trafic de qualité de service à un pod et limiter efficacement sa bande passante disponible. Le trafic sortant (du pod) est géré par la police, qui laisse simplement tomber les paquets dépassant le taux configuré. Le trafic entrant (vers le module) est géré par la mise en forme des paquets en file d'attente afin de traiter efficacement les données. Les limites que vous imposez à un module n'affectent pas la bande passante des autres modules.

Procédure

Pour limiter la bande passante sur un pod :

1. Rédigez un fichier JSON de définition d'objet et spécifiez la vitesse de circulation des données à l'aide des annotations **kubernetes.io/ingress-bandwidth** et **kubernetes.io/egress-bandwidth**. Par exemple, pour limiter la bande passante de sortie et d'entrée des pods à 10M/s :

   Définition limitée de l'objet Pod

```
{
    "kind": "Pod",
    "spec": {
        "containers": [
            {
                "image": "openshift/hello-openshift",
                "name": "hello-openshift"
            }
        ]
    },
    "apiVersion": "v1",
    "metadata": {
        "name": "iperf-slow",
        "annotations": {
            "kubernetes.io/ingress-bandwidth": "10M",
            "kubernetes.io/egress-bandwidth": "10M"
        }
    }
}
```

2. Créer le pod à l'aide de la définition de l'objet :

```
oc create -f <file_ou_dir_path>
```

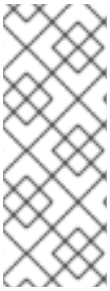
2.3.3. Comprendre comment utiliser les budgets de perturbation des pods pour spécifier le nombre de pods qui doivent être opérationnels

Un *pod disruption budget* fait partie de l'API Kubernetes, qui peut être géré avec des commandes **oc** comme d'autres types d'objets. Ils permettent de spécifier des contraintes de sécurité sur les pods pendant les opérations, comme la vidange d'un nœud pour la maintenance.

PodDisruptionBudget est un objet API qui spécifie le nombre ou le pourcentage minimum de répliques qui doivent être en service à un moment donné. La définition de ces valeurs dans les projets peut être utile lors de la maintenance des nœuds (par exemple, lors de la réduction ou de la mise à niveau d'un cluster) et n'est honorée qu'en cas d'éviction volontaire (et non en cas de défaillance d'un nœud).

La configuration d'un objet **PodDisruptionBudget** se compose des éléments clés suivants :

- Un sélecteur d'étiquettes, qui est une requête d'étiquettes sur un ensemble de pods.
- Un niveau de disponibilité, qui spécifie le nombre minimum de pods qui doivent être disponibles simultanément, soit :
    - **minAvailable** est le nombre de pods qui doivent toujours être disponibles, même en cas d'interruption.
    - **maxUnavailable** est le nombre de pods qui peuvent être indisponibles lors d'une perturbation.



NOTE

Available fait référence au nombre de pods qui ont la condition **Ready=True**. **Ready=True** fait référence au pod qui est capable de servir les requêtes et qui devrait être ajouté aux pools d'équilibrage de charge de tous les services correspondants.

Un **maxUnavailable** de **0%** ou **0** ou un **minAvailable** de **100%** ou égal au nombre de répliques est autorisé mais peut bloquer la vidange des nœuds.

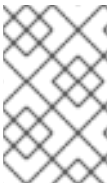
Vous pouvez vérifier les budgets de perturbation des pods dans tous les projets en procédant comme suit :

```
$ oc get poddisruptionbudget --all-namespaces
```

Exemple de sortie

```
NAMESPACE         NAME          MIN-AVAILABLE   SELECTOR
another-project   another-pdb   4               bar=foo
test-project      my-pdb        2               foo=bar
```

Le site **PodDisruptionBudget** est considéré comme sain lorsqu'il y a au moins **minAvailable** pods en cours d'exécution dans le système. Chaque pod dépassant cette limite peut être expulsé.



NOTE

En fonction des paramètres de priorité et de préemption des pods, les pods de moindre priorité peuvent être supprimés en dépit de leurs exigences en matière de budget de perturbation des pods.

2.3.3.1. Spécification du nombre de pods qui doivent être opérationnels avec des budgets de perturbation de pods

Vous pouvez utiliser un objet **PodDisruptionBudget** pour spécifier le nombre ou le pourcentage minimum de répliques qui doivent être opérationnelles à un moment donné.

Procédure

Pour configurer un budget de perturbation de pods :

1. Créez un fichier YAML avec une définition d'objet similaire à la suivante :

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2 2
  selector: 3
    matchLabels:
      foo: bar
```

1. **PodDisruptionBudget** fait partie du groupe **policy/v1** API.
2. Le nombre minimum de pods qui doivent être disponibles simultanément. Il peut s'agir d'un

nombre entier ou d'une chaîne de caractères spécifiant un pourcentage, par exemple **20%**.

3 Une requête d'étiquette sur un ensemble de ressources. Les résultats de **matchLabels** et **matchExpressions** sont logiquement joints. Laissez ce paramètre vide, par exemple **selector {}**, pour sélectionner tous les pods du projet.

Ou bien :

```
apiVersion: policy/v1 1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  maxUnavailable: 25% 2
  selector: 3
    matchLabels:
      foo: bar
```

1 **PodDisruptionBudget** fait partie du groupe **policy/v1** API.

2 Le nombre maximum de pods qui peuvent être indisponibles simultanément. Il peut s'agir d'un nombre entier ou d'une chaîne de caractères spécifiant un pourcentage, par exemple **20%**.

3 Une requête d'étiquette sur un ensemble de ressources. Les résultats de **matchLabels** et **matchExpressions** sont logiquement joints. Laissez ce paramètre vide, par exemple **selector {}**, pour sélectionner tous les pods du projet.

2. Exécutez la commande suivante pour ajouter l'objet au projet :

```
$ oc create -f </path/to/file> -n <project_name>
```

2.3.4. Prévenir l'enlèvement des nacelles à l'aide de nacelles critiques

Un certain nombre de composants essentiels au bon fonctionnement d'une grappe sont exécutés sur un nœud ordinaire de la grappe plutôt que sur le nœud principal. Un cluster peut cesser de fonctionner correctement si un add-on critique est expulsé.

Les pods marqués comme critiques ne sont pas autorisés à être expulsés.

Procédure

Rendre une nacelle critique :

1. Créer un spec **Pod** ou modifier les pods existants pour inclure la classe de priorité **system-cluster-critical**:

```
spec:
  template:
    metadata:
      name: critical-pod
    priorityClassName: system-cluster-critical 1
```

1 Classe de priorité par défaut pour les pods qui ne doivent jamais être expulsés d'un nœud.

Vous pouvez également spécifier **system-node-critical** pour les pods qui sont importants pour le cluster mais qui peuvent être supprimés si nécessaire.

2. Créer la capsule :

```
oc create -f <nom-de-fichier>.yaml
```

2.3.5. Réduction des délais d'attente des pods lors de l'utilisation de volumes persistants avec un nombre élevé de fichiers

Si un volume de stockage contient de nombreux fichiers (~1.000.000 ou plus), vous pouvez rencontrer des dépassements de temps de pod.

Cela peut se produire car, lorsque les volumes sont montés, OpenShift Container Platform modifie récursivement la propriété et les permissions du contenu de chaque volume afin de correspondre à l'adresse **fsGroup** spécifiée dans un pod **securityContext**. Pour les gros volumes, la vérification et la modification de la propriété et des autorisations peuvent prendre du temps, ce qui entraîne un démarrage très lent du pod.

Vous pouvez réduire ce délai en appliquant l'une des solutions suivantes :

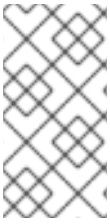
- Utilisez une contrainte de contexte de sécurité (SCC) pour ignorer le réétiquetage SELinux d'un volume.
- Utilisez le champ **fsGroupChangePolicy** dans un SCC pour contrôler la façon dont OpenShift Container Platform vérifie et gère la propriété et les permissions pour un volume.
- Utiliser une classe d'exécution pour ignorer le réétiquetage SELinux d'un volume.

Pour plus d'informations, voir Lors de l'utilisation de volumes persistants avec un nombre élevé de fichiers dans OpenShift, pourquoi les pods ne démarrent-ils pas ou prennent-ils un temps excessif pour atteindre l'état " Ready " ?

2.4. MISE À L'ÉCHELLE AUTOMATIQUE DES PODS AVEC LE POD AUTOSCALER HORIZONTAL

En tant que développeur, vous pouvez utiliser un pod autoscaler horizontal (HPA) pour spécifier comment OpenShift Container Platform doit automatiquement augmenter ou diminuer l'échelle d'un contrôleur de réplication ou d'une configuration de déploiement, en fonction des métriques collectées à partir des pods qui appartiennent à ce contrôleur de réplication ou à cette configuration de déploiement. Vous pouvez créer un HPA pour n'importe quel déploiement, configuration de déploiement, ensemble de répliques, contrôleur de réplication ou ensemble avec état.

Pour plus d'informations sur la mise à l'échelle des pods en fonction de mesures personnalisées, voir Mise à l'échelle automatique des pods en fonction de mesures personnalisées .



NOTE

Il est recommandé d'utiliser un objet **Deployment** ou **ReplicaSet**, sauf si vous avez besoin d'une fonctionnalité ou d'un comportement spécifique fourni par d'autres objets. Pour plus d'informations sur ces objets, voir Comprendre les objets Deployment et DeploymentConfig.

2.4.1. Comprendre les autoscalers de pods horizontaux

Vous pouvez créer un autoscaler de pods horizontal pour spécifier le nombre minimum et maximum de pods que vous souhaitez exécuter, ainsi que l'utilisation du CPU ou de la mémoire que vos pods doivent cibler.

Après avoir créé un autoscaler de pods horizontaux, OpenShift Container Platform commence à interroger les métriques de ressources CPU et/ou mémoire sur les pods. Lorsque ces métriques sont disponibles, l'autoscaler de pods horizontaux calcule le rapport entre l'utilisation actuelle de la métrique et l'utilisation souhaitée de la métrique, et augmente ou réduit l'échelle en conséquence. L'interrogation et la mise à l'échelle se produisent à intervalles réguliers, mais il peut s'écouler une à deux minutes avant que les mesures ne soient disponibles.

Pour les contrôleurs de réplication, cette mise à l'échelle correspond directement aux répliques du contrôleur de réplication. Pour les configurations de déploiement, la mise à l'échelle correspond directement au nombre de répliques de la configuration de déploiement. Notez que la mise à l'échelle automatique ne s'applique qu'au dernier déploiement dans la phase **Complete**.

OpenShift Container Platform prend automatiquement en compte les ressources et évite une mise à l'échelle automatique inutile lors des pics de ressources, comme lors du démarrage. Les pods dans l'état **unready** ont une utilisation de **0 CPU** lors de la mise à l'échelle et l'autoscaler ignore les pods lors de la mise à l'échelle. Les pods sans métriques connues ont une utilisation de **0\fPU** lors de la montée en charge et de **100\fPU** lors de la descente en charge. Cela permet une plus grande stabilité lors de la décision HPA. Pour utiliser cette fonctionnalité, vous devez configurer des contrôles de disponibilité afin de déterminer si un nouveau module est prêt à être utilisé.

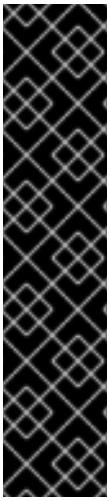
Pour utiliser les autoscalers de pods horizontaux, votre administrateur de cluster doit avoir correctement configuré les métriques de cluster.

2.4.1.1. Mesures prises en charge

Les métriques suivantes sont prises en charge par les autoscalers de pods horizontaux :

Tableau 2.1. Metrics

Métrique	Description	Version de l'API
Utilisation de l'unité centrale	Nombre de cœurs de CPU utilisés. Peut être utilisé pour calculer un pourcentage de l'unité centrale demandée par le pod.	**autoscaling/v1**, **autoscaling/v2**
Utilisation de la mémoire	Quantité de mémoire utilisée. Peut être utilisé pour calculer un pourcentage de la mémoire demandée par le pod.	**autoscaling/v2**



IMPORTANT

Pour l'autoscaling basé sur la mémoire, l'utilisation de la mémoire doit augmenter et diminuer proportionnellement au nombre de répliques. En moyenne :

- Une augmentation du nombre de répliques doit entraîner une diminution globale de l'utilisation de la mémoire (ensemble de travail) par pod.
- Une diminution du nombre de répliques doit entraîner une augmentation globale de l'utilisation de la mémoire par pod.

Utilisez la console web d'OpenShift Container Platform pour vérifier le comportement de la mémoire de votre application et assurez-vous que votre application répond à ces exigences avant d'utiliser l'autoscaling basé sur la mémoire.

L'exemple suivant illustre la mise à l'échelle automatique de l'objet **image-registry Deployment** . Le déploiement initial nécessite 3 pods, l'objet HPA augmente le minimum à 5 pods. L'objet HPA augmente le minimum à 5. Si l'utilisation du CPU sur les pods atteint 75 %, les pods passent à 7 :

```
$ oc autoscale deployment/image-registry --min=5 --max=7 --cpu-percent=75
```

Exemple de sortie

```
horizontalpodautoscaler.autoscaling/image-registry autoscaled
```

Exemple d'APS pour l'objet image-registry Deployment avec minReplicas fixé à 3

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: image-registry
  namespace: default
spec:
  maxReplicas: 7
  minReplicas: 3
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: image-registry
  targetCPUUtilizationPercentage: 75
status:
  currentReplicas: 5
  desiredReplicas: 0
```

1. Afficher le nouvel état du déploiement :

   ```
   $ oc get deployment image-registry
   ```

   Il y a maintenant 5 pods dans le déploiement :

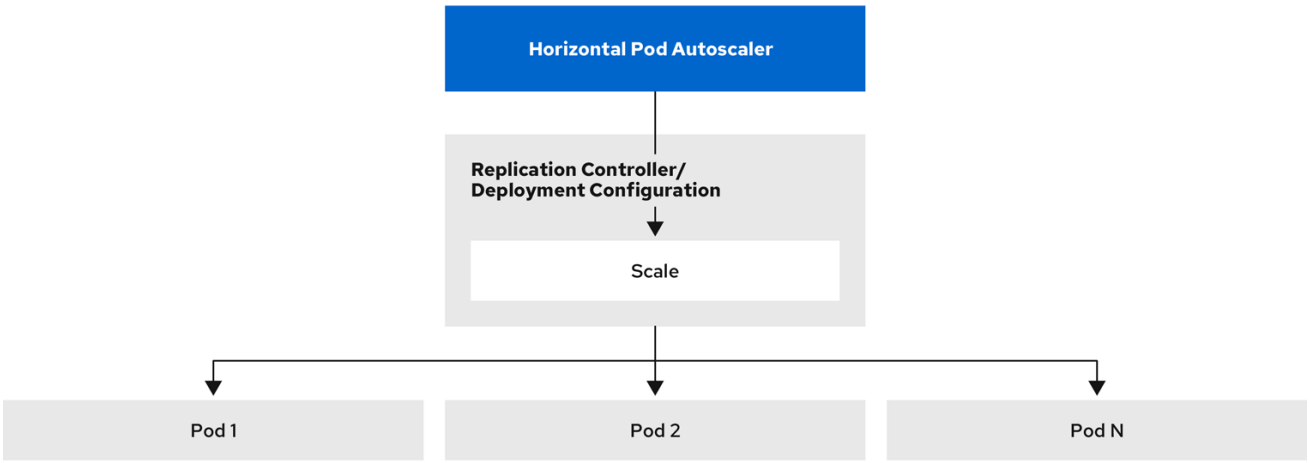
   ### Exemple de sortie

   ```
   NAME           REVISION   DESIRED   CURRENT   TRIGGERED BY
   image-registry   1          5         5         config
   ```

2.4.2. Comment fonctionne l'APH ?

Le pod autoscaler horizontal (HPA) étend le concept de pod autoscaling. Le HPA vous permet de créer et de gérer un groupe de nœuds à charge équilibrée. Le HPA augmente ou diminue automatiquement le nombre de pods lorsqu'un seuil donné de CPU ou de mémoire est franchi.

Figure 2.1. Flux de travail de haut niveau de l'APH



L'HPA est une ressource API dans le groupe API Kubernetes autoscaling. L'autoscaler fonctionne comme une boucle de contrôle avec une valeur par défaut de 15 secondes pour la période de synchronisation. Au cours de cette période, le gestionnaire du contrôleur interroge l'utilisation du CPU, de la mémoire, ou des deux, par rapport à ce qui est défini dans le fichier YAML pour l'HPA. Le gestionnaire de contrôleur obtient les métriques d'utilisation de l'API de métriques de ressources pour les métriques de ressources par pod comme le CPU ou la mémoire, pour chaque pod ciblé par l'HPA.

Si une valeur cible d'utilisation est définie, le contrôleur calcule la valeur d'utilisation en tant que pourcentage de la demande de ressources équivalente sur les conteneurs de chaque pod. Le contrôleur prend ensuite la moyenne de l'utilisation dans tous les pods ciblés et produit un ratio qui est utilisé pour mettre à l'échelle le nombre de répliques souhaitées. L'APH est configuré pour récupérer les métriques sur **metrics.k8s.io**, qui est fourni par le serveur de métriques. En raison de la nature dynamique de l'évaluation des métriques, le nombre de réplicas peut fluctuer lors de la mise à l'échelle d'un groupe de réplicas.

NOTE

Pour mettre en œuvre l'APH, tous les pods ciblés doivent avoir une demande de ressource définie sur leurs conteneurs.

2.4.3. A propos des demandes et des limites

L'ordonnanceur utilise la demande de ressources que vous spécifiez pour les conteneurs d'un module, afin de décider sur quel nœud placer le module. Le kubelet applique la limite de ressources que vous spécifiez pour un conteneur afin de s'assurer que le conteneur n'est pas autorisé à utiliser plus que la limite spécifiée. Le kubelet réserve également la quantité demandée de cette ressource système spécifiquement pour l'utilisation de ce conteneur.

Comment utiliser les indicateurs de ressources ?

Dans les spécifications du pod, vous devez spécifier les demandes de ressources, telles que le CPU et la mémoire. Le HPA utilise cette spécification pour déterminer l'utilisation des ressources, puis augmente ou réduit la cible.

Par exemple, l'objet HPA utilise la source de métriques suivante :

```
type: Resource
resource:
  name: cpu
  target:
    type: Utilization
    averageUtilization: 60
```

Dans cet exemple, l'APH maintient l'utilisation moyenne des modules dans la cible de mise à l'échelle à 60 %. L'utilisation est le rapport entre l'utilisation actuelle de la ressource et la ressource demandée pour le module.

2.4.4. Meilleures pratiques

Tous les pods doivent avoir des demandes de ressources configurées

Le HPA prend une décision de mise à l'échelle basée sur les valeurs observées d'utilisation du CPU ou de la mémoire des pods dans un cluster OpenShift Container Platform. Les valeurs d'utilisation sont calculées en pourcentage des demandes de ressources de chaque pod. Des valeurs de demandes de ressources manquantes peuvent affecter les performances optimales de l'APH.

Configurer la période de refroidissement

Lors de la mise à l'échelle automatique des pods horizontaux, il peut y avoir une mise à l'échelle rapide des événements sans intervalle de temps. Configurez la période de refroidissement pour éviter les fluctuations fréquentes des répliques. Vous pouvez spécifier une période de refroidissement en configurant le champ **stabilizationWindowSeconds**. La fenêtre de stabilisation est utilisée pour limiter la fluctuation du nombre de répliques lorsque les métriques utilisées pour la mise à l'échelle continuent de fluctuer. L'algorithme de mise à l'échelle automatique utilise cette fenêtre pour déduire un état antérieur souhaité et éviter les changements non désirés de l'échelle de la charge de travail.

Par exemple, une fenêtre de stabilisation est spécifiée pour le champ **scaleDown**:

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
```

Dans l'exemple ci-dessus, tous les états souhaités au cours des cinq dernières minutes sont pris en compte. Cela permet d'obtenir un maximum glissant et d'éviter que l'algorithme de mise à l'échelle ne supprime fréquemment des nacelles pour en recréer une équivalente quelques instants plus tard.

2.4.4.1. Politiques d'échelonnement

L'API **autoscaling/v2** vous permet d'ajouter *scaling policies* à un pod autoscaler horizontal. Une politique de mise à l'échelle contrôle la façon dont l'autoscaler de pods horizontaux (HPA) de OpenShift Container Platform met à l'échelle les pods. Les politiques de mise à l'échelle vous permettent de limiter le taux de mise à l'échelle des pods par les HPA en définissant un nombre ou un pourcentage spécifique à mettre à l'échelle dans une période de temps spécifiée. Vous pouvez également définir une politique de mise à l'échelle *stabilization window*, qui utilise les états souhaités calculés précédemment pour contrôler la mise à l'échelle si les métriques fluctuent. Vous pouvez créer plusieurs politiques pour la

même direction de mise à l'échelle et déterminer la politique à utiliser en fonction de la quantité de changement. Vous pouvez également limiter la mise à l'échelle par des itérations temporelles. L'APH met à l'échelle les pods au cours d'une itération, puis effectue la mise à l'échelle, si nécessaire, au cours des itérations suivantes.

Exemple d'objet HPA avec une politique de mise à l'échelle

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
  behavior:
    scaleDown: 1
      policies: 2
      - type: Pods 3
        value: 4 4
        periodSeconds: 60 5
      - type: Percent
        value: 10 6
        periodSeconds: 60
      selectPolicy: Min 7
      stabilizationWindowSeconds: 300 8
    scaleUp: 9
      policies:
      - type: Pods
        value: 5 10
        periodSeconds: 70
      - type: Percent
        value: 12 11
        periodSeconds: 80
      selectPolicy: Max
      stabilizationWindowSeconds: 0
...
```

1. Spécifie la direction de la politique de mise à l'échelle, soit **scaleDown** ou **scaleUp**. Cet exemple crée une politique de mise à l'échelle vers le bas.
2. Définit la politique de mise à l'échelle.
3. Détermine si la politique s'adapte à un nombre spécifique de pods ou à un pourcentage de pods à chaque itération. La valeur par défaut est **pods**.
4. Détermine la quantité de mise à l'échelle, soit le nombre de pods, soit le pourcentage de pods, à chaque itération. Il n'y a pas de valeur par défaut pour la mise à l'échelle par nombre de modules.
5. Détermine la durée d'une itération de mise à l'échelle. La valeur par défaut est **15** secondes.
6. La valeur par défaut de la réduction d'échelle en pourcentage est de 100 %.
7. Détermine la politique à utiliser en premier, si plusieurs politiques sont définies. Spécifiez **Max** pour utiliser la stratégie qui autorise le plus grand nombre de changements, **Min** pour utiliser la stratégie qui autorise le plus petit nombre de changements, ou **Disabled** pour empêcher l'HPA de se mettre

à l'échelle dans cette direction de stratégie. La valeur par défaut est **Max**.

8 Détermine la période de temps pendant laquelle l'APH doit revenir sur les états souhaités. La valeur par défaut est **0**.

9 Cet exemple permet d'élaborer une politique de mise à l'échelle.

10 Le montant de la mise à l'échelle du nombre de pods. La valeur par défaut de l'augmentation du nombre de modules est de 4 %.

11 Le montant de la mise à l'échelle par le pourcentage de pods. La valeur par défaut de la mise à l'échelle par pourcentage est de 100 %.

Exemple de politique de réduction d'échelle

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory
  namespace: default
spec:
...
  minReplicas: 20
...
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Pods
        value: 4
        periodSeconds: 30
      - type: Percent
        value: 10
        periodSeconds: 60
      selectPolicy: Max
    scaleUp:
      selectPolicy: Disabled
```

Dans cet exemple, lorsque le nombre de pods est supérieur à 40, la politique basée sur le pourcentage est utilisée pour la réduction d'échelle, car cette politique entraîne un changement plus important, comme l'exige le site **selectPolicy**.

S'il y a 80 répliques de pods, lors de la première itération, l'APH réduit les pods de 8, soit 10 % des 80 pods (sur la base des paramètres **type: Percent** et **value: 10**), en une minute (**periodSeconds: 60**). Pour l'itération suivante, le nombre de nacelles est de 72. L'APH calcule que 10 % des nacelles restantes représentent 7,2, qu'il arrondit à 8 et réduit de 8 nacelles. À chaque itération suivante, le nombre de nacelles à réduire est recalculé en fonction du nombre de nacelles restantes. Lorsque le nombre de nacelles est inférieur à 40, la politique basée sur les nacelles est appliquée, car le nombre basé sur les nacelles est supérieur au nombre basé sur le pourcentage. Le HPA réduit 4 pods à la fois (**type: Pods** et **value: 4**), sur 30 secondes (**periodSeconds: 30**), jusqu'à ce qu'il reste 20 répliques (**minReplicas**).

Le paramètre **selectPolicy: Disabled** empêche l'APH de mettre à l'échelle les pods. Vous pouvez augmenter manuellement l'échelle en ajustant le nombre de répliques dans l'ensemble de répliques ou l'ensemble de déploiement, si nécessaire.

Si elle est définie, vous pouvez visualiser la politique de mise à l'échelle à l'aide de la commande **oc edit**:

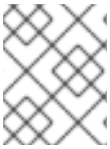
```
$ oc edit hpa hpa-resource-metrics-memory
```

Exemple de sortie

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  annotations:
    autoscaling.alpha.kubernetes.io/behavior:\
'{"ScaleUp":{"StabilizationWindowSeconds":0,"SelectPolicy":"Max","Policies":
[{"Type":"Pods","Value":4,"PeriodSeconds":15},{"Type":"Percent","Value":100,"PeriodSeconds":15}]},\
"ScaleDown":{"StabilizationWindowSeconds":300,"SelectPolicy":"Min","Policies":
[{"Type":"Pods","Value":4,"PeriodSeconds":60},{"Type":"Percent","Value":10,"PeriodSeconds":60}]}}'
...
```

2.4.5. Création d'un pod autoscaler horizontal à l'aide de la console web

Depuis la console web, vous pouvez créer un pod autoscaler horizontal (HPA) qui spécifie le nombre minimum et maximum de pods que vous souhaitez exécuter sur un objet **Deployment** ou **DeploymentConfig**. Vous pouvez également définir la quantité d'utilisation de CPU ou de mémoire que vos pods doivent cibler.



NOTE

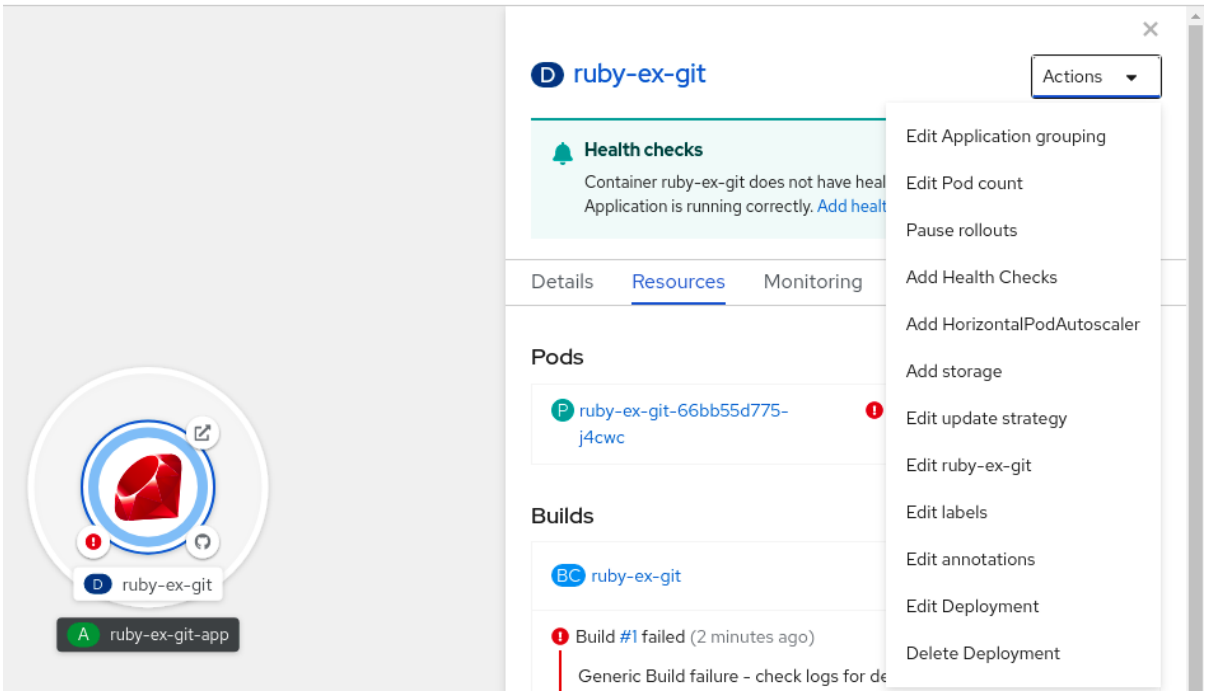
Un HPA ne peut pas être ajouté aux déploiements qui font partie d'un service soutenu par un opérateur, d'un service Knative ou d'une carte Helm.

Procédure

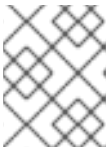
Pour créer un HPA dans la console web :

1. Dans la vue **Topology**, cliquez sur le nœud pour faire apparaître le volet latéral.
2. Dans la liste déroulante **Actions**, sélectionnez **Add HorizontalPodAutoscaler** pour ouvrir le formulaire **Add HorizontalPodAutoscaler**.

Figure 2.2. Ajouter HorizontalPodAutoscaler



3. Dans le formulaire **Add HorizontalPodAutoscaler**, définissez le nom, les limites minimales et maximales du pod, l'utilisation du processeur et de la mémoire, et cliquez sur **Save**.

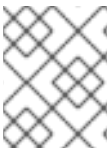


NOTE

Si l'une des valeurs de l'utilisation du processeur et de la mémoire est manquante, un avertissement s'affiche.

Pour éditer un HPA dans la console web :

1. Dans la vue **Topology**, cliquez sur le nœud pour faire apparaître le volet latéral.
2. Dans la liste déroulante **Actions**, sélectionnez **Edit HorizontalPodAutoscaler** pour ouvrir le formulaire **Edit Horizontal Pod Autoscaler**.
3. Dans le formulaire **Edit Horizontal Pod Autoscaler**, modifiez les limites minimales et maximales du pod et l'utilisation du CPU et de la mémoire, puis cliquez sur **Save**.



NOTE

Lors de la création ou de la modification du pod autoscaler horizontal dans la console web, vous pouvez passer de **Form view** à **YAML view**.

Pour supprimer un HPA dans la console web :

1. Dans la vue **Topology**, cliquez sur le nœud pour faire apparaître le panneau latéral.
2. Dans la liste déroulante **Actions**, sélectionnez **Remove HorizontalPodAutoscaler**.
3. Dans la fenêtre de confirmation, cliquez sur **Remove** pour supprimer le HPA.

2.4.6. Création d'un pod autoscaler horizontal pour l'utilisation du CPU à l'aide de la CLI

À l'aide de la CLI d'OpenShift Container Platform, vous pouvez créer un pod autoscaler horizontal (HPA) pour mettre automatiquement à l'échelle un objet existant **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, ou **StatefulSet**. Le HPA met à l'échelle les pods associés à cet objet pour maintenir l'utilisation du CPU que vous spécifiez.

NOTE

Il est recommandé d'utiliser un objet **Deployment** ou **ReplicaSet**, sauf si vous avez besoin d'une fonction ou d'un comportement spécifique fourni par d'autres objets.

Le HPA augmente et diminue le nombre de répliques entre les nombres minimum et maximum pour maintenir l'utilisation spécifiée du CPU dans tous les pods.

Lors de l'autoscaling pour l'utilisation du CPU, vous pouvez utiliser la commande **oc autoscale** et spécifier le nombre minimum et maximum de pods que vous souhaitez exécuter à tout moment et l'utilisation moyenne du CPU que vos pods doivent cibler. Si vous ne spécifiez pas de minimum, les pods reçoivent des valeurs par défaut du serveur OpenShift Container Platform.

Pour adapter l'autoscale à une valeur de CPU spécifique, créez un objet **HorizontalPodAutoscaler** avec la CPU cible et les limites du pod.

Conditions préalables

Pour utiliser les autoscalers de pods horizontaux, votre administrateur de cluster doit avoir correctement configuré les métriques du cluster. Vous pouvez utiliser la commande **oc describe PodMetrics <pod-name>** pour déterminer si les métriques sont configurées. Si les métriques sont configurées, la sortie est similaire à ce qui suit, avec **Cpu** et **Memory** affichés sous **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Exemple de sortie

```
Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

```
Timestamp:          2019-05-23T18:47:56Z
Window:             1m0s
Events:             <none>
```

Procédure

Pour créer un pod autoscaler horizontal pour l'utilisation du CPU :

1. Effectuez l'une des opérations suivantes :

   - Pour mettre à l'échelle en fonction du pourcentage d'utilisation du processeur, créez un objet **HorizontalPodAutoscaler** pour un objet existant :

     ```
     $ oc autoscale <object_type>/<name> \ 1
       --min <number> \ 2
       --max <number> \ 3
       --cpu-percent=<percent> 4
     ```

     1. Spécifiez le type et le nom de l'objet à mettre à l'échelle. L'objet doit exister et être un **Deployment**, **DeploymentConfig**/**dc**, **ReplicaSet**/**rs**, **ReplicationController**/**rc**, ou **StatefulSet**.
     2. Optionnellement, spécifier le nombre minimum de répliques lors d'une réduction d'échelle.
     3. Spécifiez le nombre maximum de répliques lors de la mise à l'échelle.
     4. Spécifiez l'utilisation moyenne de l'unité centrale cible sur tous les pods, représentée en pourcentage de l'unité centrale demandée. Si elle n'est pas spécifiée ou si elle est négative, une politique de mise à l'échelle automatique par défaut est utilisée.

     Par exemple, la commande suivante montre la mise à l'échelle automatique de l'objet **image-registry Deployment** . Le déploiement initial nécessite 3 pods, l'objet HPA augmente le minimum à 5. L'objet HPA augmente le minimum à 5. Si l'utilisation du CPU sur les pods atteint 75 %, les pods passeront à 7 :

     ```
     $ oc autoscale deployment/image-registry --min=5 --max=7 --cpu-percent=75
     ```

   - Pour mettre à l'échelle une valeur de CPU spécifique, créez un fichier YAML similaire au suivant pour un objet existant :

     a. Créez un fichier YAML similaire au suivant :

     ```
     apiVersion: autoscaling/v2 1
     kind: HorizontalPodAutoscaler
     metadata:
       name: cpu-autoscale 2
       namespace: default
     spec:
       scaleTargetRef:
         apiVersion: apps/v1 3
         kind: Deployment 4
         name: example 5
       minReplicas: 1 6
     ```

```
maxReplicas: 10 7
metrics: 8
- type: Resource
  resource:
    name: cpu 9
    target:
      type: AverageValue 10
      averageValue: 500m 11
```

1. Utilisez l'API **autoscaling/v2**.
2. Spécifiez un nom pour cet objet autoscaler de pods horizontaux.
3. Indiquez la version API de l'objet à mettre à l'échelle :
   - Pour un objet **Deployment**, **ReplicaSet**, **Statefulset**, utilisez **apps/v1**.
   - Pour un **ReplicationController**, utilisez **v1**.
   - Pour un **DeploymentConfig**, utilisez **apps.openshift.io/v1**.
4. Spécifiez le type d'objet. L'objet doit être un **Deployment**, **DeploymentConfig/dc**, **ReplicaSet/rs**, **ReplicationController/rc**, ou **StatefulSet**.
5. Indiquez le nom de l'objet à mettre à l'échelle. L'objet doit exister.
6. Spécifiez le nombre minimum de répliques lors de la réduction d'échelle.
7. Spécifiez le nombre maximum de répliques lors de la mise à l'échelle.
8. Utilisez le paramètre **metrics** pour l'utilisation de la mémoire.
9. Spécifiez **cpu** pour l'utilisation de l'unité centrale.
10. Régler sur **AverageValue**.
11. Réglé sur **averageValue** avec la valeur ciblée de l'unité centrale.

b. Créer le pod autoscaler horizontal :

```
oc create -f <nom-de-fichier>.yaml
```

2. Vérifiez que le pod horizontal autoscaler a été créé :

```
$ oc get hpa cpu-autoscale
```

Exemple de sortie

```
NAME          REFERENCE            TARGETS      MINPODS   MAXPODS   REPLICAS
AGE
cpu-autoscale Deployment/example   173m/500m    1         10        1         20m
```

2.4.7. Création d'un objet autoscaler pod horizontal pour l'utilisation de la mémoire à l'aide de la CLI

À l'aide de la CLI d'OpenShift Container Platform, vous pouvez créer un pod autoscaler horizontal (HPA) pour mettre automatiquement à l'échelle un objet existant **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, ou **StatefulSet**. Le HPA met à l'échelle les pods associés à cet objet pour maintenir l'utilisation moyenne de la mémoire que vous spécifiez, soit une valeur directe, soit un pourcentage de la mémoire demandée.

NOTE

Il est recommandé d'utiliser un objet **Deployment** ou **ReplicaSet**, sauf si vous avez besoin d'une fonction ou d'un comportement spécifique fourni par d'autres objets.

L'APH augmente et diminue le nombre de répliques entre les nombres minimum et maximum pour maintenir l'utilisation de la mémoire spécifiée dans tous les pods.

Pour l'utilisation de la mémoire, vous pouvez spécifier le nombre minimum et maximum de pods et l'utilisation moyenne de la mémoire que vos pods doivent viser. Si vous ne spécifiez pas de minimum, les pods reçoivent des valeurs par défaut du serveur OpenShift Container Platform.

Conditions préalables

Pour utiliser les autoscalers de pods horizontaux, votre administrateur de cluster doit avoir correctement configuré les métriques du cluster. Vous pouvez utiliser la commande **oc describe PodMetrics <pod-name>** pour déterminer si les métriques sont configurées. Si les métriques sont configurées, la sortie est similaire à ce qui suit, avec **Cpu** et **Memory** affichés sous **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-129-223.compute.internal -n openshift-kube-scheduler
```

Exemple de sortie

```
Name:         openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Cpu:     0
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2020-02-14T22:21:14Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-129-223.compute.internal
Timestamp:             2020-02-14T22:21:14Z
Window:                5m0s
Events:                <none>
```

Procédure

Pour créer un autoscaler de pods horizontaux pour l'utilisation de la mémoire :

1. Créez un fichier YAML pour l'un des éléments suivants :

   - Pour mettre à l'échelle une valeur de mémoire spécifique, créez un objet **HorizontalPodAutoscaler** similaire au suivant pour un objet existant :

```
apiVersion: autoscaling/v2 (1)
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-resource-metrics-memory (2)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (3)
    kind: Deployment (4)
    name: example (5)
  minReplicas: 1 (6)
  maxReplicas: 10 (7)
  metrics: (8)
  - type: Resource
    resource:
      name: memory (9)
      target:
        type: AverageValue (10)
        averageValue: 500Mi (11)
  behavior: (12)
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Pods
        value: 4
        periodSeconds: 60
      - type: Percent
        value: 10
        periodSeconds: 60
      selectPolicy: Max
```

(1) Utilisez l'API **autoscaling/v2**.

(2) Spécifiez un nom pour cet objet autoscaler de pods horizontaux.

(3) Indiquez la version API de l'objet à mettre à l'échelle :

- Pour un objet **Deployment**, **ReplicaSet**, ou **Statefulset**, utilisez **apps/v1**.
- Pour un **ReplicationController**, utilisez **v1**.
- Pour un **DeploymentConfig**, utilisez **apps.openshift.io/v1**.

(4) Spécifiez le type d'objet. L'objet doit être un **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, ou **StatefulSet**.

5. Indiquez le nom de l'objet à mettre à l'échelle. L'objet doit exister.
6. Spécifiez le nombre minimum de répliques lors de la réduction d'échelle.
7. Spécifiez le nombre maximum de répliques lors de la mise à l'échelle.
8. Utilisez le paramètre **metrics** pour l'utilisation de la mémoire.
9. Spécifiez **memory** pour l'utilisation de la mémoire.
10. Réglez le type sur **AverageValue**.
11. Spécifiez **averageValue** et une valeur de mémoire spécifique.
12. Facultatif : Spécifiez une politique de mise à l'échelle pour contrôler le taux de mise à l'échelle vers le haut ou vers le bas.

- Pour mettre à l'échelle un pourcentage, créez un objet **HorizontalPodAutoscaler** similaire au suivant pour un objet existant :

```
apiVersion: autoscaling/v2 (1)
kind: HorizontalPodAutoscaler
metadata:
  name: memory-autoscale (2)
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (3)
    kind: Deployment (4)
    name: example (5)
  minReplicas: 1 (6)
  maxReplicas: 10 (7)
  metrics: (8)
  - type: Resource
    resource:
      name: memory (9)
      target:
        type: Utilization (10)
        averageUtilization: 50 (11)
  behavior: (12)
    scaleUp:
      stabilizationWindowSeconds: 180
      policies:
      - type: Pods
        value: 6
        periodSeconds: 120
      - type: Percent
        value: 10
        periodSeconds: 120
      selectPolicy: Max
```

1. Utilisez l'API **autoscaling/v2**.
2. Spécifiez un nom pour cet objet autoscaler de pods horizontaux.

3. Indiquez la version API de l'objet à mettre à l'échelle :
   - Pour un ReplicationController, utilisez **v1**.
   - Pour un DeploymentConfig, utilisez **apps.openshift.io/v1**.
   - Pour un objet Deployment, ReplicaSet, Statefulset, utilisez **apps/v1**.
4. Spécifiez le type d'objet. L'objet doit être un **Deployment**, **DeploymentConfig**, **ReplicaSet**, **ReplicationController**, ou **StatefulSet**.
5. Indiquez le nom de l'objet à mettre à l'échelle. L'objet doit exister.
6. Spécifiez le nombre minimum de répliques lors de la réduction d'échelle.
7. Spécifiez le nombre maximum de répliques lors de la mise à l'échelle.
8. Utilisez le paramètre **metrics** pour l'utilisation de la mémoire.
9. Spécifiez **memory** pour l'utilisation de la mémoire.
10. Régler sur **Utilization**.
11. Spécifiez **averageUtilization** et un objectif d'utilisation moyenne de la mémoire sur tous les pods, représenté en pourcentage de la mémoire demandée. Les pods cibles doivent avoir des demandes de mémoire configurées.
12. Facultatif : Spécifiez une politique de mise à l'échelle pour contrôler le taux de mise à l'échelle vers le haut ou vers le bas.

2. Créer le pod autoscaler horizontal :

```
oc create -f <nom-de-fichier>.yaml
```

Par exemple :

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

Exemple de sortie

```
horizontalpodautoscaler.autoscaling/hpa-resource-metrics-memory created
```

3. Vérifiez que le pod horizontal autoscaler a été créé :

```
$ oc get hpa hpa-resource-metrics-memory
```

Exemple de sortie

```
NAME                        REFERENCE           TARGETS        MINPODS   MAXPODS
REPLICAS   AGE
hpa-resource-metrics-memory   Deployment/example   2441216/500Mi   1         10        1
20m
```

```
$ oc describe hpa hpa-resource-metrics-memory
```

Exemple de sortie

```
Name:                       hpa-resource-metrics-memory
Namespace:                  default
Labels:                     <none>
Annotations:                <none>
CreationTimestamp:          Wed, 04 Mar 2020 16:31:37 +0530
Reference:                  Deployment/example
Metrics:                    ( current / target )
  resource memory on pods:  2441216 / 500Mi
Min replicas:               1
Max replicas:               10
ReplicationController pods: 1 current / 1 desired
Conditions:
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    recommended size matches current size
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from memory resource
  ScalingLimited  False   DesiredWithinRange  the desired count is within the acceptable range
Events:
  Type    Reason              Age               From                       Message
  ----    ------              ----              ----                       -------
  Normal  SuccessfulRescale   6m34s             horizontal-pod-autoscaler  New size: 1; reason: All metrics below target
```

2.4.8. Comprendre les conditions d'état du pod autoscaler horizontal à l'aide de la CLI

Vous pouvez utiliser les conditions d'état définies pour déterminer si l'autoscaler de pods horizontaux (HPA) est capable ou non de se mettre à l'échelle et s'il est actuellement limité de quelque manière que ce soit.

Les conditions d'état HPA sont disponibles avec la version **v2** de l'API de mise à l'échelle automatique.

L'APH répond par les états suivants :

- La condition **AbleToScale** indique si HPA est en mesure de récupérer et de mettre à jour les mesures, et si des conditions liées au backoff risquent d'empêcher la mise à l'échelle.
  - Une condition **True** indique que la mise à l'échelle est autorisée.
  - Une condition **False** indique que la mise à l'échelle n'est pas autorisée pour la raison spécifiée.
- La condition **ScalingActive** indique si le HPA est activé (par exemple, le nombre de répliques de la cible n'est pas nul) et s'il est en mesure de calculer les mesures souhaitées.
  - Une condition **True** indique que les mesures fonctionnent correctement.
  - Une condition **False** indique généralement un problème de récupération des données.

- La condition **ScalingLimited** indique que l'échelle souhaitée a été plafonnée par le maximum ou le minimum de l'échelle automatique du pod horizontal.
  - Une condition **True** indique que vous devez augmenter ou diminuer le nombre minimum ou maximum de répliques afin de procéder à une mise à l'échelle.
  - La condition **False** indique que la mise à l'échelle demandée est autorisée.

```
$ oc describe hpa cm-test
```

Exemple de sortie

```
Name:                       cm-test
Namespace:                  prom
Labels:                     <none>
Annotations:                <none>
CreationTimestamp:          Fri, 16 Jun 2017 18:09:22 +0000
Reference:                  ReplicationController/cm-test
Metrics:                    ( current / target )
  "http_requests" on pods:  66m / 500m
Min replicas:               1
Max replicas:               4
ReplicationController pods: 1 current / 1 desired
Conditions: 1
  Type            Status  Reason              Message
  ----            ------  ------              -------
  AbleToScale     True    ReadyForNewScale    the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True    ValidMetricFound    the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False   DesiredWithinRange  the desired replica count is within the acceptable range
Events:
```

1 Les messages d'état de l'autoscaler du pod horizontal.

Voici un exemple d'un module qui ne peut pas être mis à l'échelle :

Exemple de sortie

```
Conditions:
  Type        Status  Reason          Message
  ----        ------  ------          -------
  AbleToScale  False   FailedGetScale  the HPA controller was unable to get the target's current scale: no matches for kind "ReplicationController" in group "apps"
Events:
  Type     Reason          Age               From                    Message
  ----     ------          ----              ----                    -------
  Warning  FailedGetScale  6s (x3 over 36s)  horizontal-pod-autoscaler  no matches for kind "ReplicationController" in group "apps"
```

Voici un exemple de pod qui n'a pas pu obtenir les métriques nécessaires à la mise à l'échelle :

Exemple de sortie

```
Conditions:
  Type                Status    Reason                  Message
  ----                ------    ------                  -------
  AbleToScale         True      SucceededGetScale       the HPA controller was able to get the target's current scale
  ScalingActive       False     FailedGetResourceMetric the HPA was unable to compute the replica count: failed to get cpu utilization: unable to get metrics for resource cpu: no metrics returned from resource metrics API
```

Voici un exemple de pod où l'autoscaling demandé était inférieur aux minimums requis :

Exemple de sortie

```
Conditions:
  Type            Status    Reason              Message
  ----            ------    ------              -------
  AbleToScale     True      ReadyForNewScale    the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True      ValidMetricFound    the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable range
```

2.4.8.1. Visualisation des conditions d'état des pods horizontaux autoscaler à l'aide de la CLI

Vous pouvez visualiser les conditions d'état définies sur un pod par le pod autoscaler horizontal (HPA).

NOTE

Les conditions d'état de l'autoscaler de pods horizontaux sont disponibles avec la version **v2** de l'API d'autoscaling.

Conditions préalables

Pour utiliser les autoscalers de pods horizontaux, votre administrateur de cluster doit avoir correctement configuré les métriques du cluster. Vous pouvez utiliser la commande **oc describe PodMetrics <pod-name>** pour déterminer si les métriques sont configurées. Si les métriques sont configurées, la sortie est similaire à ce qui suit, avec **Cpu** et **Memory** affichés sous **Usage**.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Exemple de sortie

```
Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Memory:  0
  Name:      scheduler
```

```
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Timestamp:             2019-05-23T18:47:56Z
Window:                1m0s
Events:                <none>
```

Procédure

Pour afficher les conditions d'état d'un module, utilisez la commande suivante avec le nom du module :

```
oc describe hpa <pod-name> $ oc describe hpa <pod-name>
```

Par exemple :

```
$ oc describe hpa cm-test
```

Les conditions apparaissent dans le champ **Conditions** de la sortie.

Exemple de sortie

```
Name:                         cm-test
Namespace:                    prom
Labels:                       <none>
Annotations:                  <none>
CreationTimestamp:            Fri, 16 Jun 2017 18:09:22 +0000
Reference:                    ReplicationController/cm-test
Metrics:                      ( current / target )
  "http_requests" on pods:    66m / 500m
Min replicas:                 1
Max replicas:                 4
ReplicationController pods:   1 current / 1 desired
Conditions: (1)
  Type            Status    Reason              Message
  ----            ------    ------              -------
  AbleToScale     True      ReadyForNewScale    the last scale time was sufficiently old as to warrant a new scale
  ScalingActive   True      ValidMetricFound    the HPA was able to successfully calculate a replica count from pods metric http_request
  ScalingLimited  False     DesiredWithinRange  the desired replica count is within the acceptable range
```

2.4.9. Ressources supplémentaires

- Pour plus d'informations sur les contrôleurs de réplication et les contrôleurs de déploiement, voir Comprendre les déploiements et les configurations de déploiement .

- Pour un exemple d'utilisation de HPA, voir Horizontal Pod Autoscaling of Quarkus Application Based on Memory Utilization (mise à l'échelle horizontale de l'application Quarkus en fonction de l'utilisation de la mémoire).

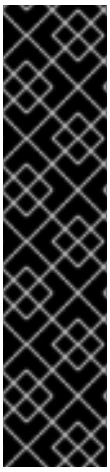
2.5. MISE À L'ÉCHELLE AUTOMATIQUE DES PODS EN FONCTION DE MESURES PERSONNALISÉES

En tant que développeur, vous pouvez utiliser le custom metrics autoscaler pour spécifier comment OpenShift Container Platform doit automatiquement augmenter ou diminuer le nombre de pods pour un déploiement, un stateful set, une ressource personnalisée ou un job basé sur des métriques personnalisées qui ne sont pas basées uniquement sur le CPU ou la mémoire.

L'opérateur Custom Metrics Autoscaler pour Red Hat OpenShift est un opérateur optionnel, basé sur Kubernetes Event Driven Autoscaler (KEDA), qui permet aux charges de travail d'être mises à l'échelle en utilisant des sources de métriques supplémentaires autres que les métriques de pod.

NOTE

L'autoscaler de métriques personnalisées ne prend actuellement en charge que les métriques Prometheus, CPU, mémoire et Apache Kafka.



IMPORTANT

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

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

2.5.1. Notes de mise à jour de Custom Metrics Autoscaler Operator

Les notes de mise à jour de Custom Metrics Autoscaler Operator pour Red Hat Openshift décrivent les nouvelles fonctionnalités et améliorations, les fonctionnalités obsolètes et les problèmes connus.

Le Custom Metrics Autoscaler Operator utilise l'Event Driven Autoscaler (KEDA) basé sur Kubernetes et est construit sur le pod autoscaler horizontal (HPA) d'OpenShift Container Platform.

NOTE

Custom Metrics Autoscaler Operator pour Red Hat OpenShift est fourni en tant que composant installable, avec un cycle de publication distinct de celui de la plate-forme OpenShift Container Platform. La politique de cycle de vie de Red Hat OpenShift Container Platform décrit la compatibilité des versions.

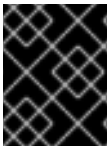
2.5.1.1. Versions prises en charge

Le tableau suivant définit les versions de Custom Metrics Autoscaler Operator pour chaque version d'OpenShift Container Platform.

Version	Version d'OpenShift Container Platform	Disponibilité générale
2.8.2-174	4.12	Avant-première technologique
2.8.2-174	4.11	Avant-première technologique
2.8.2-174	4.10	Avant-première technologique

2.5.1.2. Notes de publication de Custom Metrics Autoscaler Operator 2.8.2-174

Cette version de Custom Metrics Autoscaler Operator 2.8.2-174 fournit de nouvelles fonctionnalités et des corrections de bugs pour l'exécution de l'Operator dans un cluster OpenShift Container Platform. Les composants de Custom Metrics Autoscaler Operator 2.8.2-174 ont été publiés dans RHEA-2023:1683.



IMPORTANT

Le Custom Metrics Autoscaler Operator est actuellement une fonctionnalité en avant-première technologique.

2.5.1.2.1. Nouvelles fonctionnalités et améliorations

2.5.1.2.1.1. Aide à la mise à niveau de l'opérateur

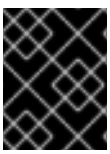
Vous pouvez désormais mettre à niveau une version antérieure de l'opérateur Custom Metrics Autoscaler. Voir "Changer le canal de mise à jour d'un opérateur" dans "Ressources supplémentaires" pour plus d'informations sur la mise à niveau d'un opérateur.

2.5.1.2.1.2. soutien indispensable

Vous pouvez désormais collecter des données sur l'opérateur Custom Metrics Autoscaler et ses composants en utilisant l'outil OpenShift Container Platform **must-gather**. Actuellement, le processus d'utilisation de l'outil **must-gather** avec Custom Metrics Autoscaler est différent de celui des autres opérateurs. Voir "Gathering debugging data in the \N- Additional resources\N" (Collecte de données de débogage dans les ressources supplémentaires) pour plus d'informations.

2.5.1.3. Notes de publication de Custom Metrics Autoscaler Operator 2.8.2

Cette version de Custom Metrics Autoscaler Operator 2.8.2 apporte de nouvelles fonctionnalités et des corrections de bugs pour l'exécution de l'Operator dans un cluster OpenShift Container Platform. Les composants de Custom Metrics Autoscaler Operator 2.8.2 ont été publiés dans la RHSA-2023:1042.



IMPORTANT

Le Custom Metrics Autoscaler Operator est actuellement une fonctionnalité en avant-première technologique.

2.5.1.3.1. Nouvelles fonctionnalités et améliorations

2.5.1.3.1.1. Enregistrement des audits

Vous pouvez désormais rassembler et afficher les journaux d'audit de Custom Metrics Autoscaler Operator et de ses composants associés. Les journaux d'audit sont des ensembles chronologiques d'enregistrements relatifs à la sécurité qui documentent la séquence des activités qui ont affecté le système par des utilisateurs individuels, des administrateurs ou d'autres composants du système.

2.5.1.3.1.2. Faire évoluer les applications en fonction des mesures d'Apache Kafka

Vous pouvez maintenant utiliser le KEDA Apache kafka trigger/scaler pour mettre à l'échelle des déploiements basés sur un sujet Apache Kafka.

IMPORTANT

L'autoscaling basé sur les métriques Apache Kafka est une fonctionnalité Technology Preview (TP) dans toutes les versions TP de Custom Metrics Autoscaler et dans la version Custom Metrics Autoscaler General Availability.

Les fonctionnalités de l'aperçu technologique ne sont pas prises en charge par les accords de niveau de service (SLA) de production de Red Hat et peuvent ne pas être complètes sur le plan fonctionnel. Red Hat ne recommande pas leur utilisation en production.

2.5.1.3.1.3. Mise à l'échelle des applications en fonction des paramètres de l'unité centrale

Vous pouvez désormais utiliser le déclencheur/échelle de CPU de KEDA pour échelonner les déploiements en fonction des métriques de CPU.

2.5.1.3.1.4. Mise à l'échelle des applications en fonction des mesures de la mémoire

Vous pouvez désormais utiliser le déclencheur/échelle de mémoire KEDA pour échelonner les déploiements en fonction des mesures de mémoire.

Ressources supplémentaires

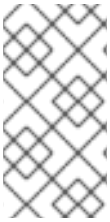
- Changer le canal de mise à jour d'un opérateur
- Collecte de données de débogage
- Configuration de la journalisation des audits
- Comprendre le déclenchement de l'unité centrale
- Comprendre le déclenchement de la mémoire
- Comprendre le déclencheur Kafka

2.5.2. Comprendre l'autoscaler de métriques personnalisées

L'opérateur Autoscaler Custom Metrics fait évoluer vos pods vers le haut ou vers le bas en fonction de mesures externes personnalisées provenant d'applications spécifiques. Vos autres applications continuent d'utiliser d'autres méthodes de mise à l'échelle. Vous configurez *triggers*, également appelé scalers, qui est la source d'événements et de métriques que l'autoscaler de métriques personnalisées

utilise pour déterminer comment procéder à la mise à l'échelle. L'autoscaler de métriques personnalisées utilise une API de métriques pour convertir les métriques externes en une forme utilisable par OpenShift Container Platform. Le custom metrics autoscaler crée un pod autoscaler horizontal (HPA) qui effectue la mise à l'échelle réelle.

Pour utiliser l'autoscaler de métriques personnalisées, vous devez créer un objet **ScaledObject** ou **ScaledJob**, qui est une ressource personnalisée (CR) définissant les métadonnées de mise à l'échelle. Vous spécifiez le déploiement ou le travail à mettre à l'échelle, la source des métriques à mettre à l'échelle (déclencheur) et d'autres paramètres tels que les nombres minimum et maximum de réplicas autorisés.



NOTE

Vous ne pouvez créer qu'un seul objet ou travail mis à l'échelle pour chaque charge de travail que vous souhaitez mettre à l'échelle. Vous ne pouvez pas non plus utiliser un objet ou un travail mis à l'échelle et le pod autoscaler horizontal (HPA) sur la même charge de travail.

Contrairement au HPA, l'autoscaler de métriques personnalisé peut s'adapter à zéro. Si vous définissez la valeur **minReplicaCount** dans le Custom Metrics Autoscaler CR sur **0**, le Custom Metrics Autoscaler met à l'échelle la charge de travail de 1 à 0 réplica ou de 0 à 1 réplica. C'est ce qu'on appelle *activation phase*. Après avoir mis à l'échelle jusqu'à 1 réplica, le HPA prend le contrôle de la mise à l'échelle. C'est ce qu'on appelle *scaling phase*.

Certains déclencheurs permettent de modifier le nombre de répliques mises à l'échelle par l'autoscaler de métriques de cluster. Dans tous les cas, le paramètre permettant de configurer la phase d'activation utilise toujours la même phrase, préfixée par *activation*. Par exemple, si le paramètre **threshold** configure la mise à l'échelle, **activationThreshold** configure l'activation. La configuration des phases d'activation et de mise à l'échelle vous offre une plus grande flexibilité dans vos politiques de mise à l'échelle. Par exemple, vous pouvez configurer une phase d'activation plus élevée pour empêcher la mise à l'échelle vers le haut ou vers le bas si la métrique est particulièrement basse.

La valeur d'activation est plus prioritaire que la valeur de mise à l'échelle en cas de décisions différentes pour chacune d'entre elles. Par exemple, si **threshold** est défini sur **10** et **activationThreshold** sur **50**, si la métrique indique **40**, le scaler n'est pas actif et les pods sont mis à l'échelle à zéro, même si l'APH nécessite 4 instances.

Vous pouvez vérifier que la mise à l'échelle automatique a eu lieu en examinant le nombre de pods dans votre ressource personnalisée ou en examinant les journaux de l'opérateur Autoscaler de Custom Metrics à la recherche de messages similaires à ceux qui suivent :

```
Successfully set ScaleTarget replica count
```

```
Successfully updated ScaleTarget
```

Vous pouvez interrompre temporairement la mise à l'échelle automatique d'un objet de charge de travail, si nécessaire. Par exemple, vous pouvez interrompre la mise à l'échelle automatique avant d'effectuer la maintenance du cluster.

2.5.3. Installation de l'autoscaler de métriques personnalisé

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer l'opérateur Custom Metrics Autoscaler.

L'installation crée cinq CRD :

- **ClusterTriggerAuthentication**
- **KedaController**
- **ScaledJob**
- **ScaledObject**
- **TriggerAuthentication**

Conditions préalables

- Si vous utilisez la communauté KEDA :
  - Désinstallez la communauté KEDA. Vous ne pouvez pas exécuter à la fois KEDA et l'autoscaler de métriques personnalisé sur le même cluster OpenShift Container Platform.
  - Supprimez les définitions de ressources personnalisées de KEDA 1.x en exécutant les commandes suivantes :

```
$ oc delete crd scaledobjects.keda.k8s.io
```

```
$ oc delete crd triggerauthentications.keda.k8s.io
```

Procédure

1. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
2. Choisissez **Custom Metrics Autoscaler** dans la liste des opérateurs disponibles et cliquez sur **Install**.
3. Sur la page **Install Operator**, assurez-vous que l'option **All namespaces on the cluster (default)** est sélectionnée pour **Installation Mode**. Cela permet d'installer l'opérateur dans tous les espaces de noms.
4. Assurez-vous que l'espace de noms **openshift-keda** est sélectionné pour **Installed Namespace**. OpenShift Container Platform crée l'espace de noms, s'il n'est pas présent dans votre cluster.
5. Cliquez sur **Install**.
6. Vérifiez l'installation en listant les composants de Custom Metrics Autoscaler Operator :
   a. Navigate to **Workloads → Pods**.
   b. Sélectionnez le projet **openshift-keda** dans le menu déroulant et vérifiez que le module **custom-metrics-autoscaler-operator-*** est en cours d'exécution.
   c. Naviguez jusqu'à **Workloads → Deployments** pour vérifier que le déploiement **custom-metrics-autoscaler-operator** est en cours.
7. Facultatif : Vérifiez l'installation dans le CLI OpenShift à l'aide des commandes suivantes :

```
$ oc get all -n openshift-keda
```

Le résultat ressemble à ce qui suit :

Exemple de sortie

```
NAME                                             READY   STATUS    RESTARTS   AGE
pod/custom-metrics-autoscaler-operator-5fd8d9ffd8-xt4xp   1/1     Running   0          18m

NAME                                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/custom-metrics-autoscaler-operator   1/1     1            1           18m

NAME                                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/custom-metrics-autoscaler-operator-5fd8d9ffd8   1         1         1       18m
```

8. Installez la ressource personnalisée **KedaController**, qui crée les CRD nécessaires :

   a. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.

   b. Cliquez sur **Custom Metrics Autoscaler**.

   c. Sur la page **Operator Details**, cliquez sur l'onglet **KedaController**.

   d. Dans l'onglet **KedaController**, cliquez sur **Create KedaController** et modifiez le fichier.

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
metadata:
  name: keda
  namespace: openshift-keda
spec:
  watchNamespace: '' (1)
  operator:
    logLevel: info (2)
    logEncoder: console (3)
  metricsServer:
    logLevel: '0' (4)
    auditConfig: (5)
      logFormat: "json"
      logOutputVolumeClaim: "persistentVolumeClaimName"
      policy:
        rules:
        - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
      lifetime:
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
  serviceAccount: {}
```

(1) (1) Spécifie les espaces de noms que l'autoscaler personnalisé doit surveiller. Saisissez les noms dans une liste séparée par des virgules. Omettre ou définir empty pour surveiller tous les espaces de noms. La valeur par défaut est empty.

(2) Spécifie le niveau de verbosité des messages du journal de l'opérateur Autoscaler de Custom Metrics. Les valeurs autorisées sont **debug**, **info**, **error**. La valeur par défaut

Custom Metrics. Les valeurs autorisées sont **debug**, **info**, **error**. La valeur par défaut est **info**.

3 Spécifie le format de consignation des messages de consignation de l'opérateur Autoscaler de Custom Metrics. Les valeurs autorisées sont **console** ou **json**. La valeur par défaut est **console**.

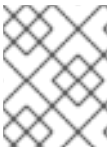
4 Spécifie le niveau de journalisation pour le serveur Autoscaler Metrics de Custom Metrics. Les valeurs autorisées sont **0** pour **info** et **4** ou **debug**. La valeur par défaut est **0**.

5 Active la journalisation des audits pour l'opérateur Custom Metrics Autoscaler et spécifie la politique d'audit à utiliser, comme décrit dans la section "Configuration de la journalisation des audits".

e. Cliquez sur **Create** pour créer le contrôleur KEDAC.

2.5.4. Comprendre les déclencheurs d'autoscaler de métriques personnalisées

Les déclencheurs, également appelés scalers, fournissent les métriques que l'opérateur Custom Metrics Autoscaler utilise pour mettre à l'échelle vos pods.



NOTE

L'autoscaler de métriques personnalisé ne prend actuellement en charge que les déclencheurs Prometheus, CPU, memory et Apache Kafka.

Vous utilisez une ressource personnalisée **ScaledObject** ou **ScaledJob** pour configurer des déclencheurs pour des objets spécifiques, comme décrit dans les sections suivantes.

2.5.4.1. Comprendre le déclencheur Prometheus

Vous pouvez mettre à l'échelle les pods en fonction des métriques Prometheus, qui peuvent utiliser la surveillance OpenShift Container Platform installée ou un serveur Prometheus externe comme source de métriques. Voir *Additional resources* pour plus d'informations sur les configurations requises pour utiliser la surveillance OpenShift Container Platform comme source de métriques.

NOTE

Si Prometheus prend des mesures de l'application que l'autoscaler de mesures personnalisé met à l'échelle, ne définissez pas les répliques minimales à **0** dans la ressource personnalisée. S'il n'y a pas de pods d'application, l'autoscaler de métriques personnalisé n'a pas de métriques à mettre à l'échelle.

Exemple d'objet mis à l'échelle avec une cible Prometheus

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: prom-scaledobject
  namespace: my-namespace
spec:
 ...
  triggers:
```

```
- type: prometheus (1)
  metadata:
    serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092 (2)
    namespace: kedatest (3)
    metricName: http_requests_total (4)
    threshold: '5' (5)
    query: sum(rate(http_requests_total{job="test-app"}[1m])) (6)
    authModes: "basic" (7)
    cortexOrgID: my-org (8)
    ignoreNullValues: false (9)
    unsafeSsl: "false" (10)
```

(1) Spécifie Prometheus comme type de mesureur/déclencheur.

(2) Spécifie l'adresse du serveur Prometheus. Cet exemple utilise la surveillance de OpenShift Container Platform.

(3) Facultatif : Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle. Ce paramètre est obligatoire si la surveillance de OpenShift Container Platform sert de source pour les métriques.

(4) Spécifie le nom permettant d'identifier la métrique dans l'API **external.metrics.k8s.io**. Si vous utilisez plusieurs déclencheurs, tous les noms de métriques doivent être uniques.

(5) Spécifie la valeur pour laquelle la mise à l'échelle doit commencer.

(6) Spécifie la requête Prometheus à utiliser.

(7) Spécifie la méthode d'authentification à utiliser. Les scalers Prometheus prennent en charge l'authentification par support (**bearer**), l'authentification de base (**basic**) ou l'authentification TLS (**tls**). Vous configurez les paramètres d'authentification spécifiques dans un déclencheur d'authentification, comme indiqué dans la section suivante. Si nécessaire, vous pouvez également utiliser un secret.

(8) Facultatif : Transmet l'en-tête **X-Scope-OrgID** au stockage multi-tenant Cortex ou Mimir pour Prometheus. Ce paramètre n'est requis qu'avec le stockage Prometheus multi-tenant, afin d'indiquer les données que Prometheus doit renvoyer.

(9) Facultatif : Spécifie comment le déclencheur doit procéder si la cible Prometheus est perdue.

- Si **true**, le déclencheur continue à fonctionner si la cible Prometheus est perdue. Il s'agit de la valeur par défaut.
- Si **false**, le déclencheur renvoie une erreur si la cible Prometheus est perdue.

(10) Facultatif : Indique si la vérification du certificat doit être ignorée. Par exemple, vous pouvez ignorer la vérification si vous utilisez des certificats auto-signés au niveau du point final Prometheus.

- Si **true**, la vérification du certificat est effectuée.
- Si **false**, la vérification du certificat n'est pas effectuée. Il s'agit de la valeur par défaut.

2.5.4.2. Comprendre le déclenchement de l'unité centrale

Vous pouvez dimensionner les pods en fonction des métriques de CPU. Ce déclencheur utilise les métriques de cluster comme source de métriques.

L'autoscaler de métriques personnalisées met à l'échelle les pods associés à un objet afin de maintenir l'utilisation du CPU que vous avez spécifiée. L'autoscaler augmente ou diminue le nombre de répliques entre les nombres minimum et maximum pour maintenir l'utilisation de l'UC spécifiée dans tous les pods. Le déclencheur de mémoire prend en compte l'utilisation de la mémoire de l'ensemble du module. Si le pod a plusieurs conteneurs, l'utilisation de la mémoire est la somme de tous les conteneurs.

NOTE

- Ce déclencheur ne peut pas être utilisé avec la ressource personnalisée **ScaledJob**.
- Lorsque vous utilisez un déclencheur de mémoire pour mettre un objet à l'échelle, l'objet n'est pas mis à l'échelle sur **0**, même si vous utilisez plusieurs déclencheurs.

Exemple d'objet mis à l'échelle avec une cible CPU

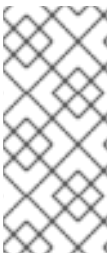
```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: cpu-scaledobject
  namespace: my-namespace
spec:

 ...

  triggers:
  - type: cpu (1)
    metricType: Utilization (2)
    metadata:
      value: "60" (3)
      containerName: "api" (4)
```

1. Spécifie l'unité centrale comme type d'échelle/déclencheur.
2. Spécifie le type de métrique à utiliser, soit **Utilization** ou **AverageValue**.
3. Spécifie la valeur à partir de laquelle les actions de mise à l'échelle doivent être déclenchées :
   - Lorsque l'on utilise **Utilization**, la valeur cible est la moyenne des mesures de la ressource pour tous les pods concernés, représentée en pourcentage de la valeur demandée de la ressource pour les pods.
   - Lorsque l'on utilise **AverageValue**, la valeur cible est la moyenne des mesures de tous les pods concernés.
4. Facultatif. Spécifie un conteneur individuel à mettre à l'échelle, en fonction de l'utilisation de la mémoire de ce conteneur uniquement, plutôt que de l'ensemble du pod. Ici, seul le conteneur nommé **api** doit être mis à l'échelle.

2.5.4.3. Comprendre le déclenchement de la mémoire

Vous pouvez mettre à l'échelle les pods en fonction des métriques de mémoire. Ce déclencheur utilise les métriques de cluster comme source de métriques.

L'autoscaler de métriques personnalisées met à l'échelle les pods associés à un objet afin de maintenir l'utilisation moyenne de la mémoire que vous avez spécifiée. L'autoscaler augmente et diminue le nombre de réplicas entre les nombres minimum et maximum pour maintenir l'utilisation de la mémoire spécifiée dans tous les pods. Le déclencheur de mémoire prend en compte l'utilisation de la mémoire de l'ensemble du module. Si le pod possède plusieurs conteneurs, l'utilisation de la mémoire est la somme de tous les conteneurs.



NOTE

- Ce déclencheur ne peut pas être utilisé avec la ressource personnalisée **ScaledJob**.
- Lorsque vous utilisez un déclencheur de mémoire pour mettre un objet à l'échelle, l'objet n'est pas mis à l'échelle sur **0**, même si vous utilisez plusieurs déclencheurs.

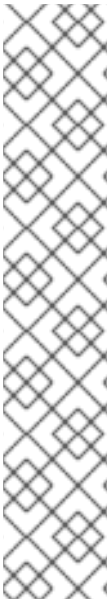
Exemple d'objet mis à l'échelle avec une cible de mémoire

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: memory-scaledobject
  namespace: my-namespace
spec:

 ...

  triggers:
  - type: memory 1
    metricType: Utilization 2
    metadata:
      value: "60" 3
      containerName: "api" 4
```

1. Spécifie la mémoire comme type d'échelle/déclencheur.
2. Spécifie le type de métrique à utiliser, soit **Utilization** ou **AverageValue**.
3. Spécifie la valeur pour laquelle des actions de mise à l'échelle doivent être déclenchées :
   - Lorsque l'on utilise **Utilization**, la valeur cible est la moyenne des mesures de la ressource pour tous les pods concernés, représentée en pourcentage de la valeur demandée de la ressource pour les pods.
   - Lorsque l'on utilise **AverageValue**, la valeur cible est la moyenne des mesures de tous les pods concernés.
4. Facultatif. Spécifie un conteneur individuel à mettre à l'échelle, en fonction de l'utilisation de la mémoire de ce conteneur uniquement, plutôt que de l'ensemble du pod. Ici, seul le conteneur nommé **api** doit être mis à l'échelle.

2.5.4.4. Comprendre le déclencheur Kafka

Vous pouvez mettre à l'échelle les pods en fonction d'un sujet Apache Kafka ou d'autres services qui prennent en charge le protocole Kafka. L'autoscaler de métriques personnalisées n'augmente pas le nombre de partitions Kafka, sauf si vous définissez le paramètre **allowIdleConsumers** sur **true** dans l'objet ou le travail mis à l'échelle.



NOTE

Si le nombre de groupes de consommateurs dépasse le nombre de partitions d'un thème, les groupes de consommateurs supplémentaires restent inactifs.

Pour éviter cela, le nombre de répliques est limité par défaut :

- Le nombre de partitions sur un sujet, si un sujet est spécifié.
- Le nombre de partitions de tous les thèmes du groupe de consommateurs, si aucun thème n'est spécifié.
- Le site **maxReplicaCount** spécifié dans l'objet mis à l'échelle ou le travail mis à l'échelle CR.

Vous pouvez utiliser le paramètre **allowIdleConsumers** pour désactiver ces comportements par défaut.

Exemple d'objet mis à l'échelle avec une cible Kafka

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: kafka-scaledobject
  namespace: my-namespace
spec:
 ...
  triggers:
  - type: kafka 1
    metadata:
      topic: my-topic 2
      bootstrapServers: my-cluster-kafka-bootstrap.openshift-operators.svc:9092 3
      consumerGroup: my-group 4
      lagThreshold: '10' 5
      activationLagThreshold 6
      offsetResetPolicy: 'latest' 7
      allowIdleConsumers: true 8
      scaleToZeroOnInvalidOffset: false 9
      excludePersistentLag: false 10
      version: 1.0.0 11
      partitionLimitation: '1,2,10-20,31' 12
```

1. Spécifie Kafka comme type de scaler/déclencheur.
2. Spécifie le nom du sujet Kafka sur lequel Kafka traite le décalage.

3. Spécifie une liste de brokers Kafka séparés par des virgules à laquelle se connecter.

4. Spécifie le nom du groupe de consommateurs Kafka utilisé pour vérifier le décalage sur le sujet et traiter le lag correspondant.

5. Facultatif : Spécifie la valeur cible moyenne pour déclencher les actions de mise à l'échelle. La valeur par défaut est **5**.

6. Facultatif : Spécifie la valeur cible pour la phase d'activation.

7. Facultatif : Spécifie la politique de réinitialisation du décalage Kafka pour le consommateur Kafka. Les valeurs disponibles sont : **latest** et **earliest**. La valeur par défaut est **latest**.

8. Facultatif : Indique si le nombre de répliques Kafka peut dépasser le nombre de partitions sur un sujet.

    - Si **true**, le nombre de répliques Kafka peut dépasser le nombre de partitions sur un sujet. Cela permet d'avoir des consommateurs Kafka inactifs.
    - Si **false**, le nombre de répliques Kafka ne peut pas dépasser le nombre de partitions sur un sujet. Il s'agit de la valeur par défaut.

9. Spécifie le comportement du déclencheur lorsqu'une partition Kafka n'a pas de décalage valide.

    - Si **true**, les consommateurs sont ramenés à zéro pour cette partition.
    - Si **false**, le scaler conserve un seul consommateur pour cette partition. Il s'agit de la valeur par défaut.

10. Facultatif : Spécifie si le déclencheur inclut ou exclut le décalage de partition pour les partitions dont le décalage actuel est le même que le décalage actuel du cycle d'interrogation précédent.

    - Si **true**, le mesureur exclut le décalage de partition dans ces partitions.
    - Si **false**, le déclencheur inclut tous les décalages de consommation dans toutes les partitions. Il s'agit de la valeur par défaut.

11. Facultatif : Spécifie la version de vos brokers Kafka. La valeur par défaut est **1.0.0**.

12. Facultatif : Spécifie une liste d'ID de partitions séparées par des virgules pour étendre la mise à l'échelle. Si cette option est activée, seuls les ID répertoriés sont pris en compte lors du calcul du décalage. Par défaut, toutes les partitions sont prises en compte.

Ressources supplémentaires

- Configurer l'autoscaler de métriques personnalisé pour utiliser la surveillance de OpenShift Container Platform

2.5.5. Comprendre les métriques personnalisées Autoscaler trigger authentications

Une authentification de déclenchement vous permet d'inclure des informations d'authentification dans un objet ou un travail de mise à l'échelle qui peut être utilisé par les conteneurs associés. Vous pouvez utiliser des authentifications de déclenchement pour transmettre des secrets OpenShift Container Platform, des mécanismes d'authentification de pods natifs de la plateforme, des variables d'environnement, etc.

Vous définissez un objet **TriggerAuthentication** dans le même espace de noms que l'objet que vous souhaitez mettre à l'échelle. Cette authentification de déclenchement ne peut être utilisée que par les objets de cet espace de noms.

Pour partager des informations d'identification entre des objets de plusieurs espaces de noms, vous pouvez également créer un objet **ClusterTriggerAuthentication** qui peut être utilisé dans tous les espaces de noms.

Les authentifications par déclenchement et les authentifications par déclenchement en grappe utilisent la même configuration. Toutefois, l'authentification par déclenchement de grappe nécessite un paramètre supplémentaire **kind** dans la référence d'authentification de l'objet mis à l'échelle.

Exemple : déclencher l'authentification avec un secret

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: secret-triggerauthentication
  namespace: my-namespace (1)
spec:
  secretTargetRef: (2)
  - parameter: user-name (3)
    name: my-secret (4)
    key: USER_NAME (5)
  - parameter: password
    name: my-secret
    key: USER_PASSWORD
```

1. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.
2. Spécifie que l'authentification de ce déclencheur utilise un secret pour l'autorisation.
3. Spécifie le paramètre d'authentification à fournir en utilisant le secret.
4. Spécifie le nom du secret à utiliser.
5. Spécifie la clé du secret à utiliser avec le paramètre spécifié.

Exemple d'authentification par déclenchement de cluster avec un secret

```
kind: ClusterTriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata: (1)
  name: secret-cluster-triggerauthentication
spec:
  secretTargetRef: (2)
  - parameter: user-name (3)
    name: secret-name (4)
    key: USER_NAME (5)
  - parameter: user-password
    name: secret-name
    key: USER_PASSWORD
```

1. Notez qu'aucun espace de noms n'est utilisé avec un déclencheur d'authentification en grappe.
2. Spécifie que l'authentification de ce déclencheur utilise un secret pour l'autorisation.
3. Spécifie le paramètre d'authentification à fournir en utilisant le secret.
4. Spécifie le nom du secret à utiliser.
5. Spécifie la clé du secret à utiliser avec le paramètre spécifié.

Exemple de déclenchement de l'authentification avec un jeton

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: token-triggerauthentication
  namespace: my-namespace (1)
spec:
  secretTargetRef: (2)
  - parameter: bearerToken (3)
    name: my-token-2vzfq (4)
    key: token (5)
  - parameter: ca
    name: my-token-2vzfq
    key: ca.crt
```

1. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.
2. Spécifie que l'authentification de ce déclencheur utilise un secret pour l'autorisation.
3. Spécifie le paramètre d'authentification à fournir en utilisant le jeton.
4. Spécifie le nom du jeton à utiliser.
5. Spécifie la clé du jeton à utiliser avec le paramètre spécifié.

Exemple de déclenchement de l'authentification à l'aide d'une variable d'environnement

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: env-var-triggerauthentication
  namespace: my-namespace (1)
spec:
  env: (2)
  - parameter: access_key (3)
    name: ACCESS_KEY (4)
    containerName: my-container (5)
```

1. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.
2. Spécifie que ce déclencheur d'authentification utilise des variables d'environnement pour l'autorisation.

3. Spécifiez le paramètre à définir avec cette variable.
4. Indiquez le nom de la variable d'environnement.
5. Facultatif : Indiquez un conteneur qui nécessite une authentification. Le conteneur doit se trouver dans la même ressource que celle référencée par **scaleTargetRef** dans l'objet mis à l'échelle.

Exemple d'authentification par déclenchement avec des fournisseurs d'authentification de pods

```
kind: TriggerAuthentication
apiVersion: keda.sh/v1alpha1
metadata:
  name: pod-id-triggerauthentication
  namespace: my-namespace (1)
spec:
  podIdentity: (2)
    provider: aws-eks (3)
```

1. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.
2. Spécifie que ce déclencheur d'authentification utilise une méthode d'authentification podale native pour l'autorisation.
3. Spécifie l'identité d'un pod. Les valeurs prises en charge sont **none**, **azure**, **aws-eks** ou **aws-kiam**. La valeur par défaut est **none**.

Ressources supplémentaires

- Pour plus d'informations sur les secrets de OpenShift Container Platform, voir Fournir des données sensibles aux pods.

2.5.5.1. Utilisation des authentifications par déclenchement

Vous utilisez les authentifications de déclenchement et les authentifications de déclenchement de cluster en utilisant une ressource personnalisée pour créer l'authentification, puis en ajoutant une référence à un objet ou à un travail mis à l'échelle.

Conditions préalables

- L'opérateur Custom Metrics Autoscaler doit être installé.
- Si vous utilisez un secret, l'objet **Secret** doit exister, par exemple :

  #### Exemple de secret

  ```
  apiVersion: v1
  kind: Secret
  metadata:
    name: my-secret
  ```

```
data:
  user-name: <base64_USER_NAME>
  password: <base64_USER_PASSWORD>
```

Procédure

1. Créer l'objet **TriggerAuthentication** ou **ClusterTriggerAuthentication**.
   a. Créer un fichier YAML qui définit l'objet :

      Exemple : déclencher l'authentification avec un secret

      ```
      kind: TriggerAuthentication
      apiVersion: keda.sh/v1alpha1
      metadata:
        name: prom-triggerauthentication
        namespace: my-namespace
      spec:
        secretTargetRef:
        - parameter: user-name
          name: my-secret
          key: USER_NAME
        - parameter: password
          name: my-secret
          key: USER_PASSWORD
      ```

   b. Créer l'objet **TriggerAuthentication**:

      ```
      oc create -f <nom-de-fichier>.yaml
      ```

2. Créer ou modifier un fichier YAML **ScaledObject**:

   Exemple d'objet mis à l'échelle

   ```
   apiVersion: keda.sh/v1alpha1
   kind: ScaledObject
   metadata:
     name: scaledobject
     namespace: my-namespace
   spec:
     scaleTargetRef:
       name: example-deployment
     maxReplicaCount: 100
     minReplicaCount: 0
     pollingInterval: 30
     triggers:
     - authenticationRef:
       type: prometheus
       metadata:
         serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
         namespace: kedatest # replace <NAMESPACE>
         metricName: http_requests_total
         threshold: '5'
         query: sum(rate(http_requests_total{job="test-app"}[1m]))
         authModes: "basic"
   ```

```
- authenticationRef: 1
    name: prom-triggerauthentication
  metadata:
    name: prom-triggerauthentication
  type: object
- authenticationRef: 2
    name: prom-cluster-triggerauthentication
    kind: ClusterTriggerAuthentication
  metadata:
    name: prom-cluster-triggerauthentication
  type: object
```

1 Facultatif : Spécifiez un déclencheur d'authentification.

2 Facultatif : Spécifiez une authentification de déclenchement de cluster. Vous devez inclure le paramètre **kind: ClusterTriggerAuthentication**.

NOTE

Il n'est pas nécessaire de spécifier à la fois une authentification de déclenchement de l'espace de noms et une authentification de déclenchement du cluster.

3. Créer l'objet. Par exemple :

```
$ oc apply -f <nom-de-fichier>
```

2.5.6. Configurer l'autoscaler de métriques personnalisé pour utiliser la surveillance de OpenShift Container Platform

Vous pouvez utiliser la surveillance Prometheus de OpenShift Container Platform installée comme source pour les métriques utilisées par l'autoscaler de métriques personnalisé. Cependant, vous devez effectuer quelques configurations supplémentaires.

NOTE

Ces étapes ne sont pas nécessaires pour une source Prometheus externe.

Vous devez effectuer les tâches suivantes, décrites dans cette section :

- Créer un compte de service pour obtenir un jeton.
- Créer un rôle.
- Ajoutez ce rôle au compte de service.
- Référencer le jeton dans l'objet d'authentification du déclencheur utilisé par Prometheus.

Conditions préalables

- La surveillance de OpenShift Container Platform doit être installée.

- La surveillance des charges de travail définies par l'utilisateur doit être activée dans la surveillance de OpenShift Container Platform, comme décrit dans la section **Creating a user-defined workload monitoring config map**.
- L'opérateur Custom Metrics Autoscaler doit être installé.

Procédure

1. Passez au projet contenant l'objet que vous souhaitez mettre à l'échelle :

```
$ oc project my-project
```

2. Utilisez la commande suivante pour créer un compte de service, si votre cluster n'en possède pas :

```
oc create serviceaccount <service_account> $ oc create serviceaccount <service_account>
```

où :

-YRFFGUNA_compte_de_service>

Spécifie le nom du compte de service.

3. Utilisez la commande suivante pour localiser le jeton attribué au compte de service :

```
oc describe serviceaccount <service_account> $ oc describe serviceaccount
<service_account>
```

où :

-YRFFGUNA_compte_de_service>

Spécifie le nom du compte de service.

Exemple de sortie

```
Name:                thanos
Namespace:           my-project
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-nnwgj
Mountable secrets:   thanos-dockercfg-nnwgj
Tokens:              thanos-token-9g4n5 (1)
Events:              <none>
```

(1) Utilisez ce jeton dans l'authentification du déclencheur.

4. Créer un déclencheur d'authentification avec le jeton de compte de service :

a. Créez un fichier YAML similaire au suivant :

```
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-trigger-auth-prometheus
```

```
spec:
  secretTargetRef: (1)
  - parameter: bearerToken (2)
    name: thanos-token-9g4n5 (3)
    key: token (4)
  - parameter: ca
    name: thanos-token-9g4n5
    key: ca.crt
```

(1) Spécifie que cet objet utilise un secret pour l'autorisation.

(2) Spécifie le paramètre d'authentification à fournir en utilisant le jeton.

(3) Spécifie le nom du jeton à utiliser.

(4) Spécifie la clé du jeton à utiliser avec le paramètre spécifié.

b. Créer l'objet CR :

```
oc create -f <nom-de-fichier>.yaml
```

5. Créer un rôle pour lire les métriques de Thanos :

a. Créez un fichier YAML avec les paramètres suivants :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
- apiGroups:
  - metrics.k8s.io
  resources:
  - pods
  - nodes
  verbs:
  - get
  - list
  - watch
```

b. Créer l'objet CR :

```
oc create -f <nom-de-fichier>.yaml
```

6. Créer un lien de rôle pour la lecture des métriques de Thanos :

a. Créez un fichier YAML similaire au suivant :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: thanos-metrics-reader (1)
  namespace: my-project (2)
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: thanos-metrics-reader
subjects:
- kind: ServiceAccount
  name: thanos (3)
  namespace: my-project (4)
```

1. Spécifie le nom du rôle que vous avez créé.
2. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.
3. Spécifie le nom du compte de service à lier au rôle.
4. Spécifie l'espace de noms de l'objet que vous souhaitez mettre à l'échelle.

b. Créer l'objet CR :

```
oc create -f <nom-de-fichier>.yaml
```

Vous pouvez maintenant déployer un objet ou un travail mis à l'échelle pour activer la mise à l'échelle automatique de votre application, comme décrit dans les sections suivantes. Pour utiliser la surveillance d'OpenShift Container Platform comme source, dans le déclencheur ou le scaler, spécifiez le type **prometheus** et utilisez **https://thanos-querier.openshift-monitoring.svc.cluster.local:9092** comme **serverAddress**.

Ressources supplémentaires

- Pour plus d'informations sur l'activation de la surveillance des charges de travail définies par l'utilisateur, voir Création d'une carte de configuration pour la surveillance des charges de travail définies par l'utilisateur.

2.5.7. Mise en pause de l'autoscaler de métriques personnalisées pour une charge de travail

Vous pouvez interrompre la mise à l'échelle automatique d'une charge de travail, si nécessaire, en ajoutant l'annotation **autoscaling.keda.sh/paused-replicas** à l'autoscaler de métriques personnalisé pour cette charge de travail. L'autoscaler de métriques personnalisées met à l'échelle les réplicas pour cette charge de travail jusqu'à la valeur spécifiée et interrompt l'autoscaling jusqu'à ce que l'annotation soit supprimée.

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4"
...
```

Pour redémarrer l'autoscaling, modifiez le CR **ScaledObject** pour supprimer l'annotation.

Par exemple, vous pouvez vouloir mettre en pause l'autoscaling avant d'effectuer la maintenance du cluster ou pour éviter la pénurie de ressources en supprimant les charges de travail non critiques.

Procédure

1. Utilisez la commande suivante pour éditer le **ScaledObject** CR pour votre charge de travail :

```
$ oc edit ScaledObject scaledobject
```

2. Ajouter l'annotation **autoscaling.keda.sh/paused-replicas** avec n'importe quelle valeur :

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
    autoscaling.keda.sh/paused-replicas: "4" 1
  creationTimestamp: "2023-02-08T14:41:01Z"
  generation: 1
  name: scaledobject
  namespace: my-project
  resourceVersion: "65729"
  uid: f5aec682-acdf-4232-a783-58b5b82f5dd0
```

1 Spécifie que l'opérateur de mise à l'échelle automatique des métriques personnalisées doit mettre à l'échelle les répliques jusqu'à la valeur spécifiée et arrêter la mise à l'échelle automatique.

2.5.8. Configuration de la journalisation des audits

Vous pouvez rassembler des journaux d'audit, qui sont un ensemble chronologique d'enregistrements relatifs à la sécurité documentant la séquence des activités qui ont affecté le système par des utilisateurs individuels, des administrateurs ou d'autres composants du système.

Par exemple, les journaux d'audit peuvent vous aider à comprendre d'où provient une demande d'autoscaling. Il s'agit d'une information clé lorsque les backends sont surchargés par des requêtes d'autoscaling effectuées par des applications utilisateur et que vous devez déterminer quelle est l'application qui pose problème. Vous pouvez configurer l'audit pour le Custom Metrics Autoscaler Operator en modifiant la ressource personnalisée **KedaController**. Les journaux sont envoyés à un fichier journal d'audit sur un volume sécurisé à l'aide d'une revendication de volume persistant dans le CR **KedaController**.

Conditions préalables

- L'opérateur Custom Metrics Autoscaler doit être installé.

Procédure

1. Modifiez la ressource personnalisée **KedaController** pour ajouter la strophe **auditConfig**:

```
kind: KedaController
apiVersion: keda.sh/v1alpha1
```

```
metadata:
  name: keda
  namespace: openshift-keda
spec:
 ...
  metricsServer:
 ...
    auditConfig:
      logFormat: "json" 1
      logOutputVolumeClaim: "pvc-audit-log" 2
      policy:
        rules: 3
        - level: Metadata
        omitStages: "RequestReceived" 4
        omitManagedFields: false 5
      lifetime: 6
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
```

1. Spécifie le format de sortie du journal d'audit, soit **legacy** ou **json**.

2. Spécifie une revendication de volume persistant existante pour le stockage des données de journalisation. Toutes les demandes adressées au serveur API sont consignées dans cette revendication de volume persistant. Si vous laissez ce champ vide, les données du journal sont envoyées à stdout.

3. Spécifie quels événements doivent être enregistrés et quelles données doivent être incluses :

   - **None**: Ne pas enregistrer les événements.
   - **Metadata**: N'enregistrez que les métadonnées de la demande, telles que l'utilisateur, l'horodatage, etc. Ne pas enregistrer le texte de la demande et le texte de la réponse. Il s'agit de la valeur par défaut.
   - **Request**: Enregistrer uniquement les métadonnées et le texte de la demande, mais pas le texte de la réponse. Cette option ne s'applique pas aux demandes ne portant pas sur des ressources.
   - **RequestResponse**: Métadonnées de l'événement, texte de la demande et texte de la réponse. Cette option ne s'applique pas aux demandes ne portant pas sur des ressources.

4. Spécifie les étapes pour lesquelles aucun événement n'est créé.

5. Indique s'il faut omettre les champs gérés des corps de demande et de réponse dans le journal d'audit de l'API, soit **true** pour omettre les champs, soit **false** pour les inclure.

6. Spécifie la taille et la durée de vie des journaux d'audit.

   - **maxAge**: Nombre maximal de jours pendant lesquels les fichiers journaux d'audit doivent être conservés, sur la base de l'horodatage encodé dans leur nom de fichier.
   - **maxBackup**: Nombre maximal de fichiers journaux d'audit à conserver. La valeur **0** permet de conserver tous les fichiers journaux d'audit.

- **maxSize**: Taille maximale en mégaoctets d'un fichier journal d'audit avant qu'il ne fasse l'objet d'une rotation.

Vérification

1. Consulter directement le fichier journal de l'audit :

   a. Obtenir le nom du pod **keda-metrics-apiserver-***:

   ```
   oc get pod -n openshift-keda
   ```

   Exemple de sortie

   ```
   NAME                                      READY   STATUS    RESTARTS   AGE
   custom-metrics-autoscaler-operator-5cb44cd75d-9v4lv   1/1     Running   0          8m20s
   keda-metrics-apiserver-65c7cc44fd-rrl4r               1/1     Running   0          2m55s
   keda-operator-776cbb6768-zpj5b                        1/1     Running   0          2m55s
   ```

   b. Affichez les données du journal à l'aide d'une commande similaire à la suivante :

   ```
   oc logs keda-metrics-apiserver-<hash>|grep -i metadata 1
   ```

   1 Facultatif : vous pouvez utiliser la commande **grep** pour spécifier le niveau de journal à afficher : **Metadata**, **Request**, **RequestResponse**.

   Par exemple :

   ```
   $ oc logs keda-metrics-apiserver-65c7cc44fd-rrl4r|grep -i metadata
   ```

   Exemple de sortie

   ```
   ...
   {"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"4c81d41b-
   3dab-4675-90ce-
   20b87ce24013","stage":"ResponseComplete","requestURI":"/healthz","verb":"get","user":
   {"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":
   ["10.131.0.1"],"userAgent":"kube-probe/1.26","responseStatus":{"metadata":
   {},"code":200},"requestReceivedTimestamp":"2023-02-
   16T13:00:03.554567Z","stageTimestamp":"2023-02-
   16T13:00:03.555032Z","annotations":
   {"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
   ...
   ```

2. Vous pouvez également consulter un journal spécifique :

   a. Utilisez une commande similaire à la suivante pour vous connecter au pod **keda-metrics-apiserver-***:

   ```
   oc rsh pod/keda-metrics-apiserver-<hash> -n openshift-keda
   ```

   Par exemple :

   ```
   ```

```
$ oc rsh pod/keda-metrics-apiserver-65c7cc44fd-rrl4r -n openshift-keda
```

b. Allez dans le répertoire **/var/audit-policy/**:

```
sh-4.4$ cd /var/audit-policy/
```

c. Liste des journaux disponibles :

```
sh-4.4$ ls
```

Exemple de sortie

```
log-2023.02.17-14:50  policy.yaml
```

d. Consulter le journal, si nécessaire :

```
sh-4.4$ cat <log_name>/<pvc_name>|grep -i <log_level>
```
(1)

(1) Facultatif : vous pouvez utiliser la commande **grep** pour spécifier le niveau de journal à afficher : **Metadata**, **Request**, **RequestResponse**.

Par exemple :

```
sh-4.4$ cat log-2023.02.17-14:50/pvc-audit-log|grep -i Request
```

Exemple de sortie

```
 ...
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Request","auditID":"63e7f68c-04ec-4f4d-8749-bf1656572a41","stage":"ResponseComplete","requestURI":"/openapi/v2","verb":"get","user":{"username":"system:aggregator","groups":["system:authenticated"]},"sourceIPs":["10.128.0.1"],"responseStatus":{"metadata":{},"code":304},"requestReceivedTimestamp":"2023-02-17T13:12:55.035478Z","stageTimestamp":"2023-02-17T13:12:55.038346Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:discovery\" of ClusterRole \"system:discovery\" to Group \"system:authenticated\""}}
 ...
```

Ressources supplémentaires

- Configuration de la journalisation des audits

2.5.9. Collecte de données de débogage

Lorsque vous ouvrez un dossier d'assistance, il est utile de fournir des informations de débogage sur votre cluster à l'équipe d'assistance de Red Hat.

Il est recommandé de fournir les informations suivantes :

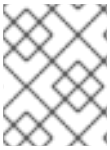
- Données recueillies à l'aide de l'outil **must-gather**.
- L'identifiant unique du cluster.

Vous pouvez utiliser l'outil **must-gather** pour collecter des données sur l'opérateur Custom Metrics Autoscaler et ses composants, notamment

- L'espace de noms **openshift-keda** et ses objets enfants.
- Les objets d'installation Custom Metric Autoscaler Operator.
- Les objets Custom Metric Autoscaler Operator CRD.

La commande suivante exécute l'outil **must-gather** pour l'opérateur Custom Metrics Autoscaler :

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
-o jsonpath='{.status.channels[?(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```



NOTE

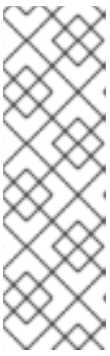
La commande standard d'OpenShift Container Platform **must-gather**, **oc adm must-gather**, ne collecte pas les données Custom Metrics Autoscaler Operator.

Conditions préalables

- Accès au cluster en tant qu'utilisateur ayant le rôle **cluster-admin**.
- L'OpenShift Container Platform CLI (**oc**) est installé.

Procédure

1. Accédez au répertoire dans lequel vous souhaitez stocker les données **must-gather**.



NOTE

Si votre cluster utilise un réseau restreint, vous devez prendre des mesures supplémentaires. Si votre registre miroir dispose d'une autorité de certification approuvée, vous devez d'abord ajouter cette dernière au cluster. Pour tous les clusters sur des réseaux restreints, vous devez importer l'image par défaut **must-gather** en tant que flux d'images en exécutant la commande suivante.

```
$ oc import-image is/must-gather -n openshift
```

2. Effectuez l'une des opérations suivantes :
   - Pour obtenir uniquement les données Custom Metrics Autoscaler Operator **must-gather**, utilisez la commande suivante :

```
$ oc adm must-gather --image="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator \
-n openshift-marketplace \
```

```
-o jsonpath='{.status.channels[?
(@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
```

L'image personnalisée de la commande **must-gather** est tirée directement des manifestes du paquet Operator, de sorte qu'elle fonctionne sur n'importe quel cluster où l'opérateur Custom Metric Autoscaler est disponible.

- Pour collecter les données par défaut **must-gather** en plus des informations de l'opérateur Autoscaler de métriques personnalisées :

  a. Utilisez la commande suivante pour obtenir l'image Custom Metrics Autoscaler Operator et la définir comme variable d'environnement :

  ```
  $ IMAGE="$(oc get packagemanifests openshift-custom-metrics-autoscaler-operator
  \
    -n openshift-marketplace \
    -o jsonpath='{.status.channels[?
  (@.name=="stable")].currentCSVDesc.annotations.containerImage}')"
  ```

  b. Utilisez le site **oc adm must-gather** avec l'image Custom Metrics Autoscaler Operator :

  ```
  $ oc adm must-gather --image-stream=openshift/must-gather --image=${IMAGE}
  ```

Exemple 2.1. Exemple de sortie de collecte obligatoire pour le Custom Metric Autoscaler :

```
└── openshift-keda
    ├── apps
    │   ├── daemonsets.yaml
    │   ├── deployments.yaml
    │   ├── replicasets.yaml
    │   └── statefulsets.yaml
    ├── apps.openshift.io
    │   └── deploymentconfigs.yaml
    ├── autoscaling
    │   └── horizontalpodautoscalers.yaml
    ├── batch
    │   ├── cronjobs.yaml
    │   └── jobs.yaml
    ├── build.openshift.io
    │   ├── buildconfigs.yaml
    │   └── builds.yaml
    ├── core
    │   ├── configmaps.yaml
    │   ├── endpoints.yaml
    │   ├── events.yaml
    │   ├── persistentvolumeclaims.yaml
    │   ├── pods.yaml
    │   ├── replicationcontrollers.yaml
    │   ├── secrets.yaml
    │   └── services.yaml
    ├── discovery.k8s.io
    │   └── endpointslices.yaml
    ├── image.openshift.io
    │   └── imagestreams.yaml
```

```
├── k8s.ovn.org
│   ├── egressfirewalls.yaml
│   └── egressqoses.yaml
├── keda.sh
│   ├── kedacontrollers
│   │   └── keda.yaml
│   ├── scaledobjects
│   │   └── example-scaledobject.yaml
│   └── triggerauthentications
│       └── example-triggerauthentication.yaml
├── monitoring.coreos.com
│   └── servicemonitors.yaml
├── networking.k8s.io
│   └── networkpolicies.yaml
├── openshift-keda.yaml
├── pods
│   ├── custom-metrics-autoscaler-operator-58bd9f458-ptgwx
│   │   ├── custom-metrics-autoscaler-operator
│   │   │   └── custom-metrics-autoscaler-operator
│   │   │       └── logs
│   │   │           ├── current.log
│   │   │           ├── previous.insecure.log
│   │   │           └── previous.log
│   │   └── custom-metrics-autoscaler-operator-58bd9f458-ptgwx.yaml
│   ├── custom-metrics-autoscaler-operator-58bd9f458-thbsh
│   │   └── custom-metrics-autoscaler-operator
│   │       └── custom-metrics-autoscaler-operator
│   │           └── logs
│   ├── keda-metrics-apiserver-65c7cc44fd-6wq4g
│   │   ├── keda-metrics-apiserver
│   │   │   └── keda-metrics-apiserver
│   │   │       └── logs
│   │   │           ├── current.log
│   │   │           ├── previous.insecure.log
│   │   │           └── previous.log
│   │   └── keda-metrics-apiserver-65c7cc44fd-6wq4g.yaml
│   └── keda-operator-776cbb6768-fb6m5
│       ├── keda-operator
│       │   └── keda-operator
│       │       └── logs
│       │           ├── current.log
│       │           ├── previous.insecure.log
│       │           └── previous.log
│       └── keda-operator-776cbb6768-fb6m5.yaml
├── policy
│   └── poddisruptionbudgets.yaml
└── route.openshift.io
    └── routes.yaml
```

3. Créez un fichier compressé à partir du répertoire **must-gather** qui a été créé dans votre répertoire de travail. Par exemple, sur un ordinateur utilisant un système d'exploitation Linux, exécutez la commande suivante :

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

1 Remplacez **must-gather-local.5421342344627712289/** par le nom du répertoire.

4. Joignez le fichier compressé à votre demande d'assistance sur le portail client de Red Hat .

Ressources supplémentaires

- À propos de l'outil de collecte obligatoire
- Obtention de l'identifiant du cluster

2.5.10. Accès aux métriques de performance

L'opérateur Autoscaler Custom Metrics expose des mesures prêtes à l'emploi qu'il extrait du composant de surveillance du cluster. Vous pouvez interroger les mesures à l'aide du Prometheus Query Language (PromQL) pour analyser et diagnostiquer les problèmes. Toutes les mesures sont réinitialisées lorsque le pod contrôleur redémarre.

Vous pouvez accéder aux métriques et exécuter des requêtes en utilisant la console web d'OpenShift Container Platform.

Procédure

1. Sélectionnez la perspective **Administrator** dans la console web de OpenShift Container Platform.
2. Sélectionnez **Observe → Metrics**.
3. Pour créer une requête personnalisée, ajoutez votre requête PromQL au champ **Expression**.
4. Pour ajouter plusieurs requêtes, sélectionnez **Add Query**.

2.5.10.1. Métriques fournies

Le Custom Metrics Autoscaler Operator expose les métriques suivantes, que vous pouvez visualiser en utilisant la console web d'OpenShift Container Platform.

Tableau 2.2. Métriques personnalisées de l'opérateur Autoscaler

Nom de la métrique	Description
keda_scaler_activity	Indique si l'analyseur est actif ou inactif. Une valeur de **1** indique que le mesureur est actif ; une valeur de **0** indique que le mesureur est inactif.
keda_scaler_metrics_value	Valeur actuelle de la métrique de chaque scaler, utilisée par l'Horizontal Pod Autoscaler (HPA) dans le calcul de la moyenne cible.
keda_scaler_metrics_latency	Le temps de latence pour récupérer la métrique actuelle de chaque mesureur.
keda_scaler_errors	Le nombre d'erreurs qui se sont produites pour chaque mesureur.
keda_scaler_errors_total	Le nombre total d'erreurs rencontrées pour tous les scalers.

Nom de la métrique	Description
keda_scaled_object_errors	Le nombre d'erreurs qui se sont produites pour chaque objet mis à l'échelle.
keda_resource_totals	Nombre total de ressources personnalisées Custom Metrics Autoscaler dans chaque espace de noms pour chaque type de ressource personnalisée.
keda_trigger_totals	Le nombre total de déclencheurs par type de déclencheur.

Métriques personnalisées Métriques Autoscaler Admission webhook

Le webhook Custom Metrics Autoscaler Admission expose également les métriques Prometheus suivantes.

Nom de la métrique	Description
keda_scaled_object_validation_total	Le nombre de validations d'objets mis à l'échelle.
keda_scaled_object_validation_errors	Le nombre d'erreurs de validation.

2.5.11. Comprendre comment ajouter des autoscalers de métriques personnalisés

Pour ajouter un autoscaler de métriques personnalisé, créez une ressource personnalisée **ScaledObject** pour un déploiement, un ensemble avec état ou une ressource personnalisée. Créez une ressource personnalisée **ScaledJob** pour un travail.

Vous ne pouvez créer qu'un seul objet ou travail mis à l'échelle pour chaque charge de travail que vous souhaitez mettre à l'échelle. Vous ne pouvez pas non plus utiliser un objet ou un travail mis à l'échelle et le pod autoscaler horizontal (HPA) sur la même charge de travail.

2.5.11.1. Ajout d'un autoscaler de métriques personnalisé à une charge de travail

Vous pouvez créer un autoscaler de métriques personnalisé pour une charge de travail créée par un objet **Deployment**, **StatefulSet** ou **custom resource**.

Conditions préalables

- L'opérateur Custom Metrics Autoscaler doit être installé.
- Si vous utilisez un autoscaler de métriques personnalisé pour une mise à l'échelle basée sur le CPU ou la mémoire :
  - L'administrateur de votre cluster doit avoir correctement configuré les métriques du cluster. Vous pouvez utiliser la commande **oc describe PodMetrics <pod-name>** pour déterminer si les métriques sont configurées. Si les métriques sont configurées, la sortie est similaire à ce qui suit, avec le CPU et la mémoire affichés sous Usage.

```
$ oc describe PodMetrics openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
```

Exemple de sortie

```
Name:         openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Namespace:    openshift-kube-scheduler
Labels:       <none>
Annotations:  <none>
API Version:  metrics.k8s.io/v1beta1
Containers:
  Name:  wait-for-host-port
  Usage:
    Memory:  0
  Name:      scheduler
  Usage:
    Cpu:     8m
    Memory:  45440Ki
Kind:        PodMetrics
Metadata:
  Creation Timestamp:  2019-05-23T18:47:56Z
  Self Link:           /apis/metrics.k8s.io/v1beta1/namespaces/openshift-kube-
scheduler/pods/openshift-kube-scheduler-ip-10-0-135-131.ec2.internal
Timestamp:             2019-05-23T18:47:56Z
Window:                1m0s
Events:                <none>
```

- Les pods associés à l'objet que vous souhaitez mettre à l'échelle doivent inclure les limites de mémoire et de CPU spécifiées. Par exemple :

Exemple de spécification de pod

```
apiVersion: v1
kind: Pod
 ...
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      limits:
        memory: "128Mi"
        cpu: "500m"
```

Procédure

1. Créez un fichier YAML similaire au suivant. Seuls le nom **<2>**, le nom de l'objet **<4>** et le type d'objet **<5>** sont nécessaires :

Exemple d'objet mis à l'échelle

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  annotations:
```

```
    autoscaling.keda.sh/paused-replicas: "0" (1)
  name: scaledobject (2)
  namespace: my-namespace
spec:
  scaleTargetRef:
    apiVersion: apps/v1 (3)
    name: example-deployment (4)
    kind: Deployment (5)
    envSourceContainerName: .spec.template.spec.containers[0] (6)
  cooldownPeriod:  200 (7)
  maxReplicaCount: 100 (8)
  minReplicaCount: 0 (9)
  metricsServer: (10)
    auditConfig:
      logFormat: "json"
      logOutputVolumeClaim: "persistentVolumeClaimName"
      policy:
        rules:
        - level: Metadata
        omitStages: "RequestReceived"
        omitManagedFields: false
      lifetime:
        maxAge: "2"
        maxBackup: "1"
        maxSize: "50"
  fallback: (11)
    failureThreshold: 3
    replicas: 6
  pollingInterval: 30 (12)
  advanced:
    restoreToOriginalReplicaCount: false (13)
    horizontalPodAutoscalerConfig:
      name: keda-hpa-scale-down (14)
      behavior: (15)
        scaleDown:
          stabilizationWindowSeconds: 300
          policies:
          - type: Percent
            value: 100
            periodSeconds: 15
  triggers:
  - type: prometheus (16)
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: kedatest
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "basic"
  - authenticationRef: (17)
      name: prom-triggerauthentication
    metadata:
      name: prom-triggerauthentication
    type: object
```

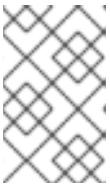
```
- authenticationRef: 18
    name: prom-cluster-triggerauthentication
  metadata:
    name: prom-cluster-triggerauthentication
  type: object
```

1. Facultatif : Spécifie que l'opérateur Custom Metrics Autoscaler doit mettre à l'échelle les répliques jusqu'à la valeur spécifiée et arrêter la mise à l'échelle automatique, comme décrit dans la section "Mise en pause du Custom Metrics Autoscaler pour une charge de travail".
2. Spécifie un nom pour cet autoscaler de métriques personnalisé.
3. Facultatif : Spécifie la version API de la ressource cible. La valeur par défaut est **apps/v1**.
4. Spécifie le nom de l'objet que vous souhaitez mettre à l'échelle.
5. Spécifie que le site **kind** est **Deployment**, **StatefulSet** ou **CustomResource**.
6. Facultatif : Spécifie le nom du conteneur dans la ressource cible, à partir duquel l'autoscaler de métriques personnalisées obtient les variables d'environnement contenant les secrets, etc. La valeur par défaut est **.spec.template.spec.containers[0]**.
7. Facultatif. Spécifie la période en secondes à attendre après le signalement du dernier déclencheur avant de ramener le déploiement à **0** si **minReplicaCount** est défini sur **0**. La valeur par défaut est **300**.
8. Facultatif : Spécifie le nombre maximum de répliques lors de la mise à l'échelle. La valeur par défaut est **100**.
9. Facultatif : Spécifie le nombre minimum de répliques lors de la réduction d'échelle.
10. Facultatif : Spécifie les paramètres des journaux d'audit, comme décrit dans la section "Configuration de la journalisation des audits".
11. Facultatif : Spécifie le nombre de répliques à utiliser si un scaler ne parvient pas à obtenir les métriques de la source pendant le nombre de fois défini par le paramètre **failureThreshold**. Pour plus d'informations sur le comportement de repli, voir la documentation KEDA.
12. Facultatif : Spécifie l'intervalle en secondes pour vérifier chaque déclencheur. La valeur par défaut est **30**.
13. Facultatif : Indique s'il faut ramener la ressource cible au nombre de répliques d'origine après la suppression de l'objet mis à l'échelle. La valeur par défaut est **false**, qui conserve le nombre de répliques tel qu'il est lorsque l'objet mis à l'échelle est supprimé.
14. Facultatif : Spécifie un nom pour le pod horizontal autoscaler. La valeur par défaut est **keda-hpa-{scaled-object-name}**.
15. Facultatif : Spécifie une politique de mise à l'échelle à utiliser pour contrôler le taux de mise à l'échelle des pods à la hausse ou à la baisse, comme décrit dans la section "Politiques de mise à l'échelle".
16. Spécifie le déclencheur à utiliser comme base pour la mise à l'échelle, comme décrit dans la section "Understanding the custom metrics autoscaler triggers" (Comprendre les déclencheurs d'autoscaler de métriques personnalisées). Cet exemple utilise la surveillance

de OpenShift Container Platform.

17 Facultatif : Spécifie une authentification de déclenchement, comme décrit dans la section "Création d'une authentification de déclenchement autoscaler de métriques personnalisée".

18 Facultatif : Spécifie une authentification de déclenchement de cluster, comme décrit dans la section "Création d'une authentification de déclenchement autoscaler de métriques personnalisée".



NOTE

Il n'est pas nécessaire de spécifier à la fois une authentification de déclenchement de l'espace de noms et une authentification de déclenchement du cluster.

2. Créez l'autoscaler de métriques personnalisé :

```
oc create -f <nom-de-fichier>.yaml
```

Vérification

- Affichez la sortie de la commande pour vérifier que l'autoscaler de métriques personnalisé a été créé :

```
oc get scaledobject <scaled_object_name>
```

Exemple de sortie

```
NAME              SCALETARGETKIND      SCALETARGETNAME      MIN   MAX   TRIGGERS
AUTHENTICATION              READY   ACTIVE   FALLBACK   AGE
scaledobject   apps/v1.Deployment   example-deployment   0     50    prometheus   prom-
triggerauthentication   True    True     True       17s
```

Notez les champs suivants dans le résultat :

- **TRIGGERS**: Indique le déclencheur, ou l'échelle, utilisé.
- **AUTHENTICATION**: Indique le nom de l'authentification de déclenchement utilisée.
- **READY**: Indique si l'objet mis à l'échelle est prêt à commencer la mise à l'échelle :
  - Si **True**, l'objet mis à l'échelle est prêt.
  - Si **False**, l'objet mis à l'échelle n'est pas prêt en raison d'un problème dans un ou plusieurs des objets que vous avez créés.
- **ACTIVE**: Indique si la mise à l'échelle est en cours :
  - Si **True**, il y a mise à l'échelle.
  - Si **False**, la mise à l'échelle n'a pas lieu parce qu'il n'y a pas de métriques ou qu'il y a un problème dans un ou plusieurs des objets que vous avez créés.

- **FALLBACK**: Indique si l'autoscaler de métriques personnalisé est capable d'obtenir des métriques de la source
  - Si **False**, l'autoscaler de métriques personnalisées reçoit des métriques.
  - Si **True**, l'autoscaler de métriques personnalisées reçoit des métriques parce qu'il n'y en a pas ou qu'il y a un problème dans un ou plusieurs des objets que vous avez créés.

Ressources supplémentaires

- Politiques d'échelonnement
- Comprendre les déclencheurs d'autoscaler de métriques personnalisées
- Comprendre les métriques personnalisées Autoscaler trigger authentications

2.5.11.2. Ajout d'un autoscaler de métriques personnalisé à une tâche

Vous pouvez créer un autoscaler de métriques personnalisé pour n'importe quel objet **Job**.

Conditions préalables

- L'opérateur Custom Metrics Autoscaler doit être installé.

Procédure

1. Créez un fichier YAML similaire au suivant :

```
kind: ScaledJob
apiVersion: keda.sh/v1alpha1
metadata:
  name: scaledjob
  namespace: my-namespace
spec:
  failedJobsHistoryLimit: 5
  jobTargetRef:
    activeDeadlineSeconds: 600 (1)
    backoffLimit: 6 (2)
    parallelism: 1 (3)
    completions: 1 (4)
    template: (5)
      metadata:
        name: pi
      spec:
        containers:
        - name: pi
          image: perl
          command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
  maxReplicaCount: 100 (6)
  pollingInterval: 30 (7)
  successfulJobsHistoryLimit: 5 (8)
  failedJobsHistoryLimit: 5 (9)
  envSourceContainerName: (10)
  rolloutStrategy: gradual (11)
```

```
  scalingStrategy: (12)
    strategy: "custom"
    customScalingQueueLengthDeduction: 1
    customScalingRunningJobPercentage: "0.5"
    pendingPodConditions:
      - "Ready"
      - "PodScheduled"
      - "AnyOtherCustomPodCondition"
    multipleScalersCalculation : "max"
  triggers:
  - type: prometheus (13)
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9092
      namespace: kedatest
      metricName: http_requests_total
      threshold: '5'
      query: sum(rate(http_requests_total{job="test-app"}[1m]))
      authModes: "bearer"
  - authenticationRef: (14)
      name: prom-triggerauthentication
    metadata:
      name: prom-triggerauthentication
    type: object
  - authenticationRef: (15)
      name: prom-cluster-triggerauthentication
    metadata:
      name: prom-cluster-triggerauthentication
    type: object
```

1. Spécifie la durée maximale d'exécution du travail.
2. Spécifie le nombre de tentatives pour un travail. La valeur par défaut est **6**.
3. Facultatif : Spécifie le nombre de répliques de pods qu'un travail doit exécuter en parallèle ; la valeur par défaut est **1**.
   - Pour les travaux non parallèles, laissez le paramètre non défini. Si elle n'est pas définie, la valeur par défaut est **1**.
4. Facultatif : Indique le nombre de pods terminés avec succès pour qu'un travail soit considéré comme terminé.
   - Pour les travaux non parallèles, laissez le paramètre non défini. Si elle n'est pas définie, la valeur par défaut est **1**.
   - Pour les travaux parallèles avec un nombre d'achèvements fixe, indiquez le nombre d'achèvements.
   - Pour les travaux parallèles avec une file d'attente, laissez le paramètre non défini. Lorsqu'il n'est pas défini, la valeur par défaut est celle du paramètre **parallelism**.
5. Spécifie le modèle du module créé par le contrôleur.
6. Facultatif : Spécifie le nombre maximum de répliques lors de la mise à l'échelle. La valeur par défaut est **100**.

7 Facultatif : Spécifie l'intervalle en secondes pour vérifier chaque déclencheur. La valeur par défaut est **30**.

8 Facultatif : Spécifie le nombre de travaux terminés avec succès qui doivent être conservés. La valeur par défaut est **100**.

9 Facultatif : Spécifie le nombre de travaux ayant échoué qui doivent être conservés. La valeur par défaut est **100**.

10 Facultatif : Spécifie le nom du conteneur dans la ressource cible, à partir duquel l'autoscaler personnalisé obtient les variables d'environnement contenant les secrets, etc. La valeur par défaut est **.spec.template.spec.containers[0]**.

11 Facultatif : Indique si les travaux existants sont interrompus lorsqu'un travail mis à l'échelle est mis à jour :

- **default**: L'autoscaler met fin à un travail existant si le travail de mise à l'échelle qui lui est associé est mis à jour. L'autoscaler recrée le travail avec les dernières spécifications.
- **gradual**: L'autoscaler ne met pas fin à un travail existant si le travail de mise à l'échelle qui lui est associé est mis à jour. L'autoscaler crée de nouveaux travaux avec les dernières spécifications.

12 Facultatif : Spécifie une stratégie de mise à l'échelle : **default custom** ou **accurate**. La valeur par défaut est **default**. Pour plus d'informations, voir le lien dans la section "Ressources supplémentaires" qui suit.

13 Spécifie le déclencheur à utiliser comme base pour la mise à l'échelle, comme décrit dans la section "Understanding the custom metrics autoscaler triggers" (Comprendre les déclencheurs d'autoscaler de mesures personnalisées).

14 Facultatif : Spécifie une authentification de déclenchement, comme décrit dans la section "Création d'une authentification de déclenchement autoscaler de métriques personnalisée".

15 Facultatif : Spécifie une authentification de déclenchement de cluster, comme décrit dans la section "Création d'une authentification de déclenchement autoscaler de métriques personnalisée".

NOTE

Il n'est pas nécessaire de spécifier à la fois une authentification de déclenchement de l'espace de noms et une authentification de déclenchement du cluster.

2. Créez l'autoscaler de métriques personnalisé :

```
oc create -f <nom-de-fichier>.yaml
```

Vérification

- Affichez la sortie de la commande pour vérifier que l'autoscaler de métriques personnalisé a été créé :

-

```
oc get scaledjob <scaled_job_name>
```

Exemple de sortie

```
NAME        MAX   TRIGGERS     AUTHENTICATION              READY   ACTIVE   AGE
scaledjob   100   prometheus   prom-triggerauthentication  True    True     8s
```

Notez les champs suivants dans le résultat :

- **TRIGGERS**: Indique le déclencheur, ou l'échelle, utilisé.
- **AUTHENTICATION**: Indique le nom de l'authentification de déclenchement utilisée.
- **READY**: Indique si l'objet mis à l'échelle est prêt à commencer la mise à l'échelle :
  - Si **True**, l'objet mis à l'échelle est prêt.
  - Si **False**, l'objet mis à l'échelle n'est pas prêt en raison d'un problème dans un ou plusieurs des objets que vous avez créés.
- **ACTIVE**: Indique si la mise à l'échelle est en cours :
  - Si **True**, il y a mise à l'échelle.
  - Si **False**, la mise à l'échelle n'a pas lieu parce qu'il n'y a pas de métriques ou qu'il y a un problème dans un ou plusieurs des objets que vous avez créés.

Ressources supplémentaires

- Comprendre les déclencheurs d'autoscaler de métriques personnalisées
- Comprendre les métriques personnalisées Autoscaler trigger authentications
- Exécuter des tâches dans des pods à l'aide de jobs

2.5.12. Désinstallation de l'opérateur Autoscaler de mesures personnalisées

Vous pouvez supprimer le Custom Metrics Autoscaler de votre cluster OpenShift Container Platform. Après avoir supprimé l'opérateur Custom Metrics Autoscaler, supprimez les autres composants associés à l'opérateur afin d'éviter tout problème potentiel.

NOTE

Vous devez d'abord supprimer la ressource personnalisée (CR) **KedaController**. Si vous ne supprimez pas spécifiquement la CR, OpenShift Container Platform peut se bloquer lorsque vous supprimez le projet **openshift-keda**. Si vous supprimez le Custom Metrics Autoscaler Operator avant de supprimer la CR, vous ne pourrez pas supprimer la CR.

Conditions préalables

- L'opérateur Custom Metrics Autoscaler doit être installé.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Passez au projet **openshift-keda**.
3. Supprimer la ressource personnalisée **KedaController**.
   a. Trouvez l'opérateur **CustomMetricsAutoscaler** et cliquez sur l'onglet **KedaController**.
   b. Recherchez la ressource personnalisée, puis cliquez sur **Delete KedaController**.
   c. Cliquez sur **Uninstall**.
4. Supprimez l'opérateur Autoscaler de métriques personnalisées :
   a. Cliquez sur **Operators → Installed Operators**.
   b. Trouvez l'opérateur **CustomMetricsAutoscaler** et cliquez sur le menu **Options** et sélectionnez **Uninstall Operator**.
   c. Cliquez sur **Uninstall**.
5. Optionnel : Utilisez le CLI OpenShift pour supprimer les composants autoscaler des métriques personnalisées :
   a. Supprimez les CRD de métriques personnalisées autoscaler :
      - **clustertriggerauthentications.keda.sh**
      - **kedacontrollers.keda.sh**
      - **scaledjobs.keda.sh**
      - **scaledobjects.keda.sh**
      - **triggerauthentications.keda.sh**

      ```
      $ oc delete crd clustertriggerauthentications.keda.sh kedacontrollers.keda.sh
      scaledjobs.keda.sh scaledobjects.keda.sh triggerauthentications.keda.sh
      ```

      La suppression des CRD supprime les rôles associés, les rôles de cluster et les liaisons de rôles. Toutefois, il se peut que quelques rôles de cluster doivent être supprimés manuellement.
   b. Liste des rôles de cluster autoscaler de métriques personnalisées :

      ```
      $ oc get clusterrole | grep keda.sh
      ```

   c. Supprimez les rôles de cluster autoscaler de métriques personnalisées répertoriés. Par exemple :

      ```
      $ oc delete clusterrole.keda.sh-v1alpha1-admin
      ```

   d. Liste de toutes les liaisons de rôles de cluster autoscaler de métriques personnalisées :

```
$ oc get clusterrolebinding | grep keda.sh
```

e. Supprimez les liaisons de rôles de cluster autoscaler de métriques personnalisées répertoriées. Par exemple :

```
$ oc delete clusterrolebinding.keda.sh-v1alpha1-admin
```

6. Supprimez le projet custom metrics autoscaler :

```
$ oc delete project openshift-keda
```

7. Supprimez l'opérateur Autoscaler de métrique de cluster :

```
$ oc delete operator/openshift-custom-metrics-autoscaler-operator.openshift-keda
```

2.6. AJUSTEZ AUTOMATIQUEMENT LES NIVEAUX DE RESSOURCES DES PODS GRÂCE À L'AUTOSCALER VERTICAL DE PODS

Le Vertical Pod Autoscaler Operator (VPA) d'OpenShift Container Platform examine automatiquement les ressources historiques et actuelles de CPU et de mémoire pour les conteneurs dans les pods et peut mettre à jour les limites de ressources et les demandes en fonction des valeurs d'utilisation qu'il apprend. Le VPA utilise des ressources personnalisées individuelles (CR) pour mettre à jour tous les pods associés à un objet de charge de travail, tel que **Deployment**, **DeploymentConfig**, **StatefulSet**, **Job**, **DaemonSet**, **ReplicaSet**, ou **ReplicationController**, dans un projet.

L'APV vous aide à comprendre l'utilisation optimale du CPU et de la mémoire pour vos pods et peut automatiquement maintenir les ressources des pods tout au long de leur cycle de vie.

2.6.1. À propos de l'opérateur de l'autoscaler à nacelle verticale

Le Vertical Pod Autoscaler Operator (VPA) est implémenté en tant que ressource API et ressource personnalisée (CR). La CR détermine les actions que le Vertical Pod Autoscaler Operator doit entreprendre avec les pods associés à un objet de charge de travail spécifique, tel qu'un ensemble de démons, un contrôleur de réplication, etc. dans un projet.

Vous pouvez utiliser le système de recommandation par défaut ou utiliser votre propre système de recommandation pour procéder à une mise à l'échelle automatique sur la base de vos propres algorithmes.

Le recommandeur par défaut calcule automatiquement l'utilisation historique et actuelle du processeur et de la mémoire pour les conteneurs de ces modules et utilise ces données pour déterminer des limites de ressources et des demandes optimisées afin de garantir que ces modules fonctionnent efficacement à tout moment. Par exemple, le recommandeur par défaut suggère de réduire les ressources pour les conteneurs qui demandent plus de ressources qu'ils n'en utilisent et d'augmenter les ressources pour les conteneurs qui n'en demandent pas assez.

L'APV supprime ensuite automatiquement tous les pods qui ne sont pas conformes à ces recommandations, un par un, afin que vos applications puissent continuer à répondre aux demandes sans interruption de service. Les objets de charge de travail redéploient ensuite les pods avec les limites de ressources et les demandes d'origine. L'APV utilise un webhook d'admission en mutation pour mettre à jour les modules avec des limites de ressources et des demandes optimisées avant que les modules ne

soient admis sur un nœud. Si vous ne souhaitez pas que l'APV supprime des modules, vous pouvez afficher les limites de ressources et les demandes de l'APV et mettre à jour manuellement les modules si nécessaire.

NOTE

Par défaut, les objets de charge de travail doivent spécifier un minimum de deux répliques pour que l'APV supprime automatiquement leurs pods. Les objets de charge de travail qui spécifient moins de répliques que ce minimum ne sont pas supprimés. Si vous supprimez manuellement ces modules, lorsque l'objet de charge de travail redéploie les modules, l'APV met à jour les nouveaux modules avec ses recommandations. Vous pouvez modifier ce minimum en modifiant l'objet **VerticalPodAutoscalerController** comme indiqué à l'adresse *Changing the VPA minimum value*.

Par exemple, si vous avez un module qui utilise 50 % de l'UC mais n'en demande que 10 %, l'APV détermine que le module consomme plus d'UC que ce qui est demandé et le supprime. L'objet de charge de travail, tel que l'ensemble de répliques, redémarre les modules et l'APV met à jour le nouveau module avec les ressources recommandées.

Pour les développeurs, vous pouvez utiliser l'APV pour vous assurer que vos pods restent opérationnels pendant les périodes de forte demande en planifiant les pods sur des nœuds qui disposent des ressources appropriées pour chaque pod.

Les administrateurs peuvent utiliser l'APV pour mieux utiliser les ressources du cluster, par exemple en empêchant les pods de réserver plus de ressources CPU que nécessaire. L'APV surveille les ressources que les charges de travail utilisent réellement et ajuste les besoins en ressources afin que la capacité soit disponible pour d'autres charges de travail. L'APV maintient également les ratios entre les limites et les demandes qui sont spécifiés dans la configuration initiale du conteneur.

NOTE

Si vous arrêtez de faire fonctionner l'APV ou si vous supprimez un CR APV spécifique dans votre cluster, les demandes de ressources pour les pods déjà modifiés par l'APV ne changent pas. Tout nouveau module obtient les ressources définies dans l'objet de charge de travail, et non les recommandations précédentes faites par l'APPV.

2.6.2. Installation de l'opérateur Autoscaler Vertical Pod

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer l'opérateur Vertical Pod Autoscaler (VPA).

Procédure

1. Dans la console Web OpenShift Container Platform, cliquez sur **Operators → OperatorHub**.
2. Choisissez **VerticalPodAutoscaler** dans la liste des opérateurs disponibles et cliquez sur **Install**.
3. Sur la page **Install Operator**, assurez-vous que l'option **Operator recommended namespace** est sélectionnée. Cela permet d'installer l'opérateur dans l'espace de noms obligatoire **openshift-vertical-pod-autoscaler**, qui est automatiquement créé s'il n'existe pas.
4. Cliquez sur **Install**.
5. Vérifiez l'installation en dressant la liste des composants de l'opérateur VPA :

a. Navigate to **Workloads → Pods**.

b. Sélectionnez le projet **openshift-vertical-pod-autoscaler** dans le menu déroulant et vérifiez que quatre pods sont en cours d'exécution.

c. Naviguez jusqu'à **Workloads → Deployments** pour vérifier que quatre déploiements sont en cours.

6. Facultatif. Vérifiez l'installation dans le CLI de OpenShift Container Platform à l'aide de la commande suivante :

```
$ oc get all -n openshift-vertical-pod-autoscaler
```

La sortie montre quatre pods et quatre déploiements :

Exemple de sortie

```
NAME                                            READY   STATUS    RESTARTS   AGE
pod/vertical-pod-autoscaler-operator-85b4569c47-2gmhc   1/1     Running   0          3m13s
pod/vpa-admission-plugin-default-67644fc87f-xq7k9       1/1     Running   0          2m56s
pod/vpa-recommender-default-7c54764b59-8gckt            1/1     Running   0          2m56s
pod/vpa-updater-default-7f6cc87858-47vw9                1/1     Running   0          2m56s

NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/vpa-webhook   ClusterIP   172.30.53.206   <none>        443/TCP   2m56s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/vertical-pod-autoscaler-operator   1/1     1            1           3m13s
deployment.apps/vpa-admission-plugin-default       1/1     1            1           2m56s
deployment.apps/vpa-recommender-default            1/1     1            1           2m56s
deployment.apps/vpa-updater-default                1/1     1            1           2m56s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/vertical-pod-autoscaler-operator-85b4569c47   1         1         1       3m13s
replicaset.apps/vpa-admission-plugin-default-67644fc87f       1         1         1       2m56s
replicaset.apps/vpa-recommender-default-7c54764b59            1         1         1       2m56s
replicaset.apps/vpa-updater-default-7f6cc87858                1         1         1       2m56s
```

2.6.3. À propos de l'utilisation de l'opérateur d'autoscaler de nacelles verticales

Pour utiliser l'opérateur Vertical Pod Autoscaler (VPA), vous créez une ressource personnalisée (CR) VPA pour un objet de charge de travail dans votre cluster. Le VPA apprend et applique les ressources optimales de CPU et de mémoire pour les pods associés à cet objet de charge de travail. Vous pouvez utiliser une APV avec un déploiement, un ensemble avec état, un travail, un ensemble de démons, un ensemble de réplicas ou un objet de charge de travail de contrôleur de réplication. L'APV CR doit se trouver dans le même projet que les modules que vous souhaitez surveiller.

La CR APV permet d'associer un objet de charge de travail et de spécifier le mode de fonctionnement de l'APV :

- Les modes **Auto** et **Recreate** appliquent automatiquement les recommandations de l'APV en matière de CPU et de mémoire tout au long de la durée de vie du pod. L'APV supprime tous les modules du projet qui ne sont pas conformes à ses recommandations. Lorsqu'il est redéployé par l'objet de charge de travail, l'APV met à jour les nouveaux modules avec ses recommandations.

- Le mode **Initial** applique automatiquement les recommandations de l'APV uniquement lors de la création d'un pod.
- Le mode **Off** ne fournit que les limites de ressources et les demandes recommandées, ce qui vous permet d'appliquer manuellement les recommandations. Le mode **off** ne met pas à jour les pods.

Vous pouvez également utiliser le CR pour exclure certains conteneurs de l'évaluation et des mises à jour de l'APV.

Par exemple, un pod a les limites et les demandes suivantes :

```
resources:
  limits:
    cpu: 1
    memory: 500Mi
  requests:
    cpu: 500m
    memory: 100Mi
```

Après la création d'une APV configurée sur **auto**, l'APV apprend l'utilisation des ressources et supprime le module. Lorsqu'il est redéployé, le module utilise les nouvelles limites et demandes de ressources :

```
resources:
  limits:
    cpu: 50m
    memory: 1250Mi
  requests:
    cpu: 25m
    memory: 262144k
```

Vous pouvez afficher les recommandations de l'APV à l'aide de la commande suivante :

```
$ oc get vpa <vpa-name> --output yaml
```

Après quelques minutes, l'écran affiche les recommandations pour les demandes de CPU et de mémoire, comme suit :

Exemple de sortie

```
...
status:
...
  recommendation:
    containerRecommendations:
    - containerName: frontend
      lowerBound:
        cpu: 25m
        memory: 262144k
      target:
        cpu: 25m
        memory: 262144k
      uncappedTarget:
        cpu: 25m
        memory: 262144k
```

```
      upperBound:
        cpu: 262m
        memory: "274357142"
    - containerName: backend
      lowerBound:
        cpu: 12m
        memory: 131072k
      target:
        cpu: 12m
        memory: 131072k
      uncappedTarget:
        cpu: 12m
        memory: 131072k
      upperBound:
        cpu: 476m
        memory: "498558823"
...
```

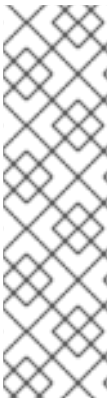
Le résultat montre les ressources recommandées, **target**, les ressources minimales recommandées, **lowerBound**, les ressources les plus élevées recommandées, **upperBound**, et les recommandations de ressources les plus récentes, **uncappedTarget**.

L'APV utilise les valeurs **lowerBound** et **upperBound** pour déterminer si un module doit être mis à jour. Si un module a des demandes de ressources inférieures aux valeurs **lowerBound** ou supérieures aux valeurs **upperBound**, l'APV met fin au module et le recrée avec les valeurs **target**.

2.6.3.1. Modification de la valeur minimale de l'APV

Par défaut, les objets de charge de travail doivent spécifier un minimum de deux répliques pour que l'APV supprime et mette à jour automatiquement leurs pods. Par conséquent, les objets de charge de travail qui spécifient moins de deux répliques ne sont pas automatiquement pris en compte par l'APPV. L'APP met à jour les nouveaux modules de ces objets de charge de travail si les modules sont redémarrés par un processus externe à l'APP. Vous pouvez modifier cette valeur minimale à l'échelle du cluster en modifiant le paramètre **minReplicas** dans la ressource personnalisée (CR) **VerticalPodAutoscalerController**.

Par exemple, si vous définissez **minReplicas** sur **3**, l'APV ne supprime pas et ne met pas à jour les pods pour les objets de charge de travail qui spécifient moins de trois répliques.



NOTE

Si vous définissez **minReplicas** sur **1**, l'APV peut supprimer le seul module d'un objet de charge de travail qui ne spécifie qu'une seule réplique. Vous ne devez utiliser ce paramètre avec les objets à une réplique que si votre charge de travail peut tolérer des temps d'arrêt chaque fois que l'APV supprime un module pour ajuster ses ressources. Pour éviter les temps d'arrêt indésirables avec les objets à réplique unique, configurez les CR de l'APV avec le paramètre **podUpdatePolicy** défini sur **Initial**, qui met automatiquement à jour le module uniquement lorsqu'il est redémarré par un processus externe à l'APV, ou **Off**, qui vous permet de mettre à jour le module manuellement à un moment approprié pour votre application.

Exemple d'objet VerticalPodAutoscalerController

```
apiVersion: autoscaling.openshift.io/v1
kind: VerticalPodAutoscalerController
```

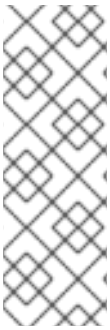
```
metadata:
  creationTimestamp: "2021-04-21T19:29:49Z"
  generation: 2
  name: default
  namespace: openshift-vertical-pod-autoscaler
  resourceVersion: "142172"
  uid: 180e17e9-03cc-427f-9955-3b4d7aeb2d59
spec:
  minReplicas: 3 1
  podMinCPUMillicores: 25
  podMinMemoryMb: 250
  recommendationOnly: false
  safetyMarginFraction: 0.15
```

1 Spécifiez le nombre minimum de répliques d'un objet de charge de travail sur lequel l'APV doit agir. Les objets dont le nombre de répliques est inférieur au minimum ne sont pas automatiquement supprimés par l'APV.

2.6.3.2. Appliquer automatiquement les recommandations de l'APV

Pour utiliser l'APV afin de mettre à jour automatiquement les modules, créez un CR APV pour un objet de charge de travail spécifique avec **updateMode** défini sur **Auto** ou **Recreate**.

Lorsque les modules sont créés pour l'objet de charge de travail, l'APV surveille constamment les conteneurs pour analyser leurs besoins en CPU et en mémoire. L'APV supprime tous les modules qui ne répondent pas aux recommandations de l'APV en matière de CPU et de mémoire. Lors du redéploiement, les modules utilisent les nouvelles limites de ressources et les demandes basées sur les recommandations de l'APV, tout en respectant le budget de perturbation des modules défini pour vos applications. Les recommandations sont ajoutées au champ **status** du CR VPA pour référence.



NOTE

Par défaut, les objets de charge de travail doivent spécifier un minimum de deux répliques pour que l'APV supprime automatiquement leurs pods. Les objets de charge de travail qui spécifient moins de répliques que ce minimum ne sont pas supprimés. Si vous supprimez manuellement ces modules, lorsque l'objet de charge de travail redéploie les modules, l'APV met à jour les nouveaux modules avec ses recommandations. Vous pouvez modifier ce minimum en modifiant l'objet **VerticalPodAutoscalerController** comme indiqué à l'adresse *Changing the VPA minimum value*.

Exemple d'APV CR pour le mode Auto

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment 1
    name:       frontend 2
  updatePolicy:
    updateMode: "Auto" 3
```

1. Le type d'objet de charge de travail que vous voulez que ce CR VPA gère.
2. Le nom de l'objet de charge de travail que vous voulez que ce CR VPA gère.
3. Réglez le mode sur **Auto** ou **Recreate**:

    - **Auto**. L'APV attribue des demandes de ressources lors de la création d'un module et met à jour les modules existants en les supprimant lorsque les ressources demandées diffèrent sensiblement de la nouvelle recommandation.
    - **Recreate**. L'APV attribue des demandes de ressources lors de la création d'un pod et met à jour les pods existants en les terminant lorsque les ressources demandées diffèrent de manière significative de la nouvelle recommandation. Ce mode ne doit être utilisé que rarement, uniquement si vous devez vous assurer que les modules sont redémarrés chaque fois que la demande de ressources change.

NOTE

Il faut qu'il y ait des nacelles en fonctionnement dans le projet pour que l'APV puisse déterminer les ressources recommandées et appliquer les recommandations aux nouvelles nacelles.

2.6.3.3. Appliquer automatiquement les recommandations de l'APV lors de la création d'un pod

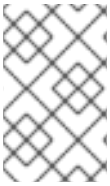
Pour utiliser l'APV afin d'appliquer les ressources recommandées uniquement lorsqu'un module est déployé pour la première fois, créez un APV CR pour un objet de charge de travail spécifique avec **updateMode** défini sur **Initial**.

Ensuite, supprimez manuellement tous les modules associés à l'objet de charge de travail pour lequel vous souhaitez utiliser les recommandations de l'APV. En mode **Initial**, l'APV ne supprime pas les modules et ne les met pas à jour lorsqu'il apprend de nouvelles recommandations de ressources.

Exemple d'APV CR pour le mode Initial

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment (1)
    name:       frontend (2)
  updatePolicy:
    updateMode: "Initial" (3)
```

1. Le type d'objet de charge de travail que vous voulez que ce CR VPA gère.
2. Le nom de l'objet de charge de travail que vous voulez que ce CR VPA gère.
3. Définissez le mode sur **Initial**. L'APV attribue des ressources lors de la création des pods et ne modifie pas les ressources pendant la durée de vie du pod.



NOTE

Il faut qu'il y ait des nacelles en fonctionnement dans le projet pour qu'une APV puisse déterminer les ressources recommandées et appliquer les recommandations aux nouvelles nacelles.

2.6.3.4. Application manuelle des recommandations de l'APV

Pour utiliser l'APV afin de déterminer uniquement les valeurs recommandées pour l'unité centrale et la mémoire, créez un APV CR pour un objet de charge de travail spécifique avec **updateMode** défini sur **off**.

Lorsque les modules sont créés pour cet objet de charge de travail, l'APPV analyse les besoins en CPU et en mémoire des conteneurs et enregistre ces recommandations dans le champ **status** du CR de l'APPV. L'APV ne met pas à jour les modules au fur et à mesure qu'il détermine de nouvelles recommandations de ressources.

Exemple d'APV CR pour le mode Off

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment 1
    name:       frontend 2
  updatePolicy:
    updateMode: "Off" 3
```

1. Le type d'objet de charge de travail que vous voulez que ce CR VPA gère.
2. Le nom de l'objet de charge de travail que vous voulez que ce CR VPA gère.
3. Réglez le mode sur **Off**.

Vous pouvez consulter les recommandations à l'aide de la commande suivante.

```
$ oc get vpa <vpa-name> --output yaml
```

Avec les recommandations, vous pouvez modifier l'objet de charge de travail pour ajouter des demandes de CPU et de mémoire, puis supprimer et redéployer les pods en utilisant les ressources recommandées.

NOTE

Pour qu'un APV puisse déterminer les ressources recommandées, il faut qu'il y ait des modules d'exploitation dans le projet.

2.6.3.5. Exempter les conteneurs de l'application des recommandations de l'APV

Si votre objet de charge de travail a plusieurs conteneurs et que vous ne voulez pas que l'APV évalue et agisse sur tous les conteneurs, créez une CR APV pour un objet de charge de travail spécifique et ajoutez une **resourcePolicy** pour exclure des conteneurs spécifiques.

Lorsque l'APV met à jour les pods avec les ressources recommandées, tous les conteneurs avec **resourcePolicy** ne sont pas mis à jour et l'APV ne présente pas de recommandations pour ces conteneurs dans le pod.

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment 1
    name:       frontend 2
  updatePolicy:
    updateMode: "Auto" 3
  resourcePolicy: 4
    containerPolicies:
    - containerName: my-opt-sidecar
      mode: "Off"
```

1. Le type d'objet de charge de travail que vous voulez que ce CR VPA gère.
2. Le nom de l'objet de charge de travail que vous voulez que ce CR VPA gère.
3. Définissez le mode sur **Auto**, **Recreate** ou **Off**. Le mode **Recreate** ne doit être utilisé que rarement, uniquement si vous devez vous assurer que les pods sont redémarrés à chaque fois que la demande de ressources change.
4. Spécifiez les conteneurs que vous souhaitez exclure et définissez **mode** comme **Off**.

Par exemple, un pod a deux conteneurs, les mêmes demandes de ressources et les mêmes limites :

```
# ...
spec:
  containers:
  - name: frontend
    resources:
      limits:
        cpu: 1
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
  - name: backend
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
```

```
        cpu: 500m
        memory: 100Mi
# ...
```

Après avoir lancé un VPA CR avec le conteneur **backend** en opt-out, le VPA se termine et recrée le pod avec les ressources recommandées appliquées uniquement au conteneur **frontend**:

```
...
spec:
  containers:
    name: frontend
    resources:
      limits:
        cpu: 50m
        memory: 1250Mi
      requests:
        cpu: 25m
        memory: 262144k
...
    name: backend
    resources:
      limits:
        cpu: "1"
        memory: 500Mi
      requests:
        cpu: 500m
        memory: 100Mi
...
```

2.6.3.6. Utilisation d'un autre système de recommandation

Vous pouvez utiliser votre propre recommandeur pour une mise à l'échelle automatique basée sur vos propres algorithmes. Si vous ne spécifiez pas de recommandation alternative, OpenShift Container Platform utilise la recommandation par défaut, qui suggère des demandes de CPU et de mémoire basées sur l'utilisation historique. Comme il n'existe pas de politique de recommandation universelle qui s'applique à tous les types de charges de travail, vous pouvez vouloir créer et déployer différents recommandeurs pour des charges de travail spécifiques.

Par exemple, le recommandeur par défaut peut ne pas prédire avec précision l'utilisation future des ressources lorsque les conteneurs présentent certains comportements en matière de ressources, tels que les modèles cycliques qui alternent entre les pics d'utilisation et la marche au ralenti, tels qu'utilisés par les applications de surveillance, ou les modèles récurrents et répétitifs utilisés avec les applications d'apprentissage en profondeur. L'utilisation de la recommandation par défaut avec ces comportements d'utilisation peut entraîner un surprovisionnement important et des pertes de mémoire (OOM) pour vos applications.

NOTE

Les instructions relatives à la création d'un recommandeur dépassent le cadre de cette documentation,

Procédure

Pour utiliser un recommandeur alternatif pour vos pods :

1. Créez un compte de service pour le recommandeur alternatif et liez ce compte de service au rôle de cluster requis :

```
apiVersion: v1 (1)
kind: ServiceAccount
metadata:
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 (2)
kind: ClusterRoleBinding
metadata:
  name: system:example-metrics-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:metrics-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 (3)
kind: ClusterRoleBinding
metadata:
  name: system:example-vpa-actor
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-actor
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
---
apiVersion: rbac.authorization.k8s.io/v1 (4)
kind: ClusterRoleBinding
metadata:
  name: system:example-vpa-target-reader-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:vpa-target-reader
subjects:
- kind: ServiceAccount
  name: alt-vpa-recommender-sa
  namespace: <namespace_name>
```

(1) Crée un compte de service pour le recommandeur dans l'espace de noms où le recommandeur est déployé.

(2) Lier le compte du service de recommandation au rôle **metrics-reader**. Spécifier l'espace de noms dans lequel le service de recommandation doit être déployé.

(3)

Lier le compte du service de recommandation au rôle **vpa-actor**. Spécifier l'espace de noms dans lequel le service de recommandation doit être déployé.

4 Lier le compte du service de recommandation au rôle **vpa-target-reader**. Spécifier l'espace de noms dans lequel le service de recommandation doit être déployé.

2. Pour ajouter le recommandeur alternatif au cluster, créez un objet de déploiement similaire au suivant :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: alt-vpa-recommender
  namespace: <namespace_name>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: alt-vpa-recommender
  template:
    metadata:
      labels:
        app: alt-vpa-recommender
    spec:
      containers: 1
      - name: recommender
        image: quay.io/example/alt-recommender:latest 2
        imagePullPolicy: Always
        resources:
          limits:
            cpu: 200m
            memory: 1000Mi
          requests:
            cpu: 50m
            memory: 500Mi
        ports:
        - name: prometheus
          containerPort: 8942
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
              - ALL
          seccompProfile:
            type: RuntimeDefault
      serviceAccountName: alt-vpa-recommender-sa 3
      securityContext:
        runAsNonRoot: true
```

1 Crée un conteneur pour votre recommandeur alternatif.

2 Spécifie votre image de recommandation.

3 Associe le compte de service que vous avez créé pour le recommandeur.

Un nouveau module est créé pour le recommandeur alternatif dans le même espace de noms.

```
$ oc get pods
```

Exemple de sortie

```
NAME                                    READY   STATUS    RESTARTS   AGE
frontend-845d5478d-558zf                1/1     Running   0          4m25s
frontend-845d5478d-7z9gx                1/1     Running   0          4m25s
frontend-845d5478d-b7l4j                1/1     Running   0          4m25s
vpa-alt-recommender-55878867f9-6tp5v    1/1     Running   0          9s
```

3. Configurez un CR APV qui inclut le nom de l'objet de recommandation alternatif **Deployment**.

Exemple d'APV CR incluant le recommandeur alternatif

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
  namespace: <namespace_name>
spec:
  recommenders:
    - name: alt-vpa-recommender (1)
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment (2)
    name:       frontend
```

(1) Spécifie le nom de l'autre déploiement de recommandation.

(2) Spécifie le nom d'un objet de charge de travail existant que l'APV doit gérer.

2.6.4. Utilisation de l'opérateur de l'autoscaler à nacelle verticale

Vous pouvez utiliser le Vertical Pod Autoscaler Operator (VPA) en créant une ressource personnalisée (CR) VPA. La CR indique les pods qu'elle doit analyser et détermine les actions que l'APV doit entreprendre avec ces pods.

Conditions préalables

- L'objet de charge de travail que vous souhaitez mettre à l'échelle doit exister.
- Si vous souhaitez utiliser un autre recommandeur, un déploiement incluant ce recommandeur doit exister.

Procédure

Pour créer un CR APV pour un objet de charge de travail spécifique :

1. Passez au projet dans lequel se trouve l'objet de charge de travail que vous souhaitez mettre à l'échelle.

a. Créer un fichier YAML VPA CR :

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: vpa-recommender
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind:       Deployment 1
    name:       frontend 2
  updatePolicy:
    updateMode: "Auto" 3
  resourcePolicy: 4
    containerPolicies:
    - containerName: my-opt-sidecar
      mode: "Off"
  recommenders: 5
    - name: my-recommender
```

1. Spécifiez le type d'objet de charge de travail que vous voulez que cette APV gère : **Deployment**, **StatefulSet**, **Job**, **DaemonSet**, **ReplicaSet**, ou **ReplicationController**.

2. Spécifiez le nom d'un objet de charge de travail existant que vous voulez que cette APV gère.

3. Spécifier le mode VPA :

   - **auto** pour appliquer automatiquement les ressources recommandées aux modules associés au contrôleur. L'APV met fin aux pods existants et crée de nouveaux pods avec les limites de ressources et les demandes recommandées.

   - **recreate** pour appliquer automatiquement les ressources recommandées aux modules associés à l'objet de charge de travail. L'APV met fin aux modules existants et en crée de nouveaux avec les limites et les demandes de ressources recommandées. Le mode **recreate** ne doit être utilisé que rarement, uniquement si vous devez vous assurer que les modules sont redémarrés chaque fois que la demande de ressources change.

   - **initial** pour appliquer automatiquement les ressources recommandées lorsque les modules associés à l'objet de charge de travail sont créés. L'APV ne met pas à jour les modules au fur et à mesure qu'il apprend de nouvelles recommandations de ressources.

   - **off** de ne générer des recommandations de ressources que pour les modules associés à l'objet de charge de travail. L'APV ne met pas à jour les modules au fur et à mesure qu'il apprend de nouvelles recommandations de ressources et n'applique pas les recommandations à de nouveaux modules.

4. Facultatif. Indiquez les conteneurs que vous souhaitez exclure et définissez le mode sur **Off**.

5. Optionnel. Indiquez un autre recommandeur.

b. Créer l'APV CR :

```
oc create -f <nom-de-fichier>.yaml
```

Après quelques instants, l'APV apprend l'utilisation des ressources des conteneurs dans les pods associés à l'objet de charge de travail.

Vous pouvez afficher les recommandations de l'APV à l'aide de la commande suivante :

```
$ oc get vpa <vpa-name> --output yaml
```

La sortie montre les recommandations pour les demandes de CPU et de mémoire, comme suit :

Exemple de sortie

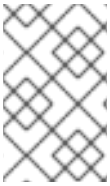
```
...
status:

...

  recommendation:
    containerRecommendations:
    - containerName: frontend
      lowerBound: 1
        cpu: 25m
        memory: 262144k
      target: 2
        cpu: 25m
        memory: 262144k
      uncappedTarget: 3
        cpu: 25m
        memory: 262144k
      upperBound: 4
        cpu: 262m
        memory: "274357142"
    - containerName: backend
      lowerBound:
        cpu: 12m
        memory: 131072k
      target:
        cpu: 12m
        memory: 131072k
      uncappedTarget:
        cpu: 12m
        memory: 131072k
      upperBound:
        cpu: 476m
        memory: "498558823"

...
```

1. **lowerBound** est le niveau minimum de ressources recommandé.
2. **target** est le niveau de ressources recommandé.

3. **upperBound** est le niveau de ressources recommandé le plus élevé.
4. **uncappedTarget** est la recommandation la plus récente en matière de ressources.

2.6.5. Désinstallation de l'opérateur d'autoscaler de nacelles verticales

Vous pouvez supprimer l'opérateur Vertical Pod Autoscaler (VPA) de votre cluster OpenShift Container Platform. Après la désinstallation, les demandes de ressources pour les pods déjà modifiés par un VPA CR existant ne changent pas. Tout nouveau pod obtient les ressources définies dans l'objet de charge de travail, et non les recommandations précédentes faites par le Vertical Pod Autoscaler Operator.



NOTE

Vous pouvez supprimer un VPA CR spécifique à l'aide de la commande **oc delete vpa <vpa-name>**. Les mêmes actions s'appliquent aux demandes de ressources que la désinstallation de l'autoscaler de pods verticaux.

Après avoir retiré l'Opérateur VPA, il est recommandé de retirer les autres composants associés à l'Opérateur afin d'éviter tout problème potentiel.

Conditions préalables

- L'opérateur Autoscaler Vertical Pod doit être installé.

Procédure

1. Dans la console web d'OpenShift Container Platform, cliquez sur **Operators → Installed Operators**.
2. Passez au projet **openshift-vertical-pod-autoscaler**.
3. Pour l'opérateur **VerticalPodAutoscaler**, cliquez sur le menu Options  et sélectionnez **Uninstall Operator**.
4. Facultatif : Pour supprimer tous les opérandes associés à l'opérateur, cochez la case **Delete all operand instances for this operator** dans la boîte de dialogue.
5. Cliquez sur **Uninstall**.
6. Optionnel : Utilisez la CLI d'OpenShift pour supprimer les composants de l'APV :
   a. Supprimer l'espace de noms de l'APV :

```
$ oc delete namespace openshift-vertical-pod-autoscaler
```

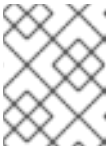
   b. Supprimer les objets de définition des ressources personnalisées (CRD) de l'APV :

```
$ oc delete crd verticalpodautoscalercheckpoints.autoscaling.k8s.io
```

```
$ oc delete crd verticalpodautoscalercontrollers.autoscaling.openshift.io
```

```
$ oc delete crd verticalpodautoscalers.autoscaling.k8s.io
```

La suppression des CRD supprime les rôles associés, les rôles de cluster et les liaisons de rôles.



NOTE

Cette action supprime du cluster tous les CR d'APV créés par l'utilisateur. Si vous réinstallez l'APV, vous devez à nouveau créer ces objets.

c. Supprimer l'opérateur APV :

```
$ oc delete operator/vertical-pod-autoscaler.openshift-vertical-pod-autoscaler
```

2.7. FOURNIR DES DONNÉES SENSIBLES AUX PODS

Certaines applications nécessitent des informations sensibles, telles que des mots de passe et des noms d'utilisateur, que vous ne voulez pas que les développeurs possèdent.

En tant qu'administrateur, vous pouvez utiliser les objets **Secret** pour fournir ces informations sans les exposer en clair.

2.7.1. Comprendre les secrets

Le type d'objet **Secret** fournit un mécanisme pour contenir des informations sensibles telles que les mots de passe, les fichiers de configuration du client OpenShift Container Platform, les informations d'identification du référentiel source privé, etc. Les secrets découplent le contenu sensible des pods. Vous pouvez monter des secrets dans des conteneurs à l'aide d'un plugin de volume ou le système peut utiliser des secrets pour effectuer des actions au nom d'un pod.

Les principales propriétés sont les suivantes

- Les données secrètes peuvent être référencées indépendamment de leur définition.
- Les volumes de données secrètes sont sauvegardés par des installations de stockage de fichiers temporaires (tmpfs) et ne s'arrêtent jamais sur un nœud.
- Les données secrètes peuvent être partagées au sein d'un espace de noms.

YAML Secret définition de l'objet

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque 1
data: 2
  username: dmFsdWUtMQ0K 3
  password: dmFsdWUtMg0KDQo=
stringData: 4
  hostname: myapp.mydomain.com 5
```

1. Indique la structure des noms et des valeurs des clés du secret.
2. Le format autorisé pour les clés du champ **data** doit respecter les directives de la valeur **DNS_SUBDOMAIN** dans le glossaire des identifiants Kubernetes.
3. La valeur associée aux clés de la carte **data** doit être encodée en base64.
4. Les entrées de la carte **stringData** sont converties en base64 et l'entrée est alors déplacée automatiquement vers la carte **data**. Ce champ est en écriture seule ; la valeur ne sera renvoyée que via le champ **data**.
5. La valeur associée aux clés de la carte **stringData** est constituée de chaînes de texte en clair.

Vous devez créer un secret avant de créer les modules qui dépendent de ce secret.

Lors de la création de secrets :

- Créer un objet secret avec des données secrètes.
- Mettre à jour le compte de service du pod pour autoriser la référence au secret.
- Créer un pod, qui consomme le secret comme une variable d'environnement ou comme un fichier (en utilisant un volume **secret**).

2.7.1.1. Types de secrets

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

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

- **kubernetes.io/service-account-token**. Utilise un jeton de compte de service.
- **kubernetes.io/basic-auth**. Utilisation avec l'authentification de base.
- **kubernetes.io/ssh-auth**. Utiliser avec l'authentification par clé SSH.
- **kubernetes.io/tls**. A utiliser avec les autorités de certification TLS.

Indiquez **type: Opaque** si vous ne souhaitez pas de validation, ce qui signifie que le secret ne prétend pas se conformer à une convention pour les noms de clés ou les valeurs. Un secret *opaque* permet d'utiliser des paires **key:value** non structurées pouvant contenir des valeurs arbitraires.

NOTE

Vous pouvez spécifier d'autres types arbitraires, tels que **example.com/my-secret-type**. Ces types ne sont pas appliqués côté serveur, mais ils indiquent que le créateur du secret avait l'intention de se conformer aux exigences clé/valeur de ce type.

Pour des exemples de différents types de secrets, voir les exemples de code sur *Using Secrets*.

2.7.1.2. Clés de données secrètes

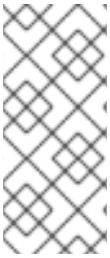
Les clés secrètes doivent se trouver dans un sous-domaine DNS.

2.7.1.3. À propos des secrets de jetons de compte de service générés automatiquement

Dans la version 4.12, OpenShift Container Platform adopte une amélioration de Kubernetes en amont , qui active la fonctionnalité **LegacyServiceAccountTokenNoAutoGeneration** par défaut. Par conséquent, lors de la création de nouveaux comptes de service (SA), un secret de jeton de compte de service n'est plus automatiquement généré. Auparavant, OpenShift Container Platform ajoutait automatiquement un jeton de compte de service à un secret pour chaque nouveau SA.

Cependant, certaines fonctionnalités et charges de travail ont besoin de secrets de jetons de compte de service pour communiquer avec le serveur API Kubernetes, par exemple, OpenShift Controller Manager. Cette exigence sera modifiée dans une prochaine version, mais elle demeure dans OpenShift Container Platform 4.12. Par conséquent, si vous avez besoin d'un secret de jeton de compte de service, vous devez utiliser manuellement l'API TokenRequest pour demander des jetons de compte de service liés ou créer un secret de jeton de compte de service.

Après la mise à jour vers la version 4.12, les secrets de jetons de compte de service existants ne sont pas supprimés et continuent de fonctionner comme prévu.



NOTE

Dans la version 4.12, les secrets des jetons des comptes de service sont toujours générés automatiquement. Au lieu de créer deux secrets par compte de service, OpenShift Container Platform n'en crée plus qu'un. Dans une prochaine version, ce nombre sera encore réduit à zéro. Notez que les secrets **dockercfg** sont toujours générés et qu'aucun secret n'est supprimé lors des mises à jour.

Ressources supplémentaires

- Pour plus d'informations sur la demande de jetons de compte de service liés, voir Utilisation de jetons de compte de service liés
- Pour plus d'informations sur la création d'un jeton secret de compte de service, voir Création d'un jeton secret de compte de service.

2.7.2. Comprendre comment créer des secrets

En tant qu'administrateur, vous devez créer un secret avant que les développeurs puissent créer les modules qui dépendent de ce secret.

Lors de la création de secrets :

1. Créez un objet secret contenant les données que vous souhaitez garder secrètes. Les données spécifiques requises pour chaque type de secret sont décrites dans les sections suivantes.

   Exemple d'objet YAML qui crée un secret opaque

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
type: Opaque 1
data: 2
  username: dmFsdWUtMQ0K
```

```
  password: dmFsdWUtMQ0KDQo=
stringData: 3
  hostname: myapp.mydomain.com
  secret.properties: |
    property1=valueA
    property2=valueB
```

1. Spécifie le type de secret.
2. Spécifie la chaîne et les données encodées.
3. Spécifie la chaîne et les données décodées.

Utilisez soit le champ **data**, soit le champ **stringdata**, mais pas les deux.

2. Mettre à jour le compte de service du pod pour référencer le secret :

YAML d'un compte de service qui utilise un secret

```
apiVersion: v1
kind: ServiceAccount
 ...
secrets:
- name: test-secret
```

3. Créer un pod, qui consomme le secret comme une variable d'environnement ou comme un fichier (en utilisant un volume **secret**) :

YAML d'un pod remplissant les fichiers d'un volume avec des données secrètes

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts: 1
          - name: secret-volume
            mountPath: /etc/secret-volume 2
            readOnly: true 3
  volumes:
    - name: secret-volume
      secret:
        secretName: test-secret 4
  restartPolicy: Never
```

1. Ajoutez un champ **volumeMounts** à chaque conteneur qui a besoin du secret.
2. Spécifie un nom de répertoire inutilisé dans lequel vous souhaitez que le secret apparaisse. Chaque clé de la carte de données secrètes devient le nom de fichier sous **mountPath**.

3 Défini à **true**. S'il est vrai, cela indique au pilote de fournir un volume en lecture seule.

4 Spécifie le nom du secret.

YAML d'un pod remplissant les variables d'environnement avec des données secrètes

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "export" ]
      env:
        - name: TEST_SECRET_USERNAME_ENV_VAR
          valueFrom:
            secretKeyRef: 1
              name: test-secret
              key: username
  restartPolicy: Never
```

1 Spécifie la variable d'environnement qui consomme la clé secrète.

YAML d'une configuration de construction remplissant les variables d'environnement avec des données secrètes

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef: 1
            name: test-secret
            key: username
```

1 Spécifie la variable d'environnement qui consomme la clé secrète.

2.7.2.1. Restrictions à la création de secrets

Pour utiliser un secret, un module doit faire référence au secret. Un secret peut être utilisé avec un module de trois façons :

- Pour remplir les variables d'environnement des conteneurs.

- En tant que fichiers dans un volume monté sur un ou plusieurs de ses conteneurs.
- Par kubelet lors de l'extraction des images pour le pod.

Les secrets de type volume écrivent des données dans le conteneur sous forme de fichier en utilisant le mécanisme de volume. Les secrets de type image pull utilisent des comptes de service pour l'injection automatique du secret dans tous les pods d'un espace de noms.

Lorsqu'un modèle contient une définition de secret, le seul moyen pour le modèle d'utiliser le secret fourni est de s'assurer que les sources de volume du secret sont validées et que la référence d'objet spécifiée pointe effectivement vers un objet **Secret**. Par conséquent, un secret doit être créé avant tout pod qui en dépend. Le moyen le plus efficace de s'en assurer est de l'injecter automatiquement par l'intermédiaire d'un compte de service.

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

La taille des secrets individuels est limitée à 1 Mo. Il s'agit de décourager la création de secrets volumineux qui pourraient épuiser la mémoire de l'apiserver et du kubelet. Cependant, la création d'un certain nombre de secrets plus petits pourrait également épuiser la mémoire.

2.7.2.2. Création d'un secret opaque

En tant qu'administrateur, vous pouvez créer un secret opaque, qui vous permet de stocker des paires **key:value** non structurées pouvant contenir des valeurs arbitraires.

Procédure

1. Créer un objet **Secret** dans un fichier YAML sur un nœud de plan de contrôle.
   Par exemple :

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque (1)
data:
  username: dXNlci1uYW1l
  password: cGFzc3dvcmQ=
```

(1) Spécifie un secret opaque.

2. La commande suivante permet de créer un objet **Secret**:

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

3. Pour utiliser le secret dans une dosette :

   a. Mettez à jour le compte de service du pod pour référencer le secret, comme indiqué dans la section "Comprendre comment créer des secrets".

   b. Créer le pod, qui consomme le secret sous forme de variable d'environnement ou de fichier (à l'aide d'un volume **secret**), comme indiqué dans la section "Comprendre comment créer des secrets".

Ressources supplémentaires

- Pour plus d'informations sur l'utilisation des secrets dans les modules, voir Comment créer des secrets.

2.7.2.3. Création d'un jeton secret de compte de service

En tant qu'administrateur, vous pouvez créer un secret de jeton de compte de service, qui vous permet de distribuer un jeton de compte de service aux applications qui doivent s'authentifier auprès de l'API.

NOTE

Il est recommandé d'obtenir des jetons de compte de service liés à l'aide de l'API TokenRequest plutôt que d'utiliser des jetons de compte de service secrets. Les jetons obtenus à l'aide de l'API TokenRequest sont plus sûrs que les jetons stockés dans les secrets, car ils ont une durée de vie limitée et ne sont pas lisibles par d'autres clients de l'API.

Vous ne devez créer un secret de jeton de compte de service que si vous ne pouvez pas utiliser l'API TokenRequest et si l'exposition à la sécurité d'un jeton n'expirant pas dans un objet API lisible est acceptable pour vous.

Pour plus d'informations sur la création de jetons de compte de service lié, reportez-vous à la section Ressources supplémentaires.

Procédure

1. Créer un objet **Secret** dans un fichier YAML sur un nœud de plan de contrôle :

   Exemple secret objet :

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-sa-sample
  annotations:
    kubernetes.io/service-account.name: "sa-name" 1
type: kubernetes.io/service-account-token 2
```

   1 Spécifie un nom de compte de service existant. Si vous créez les objets **ServiceAccount** et **Secret**, créez d'abord l'objet **ServiceAccount**.

   2 Spécifie un secret de jeton de compte de service.

2. La commande suivante permet de créer l'objet **Secret**:

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

3. Pour utiliser le secret dans une dosette :

   a. Mettez à jour le compte de service du pod pour référencer le secret, comme indiqué dans la section "Comprendre comment créer des secrets".

b. Créer le pod, qui consomme le secret sous forme de variable d'environnement ou de fichier (à l'aide d'un volume **secret**), comme indiqué dans la section "Comprendre comment créer des secrets".

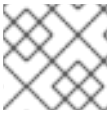
Ressources supplémentaires

- Pour plus d'informations sur l'utilisation des secrets dans les modules, voir Comment créer des secrets.
- Pour plus d'informations sur la demande de jetons de compte de service lié, voir Utilisation de jetons de compte de service lié
- Pour plus d'informations sur la création de comptes de service, voir Comprendre et créer des comptes de service.

2.7.2.4. Création d'un secret d'authentification de base

En tant qu'administrateur, vous pouvez créer un secret d'authentification de base, qui vous permet de stocker les informations d'identification nécessaires à l'authentification de base. Lorsque vous utilisez ce type de secret, le paramètre **data** de l'objet **Secret** doit contenir les clés suivantes encodées au format base64 :

- **username**le nom d'utilisateur pour l'authentification
- **password**le mot de passe ou le jeton d'authentification



NOTE

Vous pouvez utiliser le paramètre **stringData** pour utiliser un contenu en texte clair.

Procédure

1. Créer un objet **Secret** dans un fichier YAML sur un nœud de plan de contrôle :

Exemple d'objet secret

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth (1)
data:
stringData: (2)
  username: admin
  password: t0p-Secret
```

(1) Spécifie un secret d'authentification de base.

(2) Spécifie les valeurs d'authentification de base à utiliser.

2. La commande suivante permet de créer l'objet **Secret**:

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

3. Pour utiliser le secret dans une dosette :

    a. Mettez à jour le compte de service du pod pour référencer le secret, comme indiqué dans la section "Comprendre comment créer des secrets".

    b. Créer le pod, qui consomme le secret sous forme de variable d'environnement ou de fichier (à l'aide d'un volume **secret**), comme indiqué dans la section "Comprendre comment créer des secrets".

Ressources supplémentaires

- Pour plus d'informations sur l'utilisation des secrets dans les modules, voir Comment créer des secrets.

2.7.2.5. Création d'un secret d'authentification SSH

En tant qu'administrateur, vous pouvez créer un secret d'authentification SSH, qui vous permet de stocker les données utilisées pour l'authentification SSH. Lorsque vous utilisez ce type de secret, le paramètre **data** de l'objet **Secret** doit contenir l'identifiant SSH à utiliser.

Procédure

1. Créer un objet **Secret** dans un fichier YAML sur un nœud de plan de contrôle :

    Exemple secret objet :

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-ssh-auth
type: kubernetes.io/ssh-auth (1)
data:
  ssh-privatekey: | (2)
        MIIEpQIBAAKCAQEAulqb/Y ...
```

    (1) Spécifie un secret d'authentification SSH.

    (2) Spécifie la paire clé/valeur SSH en tant qu'identifiant SSH à utiliser.

2. La commande suivante permet de créer l'objet **Secret**:

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

3. Pour utiliser le secret dans une dosette :

    a. Mettez à jour le compte de service du pod pour référencer le secret, comme indiqué dans la section "Comprendre comment créer des secrets".

    b. Créer le pod, qui consomme le secret sous forme de variable d'environnement ou de fichier (à l'aide d'un volume **secret**), comme indiqué dans la section "Comprendre comment créer des secrets".

Ressources supplémentaires

- Comprendre comment créer des secrets.

2.7.2.6. Création d'un secret de configuration Docker

En tant qu'administrateur, vous pouvez créer un secret de configuration Docker, qui vous permet de stocker les informations d'identification pour accéder à un registre d'images de conteneurs.

- **kubernetes.io/dockercfg**. Utilisez ce type de secret pour stocker votre fichier de configuration Docker local. Le paramètre **data** de l'objet **secret** doit contenir le contenu d'un fichier **.dockercfg** encodé au format base64.

- **kubernetes.io/dockerconfigjson**. Utilisez ce type de secret pour stocker votre fichier JSON de configuration Docker local. Le paramètre **data** de l'objet **secret** doit contenir le contenu d'un fichier **.docker/config.json** encodé au format base64.

Procédure

1. Créer un objet **Secret** dans un fichier YAML sur un nœud de plan de contrôle.

Exemple de configuration Docker secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-cfg
  namespace: my-project
type: kubernetes.io/dockerconfig (1)
data:

.dockerconfig:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cgYXV
0aCBrZXlzCg== (2)
```

(1) Spécifie que le secret utilise un fichier de configuration Docker.

(2) La sortie d'un fichier de configuration Docker encodé en base64

Exemple de configuration Docker JSON secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-docker-json
  namespace: my-project
type: kubernetes.io/dockerconfig (1)
data:

.dockerconfigjson:bm5ubm5ubm5ubm5ubm5ubm5ubm5ubmdnZ2dnZ2dnZ2dnZ2dnZ2dnZ2cg
YXV0aCBrZXlzCg== (2)
```

(1) Spécifie que le secret utilise un fichier JSON de configuration Docker.

(2) La sortie d'un fichier JSON de configuration Docker encodé en base64

2. La commande suivante permet de créer l'objet **Secret**

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

3. Pour utiliser le secret dans une dosette :

   a. Mettez à jour le compte de service du pod pour référencer le secret, comme indiqué dans la section "Comprendre comment créer des secrets".

   b. Créer le pod, qui consomme le secret sous forme de variable d'environnement ou de fichier (à l'aide d'un volume **secret**), comme indiqué dans la section "Comprendre comment créer des secrets".

Ressources supplémentaires

- Pour plus d'informations sur l'utilisation des secrets dans les modules, voir Comment créer des secrets.

2.7.3. Comprendre comment mettre à jour les secrets

Lorsque vous modifiez la valeur d'un secret, la valeur (utilisée par un pod déjà en cours d'exécution) ne change pas dynamiquement. Pour modifier un secret, vous devez supprimer le module d'origine et en créer un nouveau (éventuellement avec un PodSpec identique).

La mise à jour d'un secret suit le même processus que le déploiement d'une nouvelle image de conteneur. Vous pouvez utiliser la commande **kubectl rolling-update**.

La valeur **resourceVersion** d'un secret n'est pas spécifiée lorsqu'il est référencé. Par conséquent, si un secret est mis à jour au moment où les modules démarrent, la version du secret utilisée pour le module n'est pas définie.

NOTE

Actuellement, il n'est pas possible de vérifier la version de la ressource d'un objet secret qui a été utilisé lors de la création d'un pod. Il est prévu que les modules communiquent cette information, de sorte qu'un contrôleur puisse redémarrer ceux qui utilisent un ancien **resourceVersion**. Dans l'intervalle, ne mettez pas à jour les données des secrets existants, mais créez-en de nouveaux avec des noms distincts.

2.7.4. Créer et utiliser des secrets

En tant qu'administrateur, vous pouvez créer un jeton de service secret. Cela vous permet de distribuer un jeton de compte de service aux applications qui doivent s'authentifier auprès de l'API.

Procédure

1. Créez un compte de service dans votre espace de noms en exécutant la commande suivante :

```
$ oc create sa <service_account_name> -n <your_namespace>
```

2. Enregistrez l'exemple YAML suivant dans un fichier nommé **service-account-token-secret.yaml**. L'exemple inclut une configuration d'objet **Secret** que vous pouvez utiliser pour générer un jeton de compte de service :

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name> 1
  annotations:
    kubernetes.io/service-account.name: "sa-name" 2
type: kubernetes.io/service-account-token 3
```

1. Remplacez **<secret_name>** par le nom de votre jeton de service secret.
2. Spécifie un nom de compte de service existant. Si vous créez les objets **ServiceAccount** et **Secret**, créez d'abord l'objet **ServiceAccount**.
3. Spécifie un type de secret de jeton de compte de service.

3. Générer le jeton de compte de service en appliquant le fichier :

```
$ oc apply -f service-account-token-secret.yaml
```

4. Obtenez le jeton du compte de service à partir du secret en exécutant la commande suivante :

```
oc get secret <sa_token_secret> -o jsonpath='{.data.token}' | base64 --decode) 1
```

Exemple de sortie

```
ayJhbGciOiJSUzI1NiIsImtpZCI6IklOb2dtck1qZ3hCSWpoNnh5YnZhSE9QMkk3YnRZMVZoclFf
QTZfRFp1YlUifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5
pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJkZWZhdWx0Iiwia3ViZXJuZXRlcy5pby9zZX
J2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6ImJ1aWxkZXItdG9rZW4tdHZrbnIiLCJrdWJlcm5l
dGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1lIjoiYnVpbGRlciIsImt1
YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjNmZGU
2MGZmLTA1NGYtNDkyZi04YzhjLTNlZjE0NDk3MmFmNyIsInN1YiI6InN5c3RlbTpzZXJ2aWNl
YWNjb3VudDpkZWZhdWx0OmJ1aWxkZXIifQ.OmqFTDuMHC_lYvvEUrjr1x453hlEEHYcxS9VK
SzmRkP1SiVZWPNPkTWlfNRp6bIUZD3U6aN3N7dMSN0eI5hu36xPgpKTdvuckKLTCnelMx6c
xOdAbrcw1mCmOClNscwjS1KO1kzMtYnnq8rXHiMJELsNlhnRyyIXRTtNBsy4t64T3283s3SLsa
ncyx0gy0ujx-Ch3uKAKdZi5iT-I8jnnQ-ds5THDs2h65RJhgglQEmSxpHrLGZFmyHAQI-
_SjvmHZPXEc482x3SkaQHNLqpmrpJorNqh1M8ZHKzlujhZgVooMvJmWPXTb2vnvi3DGn2XI-
hZxI1yD2yGH1RBpYUHA
```

1. Remplacez <sa_token_secret> par le nom de votre jeton de service secret.

5. Utilisez votre jeton de compte de service pour vous authentifier auprès de l'API de votre cluster :

```
$ curl -X GET <openshift_cluster_api> --header "Authorization : Bearer <token>" 1 2
```

1. Remplacez **<openshift_cluster_api>** par l'API de cluster OpenShift.
2. Remplacez **<token>** par le jeton de compte de service qui apparaît dans la commande précédente.

2.7.5. Utilisation de certificats signés avec des secrets

Pour sécuriser la communication avec votre service, vous pouvez configurer OpenShift Container Platform pour générer un couple certificat/clé de service signé que vous pouvez ajouter à un secret dans un projet.

Un site *service serving certificate secret* est destiné à prendre en charge des applications intergicielles complexes qui nécessitent des certificats prêts à l'emploi. Il possède les mêmes paramètres que les certificats de serveur générés par l'outil d'administration pour les nœuds et les maîtres.

Service Pod spec configuré pour un service servant des certificats secrets.

```
apiVersion: v1
kind: Service
metadata:
  name: registry
  annotations:
    service.beta.openshift.io/serving-cert-secret-name: registry-cert (1)
# ...
```

(1) Spécifier le nom du certificat

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

L'algorithme de signature pour cette fonctionnalité est **x509.SHA256WithRSA**. Pour effectuer une rotation manuelle, supprimez le secret généré. Un nouveau certificat est créé.

2.7.5.1. Génération de certificats signés à utiliser avec des secrets

Pour utiliser une paire certificat/clé de service signée avec un pod, créez ou modifiez le service pour ajouter l'annotation **service.beta.openshift.io/serving-cert-secret-name**, puis ajoutez le secret au pod.

Procédure

Pour créer un site *service serving certificate secret* :

1. Modifiez la spécification **Pod** pour votre service.
2. Ajoutez l'annotation **service.beta.openshift.io/serving-cert-secret-name** avec le nom que vous voulez utiliser pour votre secret.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
  annotations:
     service.beta.openshift.io/serving-cert-secret-name: my-cert (1)
spec:
  selector:
    app: MyApp
  ports:
```

```
  - protocol: TCP
    port: 80
    targetPort: 9376
```

Le certificat et la clé sont au format PEM et sont stockés respectivement dans **tls.crt** et **tls.key**.

3. Créer le service :

```
oc create -f <nom-de-fichier>.yaml
```

4. Consultez le secret pour vous assurer qu'il a bien été créé :

   a. Afficher une liste de tous les secrets :

   ```
   $ oc get secrets
   ```

   Exemple de sortie

   ```
   NAME                  TYPE                                  DATA      AGE
   my-cert               kubernetes.io/tls                     2         9m
   ```

   b. Voir les détails de votre secret :

   ```
   $ oc describe secret my-cert
   ```

   Exemple de sortie

   ```
   Name:         my-cert
   Namespace:    openshift-console
   Labels:       <none>
   Annotations:  service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z
                 service.beta.openshift.io/originating-service-name: my-service
                 service.beta.openshift.io/originating-service-uid: 640f0ec3-afc2-4380-bf31-
   a8c784846a11
                 service.beta.openshift.io/expiry: 2023-03-08T23:22:40Z

   Type:  kubernetes.io/tls

   Data
   ====
   tls.key:  1679 bytes
   tls.crt:  2595 bytes
   ```

5. Modifiez votre spécification **Pod** avec ce secret.

```
apiVersion: v1
kind: Pod
metadata:
  name: my-service-pod
spec:
  containers:
  - name: mypod
    image: redis
    volumeMounts:
```

```
    - name: foo
      mountPath: "/etc/foo"
  volumes:
  - name: foo
    secret:
      secretName: my-cert
      items:
      - key: username
        path: my-group/my-username
        mode: 511
```

Lorsqu'il sera disponible, votre pod s'exécutera. Le certificat sera valable pour le nom DNS du service interne, **<service.name>.<service.namespace>.svc**.

La paire certificat/clé est automatiquement remplacée lorsqu'elle est proche de l'expiration. La date d'expiration est indiquée dans l'annotation **service.beta.openshift.io/expiry** sur le secret, au format RFC3339.

NOTE

Dans la plupart des cas, le nom DNS du service **<service.name>.<service.namespace>.svc** n'est pas routable de l'extérieur. L'utilisation principale de **<service.name>.<service.namespace>.svc** est pour la communication à l'intérieur d'un cluster ou d'un service, et avec des itinéraires de re-cryptage.

2.7.6. Secrets de dépannage

Si la génération d'un certificat de service échoue avec (l'annotation **service.beta.openshift.io/serving-cert-generation-error** du service contient) :

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

Le service qui a généré le certificat n'existe plus, ou a un autre **serviceUID**. Vous devez forcer la régénération des certificats en supprimant l'ancien secret et en supprimant les annotations suivantes sur le service **service.beta.openshift.io/serving-cert-generation-error**, **service.beta.openshift.io/serving-cert-generation-error-num**:

1. Supprimer le secret :

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

2. Effacer les annotations :

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

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

NOTE

La commande de suppression d'une annotation comporte une adresse **-** après le nom de l'annotation à supprimer.

2.8. CRÉER ET UTILISER DES CARTES DE CONFIGURATION

Les sections suivantes définissent les cartes de configuration et expliquent comment les créer et les utiliser.

2.8.1. Comprendre les cartes de configuration

De nombreuses applications doivent être configurées à l'aide d'une combinaison de fichiers de configuration, d'arguments de ligne de commande et de variables d'environnement. Dans OpenShift Container Platform, ces artefacts de configuration sont découplés du contenu de l'image afin de maintenir la portabilité des applications conteneurisées.

L'objet **ConfigMap** fournit des mécanismes pour injecter des conteneurs avec des données de configuration tout en gardant les conteneurs agnostiques de OpenShift Container Platform. Une carte de configuration peut être utilisée pour stocker des informations fines comme des propriétés individuelles ou des informations grossières comme des fichiers de configuration entiers ou des blobs JSON.

L'objet API **ConfigMap** contient des paires clé-valeur de données de configuration qui peuvent être consommées dans des pods ou utilisées pour stocker des données de configuration pour des composants système tels que des contrôleurs. Par exemple, l'objet

ConfigMap Définition de l'objet

```
kind: ConfigMap
apiVersion: v1
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: example-config
  namespace: default
data: 1
  example.property.1: hello
  example.property.2: world
  example.property.file: |-
    property.1=value-1
    property.2=value-2
    property.3=value-3
binaryData:
  bar: L3Jvb3QvMTAw 2
```

1 1 Contient les données de configuration.

2 Pointe vers un fichier qui contient des données non-UTF8, par exemple un fichier keystore Java binaire. Saisissez les données du fichier en base 64.

NOTE

Vous pouvez utiliser le champ **binaryData** lorsque vous créez une carte de configuration à partir d'un fichier binaire, tel qu'une image.

Les données de configuration peuvent être consommées dans les pods de différentes manières. Une carte de configuration peut être utilisée pour :

- Remplir les valeurs des variables d'environnement dans les conteneurs
- Définir les arguments de la ligne de commande dans un conteneur
- Remplir les fichiers de configuration d'un volume

Les utilisateurs et les composants du système peuvent stocker des données de configuration dans une carte de configuration.

Une carte de configuration est similaire à un secret, mais elle est conçue pour faciliter le travail avec des chaînes de caractères qui ne contiennent pas d'informations sensibles.

Restrictions de la carte de configuration
A config map must be created before its contents can be consumed in pods.

Les contrôleurs peuvent être écrits de manière à tolérer les données de configuration manquantes. Consultez les composants individuels configurés à l'aide de cartes de configuration au cas par cas.

ConfigMap objects reside in a project.

Ils ne peuvent être référencés que par les pods du même projet.

The Kubelet only supports the use of a config map for pods it gets from the API server.

Cela inclut tous les pods créés en utilisant le CLI, ou indirectement à partir d'un contrôleur de réplication. Cela n'inclut pas les pods créés en utilisant le drapeau **--manifest-url**, le drapeau **--config** ou l'API REST du nœud OpenShift Container Platform, car ce ne sont pas des moyens courants de créer des pods.

2.8.2. Créer une carte de configuration dans la console web de OpenShift Container Platform

Vous pouvez créer une carte de configuration dans la console web d'OpenShift Container Platform.

Procédure

- Pour créer une carte de configuration en tant qu'administrateur de cluster :
  1. Dans la perspective de l'administrateur, sélectionnez **Workloads → Config Maps**.
  2. En haut à droite de la page, sélectionnez **Create Config Map**.
  3. Entrez le contenu de votre carte de configuration.
  4. Sélectionnez **Create**.
- Pour créer une carte de configuration en tant que développeur :
  1. Dans la perspective du développeur, sélectionnez **Config Maps**.

2. En haut à droite de la page, sélectionnez **Create Config Map**.
3. Entrez le contenu de votre carte de configuration.
4. Sélectionnez **Create**.

2.8.3. Création d'une carte de configuration à l'aide de la CLI

Vous pouvez utiliser la commande suivante pour créer une carte de configuration à partir de répertoires, de fichiers spécifiques ou de valeurs littérales.

Procédure

- Créer une carte de configuration :

```
oc create configmap <configmap_name> [options] $ oc create configmap
<configmap_name> [options]
```

2.8.3.1. Création d'une carte de configuration à partir d'un répertoire

Vous pouvez créer une carte de configuration à partir d'un répertoire. Cette méthode vous permet d'utiliser plusieurs fichiers d'un répertoire pour créer une carte de configuration.

Procédure

L'exemple de procédure suivant explique comment créer une carte de configuration à partir d'un répertoire.

1. Commencez par un répertoire contenant quelques fichiers qui contiennent déjà les données avec lesquelles vous voulez remplir une carte de configuration :

```
$ ls example-files
```

Exemple de sortie

```
game.properties
ui.properties
```

```
$ cat example-files/game.properties
```

Exemple de sortie

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
secret.code.passphrase=UUDDLRLRBABAS
secret.code.allowed=true
secret.code.lives=30
```

```
$ cat example-files/ui.properties
```

Exemple de sortie

```
color.good=purple
color.bad=yellow
allow.textmode=true
how.nice.to.look=fairlyNice
```

2. Créez une carte de configuration contenant le contenu de chaque fichier de ce répertoire en entrant la commande suivante :

```
$ oc create configmap game-config \
    --from-file=example-files/
```

Lorsque l'option **--from-file** pointe vers un répertoire, chaque fichier directement dans ce répertoire est utilisé pour remplir une clé dans la carte de configuration, où le nom de la clé est le nom du fichier, et la valeur de la clé est le contenu du fichier.

Par exemple, la commande précédente crée la carte de configuration suivante :

```
$ oc describe configmaps game-config
```

Exemple de sortie

```
Name:           game-config
Namespace:      default
Labels:         <none>
Annotations:    <none>

Data

game.properties:        158 bytes
ui.properties:          83 bytes
```

Vous pouvez voir que les deux clés de la carte sont créées à partir des noms de fichiers du répertoire spécifié dans la commande. Le contenu de ces clés pouvant être volumineux, la sortie de **oc describe** n'indique que les noms des clés et leur taille.

3. Entrez la commande **oc get** pour l'objet avec l'option **-o** pour voir les valeurs des clés :

```
$ oc get configmaps game-config -o yaml
```

Exemple de sortie

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
```

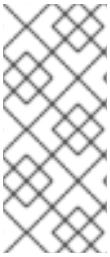
```
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:34:05Z
  name: game-config
  namespace: default
  resourceVersion: "407"
  selflink: /api/v1/namespaces/default/configmaps/game-config
  uid: 30944725-d66e-11e5-8cd0-68f728db1985
```

2.8.3.2. Création d'une carte de configuration à partir d'un fichier

Vous pouvez créer une carte de configuration à partir d'un fichier.

Procédure

L'exemple de procédure suivant explique comment créer une carte de configuration à partir d'un fichier.



NOTE

Si vous créez une carte de configuration à partir d'un fichier, vous pouvez inclure des fichiers contenant des données non-UTF8 qui sont placées dans ce champ sans corrompre les données non-UTF8. OpenShift Container Platform détecte les fichiers binaires et encode de manière transparente le fichier en tant que **MIME**. Sur le serveur, la charge utile **MIME** est décodée et stockée sans corrompre les données.

Vous pouvez transmettre l'option **--from-file** plusieurs fois à l'interface de programmation. L'exemple suivant donne des résultats équivalents à ceux de l'exemple de création à partir de répertoires.

1. Créer une carte de configuration en spécifiant un fichier spécifique :

```
$ oc create configmap game-config-2 \
    --from-file=example-files/game.properties \
    --from-file=example-files/ui.properties
```

2. Vérifier les résultats :

```
$ oc get configmaps game-config-2 -o yaml
```

Exemple de sortie

```
apiVersion: v1
data:
  game.properties: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
```

```
    secret.code.lives=30
  ui.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
    how.nice.to.look=fairlyNice
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:52:05Z
  name: game-config-2
  namespace: default
  resourceVersion: "516"
  selflink: /api/v1/namespaces/default/configmaps/game-config-2
  uid: b4952dc3-d670-11e5-8cd0-68f728db1985
```

Vous pouvez spécifier la clé à définir dans une carte de configuration pour le contenu importé à partir d'un fichier. Cette clé peut être définie en passant une expression **key=value** à l'option **--from-file**. Par exemple :

1. Créer une carte de configuration en spécifiant une paire clé-valeur :

```
$ oc create configmap game-config-3 \
    --from-file=game-special-key=example-files/game.properties
```

2. Vérifier les résultats :

```
$ oc get configmaps game-config-3 -o yaml
```

Exemple de sortie

```
apiVersion: v1
data:
  game-special-key: |- (1)
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
    secret.code.passphrase=UUDDLRLRBABAS
    secret.code.allowed=true
    secret.code.lives=30
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T18:54:22Z
  name: game-config-3
  namespace: default
  resourceVersion: "530"
  selflink: /api/v1/namespaces/default/configmaps/game-config-3
  uid: 05f8da22-d671-11e5-8cd0-68f728db1985
```

(1) Il s'agit de la clé que vous avez définie à l'étape précédente.

2.8.3.3. Création d'une carte de configuration à partir de valeurs littérales

Vous pouvez fournir des valeurs littérales pour une carte de configuration.

Procédure

L'option **--from-literal** utilise une syntaxe **key=value** qui permet de fournir des valeurs littérales directement sur la ligne de commande.

1. Créer une carte de configuration en spécifiant une valeur littérale :

```
$ oc create configmap special-config \
    --from-literal=special.how=very \
    --from-literal=special.type=charm
```

2. Vérifier les résultats :

```
$ oc get configmaps special-config -o yaml
```

Exemple de sortie

```
apiVersion: v1
data:
  special.how: very
  special.type: charm
kind: ConfigMap
metadata:
  creationTimestamp: 2016-02-18T19:14:38Z
  name: special-config
  namespace: default
  resourceVersion: "651"
  selflink: /api/v1/namespaces/default/configmaps/special-config
  uid: dadce046-d673-11e5-8cd0-68f728db1985
```

2.8.4. Cas d'utilisation : Consommer des cartes de configuration dans les pods

Les sections suivantes décrivent quelques cas d'utilisation des objets **ConfigMap** dans les pods.

2.8.4.1. Remplir les variables d'environnement dans les conteneurs à l'aide de cartes de configuration

Les cartes de configuration peuvent être utilisées pour remplir des variables d'environnement individuelles dans des conteneurs ou pour remplir des variables d'environnement dans des conteneurs à partir de toutes les clés qui forment des noms de variables d'environnement valides.

Prenons l'exemple de la carte de configuration suivante :

ConfigMap avec deux variables d'environnement

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config 1
  namespace: default 2
```

```
data:
  special.how: very 3
  special.type: charm 4
```

1 Nom de la carte de configuration.

2 Le projet dans lequel se trouve la carte de configuration. Les cartes de configuration ne peuvent être référencées que par les pods du même projet.

3 4 Variables d'environnement à injecter.

ConfigMap avec une variable d'environnement

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-config 1
  namespace: default
data:
  log_level: INFO 2
```

1 Nom de la carte de configuration.

2 Variable d'environnement à injecter.

Procédure

- Vous pouvez consommer les clés de ce **ConfigMap** dans un pod en utilisant les sections **configMapKeyRef**.

  Exemple de spécification Pod configurée pour injecter des variables d'environnement spécifiques

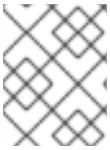
```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env: 1
        - name: SPECIAL_LEVEL_KEY 2
          valueFrom:
            configMapKeyRef:
              name: special-config 3
              key: special.how 4
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config 5
```

```
            key: special.type 6
            optional: true 7
        envFrom: 8
          - configMapRef:
              name: env-config 9
    restartPolicy: Never
```

1. Stanza pour extraire les variables d'environnement spécifiées d'un site **ConfigMap**.
2. Nom de la variable d'environnement du pod dans laquelle vous injectez la valeur de la clé.
3. 5 Nom de l'adresse **ConfigMap** à partir de laquelle des variables d'environnement spécifiques doivent être extraites.
4. 6 Variable d'environnement à extraire de **ConfigMap**.
7. Rend la variable d'environnement facultative. En tant qu'optionnel, le pod sera démarré même si les **ConfigMap** et les clés spécifiés n'existent pas.
8. Stanza pour extraire toutes les variables d'environnement d'un site **ConfigMap**.
9. Nom de l'adresse **ConfigMap** à partir de laquelle toutes les variables d'environnement doivent être extraites.

Lorsque ce module est exécuté, les journaux du module incluent la sortie suivante :

```
SPECIAL_LEVEL_KEY=very
log_level=INFO
```



NOTE

SPECIAL_TYPE_KEY=charm n'est pas listé dans l'exemple de sortie car **optional: true** est activé.

2.8.4.2. Définition des arguments de ligne de commande pour les commandes de conteneurs avec les cartes de configuration

Une carte de configuration peut également être utilisée pour définir la valeur des commandes ou des arguments dans un conteneur. Pour ce faire, on utilise la syntaxe de substitution de Kubernetes **$(VAR_NAME)**. Considérons la carte de configuration suivante :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

Procédure

- Pour injecter des valeurs dans une commande dans un conteneur, vous devez consommer les

clés que vous souhaitez utiliser comme variables d'environnement, comme dans le cas d'utilisation de ConfigMaps dans des variables d'environnement. Vous pouvez ensuite y faire référence dans la commande d'un conteneur à l'aide de la syntaxe **$(VAR_NAME)**.

Exemple de spécification Pod configurée pour injecter des variables d'environnement spécifiques

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "echo $(SPECIAL_LEVEL_KEY) $(SPECIAL_TYPE_KEY)" ]
1
      env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.how
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: special.type
  restartPolicy: Never
```

1 Injectez les valeurs dans une commande dans un conteneur en utilisant les clés que vous souhaitez utiliser comme variables d'environnement.

Lorsque ce module est exécuté, la sortie de la commande echo exécutée dans le conteneur test-container est la suivante :

```
very charm
```

2.8.4.3. Injecter du contenu dans un volume en utilisant des cartes de configuration

Vous pouvez injecter du contenu dans un volume en utilisant des cartes de configuration.

Exemple ConfigMap ressource personnalisée (CR)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  special.how: very
  special.type: charm
```

Procédure

Vous disposez de plusieurs options pour injecter du contenu dans un volume à l'aide de cartes de configuration.

- La façon la plus simple d'injecter du contenu dans un volume à l'aide d'une carte de configuration consiste à remplir le volume avec des fichiers dont la clé est le nom et le contenu la valeur de la clé :

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "cat", "/etc/config/special.how" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config 1
  restartPolicy: Never
```

1 Fichier contenant la clé.

Lorsque ce pod est exécuté, la sortie de la commande cat sera la suivante :

```
very
```

- Vous pouvez également contrôler les chemins à l'intérieur du volume où les clés de configuration sont projetées :

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "cat", "/etc/config/path/to/special-key" ]
      volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: special-config
        items:
```

```
      - key: special.how
        path: path/to/special-key 1
  restartPolicy: Never
```

1 Chemin d'accès à la clé de la carte de configuration.

Lorsque ce pod est exécuté, la sortie de la commande cat sera la suivante :

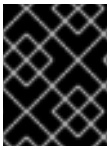
```
very
```

2.9. UTILISATION DE PLUGINS DE PÉRIPHÉRIQUES POUR ACCÉDER À DES RESSOURCES EXTERNES AVEC DES PODS

Les plugins de périphériques vous permettent d'utiliser un type de périphérique particulier (GPU, InfiniBand, ou d'autres ressources informatiques similaires qui nécessitent une initialisation et une configuration spécifiques au fournisseur) dans votre pod OpenShift Container Platform sans avoir besoin d'écrire du code personnalisé.

2.9.1. Comprendre les plugins d'appareils

Le plugin device fournit une solution cohérente et portable pour consommer des périphériques matériels à travers les clusters. Le plugin device fournit un support pour ces dispositifs à travers un mécanisme d'extension, qui rend ces dispositifs disponibles pour les conteneurs, fournit des contrôles de santé de ces dispositifs, et les partage de manière sécurisée.



IMPORTANT

OpenShift Container Platform prend en charge l'API du plugin de périphérique, mais les conteneurs de plugin de périphérique sont pris en charge par des fournisseurs individuels.

Un plugin de périphérique est un service gRPC fonctionnant sur les nœuds (en dehors du site **kubelet**) et chargé de gérer des ressources matérielles spécifiques. Tout plugin de périphérique doit prendre en charge les appels de procédure à distance (RPC) suivants :

```
service DevicePlugin {
    // GetDevicePluginOptions returns options to be communicated with Device
    // Manager
    rpc GetDevicePluginOptions(Empty) returns (DevicePluginOptions) {}

    // ListAndWatch returns a stream of List of Devices
    // Whenever a Device state change or a Device disappears, ListAndWatch
    // returns the new list
    rpc ListAndWatch(Empty) returns (stream ListAndWatchResponse) {}

    // Allocate is called during container creation so that the Device
    // Plug-in can run device specific operations and instruct Kubelet
    // of the steps to make the Device available in the container
    rpc Allocate(AllocateRequest) returns (AllocateResponse) {}

    // PreStartcontainer is called, if indicated by Device Plug-in during
    // registration phase, before each container start. Device plug-in
    // can run device specific operations such as reseting the device
```

```
    // before making devices available to the container
    rpc PreStartcontainer(PreStartcontainerRequest) returns (PreStartcontainerResponse) {}
}
```

Exemples de plugins d'appareils

- Plugin de périphérique GPU Nvidia pour système d'exploitation basé sur COS
- Plugin officiel de périphérique GPU de Nvidia
- Solarflare device plugin
- Plugins de périphériques KubeVirt : vfio et kvm
- Plugin de périphérique Kubernetes pour les cartes IBM Crypto Express (CEX)

NOTE

Pour faciliter la mise en œuvre de la référence du plugin de périphérique, il existe un plugin de périphérique fictif dans le code du gestionnaire de périphériques : ***vendor/k8s.io/kubernetes/pkg/kubelet/cm/deviceplugin/device_plugin_stub.go***.

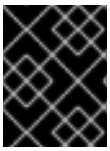
2.9.1.1. Méthodes de déploiement d'un module d'extension de dispositif

- Les ensembles de démons constituent l'approche recommandée pour les déploiements de plugins de périphériques.
- Au démarrage, le plugin de périphérique essaiera de créer un socket de domaine UNIX à l'adresse ***/var/lib/kubelet/device-plugin/*** sur le nœud pour servir les RPC du gestionnaire de périphériques.
- Comme les plugins de périphérique doivent gérer les ressources matérielles, l'accès au système de fichiers de l'hôte, ainsi que la création de sockets, ils doivent être exécutés dans un contexte de sécurité privilégié.
- Des détails plus spécifiques concernant les étapes de déploiement peuvent être trouvés avec chaque implémentation de plugin d'appareil.

2.9.2. Comprendre le gestionnaire de périphériques

Device Manager fournit un mécanisme pour annoncer les ressources matérielles spécialisées des nœuds à l'aide de plugins connus sous le nom de "device plugins".

Vous pouvez annoncer du matériel spécialisé sans avoir à modifier le code en amont.



IMPORTANT

OpenShift Container Platform prend en charge l'API du plugin de périphérique, mais les conteneurs de plugin de périphérique sont pris en charge par des fournisseurs individuels.

Le gestionnaire de dispositifs annonce les dispositifs en tant que **Extended Resources**. Les pods utilisateurs peuvent consommer les dispositifs annoncés par le Device Manager en utilisant le même mécanisme **Limit/Request** que celui utilisé pour demander n'importe quel autre **Extended Resource**.

Au démarrage, le plugin de périphérique s'enregistre auprès du gestionnaire de périphériques en

invoquant **Register** sur l'adresse */var/lib/kubelet/device-plugins/kubelet.sock* et lance un service gRPC à l'adresse */var/lib/kubelet/device-plugins/<plugin>.sock* pour répondre aux demandes du gestionnaire d'appareils.

Lors du traitement d'une nouvelle demande d'enregistrement, le gestionnaire de périphérique invoque l'appel de procédure à distance (RPC) **ListAndWatch** auprès du service d'extension de périphérique. En réponse, le gestionnaire de périphérique obtient une liste d'objets **Device** du plugin via un flux gRPC. Le gestionnaire de périphérique surveillera le flux pour les nouvelles mises à jour du module d'extension. Du côté du plugin, le plugin gardera également le flux ouvert et chaque fois qu'il y a un changement dans l'état de l'un des appareils, une nouvelle liste d'appareils est envoyée au gestionnaire de périphériques via la même connexion de flux.

Lors du traitement d'une nouvelle demande d'admission d'un pod, Kubelet transmet la demande **Extended Resources** au Device Manager pour l'attribution d'un dispositif. Le gestionnaire de périphériques vérifie dans sa base de données si un plugin correspondant existe ou non. Si le plugin existe et qu'il y a des dispositifs allouables libres ainsi que dans le cache local, **Allocate** RPC est invoqué au niveau de ce plugin de dispositif particulier.

En outre, les plugins d'appareil peuvent également effectuer plusieurs autres opérations spécifiques à l'appareil, telles que l'installation du pilote, l'initialisation de l'appareil et la réinitialisation de l'appareil. Ces fonctionnalités varient d'une implémentation à l'autre.

2.9.3. Activation du gestionnaire de périphériques

Permettre au gestionnaire de périphériques de mettre en œuvre un plugin de périphérique pour annoncer du matériel spécialisé sans aucune modification du code en amont.

Device Manager fournit un mécanisme pour annoncer les ressources matérielles spécialisées des nœuds à l'aide de plugins connus sous le nom de "device plugins".

1. Obtenez l'étiquette associée au CRD statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante. Effectuez l'une des étapes suivantes :

   a. Voir la configuration de la machine :

   ```
   # oc describe machineconfig <name>
   ```

   Par exemple :

   ```
   # oc describe machineconfig 00-worker
   ```

   Exemple de sortie

   ```
   Name:         00-worker
   Namespace:
   Labels:       machineconfiguration.openshift.io/role=worker 1
   ```

   1 Étiquette requise pour le gestionnaire de périphériques.

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration.

Exemple de configuration pour un gestionnaire de périphérique CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: devicemgr (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io: devicemgr (2)
  kubeletConfig:
    feature-gates:
      - DevicePlugins=true (3)
```

1. Attribuer un nom au CR.
2. Saisissez l'étiquette du pool de configuration de la machine.
3. Définissez **DevicePlugins** sur "true".

2. Créer le gestionnaire de périphériques :

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

Exemple de sortie

```
kubeletconfig.machineconfiguration.openshift.io/devicemgr created
```

3. Assurez-vous que le gestionnaire de périphériques a bien été activé en confirmant que l'option ***/var/lib/kubelet/device-plugins/kubelet.sock*** est créé sur le nœud. Il s'agit du socket de domaine UNIX sur lequel le serveur gRPC du Device Manager écoute les nouveaux enregistrements de plugins. Ce fichier sock est créé au démarrage de la Kubelet uniquement si Device Manager est activé.

2.10. PRISE EN COMPTE DE LA PRIORITÉ DES PODS DANS LES DÉCISIONS D'ORDONNANCEMENT DES PODS

Vous pouvez activer la priorité et la préemption des pods dans votre cluster. La priorité des pods indique l'importance d'un pod par rapport aux autres pods et met les pods en file d'attente en fonction de cette priorité. La préemption des pods permet au cluster d'expulser, ou de préempter, les pods de priorité inférieure afin que les pods de priorité supérieure puissent être planifiés s'il n'y a pas d'espace disponible sur un nœud approprié. La priorité des pods affecte également l'ordre de planification des pods et l'ordre d'expulsion en cas d'absence de ressources sur le nœud.

Pour utiliser la priorité et la préemption, vous devez créer des classes de priorité qui définissent le poids relatif de vos modules. Ensuite, faites référence à une classe de priorité dans la spécification du pod pour appliquer ce poids à la planification.

2.10.1. Comprendre la priorité des pods

Lorsque vous utilisez la fonctionnalité de priorité et de préemption des pods, l'ordonnanceur classe les pods en attente en fonction de leur priorité, et un pod en attente est placé devant d'autres pods en

attente ayant une priorité inférieure dans la file d'attente d'ordonnancement. Par conséquent, le pod le plus prioritaire peut être programmé plus tôt que les pods moins prioritaires si ses besoins de programmation sont satisfaits. Si un module ne peut pas être programmé, l'ordonnanceur continue à programmer d'autres modules moins prioritaires.

2.10.1.1. Classes de priorité pour les pods

Vous pouvez attribuer aux pods une classe de priorité, qui est un objet sans espace de noms définissant une correspondance entre un nom et la valeur entière de la priorité. Plus la valeur est élevée, plus la priorité est importante.

Un objet de classe de priorité peut prendre n'importe quelle valeur entière de 32 bits inférieure ou égale à 1000000000 (un milliard). Réservez les nombres supérieurs ou égaux à un milliard aux pods critiques qui ne doivent pas être préemptés ou évincés. Par défaut, OpenShift Container Platform a deux classes de priorité réservées pour les pods système critiques afin de garantir l'ordonnancement.

```
$ oc get priorityclasses
```

Exemple de sortie

```
NAME                     VALUE        GLOBAL-DEFAULT   AGE
system-node-critical     2000001000   false            72m
system-cluster-critical  2000000000   false            72m
openshift-user-critical  1000000000   false            3d13h
cluster-logging          1000000      false            29s
```

- **system-node-critical** - Cette classe de priorité a une valeur de 2000001000 et est utilisée pour tous les pods qui ne doivent jamais être expulsés d'un nœud. Les exemples de pods ayant cette classe de priorité sont **sdn-ovs**, **sdn**, et ainsi de suite. Un certain nombre de composants critiques incluent la classe de priorité **system-node-critical** par défaut, par exemple :
  - master-api
  - maître-contrôleur
  - master-etcd
  - sdn
  - sdn-ovs
  - synchronisation
- **system-cluster-critical** - Cette classe de priorité a une valeur de 2000000000 (deux milliards) et est utilisée avec les pods qui sont importants pour le cluster. Les modules de cette classe de priorité peuvent être expulsés d'un nœud dans certaines circonstances. Par exemple, les modules configurés avec la classe de priorité **system-node-critical** peuvent être prioritaires. Toutefois, cette classe de priorité garantit la planification. Les pods qui peuvent avoir cette classe de priorité sont par exemple fluentd, des composants complémentaires comme descheduler, etc. Un certain nombre de composants critiques incluent la classe de priorité **system-cluster-critical** par défaut, par exemple :
  - fluentd
  - serveur de métrologie

- déscheduler

- **openshift-user-critical** - Vous pouvez utiliser le champ **priorityClassName** avec des pods importants qui ne peuvent pas lier leur consommation de ressources et qui n'ont pas de comportement prévisible en matière de consommation de ressources. Les pods Prometheus sous les espaces de noms **openshift-monitoring** et **openshift-user-workload-monitoring** utilisent le champ **openshift-user-critical priorityClassName** . Les charges de travail de surveillance utilisent **system-critical** comme premier **priorityClass**, mais cela pose des problèmes lorsque la surveillance utilise trop de mémoire et que les nœuds ne peuvent pas les expulser. Par conséquent, la surveillance perd sa priorité pour donner de la flexibilité au planificateur, qui déplace les charges de travail lourdes pour maintenir les nœuds critiques en fonctionnement.

- **cluster-logging** - Cette priorité est utilisée par Fluentd pour s'assurer que les pods Fluentd sont programmés sur les nœuds avant les autres applications.

2.10.1.2. Noms de priorité des pods

Une fois que vous avez une ou plusieurs classes de priorité, vous pouvez créer des pods qui spécifient un nom de classe de priorité dans une spécification **Pod**. Le contrôleur d'admission des priorités utilise le champ du nom de la classe de priorité pour remplir la valeur entière de la priorité. Si la classe de priorité nommée n'est pas trouvée, le pod est rejeté.

2.10.2. Comprendre la préemption des pods

Lorsqu'un développeur crée un module, celui-ci est placé dans une file d'attente. Si le développeur a configuré le module pour une priorité ou une préemption, l'ordonnanceur sélectionne un module dans la file d'attente et tente de le programmer sur un nœud. Si l'ordonnanceur ne trouve pas d'espace sur un nœud approprié qui réponde à toutes les exigences du module, la logique de préemption est déclenchée pour le module en attente.

Lorsque l'ordonnanceur préempte un ou plusieurs pods sur un nœud, le champ **nominatedNodeName** du spec **Pod** de priorité supérieure est défini sur le nom du nœud, ainsi que sur le champ **nodename**. L'ordonnanceur utilise le champ **nominatedNodeName** pour garder une trace des ressources réservées aux pods et fournit également des informations à l'utilisateur sur les préemptions dans les clusters.

Une fois que l'ordonnanceur a préempté un module de priorité inférieure, il respecte la période de terminaison gracieuse du module. Si un autre nœud devient disponible pendant que l'ordonnanceur attend la fin du pod de priorité inférieure, l'ordonnanceur peut programmer le pod de priorité supérieure sur ce nœud. Par conséquent, les champs **nominatedNodeName** et **nodeName** de la spécification **Pod** peuvent être différents.

De même, si l'ordonnanceur préempte des pods sur un nœud et attend la fin, et qu'un pod ayant une priorité plus élevée que le pod en attente doit être programmé, l'ordonnanceur peut programmer le pod ayant la priorité la plus élevée à la place. Dans ce cas, l'ordonnanceur efface le site **nominatedNodeName** du pod en attente, rendant le pod éligible pour un autre nœud.

La préemption ne supprime pas nécessairement tous les modules de priorité inférieure d'un nœud. L'ordonnanceur peut programmer un module en attente en supprimant une partie des modules de priorité inférieure.

L'ordonnanceur ne considère un nœud pour la préemption de pods que si le pod en attente peut être programmé sur le nœud.

2.10.2.1. Classes de priorité non prioritaires (Technology Preview)

Les modules dont la politique de préemption est définie sur **Never** sont placés dans la file d'attente d'ordonnancement avant les modules de priorité inférieure, mais ils ne peuvent pas préempter d'autres modules. Un pod non préempté en attente d'ordonnancement reste dans la file d'attente d'ordonnancement jusqu'à ce que des ressources suffisantes soient disponibles et qu'il puisse être ordonnancé. Les modules non préemptés, comme les autres modules, sont soumis à l'arrêt de l'ordonnanceur. Cela signifie que si l'ordonnanceur tente sans succès de programmer ces modules, ils sont relancés à une fréquence moindre, ce qui permet à d'autres modules moins prioritaires d'être programmés avant eux.

Les pods non préemptés peuvent toujours être préemptés par d'autres pods à priorité élevée.

2.10.2.2. Préemption des pods et autres paramètres de l'ordonnanceur

Si vous activez la priorité et la préemption des pods, tenez compte des autres paramètres de votre ordonnanceur :

Priorité aux pods et budget de perturbation des pods

Un budget d'interruption de pod spécifie le nombre ou le pourcentage minimum de répliques qui doivent être opérationnelles à un moment donné. Si vous spécifiez des budgets de perturbation de pods, OpenShift Container Platform les respecte lors de la préemption de pods à un niveau de meilleur effort. L'ordonnanceur tente de préempter des pods sans violer le budget de perturbation des pods. Si aucun pod n'est trouvé, les pods de priorité inférieure peuvent être préemptés malgré leurs exigences en matière de budget de perturbation des pods.

Priorité et affinité des pods

L'affinité des pods exige qu'un nouveau pod soit programmé sur le même nœud que d'autres pods ayant la même étiquette.

Si un pod en attente a une affinité inter-pods avec un ou plusieurs pods de priorité inférieure sur un nœud, l'ordonnanceur ne peut pas préempter les pods de priorité inférieure sans violer les exigences d'affinité. Dans ce cas, l'ordonnanceur recherche un autre nœud pour planifier le module en attente. Cependant, il n'est pas garanti que l'ordonnanceur puisse trouver un nœud approprié et le module en attente peut ne pas être programmé.

Pour éviter cette situation, configurez soigneusement l'affinité des pods avec des pods de priorité égale.

2.10.2.3. Fin gracieuse des pods préemptés

Lorsqu'il préempte un module, l'ordonnanceur attend l'expiration de la période de terminaison gracieuse du module, ce qui permet au module de terminer son travail et de quitter le système. Si le module ne se termine pas à l'issue de cette période, l'ordonnanceur tue le module. Ce délai d'expiration crée un décalage entre le moment où l'ordonnanceur préempte le module et le moment où le module en attente peut être programmé sur le nœud.

Pour réduire cet écart, configurez une petite période de terminaison gracieuse pour les pods de moindre priorité.

2.10.3. Configuration de la priorité et de la préemption

Vous appliquez la priorité et la préemption des pods en créant un objet de classe de priorité et en associant les pods à la priorité à l'aide de **priorityClassName** dans vos spécifications **Pod**.

Exemple d'objet de classe de priorité

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority (1)
value: 1000000 (2)
preemptionPolicy: PreemptLowerPriority (3)
globalDefault: false (4)
description: "This priority class should be used for XYZ service pods only." (5)
```

1. Le nom de l'objet de la classe de priorité.
2. La valeur de priorité de l'objet.
3. Champ facultatif indiquant si cette classe de priorité est préemptable ou non préemptable. La politique de préemption est définie par défaut sur **PreemptLowerPriority**, ce qui permet aux pods de cette classe de priorité de préempter les pods de priorité inférieure. Si la politique de préemption est définie sur **Never**, les pods de cette classe de priorité ne sont pas préemptés.
4. Champ facultatif indiquant si cette classe de priorité doit être utilisée pour les pods sans nom de classe de priorité spécifié. Ce champ vaut **false** par défaut. Il ne peut y avoir qu'une seule classe de priorité avec **globalDefault** défini sur **true** dans le cluster. S'il n'y a pas de classe de priorité avec **globalDefault:true**, la priorité des modules sans nom de classe de priorité est zéro. L'ajout d'une classe de priorité avec **globalDefault:true** n'affecte que les modules créés après l'ajout de la classe de priorité et ne modifie pas les priorités des modules existants.
5. Chaîne de texte arbitraire facultative décrivant les pods que les développeurs doivent utiliser avec cette classe de priorité.

Procédure

Pour configurer votre cluster afin d'utiliser la priorité et la préemption :

1. Créer une ou plusieurs classes de priorité :
   a. Spécifiez un nom et une valeur pour la priorité.
   b. Il est possible de spécifier le champ **globalDefault** dans la classe de priorité et une description.
2. Créez un spec **Pod** ou modifiez les pods existants pour inclure le nom d'une classe de priorité, comme suit :

Exemple de Pod spec avec le nom de la classe de priorité

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
```

```
  image: nginx
  imagePullPolicy: IfNotPresent
priorityClassName: high-priority 1
```

1 Spécifiez la classe de priorité à utiliser avec ce pod.

3. Créer la capsule :

```
oc create -f <nom-de-fichier>.yaml
```

Vous pouvez ajouter le nom de la priorité directement à la configuration du module ou à un modèle de module.

2.11. PLACER DES PODS SUR DES NŒUDS SPÉCIFIQUES EN UTILISANT DES SÉLECTEURS DE NŒUDS

Un site *node selector* spécifie une carte de paires clé-valeur. Les règles sont définies à l'aide d'étiquettes personnalisées sur les nœuds et de sélecteurs spécifiés dans les pods.

Pour qu'un module puisse être exécuté sur un nœud, les paires clé-valeur indiquées doivent figurer sur l'étiquette du nœud.

Si vous utilisez l'affinité de nœuds et les sélecteurs de nœuds dans la même configuration de pods, consultez les considérations importantes ci-dessous.

2.11.1. Utilisation de sélecteurs de nœuds pour contrôler le placement des pods

Vous pouvez utiliser des sélecteurs de nœuds sur les pods et des étiquettes sur les nœuds pour contrôler l'endroit où le pod est planifié. Avec les sélecteurs de nœuds, OpenShift Container Platform planifie les pods sur les nœuds qui contiennent les étiquettes correspondantes.

Vous ajoutez des étiquettes à un nœud, à un ensemble de machines de calcul ou à une configuration de machine. L'ajout de l'étiquette à l'ensemble de machines de calcul garantit que si le nœud ou la machine tombe en panne, les nouveaux nœuds disposent de l'étiquette. Les étiquettes ajoutées à un nœud ou à une configuration de machine ne persistent pas si le nœud ou la machine tombe en panne.

Pour ajouter des sélecteurs de nœuds à un module existant, ajoutez un sélecteur de nœuds à l'objet de contrôle de ce module, tel que l'objet **ReplicaSet**, l'objet **DaemonSet**, l'objet **StatefulSet**, l'objet **Deployment** ou l'objet **DeploymentConfig**. Tous les modules existants sous cet objet de contrôle sont recréés sur un nœud avec une étiquette correspondante. Si vous créez un nouveau module, vous pouvez ajouter le sélecteur de nœud directement à la spécification **Pod**.

NOTE

Vous ne pouvez pas ajouter un sélecteur de nœud directement à un module planifié existant.

Conditions préalables

Pour ajouter un sélecteur de nœud à des modules existants, déterminez l'objet de contrôle de ce module. Par exemple, le module **router-default-66d5cf9464-m2g75** est contrôlé par l'ensemble de répliques **router-default-66d5cf9464**:

```
$ oc describe pod router-default-66d5cf9464-7pwkc

Name:             router-default-66d5cf9464-7pwkc
Namespace:        openshift-ingress

....

Controlled By:    ReplicaSet/router-default-66d5cf9464
```

La console web liste l'objet de contrôle sous **ownerReferences** dans le pod YAML :

```
ownerReferences:
  - apiVersion: apps/v1
    kind: ReplicaSet
    name: router-default-66d5cf9464
    uid: d81dd094-da26-11e9-a48a-128e7edf0312
    controller: true
    blockOwnerDeletion: true
```

Procédure

1. Ajoutez des étiquettes à un nœud en utilisant un ensemble de machines de calcul ou en éditant le nœud directement :

   - Utilisez un objet **MachineSet** pour ajouter des étiquettes aux nœuds gérés par l'ensemble de machines de calcul lors de la création d'un nœud :

     a. Exécutez la commande suivante pour ajouter des étiquettes à un objet **MachineSet**:

     ```
     $ oc patch MachineSet <name> --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
     <value>","<key>"="<value>"}}]'  -n openshift-machine-api
     ```

     Par exemple :

     ```
     $ oc patch MachineSet abc612-msrtw-worker-us-east-1c  --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
     node","region":"east"}}]'  -n openshift-machine-api
     ```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour ajouter des étiquettes à un ensemble de machines de calcul :

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b. Vérifiez que les étiquettes sont ajoutées à l'objet **MachineSet** en utilisant la commande **oc edit**:
Par exemple :

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api
```

Exemple d'objet MachineSet

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet

....

spec:
...
  template:
    metadata:
...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
....
```

- Ajouter des étiquettes directement à un nœud :

a. Modifiez l'objet **Node** pour le nœud :

```
$ oc label nodes <name> <key>=<value>
```

Par exemple, pour étiqueter un nœud :

```
$ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
```

ASTUCE

Vous pouvez également appliquer le langage YAML suivant pour ajouter des étiquettes à un nœud :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

b. Vérifiez que les étiquettes sont ajoutées au nœud :

```
$ oc get nodes -l type=user-node,region=east
```

Exemple de sortie

```
NAME                          STATUS   ROLES    AGE   VERSION
ip-10-0-142-25.ec2.internal   Ready    worker   17m   v1.25.0
```

2. Ajouter le sélecteur de nœud correspondant à un pod :

- Pour ajouter un sélecteur de nœud aux modules existants et futurs, ajoutez un sélecteur de nœud à l'objet de contrôle des modules :

Exemple ReplicaSet objet avec étiquettes

```
kind: ReplicaSet

....

spec:

....

  template:
    metadata:
      creationTimestamp: null
      labels:
        ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
        pod-template-hash: 66d5cf9464
    spec:
      nodeSelector:
        kubernetes.io/os: linux
        node-role.kubernetes.io/worker: ''
        type: user-node 1
```

1 Ajouter le sélecteur de nœud.

- Pour ajouter un sélecteur de nœud à un nouveau pod spécifique, ajoutez le sélecteur à l'objet **Pod** directement :

Exemple d'objet Pod avec un sélecteur de nœud

```
apiVersion: v1
kind: Pod

....

spec:
  nodeSelector:
    region: east
    type: user-node
```

NOTE

Vous ne pouvez pas ajouter un sélecteur de nœud directement à un module planifié existant.

CHAPITRE 3. CONTRÔLE DU PLACEMENT DES PODS SUR LES NŒUDS (SCHEDULING)

3.1. CONTRÔLER LE PLACEMENT DES PODS À L'AIDE DE L'ORDONNANCEUR

L'ordonnancement des pods est un processus interne qui détermine le placement des nouveaux pods sur les nœuds du cluster.

Le code de l'ordonnanceur a une séparation propre qui observe les nouveaux pods lorsqu'ils sont créés et identifie le nœud le plus approprié pour les héberger. Il crée ensuite des liens (liens entre le pod et le nœud) pour les pods à l'aide de l'API principale.

Ordonnancement par défaut des pods

OpenShift Container Platform est livré avec un planificateur par défaut qui répond aux besoins de la plupart des utilisateurs. Le planificateur par défaut utilise à la fois des outils inhérents et des outils de personnalisation pour déterminer ce qui convient le mieux à un pod.

Programmation avancée des nacelles

Dans les situations où vous souhaiteriez avoir plus de contrôle sur l'emplacement des nouveaux pods, les fonctionnalités de planification avancées d'OpenShift Container Platform vous permettent de configurer un pod de manière à ce qu'il soit nécessaire ou qu'il ait une préférence pour s'exécuter sur un nœud particulier ou aux côtés d'un pod spécifique.
Vous pouvez contrôler le placement des pods en utilisant les fonctions de planification suivantes :

- Profils du planificateur
- Règles d'affinité et d'anti-affinité des pods
- Affinité des nœuds
- Sélecteurs de nœuds
- Taches et tolérances
- Engagement excessif des nœuds

3.1.1. À propos de l'ordonnanceur par défaut

Le planificateur de pods par défaut d'OpenShift Container Platform est chargé de déterminer le placement des nouveaux pods sur les nœuds du cluster. Il lit les données du pod et trouve un nœud qui convient en fonction des profils configurés. Il est complètement indépendant et existe en tant que solution autonome. Il ne modifie pas le module ; il crée une liaison pour le module qui lie le module à un nœud particulier.

3.1.1.1. Comprendre la programmation par défaut

L'ordonnanceur générique existant est l'ordonnanceur par défaut fourni par la plateforme *engine* qui sélectionne un nœud pour héberger le module en trois étapes :

Filtre les nœuds

Les nœuds disponibles sont filtrés en fonction des contraintes ou des exigences spécifiées. Pour ce faire, chaque nœud est soumis à la liste des fonctions de filtrage appelée *predicates*, ou *filters*.

Hiérarchise la liste filtrée des nœuds

Pour ce faire, chaque nœud est soumis à une série de fonctions *priority* ou *scoring*, qui lui attribuent une note comprise entre 0 et 10, 0 indiquant une mauvaise adaptation et 10 une bonne adaptation à l'hébergement du module. La configuration de l'ordonnanceur peut également prendre en compte un simple *weight* (valeur numérique positive) pour chaque fonction d'évaluation. La note du nœud fournie par chaque fonction de notation est multipliée par le poids (le poids par défaut pour la plupart des notes est de 1), puis combinée en ajoutant les notes de chaque nœud fournies par toutes les notes. Cet attribut de poids peut être utilisé par les administrateurs pour donner plus d'importance à certaines notes.

Sélectionne le nœud le mieux adapté

Les nœuds sont triés en fonction de leur score et le nœud ayant le score le plus élevé est sélectionné pour héberger le pod. Si plusieurs nœuds ont le même score, l'un d'entre eux est choisi au hasard.

3.1.2. Cas d'utilisation de l'ordonnanceur

L'un des principaux cas d'utilisation de l'ordonnancement dans OpenShift Container Platform est la prise en charge de politiques d'affinité et d'anti-affinité flexibles.

3.1.2.1. Niveaux topologiques de l'infrastructure

Les administrateurs peuvent définir plusieurs niveaux topologiques pour leur infrastructure (nœuds) en spécifiant des étiquettes sur les nœuds. Par exemple : **region=r1 zone=z1** , **rack=s1**.

Ces noms d'étiquettes n'ont pas de signification particulière et les administrateurs sont libres de donner n'importe quel nom à leurs niveaux d'infrastructure, par exemple ville/bâtiment/chambre. En outre, les administrateurs peuvent définir un nombre quelconque de niveaux pour leur topologie d'infrastructure, trois niveaux étant généralement suffisants (par exemple : **regions → zones → racks**). Les administrateurs peuvent spécifier des règles d'affinité et d'anti-affinité à chacun de ces niveaux, dans n'importe quelle combinaison.

3.1.2.2. Affinité

Les administrateurs doivent pouvoir configurer l'ordonnanceur pour spécifier l'affinité à n'importe quel niveau topologique, voire à plusieurs niveaux. L'affinité à un niveau particulier indique que tous les pods appartenant au même service sont programmés sur des nœuds appartenant au même niveau. Cela permet de répondre aux exigences de latence des applications en permettant aux administrateurs de s'assurer que les pods homologues ne sont pas trop éloignés géographiquement les uns des autres. Si aucun nœud n'est disponible dans le même groupe d'affinité pour héberger le module, celui-ci n'est pas planifié.

Si vous avez besoin d'un meilleur contrôle sur l'emplacement des pods, consultez les sections Contrôle du placement des pods sur les nœuds à l'aide des règles d'affinité des nœuds et Placement des pods par rapport à d'autres pods à l'aide des règles d'affinité et d'anti-affinité.

Ces fonctions de planification avancées permettent aux administrateurs de spécifier le nœud sur lequel un pod peut être planifié et de forcer ou de rejeter la planification par rapport à d'autres pods.

3.1.2.3. Anti-affinité

Les administrateurs doivent pouvoir configurer l'ordonnanceur pour spécifier l'anti-affinité à n'importe quel niveau topologique, voire à plusieurs niveaux. L'anti-affinité (ou la "répartition") à un niveau particulier indique que tous les pods qui appartiennent au même service sont répartis sur les nœuds qui

appartiennent à ce niveau. Cela garantit que l'application est bien répartie à des fins de haute disponibilité. L'ordonnanceur tente d'équilibrer les modules de service sur tous les nœuds concernés de la manière la plus homogène possible.

Si vous avez besoin d'un meilleur contrôle sur l'emplacement des pods, consultez les sections Contrôle du placement des pods sur les nœuds à l'aide des règles d'affinité des nœuds et Placement des pods par rapport à d'autres pods à l'aide des règles d'affinité et d'anti-affinité.

Ces fonctions de planification avancées permettent aux administrateurs de spécifier le nœud sur lequel un pod peut être planifié et de forcer ou de rejeter la planification par rapport à d'autres pods.

3.2. ORDONNANCEMENT DE PODS À L'AIDE D'UN PROFIL D'ORDONNATEUR

Vous pouvez configurer OpenShift Container Platform pour utiliser un profil de planification afin de planifier les pods sur les nœuds du cluster.

3.2.1. À propos des profils de l'ordonnanceur

Vous pouvez spécifier un profil de planificateur pour contrôler la manière dont les pods sont planifiés sur les nœuds.

Les profils de planificateur suivants sont disponibles :

LowNodeUtilization

Ce profil tente de répartir les pods de manière égale sur les nœuds afin d'obtenir une faible utilisation des ressources par nœud. Ce profil fournit le comportement par défaut de l'ordonnanceur.

HighNodeUtilization

Ce profil tente de placer le plus grand nombre de pods possible sur le plus petit nombre de nœuds possible. Cela minimise le nombre de nœuds et permet une utilisation élevée des ressources par nœud.

NoScoring

Il s'agit d'un profil à faible latence qui s'efforce d'obtenir le cycle de programmation le plus rapide en désactivant tous les plugins de score. Il se peut que de meilleures décisions de programmation soient sacrifiées au profit de décisions plus rapides.

3.2.2. Configuration d'un profil d'ordonnanceur

Vous pouvez configurer l'ordonnanceur pour qu'il utilise un profil d'ordonnanceur.

Conditions préalables

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

Procédure

1. Modifiez l'objet **Scheduler**:

```
$ oc edit scheduler cluster
```

2. Spécifiez le profil à utiliser dans le champ **spec.profile**:

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  ...
  name: cluster
  resourceVersion: "601"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: b351d6d0-d06f-4a99-a26b-87af62e79f59
spec:
  mastersSchedulable: false
  profile: HighNodeUtilization 1
```

1 Régler sur **LowNodeUtilization**, **HighNodeUtilization**, ou **NoScoring**.

3. Enregistrez le fichier pour appliquer les modifications.

3.3. PLACEMENT DE NACELLES PAR RAPPORT À D'AUTRES NACELLES À L'AIDE DE RÈGLES D'AFFINITÉ ET D'ANTI-AFFINITÉ

L'affinité est une propriété des modules qui contrôle les nœuds sur lesquels ils préfèrent être programmés. L'anti-affinité est une propriété des modules qui empêche un module d'être programmé sur un nœud.

Dans OpenShift Container Platform, *pod affinity* et *pod anti-affinity* vous permettent de limiter les nœuds sur lesquels votre pod peut être planifié en fonction des étiquettes clé/valeur des autres pods.

3.3.1. Comprendre l'affinité des pods

Pod affinity et *pod anti-affinity* vous permettent de limiter les nœuds sur lesquels votre module peut être planifié en fonction des étiquettes clé/valeur des autres modules.

- L'affinité de pod peut indiquer à l'ordonnanceur de placer un nouveau pod sur le même nœud que d'autres pods si le sélecteur d'étiquette du nouveau pod correspond à l'étiquette du pod actuel.
- L'anti-affinité des pods peut empêcher l'ordonnanceur de localiser un nouveau pod sur le même nœud que les pods ayant les mêmes étiquettes si le sélecteur d'étiquette du nouveau pod correspond à l'étiquette du pod actuel.

Par exemple, les règles d'affinité permettent de répartir ou de regrouper les modules au sein d'un service ou par rapport aux modules d'autres services. Les règles anti-affinité vous permettent d'empêcher les pods d'un service particulier d'être planifiés sur les mêmes nœuds que les pods d'un autre service dont on sait qu'ils interfèrent avec les performances des pods du premier service. Vous pouvez également répartir les modules d'un service entre les nœuds, les zones de disponibilité ou les ensembles de disponibilité afin de réduire les défaillances corrélées.

NOTE

Un sélecteur d'étiquettes peut faire correspondre des pods avec plusieurs déploiements de pods. Utilisez des combinaisons uniques d'étiquettes lors de la configuration des règles d'anti-affinité afin d'éviter de faire correspondre les pods.

Il existe deux types de règles d'affinité pour les pods : *required* et *preferred*.

Les règles obligatoires **must** doivent être respectées pour qu'un pod puisse être programmé sur un nœud. Les règles préférentielles précisent que, si la règle est respectée, l'ordonnanceur tente de l'appliquer, mais ne la garantit pas.

NOTE

En fonction des paramètres de priorité et de préemption des modules, il se peut que l'ordonnanceur ne soit pas en mesure de trouver un nœud approprié pour un module sans enfreindre les exigences d'affinité. Dans ce cas, il se peut qu'un module ne soit pas planifié.

Pour éviter cette situation, configurez soigneusement l'affinité des pods avec des pods de priorité égale.

Vous configurez l'affinité/anti-affinité des pods par le biais des fichiers spec de **Pod**. Vous pouvez spécifier une règle obligatoire, une règle préférentielle ou les deux. Si vous spécifiez les deux, le nœud doit d'abord satisfaire à la règle requise, puis tente de satisfaire à la règle préférée.

L'exemple suivant montre une spécification **Pod** configurée pour l'affinité et l'anti-affinité des pods.

Dans cet exemple, la règle d'affinité des pods indique que le pod ne peut être programmé sur un nœud que si ce nœud possède au moins un pod déjà en cours d'exécution dont l'étiquette a la clé **security** et la valeur **S1**. La règle d'anti-affinité du pod indique que le pod préfère ne pas être programmé sur un nœud si ce nœud exécute déjà un pod avec une étiquette ayant la clé **security** et la valeur **S2**.

Exemple de fichier de configuration Pod avec pod affinity

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity: (1)
      requiredDuringSchedulingIgnoredDuringExecution: (2)
      - labelSelector:
          matchExpressions:
          - key: security (3)
            operator: In (4)
            values:
            - S1 (5)
        topologyKey: failure-domain.beta.kubernetes.io/zone
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
```

(1) Stanza pour configurer l'affinité des pods.

(2) Définit une règle obligatoire.

(3) (5) La clé et la valeur (étiquette) qui doivent correspondre pour appliquer la règle.

(4) L'opérateur représente la relation entre l'étiquette de la capsule existante et l'ensemble des valeurs des paramètres **matchExpression** dans la spécification de la nouvelle capsule. Il peut s'agir de **In**, **NotIn**, **Exists** ou **DoesNotExist**.

Exemple de fichier de configuration Pod avec pod anti-affinité

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security 4
              operator: In 5
              values:
              - S2
          topologyKey: kubernetes.io/hostname
  containers:
  - name: with-pod-affinity
    image: docker.io/ocpqe/hello-pod
```

1. Stanza pour configurer l'anti-affinité du pod.
2. Définit une règle préférentielle.
3. Spécifie un poids pour une règle préférentielle. Le nœud ayant le poids le plus élevé est privilégié.
4. Description de l'étiquette du pod qui détermine quand la règle anti-affinité s'applique. Spécifiez une clé et une valeur pour l'étiquette.
5. L'opérateur représente la relation entre l'étiquette de la capsule existante et l'ensemble des valeurs des paramètres **matchExpression** dans la spécification de la nouvelle capsule. Il peut s'agir de **In**, **NotIn**, **Exists** ou **DoesNotExist**.

NOTE

Si les étiquettes d'un nœud changent au moment de l'exécution, de sorte que les règles d'affinité d'un module ne sont plus respectées, le module continue de fonctionner sur le nœud.

3.3.2. Configuration d'une règle d'affinité pour les pods

Les étapes suivantes illustrent une configuration simple à deux pods qui crée un pod avec une étiquette et un pod qui utilise l'affinité pour permettre la planification avec ce pod.

Procédure

1. Créer un pod avec une étiquette spécifique dans la spécification **Pod**:

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-s1
  labels:
    security: S1
spec:
  containers:
  - name: security-s1
    image: docker.io/ocpqe/hello-pod
```

2. Lors de la création d'autres pods, modifiez la spécification **Pod** comme suit :

   a. Utilisez la strophe **podAffinity** pour configurer le paramètre **requiredDuringSchedulingIgnoredDuringExecution** ou le paramètre **preferredDuringSchedulingIgnoredDuringExecution**:

   b. Spécifiez la clé et la valeur qui doivent être respectées. Si vous souhaitez que le nouveau module soit programmé avec l'autre module, utilisez les mêmes paramètres **key** et **value** que l'étiquette du premier module.

```
podAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
  - labelSelector:
      matchExpressions:
      - key: security
        operator: In
        values:
        - S1
    topologyKey: failure-domain.beta.kubernetes.io/zone
```

   c. **operator**L'opérateur peut être **In**, **NotIn**, **Exists** ou **DoesNotExist**. Par exemple, utilisez l'opérateur **In** pour exiger que l'étiquette soit dans le nœud.

   d. Spécifiez un **topologyKey**, qui est un label Kubernetes prérempli que le système utilise pour désigner un tel domaine topologique.

3. Créer la capsule.

```
oc create -f <pod-spec>.yaml
```

3.3.3. Configuration d'une règle d'anti-affinité pour les pods

Les étapes suivantes illustrent une configuration simple à deux pods qui crée un pod avec une étiquette et un pod qui utilise une règle préférentielle anti-affinité pour tenter d'empêcher l'ordonnancement avec ce pod.

Procédure

1. Créer un pod avec une étiquette spécifique dans la spécification **Pod**:

```
$ cat team4.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: security-s2
  labels:
    security: S2
spec:
  containers:
  - name: security-s2
    image: docker.io/ocpqe/hello-pod
```

2. Lorsque vous créez d'autres pods, modifiez la spécification **Pod** pour définir les paramètres suivants :

3. Utilisez la strophe **podAntiAffinity** pour configurer le paramètre **requiredDuringSchedulingIgnoredDuringExecution** ou le paramètre **preferredDuringSchedulingIgnoredDuringExecution**:

   a. Spécifiez un poids pour le nœud, de 1 à 100. Le nœud ayant le poids le plus élevé est privilégié.

   b. Spécifiez la clé et les valeurs qui doivent être respectées. Si vous souhaitez que le nouveau module ne soit pas programmé avec l'autre module, utilisez les mêmes paramètres **key** et **value** que l'étiquette du premier module.

```
podAntiAffinity:
  preferredDuringSchedulingIgnoredDuringExecution:
  - weight: 100
    podAffinityTerm:
      labelSelector:
        matchExpressions:
        - key: security
          operator: In
          values:
          - S2
      topologyKey: kubernetes.io/hostname
```

   c. Pour une règle préférentielle, spécifiez un poids, 1-100.

   d. **operator**L'opérateur peut être **In**, **NotIn**, **Exists** ou **DoesNotExist**. Par exemple, utilisez l'opérateur **In** pour exiger que l'étiquette soit dans le nœud.

4. Spécifiez un **topologyKey**, qui est un label Kubernetes prérempli que le système utilise pour désigner un tel domaine topologique.

5. Créer la capsule.

```
oc create -f <pod-spec>.yaml
```

3.3.4. Exemple de règles d'affinité et d'anti-affinité pour les pods

Les exemples suivants illustrent l'affinité et l'anti-affinité des pods.

3.3.4.1. Pod Affinity

L'exemple suivant illustre l'affinité des pods avec des étiquettes et des sélecteurs d'étiquettes correspondants.

- Le pod **team4** porte l'étiquette **team:4**.

```
$ cat team4.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4
  labels:
    team: "4"
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Le pod **team4a** a le sélecteur d'étiquettes **team:4** sous **podAffinity**.

```
$ cat pod-team4a.yaml
apiVersion: v1
kind: Pod
metadata:
  name: team4a
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: team
            operator: In
            values:
            - "4"
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- Le module **team4a** est programmé sur le même nœud que le module **team4**.

3.3.4.2. Pod Anti-affinité

L'exemple suivant illustre l'anti-affinité des pods pour les pods dont les étiquettes et les sélecteurs d'étiquettes correspondent.

- Le pod **pod-s1** porte l'étiquette **security:s1**.

```
cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
```

```
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Le pod **pod-s2** a le sélecteur d'étiquettes **security:s1** sous **podAntiAffinity**.

```
cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s1
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-antiaffinity
    image: docker.io/ocpqe/hello-pod
```

- Le pod **pod-s2** ne peut pas être programmé sur le même nœud que **pod-s1**.

3.3.4.3. Affinité podale sans étiquettes correspondantes

L'exemple suivant illustre l'affinité des pods pour les pods ne correspondant pas aux étiquettes et aux sélecteurs d'étiquettes.

- Le pod **pod-s1** porte l'étiquette **security:s1**.

```
$ cat pod-s1.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
  labels:
    security: s1
spec:
  containers:
  - name: ocp
    image: docker.io/ocpqe/hello-pod
```

- Le pod **pod-s2** possède le sélecteur d'étiquettes **security:s2**.

```
$ cat pod-s2.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-s2
```

```
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: security
            operator: In
            values:
            - s2
        topologyKey: kubernetes.io/hostname
  containers:
  - name: pod-affinity
    image: docker.io/ocpqe/hello-pod
```

- Le pod **pod-s2** n'est pas planifié à moins qu'il n'y ait un nœud avec un pod ayant le label **security:s2**. S'il n'y a pas d'autre pod avec ce label, le nouveau pod reste en attente :

  Exemple de sortie

```
NAME     READY    STATUS    RESTARTS   AGE      IP       NODE
pod-s2   0/1      Pending   0          32s      <none>
```

3.3.5. Utilisation de l'affinité et de l'anti-affinité du pod pour contrôler l'endroit où un opérateur est installé

Par défaut, lorsque vous installez un Operator, OpenShift Container Platform installe le pod Operator sur l'un de vos nœuds de travail de manière aléatoire. Cependant, il peut y avoir des situations où vous voulez que ce pod soit planifié sur un nœud spécifique ou un ensemble de nœuds.

Les exemples suivants décrivent des situations dans lesquelles vous pourriez vouloir planifier un pod opérateur sur un nœud ou un ensemble de nœuds spécifique :

- Si un opérateur a besoin d'une plateforme particulière, telle que **amd64** ou **arm64**
- Si un opérateur nécessite un système d'exploitation particulier, tel que Linux ou Windows
- Si vous souhaitez que les opérateurs qui travaillent ensemble soient programmés sur le même hôte ou sur des hôtes situés sur le même rack
- Si vous souhaitez que les opérateurs soient dispersés dans l'infrastructure afin d'éviter les temps d'arrêt dus à des problèmes de réseau ou de matériel

Vous pouvez contrôler l'endroit où un pod d'opérateur est installé en ajoutant une affinité ou une anti-affinité de pod à l'objet **Subscription** de l'opérateur.

L'exemple suivant montre comment utiliser l'anti-affinité des pods pour empêcher l'installation de Custom Metrics Autoscaler Operator à partir de n'importe quel nœud ayant des pods avec une étiquette spécifique :

Exemple d'affinité de pod qui place le 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 Une affinité de pod qui place le pod de l'opérateur sur un nœud qui a des pods avec le label **app=test**.

Exemple d'anti-affinité de pods qui empêche le pod Operator d'accéder à 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:
      podAntiAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
            - key: cpu
              operator: In
              values:
              - high
          topologyKey: kubernetes.io/hostname
...
```

1 Une anti-affinité de pods qui empêche le pod de l'opérateur d'être planifié sur un nœud qui a des pods avec le label **cpu=high**.

Procédure

Pour contrôler l'emplacement d'une nacelle d'opérateur, procédez comme suit :

1. Installez l'opérateur comme d'habitude.
2. Si nécessaire, assurez-vous que vos nœuds sont étiquetés de manière à répondre correctement à l'affinité.
3. Modifiez l'objet Operator **Subscription** 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:
      podAntiAffinity: 1
        requiredDuringSchedulingIgnoredDuringExecution:
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                - ip-10-0-185-229.ec2.internal
            topologyKey: topology.kubernetes.io/zone
...
```

1 Ajouter un **podAffinity** ou un **podAntiAffinity**.

Vérification

- Pour 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>
```

3.4. CONTRÔLE DU PLACEMENT DES PODS SUR LES NŒUDS À L'AIDE DE RÈGLES D'AFFINITÉ DES NŒUDS

L'affinité est une propriété des pods qui contrôle les nœuds sur lesquels ils préfèrent être programmés.

Dans OpenShift Container Platform, l'affinité des nœuds est un ensemble de règles utilisées par le planificateur pour déterminer où un pod peut être placé. Les règles sont définies à l'aide d'étiquettes personnalisées sur les nœuds et de sélecteurs d'étiquettes spécifiés dans les pods.

3.4.1. Comprendre l'affinité des nœuds

L'affinité de nœud permet à un pod de spécifier une affinité envers un groupe de nœuds sur lesquels il peut être placé. Le nœud n'a pas de contrôle sur le placement.

Par exemple, vous pouvez configurer un module pour qu'il ne s'exécute que sur un nœud doté d'une unité centrale spécifique ou dans une zone de disponibilité spécifique.

Il existe deux types de règles d'affinité entre les nœuds : *required* et *preferred*.

Les règles obligatoires **must** doivent être respectées pour qu'un pod puisse être programmé sur un nœud. Les règles préférentielles précisent que, si la règle est respectée, l'ordonnanceur tente de l'appliquer, mais ne la garantit pas.

NOTE

Si les étiquettes d'un nœud changent au moment de l'exécution et que la règle d'affinité d'un nœud pour un module n'est plus respectée, le module continue de fonctionner sur le nœud.

Vous configurez l'affinité des nœuds par le biais du fichier **Pod** spec. Vous pouvez spécifier une règle obligatoire, une règle préférentielle ou les deux. Si vous spécifiez les deux, le nœud doit d'abord satisfaire à la règle requise, puis tente de satisfaire à la règle préférée.

L'exemple suivant est une spécification **Pod** avec une règle qui exige que le pod soit placé sur un nœud avec une étiquette dont la clé est **e2e-az-NorthSouth** et dont la valeur est soit **e2e-az-North** soit **e2e-az-South**:

Exemple de fichier de configuration d'un pod avec une règle d'affinité de nœud requise

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: 1
      requiredDuringSchedulingIgnoredDuringExecution: 2
        nodeSelectorTerms:
        - matchExpressions:
          - key: e2e-az-NorthSouth 3
            operator: In 4
            values:
            - e2e-az-North 5
            - e2e-az-South 6
  containers:
  - name: with-node-affinity
    image: docker.io/ocpqe/hello-pod
```

1 La strophe pour configurer l'affinité des nœuds.

2 Définit une règle obligatoire.

3 5 6 La paire clé/valeur (étiquette) qui doit être prise en compte pour appliquer la règle.

4 L'opérateur représente la relation entre l'étiquette du nœud et l'ensemble des valeurs des paramètres **matchExpression** dans la spécification **Pod**. Cette valeur peut être **In**, **NotIn**, **Exists**,

L'exemple suivant est une spécification de nœud avec une règle de préférence selon laquelle un nœud avec une étiquette dont la clé est **e2e-az-EastWest** et dont la valeur est soit **e2e-az-East** soit **e2e-az-West** est préféré pour le pod :

Exemple de fichier de configuration d'un pod avec une règle préférentielle d'affinité de nœud

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 1 3
        preference:
          matchExpressions:
          - key: e2e-az-EastWest 4
            operator: In 5
            values:
            - e2e-az-East 6
            - e2e-az-West 7
  containers:
  - name: with-node-affinity
    image: docker.io/ocpqe/hello-pod
```

1 La strophe pour configurer l'affinité des nœuds.

2 Définit une règle préférentielle.

3 Spécifie un poids pour une règle préférentielle. Le nœud ayant le poids le plus élevé est privilégié.

4 6 7 La paire clé/valeur (étiquette) qui doit être prise en compte pour appliquer la règle.

5 L'opérateur représente la relation entre l'étiquette du nœud et l'ensemble des valeurs des paramètres **matchExpression** dans la spécification **Pod**. Cette valeur peut être **In**, **NotIn**, **Exists**, ou **DoesNotExist**, **Lt**, ou **Gt**.

Il n'existe pas de concept explicite *node anti-affinity*, mais l'utilisation de l'opérateur **NotIn** ou **DoesNotExist** reproduit ce comportement.

NOTE

Si vous utilisez l'affinité de nœuds et les sélecteurs de nœuds dans la même configuration de pods, notez ce qui suit :

- Si vous configurez à la fois **nodeSelector** et **nodeAffinity**, les deux conditions doivent être remplies pour que le pod soit planifié sur un nœud candidat.
- Si vous spécifiez plusieurs **nodeSelectorTerms** associés à des types **nodeAffinity**, le module peut être programmé sur un nœud si l'un des **nodeSelectorTerms** est satisfait.
- Si vous spécifiez plusieurs **matchExpressions** associés à **nodeSelectorTerms**, le module ne peut être programmé sur un nœud que si tous les **matchExpressions** sont satisfaits.

3.4.2. Configuration d'une règle d'affinité de nœud requise

Les règles requises **must** doivent être respectées avant qu'un pod puisse être programmé sur un nœud.

Procédure

Les étapes suivantes présentent une configuration simple qui crée un nœud et un module que l'ordonnanceur doit placer sur le nœud.

1. Ajoutez une étiquette à un nœud à l'aide de la commande **oc label node**:

```
$ oc label node node1 e2e-az-name=e2e-az1
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter l'étiquette :

```
kind: Node
apiVersion: v1
metadata:
 name: <node_name>
 labels:
  e2e-az-name: e2e-az1
```

2. Dans la spécification **Pod**, utilisez la strophe **nodeAffinity** pour configurer le paramètre **requiredDuringSchedulingIgnoredDuringExecution**:

   a. Spécifiez la clé et les valeurs qui doivent être respectées. Si vous souhaitez que le nouveau module soit planifié sur le nœud que vous avez modifié, utilisez les mêmes paramètres **key** et **value** que l'étiquette du nœud.

   b. **operator**L'opérateur peut être **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt** ou **Gt**. Par exemple, utilisez l'opérateur **In** pour exiger que l'étiquette soit dans le nœud :

Exemple de sortie

```
spec:
  affinity:
    nodeAffinity:
```

```
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
```

3. Créer la capsule :

```
$ oc create -f e2e-az2.yaml
```

3.4.3. Configuration d'une règle d'affinité pour les nœuds préférés

Les règles préférentielles précisent que, si la règle est respectée, l'ordonnanceur tente d'appliquer les règles, mais n'en garantit pas l'application.

Procédure

Les étapes suivantes présentent une configuration simple qui crée un nœud et un module que l'ordonnanceur tente de placer sur le nœud.

1. Ajoutez une étiquette à un nœud à l'aide de la commande **oc label node**:

```
$ oc label node node1 e2e-az-name=e2e-az3
```

2. Dans la spécification **Pod**, utilisez la strophe **nodeAffinity** pour configurer le paramètre **preferredDuringSchedulingIgnoredDuringExecution**:

   a. Spécifiez un poids pour le nœud, sous la forme d'un nombre de 1 à 100. Le nœud ayant le poids le plus élevé est privilégié.

   b. Spécifiez la clé et les valeurs qui doivent être respectées. Si vous souhaitez que le nouveau module soit planifié sur le nœud que vous avez modifié, utilisez les mêmes paramètres **key** et **value** que l'étiquette du nœud :

```
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: e2e-az-name
            operator: In
            values:
            - e2e-az3
```

   c. **operator**L'opérateur peut être **In**, **NotIn**, **Exists**, **DoesNotExist**, **Lt** ou **Gt**. Par exemple, utilisez l'opérateur **In** pour exiger que l'étiquette soit dans le nœud.

3. Créer la capsule.

```
$ oc create -f e2e-az3.yaml
```

3.4.4. Exemple de règles d'affinité entre les nœuds

Les exemples suivants illustrent l'affinité entre les nœuds.

3.4.4.1. Affinité des nœuds avec les étiquettes correspondantes

L'exemple suivant illustre l'affinité d'un nœud et d'un module avec des étiquettes correspondantes :

- Le nœud Node1 porte l'étiquette **zone:us**:

```
$ oc label node node1 zone=us
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter l'étiquette :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: us
```

- Le pod-s1 possède la paire clé/valeur **zone** et **us** en vertu d'une règle d'affinité de nœud requise :

```
$ cat pod-s1.yaml
```

Exemple de sortie

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- Le pod-s1 peut être programmé sur le nœud 1 :

```
$ oc get pod -o wide
```

Exemple de sortie

```
NAME    READY    STATUS      RESTARTS  AGE     IP      NODE
pod-s1  1/1      Running     0         4m      IP1     node1
```

3.4.4.2. Affinité des nœuds sans étiquettes correspondantes

L'exemple suivant illustre l'affinité de nœud pour un nœud et un module sans étiquettes correspondantes :

- Le nœud Node1 porte l'étiquette **zone:emea**:

```
$ oc label node node1 zone=emea
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter l'étiquette :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    zone: emea
```

- Le pod-s1 possède la paire clé/valeur **zone** et **us** en vertu d'une règle d'affinité de nœud requise :

```
$ cat pod-s1.yaml
```

Exemple de sortie

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-s1
spec:
  containers:
    - image: "docker.io/ocpqe/hello-pod"
      name: hello-pod
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
            - key: "zone"
              operator: In
              values:
              - us
```

- Le pod-s1 ne peut pas être planifié sur le nœud 1 :

```
$ oc describe pod pod-s1
```

Exemple de sortie

```
...

Events:
 FirstSeen LastSeen Count From              SubObjectPath  Type              Reason
 --------- -------- ----- ----              -------------  --------          ------
 1m        33s      8     default-scheduler Warning        FailedScheduling  No nodes are
available that match all of the following predicates:: MatchNodeSelector (1).
```

3.4.5. Utilisation de l'affinité des nœuds pour contrôler l'emplacement d'installation d'un opérateur

Par défaut, lorsque vous installez un Operator, OpenShift Container Platform installe le pod Operator sur l'un de vos nœuds de travail de manière aléatoire. Cependant, il peut y avoir des situations où vous voulez que ce pod soit planifié sur un nœud spécifique ou un ensemble de nœuds.

Les exemples suivants décrivent des situations dans lesquelles vous pourriez vouloir planifier un pod opérateur sur un nœud ou un ensemble de nœuds spécifique :

- Si un opérateur a besoin d'une plateforme particulière, telle que **amd64** ou **arm64**
- Si un opérateur nécessite un système d'exploitation particulier, tel que Linux ou Windows
- Si vous souhaitez que les opérateurs qui travaillent ensemble soient programmés sur le même hôte ou sur des hôtes situés sur le même rack
- Si vous souhaitez que les opérateurs soient dispersés dans l'infrastructure afin d'éviter les temps d'arrêt dus à des problèmes de réseau ou de matériel

Vous pouvez contrôler l'endroit où un pod d'opérateur est installé en ajoutant des contraintes d'affinité de nœud à l'objet **Subscription** de l'opérateur.

Les exemples suivants montrent comment utiliser l'affinité entre les nœuds pour installer une instance de Custom Metrics Autoscaler Operator sur un nœud spécifique de la grappe :

Exemple d'affinité de nœud qui place le 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. Une affinité de nœud qui exige que le pod de l'opérateur soit programmé sur un nœud nommé **ip-10-0-163-94.us-west-2.compute.internal**.

Exemple d'affinité de nœud 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. Une affinité de nœud qui exige que le pod de l'opérateur soit programmé sur un nœud avec les étiquettes **kubernetes.io/arch=arm64** et **kubernetes.io/os=linux**.

Procédure

Pour contrôler l'emplacement d'une nacelle d'opérateur, procédez comme suit :

1. Installez l'opérateur comme d'habitude.
2. Si nécessaire, assurez-vous que vos nœuds sont étiquetés de manière à répondre correctement à l'affinité.
3. Modifiez l'objet Operator **Subscription** 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**.

Vérification

- Pour 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>
```

3.4.6. Ressources supplémentaires

- Pour plus d'informations sur la modification des étiquettes de nœuds, voir Comment mettre à jour les étiquettes des nœuds.

3.5. PLACER DES PODS SUR DES NŒUDS SUR-ENGAGÉS

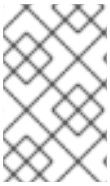
Dans l'état *overcommited*, la somme des demandes et des limites des ressources de calcul du conteneur dépasse les ressources disponibles sur le système. Le surengagement peut être souhaitable dans les environnements de développement où un compromis entre les performances garanties et la capacité est acceptable.

Les demandes et les limites permettent aux administrateurs d'autoriser et de gérer le surengagement des ressources sur un nœud. L'ordonnanceur utilise les demandes pour planifier votre conteneur et fournir une garantie de service minimum. Les limites restreignent la quantité de ressources de calcul qui peuvent être consommées sur votre nœud.

3.5.1. Comprendre le surengagement

Les demandes et les limites permettent aux administrateurs d'autoriser et de gérer le surengagement des ressources sur un nœud. L'ordonnanceur utilise les demandes pour planifier votre conteneur et fournir une garantie de service minimum. Les limites restreignent la quantité de ressources de calcul qui peuvent être consommées sur votre nœud.

Les administrateurs d'OpenShift Container Platform peuvent contrôler le niveau de surengagement et gérer la densité des conteneurs sur les nœuds en configurant les maîtres pour remplacer le rapport entre la demande et la limite définie sur les conteneurs de développement. En conjonction avec un objet **LimitRange** par projet spécifiant les limites et les valeurs par défaut, cela ajuste la limite et la demande du conteneur pour atteindre le niveau souhaité de surengagement.



NOTE

Ces dérogations n'ont aucun effet si aucune limite n'a été fixée pour les conteneurs. Créez un objet **LimitRange** avec des limites par défaut, par projet individuel ou dans le modèle de projet, pour vous assurer que les dérogations s'appliquent.

Après ces dérogations, les limites et les demandes des conteneurs doivent toujours être validées par tout objet **LimitRange** dans le projet. Il est possible, par exemple, que les développeurs spécifient une limite proche de la limite minimale, et que la demande soit ensuite remplacée en dessous de la limite minimale, ce qui entraînerait l'interdiction du pod. Cette expérience malheureuse devrait être résolue dans le cadre de travaux futurs, mais pour l'instant, il convient de configurer cette capacité et les objets **LimitRange** avec prudence.

3.5.2. Comprendre le surengagement des nœuds

Dans un environnement surchargé, il est important de configurer correctement votre nœud afin d'obtenir le meilleur comportement possible du système.

Lorsque le nœud démarre, il s'assure que les drapeaux ajustables du noyau pour la gestion de la mémoire sont correctement définis. Le noyau ne devrait jamais échouer dans l'allocation de la mémoire à moins qu'il ne soit à court de mémoire physique.

Pour garantir ce comportement, OpenShift Container Platform configure le noyau pour qu'il surengage toujours de la mémoire en définissant le paramètre **vm.overcommit_memory** sur **1**, ce qui annule le paramètre par défaut du système d'exploitation.

OpenShift Container Platform configure également le noyau pour qu'il ne panique pas lorsqu'il manque de mémoire en définissant le paramètre **vm.panic_on_oom** sur **0**. Un paramètre de 0 indique au noyau d'appeler oom_killer dans une condition de manque de mémoire (OOM), ce qui tue les processus en fonction de leur priorité

Vous pouvez afficher le paramètre actuel en exécutant les commandes suivantes sur vos nœuds :

```
$ sysctl -a |grep commit
```

Exemple de sortie

```
vm.overcommit_memory = 1
```

```
$ sysctl -a |grep panic
```

Exemple de sortie

```
vm.panic_on_oom = 0
```

NOTE

Les drapeaux ci-dessus devraient déjà être activés sur les nœuds, et aucune autre action n'est nécessaire.

Vous pouvez également effectuer les configurations suivantes pour chaque nœud :

- Désactiver ou appliquer les limites de l'unité centrale à l'aide des quotas CFS de l'unité centrale
- Réserver des ressources pour les processus du système
- Réserve de mémoire pour les différents niveaux de qualité de service

3.6. CONTRÔLE DU PLACEMENT DE PODS À L'AIDE DE TACHES DE NŒUDS

Les taches et les tolérances permettent au nœud de contrôler quels pods doivent (ou ne doivent pas) être programmés sur eux.

3.6.1. Comprendre les taches et les tolérances

Un *taint* permet à un nœud de refuser qu'un module soit programmé à moins que ce module n'ait un *toleration* correspondant.

Vous appliquez des taches à un nœud par le biais de la spécification **Node** (**NodeSpec**) et vous appliquez des tolérances à un pod par le biais de la spécification **Pod** (**PodSpec**). Lorsque vous appliquez une tare à un nœud, l'ordonnanceur ne peut pas placer un module sur ce nœud à moins que le module ne puisse tolérer la tare.

Exemple d'altération dans la spécification d'un nœud

```
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
....
```

Exemple de tolérance dans une spécification Pod

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
....
```

Les plaintes et les tolérances se composent d'une clé, d'une valeur et d'un effet.

Tableau 3.1. Composants de contamination et de tolérance

Paramètres	Description
key	Le site **key** est une chaîne de caractères quelconque, d'une longueur maximale de 253 caractères. La clé doit commencer par une lettre ou un chiffre et peut contenir des lettres, des chiffres, des traits d'union, des points et des traits de soulignement.
value	Le site **value** est une chaîne de caractères de 63 caractères maximum. La valeur doit commencer par une lettre ou un chiffre et peut contenir des lettres, des chiffres, des traits d'union, des points et des traits de soulignement.
effect	L'effet est l'un des suivants : **NoSchedule** [1] : • Les nouveaux pods qui ne correspondent pas à l'altération ne sont pas programmés sur ce nœud. • Les pods existants sur le nœud sont conservés. **PreferNoSchedule** : • Les nouveaux pods qui ne correspondent pas à l'altération peuvent être programmés sur ce nœud, mais l'ordonnanceur essaie de ne pas le faire. • Les pods existants sur le nœud sont conservés. **NoExecute** : • Les nouveaux pods qui ne correspondent pas à l'altération ne peuvent pas être programmés sur ce nœud. • Les pods existants sur le nœud qui n'ont pas de tolérance correspondante sont supprimés.
operator	**Equal** : Les paramètres **key**/**value**/**effect** doivent correspondre. Il s'agit de la valeur par défaut. **Exists** : Les paramètres **key**/**effect** doivent correspondre. Vous devez laisser vide le paramètre **value**, qui correspond à n'importe quel paramètre.

1. Si vous ajoutez une erreur **NoSchedule** à un nœud du plan de contrôle, le nœud doit avoir l'erreur **node-role.kubernetes.io/master=:NoSchedule**, qui est ajoutée par défaut.

Par exemple :

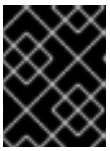
```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
...
```

Une tolérance correspond à une souillure :

- Si le paramètre **operator** est réglé sur **Equal**:
  - les paramètres de **key** sont les mêmes ;
  - les paramètres de **value** sont les mêmes ;
  - les paramètres de **effect** sont les mêmes.
- Si le paramètre **operator** est réglé sur **Exists**:
  - les paramètres de **key** sont les mêmes ;
  - les paramètres de **effect** sont les mêmes.

Les plaintes suivantes sont intégrées à OpenShift Container Platform :

- **node.kubernetes.io/not-ready**: Le nœud n'est pas prêt. Cela correspond à la condition du nœud **Ready=False**.
- **node.kubernetes.io/unreachable**: Le nœud est inaccessible depuis le contrôleur de nœud. Cela correspond à l'état du nœud **Ready=Unknown**.
- **node.kubernetes.io/memory-pressure**: Le nœud a des problèmes de pression de mémoire. Cela correspond à la condition du nœud **MemoryPressure=True**.
- **node.kubernetes.io/disk-pressure**: Le nœud a des problèmes de pression de disque. Cela correspond à l'état du nœud **DiskPressure=True**.
- **node.kubernetes.io/network-unavailable**: Le réseau du nœud est indisponible.
- **node.kubernetes.io/unschedulable**: Le nœud n'est pas ordonnançable.
- **node.cloudprovider.kubernetes.io/uninitialized**: Lorsque le contrôleur de nœuds est démarré avec un fournisseur de nuage externe, cette altération est définie sur un nœud pour le marquer comme inutilisable. Après qu'un contrôleur du cloud-controller-manager initialise ce nœud, le kubelet supprime cette taint.
- **node.kubernetes.io/pid-pressure**: Le nœud a un pid de pression. Cela correspond à la condition du nœud **PIDPressure=True**.



IMPORTANT

OpenShift Container Platform ne définit pas de pid par défaut.available **evictionHard**.

3.6.1.1. Comprendre comment utiliser les secondes de tolérance pour retarder les expulsions de nacelles

Vous pouvez spécifier la durée pendant laquelle un pod peut rester lié à un nœud avant d'être expulsé en spécifiant le paramètre **tolerationSeconds** dans la spécification **Pod** ou l'objet **MachineSet**. Si une altération ayant l'effet **NoExecute** est ajoutée à un nœud, un module qui tolère l'altération, qui a le paramètre **tolerationSeconds**, le module n'est pas expulsé avant l'expiration de ce délai.

Exemple de sortie

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

Ici, si ce pod est en cours d'exécution mais n'a pas de tolérance correspondante, le pod reste lié au nœud pendant 3 600 secondes avant d'être expulsé. Si l'altération est supprimée avant ce délai, le module n'est pas expulsé.

3.6.1.2. Comprendre comment utiliser des teintes multiples

Vous pouvez placer plusieurs taints sur le même nœud et plusieurs tolérances sur le même pod. OpenShift Container Platform traite les plaintes et tolérances multiples de la manière suivante :

1. Traiter les plaintes pour lesquelles le pod a une tolérance correspondante.
2. Les autres souillures non appariées ont les effets indiqués sur la cosse :
   - S'il y a au moins une taint non appariée avec l'effet **NoSchedule**, OpenShift Container Platform ne peut pas planifier un pod sur ce nœud.
   - S'il n'y a pas de taint non apparié avec l'effet **NoSchedule** mais qu'il y a au moins un taint non apparié avec l'effet **PreferNoSchedule**, OpenShift Container Platform essaie de ne pas planifier le pod sur le nœud.
   - S'il y a au moins une taint non appariée avec l'effet **NoExecute**, OpenShift Container Platform expulse le pod du nœud s'il est déjà en cours d'exécution sur le nœud, ou le pod n'est pas planifié sur le nœud s'il n'est pas encore en cours d'exécution sur le nœud.
     - Les pods qui ne tolèrent pas la souillure sont immédiatement expulsés.
     - Les pods qui tolèrent l'altération sans spécifier **tolerationSeconds** dans leur spécification **Pod** restent liés pour toujours.
     - Les pods qui tolèrent l'altération à l'aide d'une adresse **tolerationSeconds** spécifiée restent liés pendant la durée spécifiée.

Par exemple :

- Ajoutez au nœud les taches suivantes :

```
$ oc adm taint nodes node1 key1=value1:NoSchedule
```

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

```
$ oc adm taint nodes node1 key2=value2:NoSchedule
```

- Les tolérances suivantes s'appliquent à la nacelle :

```
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
```

Dans ce cas, le module ne peut pas être programmé sur le nœud, car il n'y a pas de tolérance correspondant à la troisième tare. Le module continue de fonctionner s'il est déjà en cours d'exécution sur le nœud lorsque la tare est ajoutée, car la troisième tare est la seule des trois qui n'est pas tolérée par le module.

3.6.1.3. Comprendre l'ordonnancement des pods et les conditions des nœuds (taint node by condition)

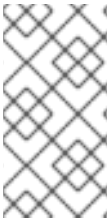
La fonction d'altération des nœuds par condition, activée par défaut, altère automatiquement les nœuds qui signalent des conditions telles que la pression de la mémoire et la pression du disque. Lorsqu'un nœud signale une condition, une erreur est ajoutée jusqu'à ce que la condition disparaisse. Les taches ont l'effet **NoSchedule**, ce qui signifie qu'aucun pod ne peut être planifié sur le nœud à moins que le pod n'ait une tolérance correspondante.

L'ordonnanceur vérifie la présence de ces anomalies sur les nœuds avant de planifier les modules. Si l'erreur est présente, le module est planifié sur un nœud différent. Comme l'ordonnanceur vérifie les anomalies et non les conditions réelles des nœuds, vous pouvez configurer l'ordonnanceur pour qu'il ignore certaines de ces conditions en ajoutant des tolérances appropriées pour les nœuds.

Pour assurer la compatibilité ascendante, le contrôleur de jeu de démons ajoute automatiquement les tolérances suivantes à tous les démons :

- node.kubernetes.io/memory-pressure
- node.kubernetes.io/disk-pressure
- node.kubernetes.io/unschedulable (1.10 ou ultérieur)
- node.kubernetes.io/network-unavailable (réseau hôte uniquement)

Vous pouvez également ajouter des tolérances arbitraires aux ensembles de démons.



NOTE

Le plan de contrôle ajoute également la tolérance **node.kubernetes.io/memory-pressure** sur les pods qui ont une classe QoS. En effet, Kubernetes gère les pods dans les classes de QoS **Guaranteed** ou **Burstable**. Les nouveaux pods **BestEffort** ne sont pas planifiés sur le nœud affecté.

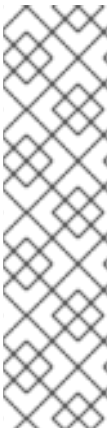
3.6.1.4. Comprendre l'éviction des pods par condition (évictions basées sur les taches)

La fonction Taint-Based Evictions, qui est activée par défaut, expulse les pods d'un nœud qui présente des conditions spécifiques, telles que **not-ready** et **unreachable**. Lorsqu'un nœud est confronté à l'une de ces conditions, OpenShift Container Platform ajoute automatiquement des taints au nœud et commence à expulser et à replanifier les pods sur différents nœuds.

Les évictions basées sur l'altération ont un effet **NoExecute**, où tout pod qui ne tolère pas l'altération est évincé immédiatement et tout pod qui tolère l'altération ne sera jamais évincé, à moins que le pod n'utilise le paramètre **tolerationSeconds**.

Le paramètre **tolerationSeconds** vous permet de spécifier la durée pendant laquelle un pod reste lié à un nœud qui a une condition de nœud. Si la condition existe toujours après la période **tolerationSeconds**, l'altération reste sur le nœud et les pods avec une tolérance correspondante sont expulsés. Si la condition disparaît avant la période **tolerationSeconds**, les pods avec les tolérances correspondantes ne sont pas supprimés.

Si vous utilisez le paramètre **tolerationSeconds** sans valeur, les pods ne sont jamais expulsés en raison des conditions "not ready" et "unreachable node".



NOTE

OpenShift Container Platform évince les pods de manière limitée afin d'éviter les évictions massives de pods dans des scénarios tels que la partition du maître par rapport aux nœuds.

Par défaut, si plus de 55 % des nœuds d'une zone donnée sont malsains, le contrôleur du cycle de vie des nœuds fait passer l'état de cette zone à **PartialDisruption** et le taux d'expulsion des pods est réduit. Pour les petits clusters (par défaut, 50 nœuds ou moins) dans cet état, les nœuds de cette zone ne sont pas altérés et les expulsions sont arrêtées.

Pour plus d'informations, voir Rate limits on eviction dans la documentation Kubernetes.

OpenShift Container Platform ajoute automatiquement une tolérance pour **node.kubernetes.io/not-ready** et **node.kubernetes.io/unreachable** avec **tolerationSeconds=300**, à moins que la configuration **Pod** ne spécifie l'une ou l'autre tolérance.

```
spec:
  tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300 (1)
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```

1 Ces tolérances garantissent que le comportement par défaut du pod est de rester lié pendant cinq minutes après la détection d'un de ces problèmes de conditions de nœuds.

Vous pouvez configurer ces tolérances selon vos besoins. Par exemple, si vous avez une application avec beaucoup d'état local, vous pourriez vouloir garder les pods liés au nœud plus longtemps en cas de partition du réseau, ce qui permettrait à la partition de se rétablir et d'éviter l'éviction des pods.

Les pods générés par un ensemble de démons sont créés avec des tolérances de **NoExecute** pour les taches suivantes, sans **tolerationSeconds**:

- **node.kubernetes.io/unreachable**
- **node.kubernetes.io/not-ready**

Par conséquent, les pods de l'ensemble des démons ne sont jamais expulsés en raison de ces conditions de nœuds.

3.6.1.5. Tolérer toutes les tares

Vous pouvez configurer un pod pour qu'il tolère toutes les plaintes en ajoutant une tolérance **operator: "Exists"** sans paramètres **key** et **value**. Les pods ayant cette tolérance ne sont pas retirés d'un nœud qui a des taches.

Pod spécification pour la tolérance de toutes les taches

```
spec:
  tolerations:
  - operator: "Exists"
```

3.6.2. Ajout de taches et de tolérances

Vous ajoutez des tolérances aux modules et des taches aux nœuds pour permettre au nœud de contrôler les modules qui doivent ou ne doivent pas être planifiés sur eux. Pour les pods et les nœuds existants, vous devez d'abord ajouter la tolérance au pod, puis ajouter la taint au nœud afin d'éviter que les pods ne soient retirés du nœud avant que vous ne puissiez ajouter la tolérance.

Procédure

1. Ajouter une tolérance à un pod en modifiant la spécification **Pod** pour y inclure une strophe **tolerations**:

   Exemple de fichier de configuration d'un pod avec un opérateur Equal

   ```
   spec:
     tolerations:
     - key: "key1" 1
       value: "value1"
       operator: "Equal"
       effect: "NoExecute"
       tolerationSeconds: 3600 2
   ```

   1 Les paramètres de tolérance, tels que décrits dans le tableau **Taint and toleration components**.

2 Le paramètre **tolerationSeconds** indique la durée pendant laquelle un pod peut rester lié à un nœud avant d'être expulsé.

Par exemple :

Exemple de fichier de configuration d'un pod avec un opérateur Exists

```
spec:
  tolerations:
   - key: "key1"
     operator: "Exists" 1
     effect: "NoExecute"
     tolerationSeconds: 3600
```

1 L'opérateur **Exists** ne prend pas de **value**.

Cet exemple place une tare sur **node1** qui a la clé **key1**, la valeur **value1**, et l'effet de tare **NoExecute**.

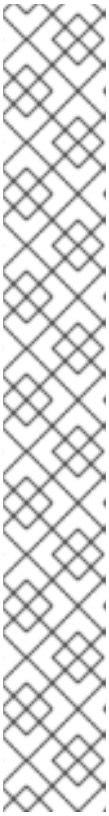
2. Ajoutez une tare à un nœud en utilisant la commande suivante avec les paramètres décrits dans le tableau **Taint and toleration components**:

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

Par exemple :

```
$ oc adm taint nodes node1 key1=value1:NoExecute
```

Cette commande place une tare sur **node1** qui a pour clé **key1**, pour valeur **value1**, et pour effet **NoExecute**.



NOTE

Si vous ajoutez une erreur **NoSchedule** à un nœud du plan de contrôle, le nœud doit avoir l'erreur **node-role.kubernetes.io/master=:NoSchedule**, qui est ajoutée par défaut.

Par exemple :

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-cdc1ab7da414629332cc4c3926e6e59c
...
spec:
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
...
```

Les tolérances du module correspondent à l'altération du nœud. Un pod avec l'une ou l'autre des tolérances peut être programmé sur **node1**.

3.6.2.1. Ajout de taches et de tolérances à l'aide d'un ensemble de machines de calcul

Vous pouvez ajouter des taches aux nœuds à l'aide d'un ensemble de machines de calcul. Tous les nœuds associés à l'objet **MachineSet** sont mis à jour avec l'erreur. Les tolérances réagissent aux taches ajoutées par un ensemble de machines de calcul de la même manière que les taches ajoutées directement aux nœuds.

Procédure

1. Ajouter une tolérance à un pod en modifiant la spécification **Pod** pour y inclure une strophe **tolerations**:

Exemple de fichier de configuration d'un pod avec l'opérateur Equal

```
spec:
  tolerations:
  - key: "key1" (1)
    value: "value1"
    operator: "Equal"
    effect: "NoExecute"
    tolerationSeconds: 3600 (2)
```

(1) Les paramètres de tolérance, tels que décrits dans le tableau **Taint and toleration components**.

(2) Le paramètre **tolerationSeconds** spécifie la durée pendant laquelle un pod est lié à un nœud avant d'être expulsé.

Par exemple :

Exemple de fichier de configuration d'un pod avec l'opérateur Exists

```
spec:
 tolerations:
 - key: "key1"
   operator: "Exists"
   effect: "NoExecute"
   tolerationSeconds: 3600
```

2. Ajouter l'altération à l'objet **MachineSet**:

   a. Modifiez le fichier YAML de **MachineSet** pour les nœuds que vous souhaitez altérer ou créez un nouvel objet **MachineSet**:

   ```
   $ oc edit machineset <machineset>
   ```

   b. Ajoutez la souillure à la section **spec.template.spec**:

   Exemple d'altération dans la spécification d'un ensemble de machines de calcul

   ```
   spec:
   ....
     template:
   ....
       spec:
         taints:
         - effect: NoExecute
           key: key1
           value: value1
   ....
   ```

   Cet exemple place une taint qui a la clé **key1**, la valeur **value1**, et l'effet de taint **NoExecute** sur les nœuds.

   c. Réduire l'échelle de la machine de calcul à 0 :

   ```
   $ oc scale --replicas=0 machineset <machineset> -n openshift-machine-api
   ```

   ASTUCE

   Vous pouvez également appliquer le YAML suivant pour mettre à l'échelle l'ensemble des machines de calcul :

   ```
   apiVersion: machine.openshift.io/v1beta1
   kind: MachineSet
   metadata:
     name: <machineset>
     namespace: openshift-machine-api
   spec:
     replicas: 0
   ```

Attendez que les machines soient retirées.

d. Augmenter l'ensemble des machines de calcul en fonction des besoins :

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

Ou bien :

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

Attendez que les machines démarrent. L'altération est ajoutée aux nœuds associés à l'objet **MachineSet**.

3.6.2.2. Lier un utilisateur à un nœud à l'aide de taches et de tolérances

Si vous souhaitez réserver un ensemble de nœuds à l'usage exclusif d'un groupe particulier d'utilisateurs, ajoutez une tolérance à leurs pods. Ajoutez ensuite une altération correspondante à ces nœuds. Les pods avec les tolérances sont autorisés à utiliser les nœuds altérés ou tout autre nœud du cluster.

Si vous voulez vous assurer que les pods sont programmés uniquement sur les nœuds altérés, ajoutez également une étiquette au même ensemble de nœuds et ajoutez une affinité de nœud aux pods de sorte que les pods ne puissent être programmés que sur des nœuds avec cette étiquette.

Procédure

Pour configurer un nœud de manière à ce que les utilisateurs ne puissent utiliser que ce nœud :

1. Ajouter une tare correspondante à ces nœuds :
   Par exemple :

```
$ oc adm taint nodes node1 dedicated=groupName:NoSchedule
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour ajouter l'altération :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: dedicated
      value: groupName
      effect: NoSchedule
```

2. Ajoutez une tolérance aux pods en écrivant un contrôleur d'admission personnalisé.

3.6.2.3. Créer un projet avec un sélecteur de nœuds et une tolérance

Vous pouvez créer un projet qui utilise un sélecteur de nœud et une tolérance, qui sont définis comme

des annotations, pour contrôler le placement des pods sur des nœuds spécifiques. Toutes les ressources ultérieures créées dans le projet sont alors planifiées sur des nœuds dont l'altération correspond à la tolérance.

Conditions préalables

- Une étiquette de sélection de nœuds a été ajoutée à un ou plusieurs nœuds en utilisant un ensemble de machines de calcul ou en éditant le nœud directement.
- Une tare a été ajoutée à un ou plusieurs nœuds en utilisant un ensemble de machines de calcul ou en modifiant le nœud directement.

Procédure

1. Créer une définition de ressource **Project**, en spécifiant un sélecteur de nœud et une tolérance dans la section **metadata.annotations**:

   Exemple de fichier project.yaml

```
kind: Project
apiVersion: project.openshift.io/v1
metadata:
  name: <project_name> 1
  annotations:
    openshift.io/node-selector: '<label>' 2
    scheduler.alpha.kubernetes.io/defaultTolerations: >-
      [{"operator": "Exists", "effect": "NoSchedule", "key":
      "<key_name>"} 3
      ]
```

   1. Le nom du projet.
   2. L'étiquette par défaut du sélecteur de nœud.
   3. Les paramètres de tolérance, tels que décrits dans le tableau **Taint and toleration components**. Cet exemple utilise l'effet **NoSchedule**, qui permet aux pods existants sur le nœud de rester, et l'opérateur **Exists**, qui ne prend pas de valeur.

2. Utilisez la commande **oc apply** pour créer le projet :

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

Toutes les ressources créées ultérieurement dans l'espace de noms **<project_name>** doivent désormais être planifiées sur les nœuds spécifiés.

Ressources supplémentaires

- Ajout manuel de taches et de tolérances aux nœuds ou avec des ensembles de machines de calcul
- Création de sélecteurs de nœuds pour l'ensemble du projet
- Placement en pods des charges de travail des opérateurs

3.6.2.4. Contrôle des nœuds avec du matériel spécial à l'aide de taches et de tolérances

Dans un cluster où un petit sous-ensemble de nœuds dispose d'un matériel spécialisé, vous pouvez utiliser les taints et les tolérances pour empêcher les pods qui n'ont pas besoin de ce matériel spécialisé d'utiliser ces nœuds, laissant ainsi les nœuds aux pods qui ont besoin de ce matériel spécialisé. Vous pouvez également exiger que les modules qui ont besoin d'un matériel spécialisé utilisent des nœuds spécifiques.

Vous pouvez y parvenir en ajoutant une tolérance aux pods qui ont besoin d'un matériel spécial et en altérant les nœuds qui disposent de ce matériel.

Procédure

Pour s'assurer que les nœuds dotés d'un matériel spécialisé sont réservés à des modules spécifiques :

1. Ajouter une tolérance aux nacelles qui ont besoin d'un matériel spécial.
   Par exemple :

```
spec:
  tolerations:
    - key: "disktype"
      value: "ssd"
      operator: "Equal"
      effect: "NoSchedule"
      tolerationSeconds: 3600
```

2. Attaquez les nœuds dotés du matériel spécialisé à l'aide de l'une des commandes suivantes :

```
oc adm taint nodes <node-name> disktype=ssd:NoSchedule
```

   Ou bien :

```
oc adm taint nodes <node-name> disktype=ssd:PreferNoSchedule
```

   ASTUCE

   Vous pouvez également appliquer le YAML suivant pour ajouter l'altération :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    ...
spec:
  taints:
    - key: disktype
      value: ssd
      effect: PreferNoSchedule
```

3.6.3. Supprimer les tares et les tolérances

Vous pouvez supprimer les tares des nœuds et les tolérances des nacelles si nécessaire. Vous devez d'abord ajouter la tolérance au module, puis ajouter l'altération au nœud afin d'éviter que des modules soient retirés du nœud avant que vous ne puissiez ajouter la tolérance.

Procédure

Éliminer les tares et les tolérances :

1. Pour supprimer une tare d'un nœud :

```
$ oc adm taint nodes <node-name> <key>-
```

Par exemple :

```
$ oc adm taint nodes ip-10-0-132-248.ec2.internal key1-
```

Exemple de sortie

```
node/ip-10-0-132-248.ec2.internal untainted
```

2. Pour supprimer une tolérance d'un pod, modifiez la spécification de **Pod** pour supprimer la tolérance :

```
spec:
  tolerations:
  - key: "key2"
    operator: "Exists"
    effect: "NoExecute"
    tolerationSeconds: 3600
```

3.7. PLACER DES PODS SUR DES NŒUDS SPÉCIFIQUES EN UTILISANT DES SÉLECTEURS DE NŒUDS

Un site *node selector* spécifie une carte de paires clé/valeur définies à l'aide d'étiquettes personnalisées sur les nœuds et de sélecteurs spécifiés dans les pods.

Pour qu'un module puisse être exécuté sur un nœud, il doit avoir le même sélecteur de nœud clé/valeur que l'étiquette du nœud.

3.7.1. À propos des sélecteurs de nœuds

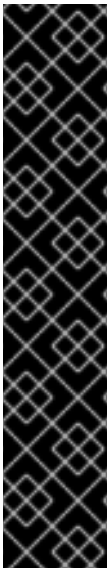
Vous pouvez utiliser des sélecteurs de nœuds sur les pods et des étiquettes sur les nœuds pour contrôler l'endroit où le pod est planifié. Avec les sélecteurs de nœuds, OpenShift Container Platform planifie les pods sur les nœuds qui contiennent les étiquettes correspondantes.

Vous pouvez utiliser un sélecteur de nœud pour placer des pods spécifiques sur des nœuds spécifiques, des sélecteurs de nœuds à l'échelle du cluster pour placer de nouveaux pods sur des nœuds spécifiques n'importe où dans le cluster, et des sélecteurs de nœuds de projet pour placer de nouveaux pods dans un projet sur des nœuds spécifiques.

Par exemple, en tant qu'administrateur de cluster, vous pouvez créer une infrastructure dans laquelle les développeurs d'applications peuvent déployer des pods uniquement sur les nœuds les plus proches de leur emplacement géographique en incluant un sélecteur de nœud dans chaque pod qu'ils créent. Dans cet exemple, le cluster se compose de cinq centres de données répartis dans deux régions. Aux États-

Unis, les nœuds sont étiquetés **us-east**, **us-central** ou **us-west**. Dans la région Asie-Pacifique (APAC), les nœuds sont étiquetés **apac-east** ou **apac-west**. Les développeurs peuvent ajouter un sélecteur de nœud aux pods qu'ils créent pour s'assurer que les pods sont planifiés sur ces nœuds.

Un pod n'est pas programmé si l'objet **Pod** contient un sélecteur de nœud, mais qu'aucun nœud n'a d'étiquette correspondante.



IMPORTANT

Si vous utilisez des sélecteurs de nœuds et des affinités de nœuds dans la même configuration de pods, les règles suivantes contrôlent le placement des pods sur les nœuds :

- Si vous configurez à la fois **nodeSelector** et **nodeAffinity**, les deux conditions doivent être remplies pour que le pod soit planifié sur un nœud candidat.
- Si vous spécifiez plusieurs **nodeSelectorTerms** associés à des types **nodeAffinity**, le module peut être programmé sur un nœud si l'un des **nodeSelectorTerms** est satisfait.
- Si vous spécifiez plusieurs **matchExpressions** associés à **nodeSelectorTerms**, le module ne peut être programmé sur un nœud que si tous les **matchExpressions** sont satisfaits.

Sélecteurs de nœuds sur des pods et des nœuds spécifiques

Vous pouvez contrôler le nœud sur lequel un pod spécifique est programmé en utilisant des sélecteurs de nœuds et des étiquettes.
Pour utiliser les sélecteurs de nœuds et les étiquettes, il faut d'abord étiqueter le nœud afin d'éviter que les modules ne soient désordonnés, puis ajouter le sélecteur de nœud au module.

NOTE

Vous ne pouvez pas ajouter un sélecteur de nœud directement à un module programmé existant. Vous devez étiqueter l'objet qui contrôle le module, tel que la configuration de déploiement.

Par exemple, l'objet **Node** suivant porte l'étiquette **region: east**:

Exemple d'objet Node avec une étiquette

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    kubernetes.io/os: linux
    failure-domain.beta.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: ''
    failure-domain.beta.kubernetes.io/region: us-east-1
```

```
    node.openshift.io/os_id: rhcos
    beta.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    beta.kubernetes.io/arch: amd64
    region: east (1)
    type: user-node
```

1. Étiquettes correspondant au sélecteur de nœuds de pods.

Un pod possède le sélecteur de nœuds **type: user-node,region: east**:

Exemple d'objet Pod avec des sélecteurs de nœuds

```
apiVersion: v1
kind: Pod

....

spec:
  nodeSelector: (1)
    region: east
    type: user-node
```

1. Sélecteurs de nœuds correspondant à l'étiquette du nœud. Le nœud doit avoir une étiquette pour chaque sélecteur de nœud.

Lorsque vous créez le pod à l'aide de la spécification de pod d'exemple, il peut être planifié sur le nœud d'exemple.

Sélecteurs de nœuds par défaut pour l'ensemble du cluster

Avec des sélecteurs de nœuds par défaut à l'échelle du cluster, lorsque vous créez un pod dans ce cluster, OpenShift Container Platform ajoute les sélecteurs de nœuds par défaut au pod et planifie le pod sur des nœuds avec des étiquettes correspondantes.
Par exemple, l'objet **Scheduler** suivant possède les sélecteurs de nœuds par défaut pour l'ensemble du cluster **region=east** et **type=user-node**:

Exemple d'ordonnanceur Opérateur Ressource personnalisée

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...

spec:
  defaultNodeSelector: type=user-node,region=east
...
```

Un nœud de cette grappe possède les étiquettes **type=user-node,region=east**:

Exemple d'objet Node

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
...
  labels:
    region: east
    type: user-node
...
```

Exemple d'objet Pod avec un sélecteur de nœud

```
apiVersion: v1
kind: Pod
...

spec:
  nodeSelector:
    region: east
...
```

Lorsque vous créez le pod à l'aide de la spécification de pod d'exemple dans le cluster d'exemple, le pod est créé avec le sélecteur de nœud à l'échelle du cluster et est planifié sur le nœud étiqueté :

Exemple de liste de pods avec le pod sur le nœud étiqueté

```
NAME    READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
pod-s1   1/1     Running   0          20s   10.131.2.6   ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>          <none>
```

NOTE

Si le projet dans lequel vous créez le module dispose d'un sélecteur de nœud de projet, ce sélecteur a la préférence sur un sélecteur de nœud à l'échelle du cluster. Votre module n'est pas créé ou planifié s'il n'a pas de sélecteur de nœud de projet.

Sélecteurs de nœuds de projet

Avec les sélecteurs de nœuds de projet, lorsque vous créez un pod dans ce projet, OpenShift Container Platform ajoute les sélecteurs de nœuds au pod et planifie les pods sur un nœud avec les étiquettes correspondantes. S'il existe un sélecteur de nœuds par défaut à l'échelle du cluster, le sélecteur de nœuds du projet est privilégié.
Par exemple, le projet suivant possède le sélecteur de nœuds **region=east**:

Exemple d'objet Namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
```

```
  annotations:
    openshift.io/node-selector: "region=east"
...
```

Le nœud suivant possède les étiquettes **type=user-node,region=east**:

Exemple d'objet Node

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
...
  labels:
    region: east
    type: user-node
...
```

Lorsque vous créez le pod à l'aide de l'exemple de spécification de pod dans cet exemple de projet, le pod est créé avec les sélecteurs de nœuds du projet et est planifié sur le nœud étiqueté :

Exemple d'objet Pod

```
apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
...
spec:
  nodeSelector:
    region: east
    type: user-node
...
```

Exemple de liste de pods avec le pod sur le nœud étiqueté

```
NAME    READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
pod-s1   1/1    Running   0          20s   10.131.2.6   ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>           <none>
```

Un module du projet n'est pas créé ou planifié s'il contient des sélecteurs de nœuds différents. Par exemple, si vous déployez le module suivant dans le projet d'exemple, il ne sera pas créé :

Exemple Pod objet avec un sélecteur de nœud invalide

```
apiVersion: v1
kind: Pod
...

spec:
  nodeSelector:
```

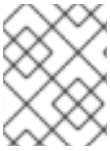
```
    region: west

....
```

3.7.2. Utilisation de sélecteurs de nœuds pour contrôler le placement des pods

Vous pouvez utiliser des sélecteurs de nœuds sur les pods et des étiquettes sur les nœuds pour contrôler l'endroit où le pod est planifié. Avec les sélecteurs de nœuds, OpenShift Container Platform planifie les pods sur les nœuds qui contiennent les étiquettes correspondantes.

Vous ajoutez des étiquettes à un nœud, à un ensemble de machines de calcul ou à une configuration de machine. L'ajout de l'étiquette à l'ensemble de machines de calcul garantit que si le nœud ou la machine tombe en panne, les nouveaux nœuds disposent de l'étiquette. Les étiquettes ajoutées à un nœud ou à une configuration de machine ne persistent pas si le nœud ou la machine tombe en panne.

Pour ajouter des sélecteurs de nœuds à un module existant, ajoutez un sélecteur de nœuds à l'objet de contrôle de ce module, tel que l'objet **ReplicaSet**, l'objet **DaemonSet**, l'objet **StatefulSet**, l'objet **Deployment** ou l'objet **DeploymentConfig**. Tous les modules existants sous cet objet de contrôle sont recréés sur un nœud avec une étiquette correspondante. Si vous créez un nouveau module, vous pouvez ajouter le sélecteur de nœud directement à la spécification **Pod**.



NOTE

Vous ne pouvez pas ajouter un sélecteur de nœud directement à un module planifié existant.

Conditions préalables

Pour ajouter un sélecteur de nœud à des modules existants, déterminez l'objet de contrôle de ce module. Par exemple, le module **router-default-66d5cf9464-m2g75** est contrôlé par l'ensemble de répliques **router-default-66d5cf9464**:

```
$ oc describe pod router-default-66d5cf9464-7pwkc

Name:             router-default-66d5cf9464-7pwkc
Namespace:        openshift-ingress

....

Controlled By:    ReplicaSet/router-default-66d5cf9464
```

La console web liste l'objet de contrôle sous **ownerReferences** dans le pod YAML :

```
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: router-default-66d5cf9464
      uid: d81dd094-da26-11e9-a48a-128e7edf0312
      controller: true
      blockOwnerDeletion: true
```

Procédure

1. Ajoutez des étiquettes à un nœud en utilisant un ensemble de machines de calcul ou en éditant le nœud directement :

   - Utilisez un objet **MachineSet** pour ajouter des étiquettes aux nœuds gérés par l'ensemble de machines de calcul lors de la création d'un nœud :

     a. Exécutez la commande suivante pour ajouter des étiquettes à un objet **MachineSet**:

     ```
     $ oc patch MachineSet <name> --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
     <value>","<key>"="<value>"}}]'  -n openshift-machine-api
     ```

     Par exemple :

     ```
     $ oc patch MachineSet abc612-msrtw-worker-us-east-1c  --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
     node","region":"east"}}]'  -n openshift-machine-api
     ```

     ASTUCE

     Vous pouvez également appliquer le YAML suivant pour ajouter des étiquettes à un ensemble de machines de calcul :

     ```
     apiVersion: machine.openshift.io/v1beta1
     kind: MachineSet
     metadata:
       name: <machineset>
       namespace: openshift-machine-api
     spec:
       template:
         spec:
           metadata:
             labels:
               region: "east"
               type: "user-node"
     ```

     b. Vérifiez que les étiquettes sont ajoutées à l'objet **MachineSet** en utilisant la commande **oc edit**:
     Par exemple :

     ```
     $ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api
     ```

     Exemple d'objet MachineSet

     ```
     apiVersion: machine.openshift.io/v1beta1
     kind: MachineSet

     ....

     spec:
     ...
       template:
         metadata:
     ```

```
...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
....
```

- Ajouter des étiquettes directement à un nœud :

  a. Modifiez l'objet **Node** pour le nœud :

  ```
  $ oc label nodes <name> <key>=<value>
  ```

  Par exemple, pour étiqueter un nœud :

  ```
  $ oc label nodes ip-10-0-142-25.ec2.internal type=user-node region=east
  ```

  ASTUCE

  Vous pouvez également appliquer le langage YAML suivant pour ajouter des étiquettes à un nœud :

  ```
  kind: Node
  apiVersion: v1
  metadata:
    name: <node_name>
    labels:
      type: "user-node"
      region: "east"
  ```

  b. Vérifiez que les étiquettes sont ajoutées au nœud :

  ```
  $ oc get nodes -l type=user-node,region=east
  ```

  Exemple de sortie

  ```
  NAME                          STATUS   ROLES    AGE   VERSION
  ip-10-0-142-25.ec2.internal   Ready    worker   17m   v1.25.0
  ```

2. Ajouter le sélecteur de nœud correspondant à un pod :

   - Pour ajouter un sélecteur de nœud aux modules existants et futurs, ajoutez un sélecteur de nœud à l'objet de contrôle des modules :

     Exemple ReplicaSet objet avec étiquettes

     ```
     kind: ReplicaSet

     ....

     spec:
     ```

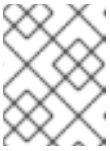
```
....

 template:
   metadata:
     creationTimestamp: null
     labels:
       ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
       pod-template-hash: 66d5cf9464
   spec:
     nodeSelector:
       kubernetes.io/os: linux
       node-role.kubernetes.io/worker: ''
       type: user-node 1
```

1 Ajouter le sélecteur de nœud.

- Pour ajouter un sélecteur de nœud à un nouveau pod spécifique, ajoutez le sélecteur à l'objet **Pod** directement :

  Exemple d'objet Pod avec un sélecteur de nœud

```
apiVersion: v1
kind: Pod

....

spec:
  nodeSelector:
    region: east
    type: user-node
```



NOTE

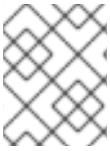
Vous ne pouvez pas ajouter un sélecteur de nœud directement à un module planifié existant.

3.7.3. Création de sélecteurs de nœuds par défaut pour l'ensemble du cluster

Vous pouvez utiliser des sélecteurs de nœuds par défaut à l'échelle du cluster sur les pods ainsi que des étiquettes sur les nœuds pour contraindre tous les pods créés dans un cluster à des nœuds spécifiques.

Avec des sélecteurs de nœuds à l'échelle du cluster, lorsque vous créez un pod dans ce cluster, OpenShift Container Platform ajoute les sélecteurs de nœuds par défaut au pod et planifie le pod sur des nœuds avec des étiquettes correspondantes.

Vous configurez les sélecteurs de nœuds à l'échelle du cluster en modifiant la ressource personnalisée (CR) de l'opérateur d'ordonnancement. Vous ajoutez des étiquettes à un nœud, à un ensemble de machines de calcul ou à une configuration de machine. L'ajout de l'étiquette à l'ensemble de machines de calcul garantit que si le nœud ou la machine tombe en panne, les nouveaux nœuds disposent de l'étiquette. Les étiquettes ajoutées à un nœud ou à une configuration de machine ne persistent pas si le nœud ou la machine tombe en panne.



NOTE

Vous pouvez ajouter des paires clé/valeur supplémentaires à un pod. Mais vous ne pouvez pas ajouter une valeur différente pour une clé par défaut.

Procédure

Pour ajouter un sélecteur de nœuds par défaut à l'échelle du cluster :

1. Modifiez le CR de l'opérateur d'ordonnancement pour ajouter les sélecteurs de nœuds par défaut à l'échelle du cluster :

```
$ oc edit scheduler cluster
```

Exemple d'opérateur d'ordonnancement CR avec un sélecteur de nœuds

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: type=user-node,region=east (1)
  mastersSchedulable: false
```

(1) Ajouter un sélecteur de nœud avec les paires **<key>:<value>** appropriées.

Après avoir effectué cette modification, attendez que les pods du projet **openshift-kube-apiserver** soient redéployés. Cela peut prendre plusieurs minutes. Le sélecteur de nœuds par défaut à l'échelle du cluster ne prend pas effet tant que les pods ne sont pas redéployés.

2. Ajoutez des étiquettes à un nœud en utilisant un ensemble de machines de calcul ou en éditant le nœud directement :

- Utiliser un ensemble de machines de calcul pour ajouter des étiquettes aux nœuds gérés par l'ensemble de machines de calcul lors de la création d'un nœud :

a. Exécutez la commande suivante pour ajouter des étiquettes à un objet **MachineSet**:

```
$ oc patch MachineSet <name> --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
<value>","<key>"="<value>"}}]'  -n openshift-machine-api (1)
```

(1) Ajouter une paire **<key>/<value>** pour chaque étiquette.

Par exemple :

```
$ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
node","region":"east"}}]'  -n openshift-machine-api
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour ajouter des étiquettes à un ensemble de machines de calcul :

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  template:
    spec:
      metadata:
        labels:
          region: "east"
          type: "user-node"
```

b. Vérifiez que les étiquettes sont ajoutées à l'objet **MachineSet** en utilisant la commande **oc edit**:
Par exemple :

```
$ oc edit MachineSet abc612-msrtw-worker-us-east-1c -n openshift-machine-api
```

Exemple d'objet MachineSet

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
  ...
spec:
  ...
  template:
    metadata:
  ...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
  ...
```

c. Redéployez les nœuds associés à cet ensemble de machines de calcul en réduisant l'échelle à **0** et en augmentant l'échelle des nœuds :
Par exemple :

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

d. Lorsque les nœuds sont prêts et disponibles, vérifiez que l'étiquette a été ajoutée aux nœuds à l'aide de la commande **oc get**:

```
$ oc get nodes -l <key>=<value>
```

Par exemple :

```
$ oc get nodes -l type=user-node
```

Exemple de sortie

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp   Ready    worker   61s   v1.25.0
```

- Ajouter des étiquettes directement à un nœud :

  a. Modifiez l'objet **Node** pour le nœud :

```
$ oc label nodes <name> <key>=<value>
```

Par exemple, pour étiqueter un nœud :

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49 type=user-node
region=east
```

ASTUCE

Vous pouvez également appliquer le langage YAML suivant pour ajouter des étiquettes à un nœud :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

  b. Vérifiez que les étiquettes sont ajoutées au nœud à l'aide de la commande **oc get**:

```
$ oc get nodes -l <key>=<value>,<key>=<valeur>
```

Par exemple :

```
$ oc get nodes -l type=user-node,region=east
```

Exemple de sortie

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.25.0
```

3.7.4. Création de sélecteurs de nœuds pour l'ensemble du projet

Vous pouvez utiliser des sélecteurs de nœuds dans un projet ainsi que des étiquettes sur les nœuds pour contraindre tous les pods créés dans ce projet aux nœuds étiquetés.

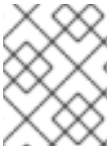
Lorsque vous créez un pod dans ce projet, OpenShift Container Platform ajoute les sélecteurs de nœuds aux pods du projet et planifie les pods sur un nœud avec des étiquettes correspondantes dans le projet. S'il existe un sélecteur de nœud par défaut à l'échelle du cluster, le sélecteur de nœud du projet a la préférence.

Vous ajoutez des sélecteurs de nœuds à un projet en modifiant l'objet **Namespace** pour ajouter le paramètre **openshift.io/node-selector**. Vous ajoutez des étiquettes à un nœud, à un ensemble de machines de calcul ou à une configuration de machine. L'ajout de l'étiquette à l'ensemble de machines de calcul garantit que si le nœud ou la machine tombe en panne, les nouveaux nœuds disposent de l'étiquette. Les étiquettes ajoutées à un nœud ou à une configuration de machine ne persistent pas si le nœud ou la machine tombe en panne.

Un pod n'est pas planifié si l'objet **Pod** contient un sélecteur de nœuds, mais qu'aucun projet n'a de sélecteur de nœuds correspondant. Lorsque vous créez un pod à partir de cette spécification, vous recevez une erreur similaire au message suivant :

Exemple de message d'erreur

```
Error from server (Forbidden): error when creating "pod.yaml": pods "pod-4" is forbidden: pod node label selector conflicts with its project node label selector
```



NOTE

Vous pouvez ajouter des paires clé/valeur supplémentaires à un module. Mais vous ne pouvez pas ajouter une valeur différente pour une clé de projet.

Procédure

Pour ajouter un sélecteur de nœud de projet par défaut :

1. Créez un espace de noms ou modifiez un espace de noms existant pour ajouter le paramètre **openshift.io/node-selector**:

```
oc edit namespace <name> $ oc edit namespace <name>
```

Exemple de sortie

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    openshift.io/node-selector: "type=user-node,region=east" 1
    openshift.io/description: ""
    openshift.io/display-name: ""
    openshift.io/requester: kube:admin
    openshift.io/sa.scc.mcs: s0:c30,c5
    openshift.io/sa.scc.supplemental-groups: 1000880000/10000
    openshift.io/sa.scc.uid-range: 1000880000/10000
  creationTimestamp: "2021-05-10T12:35:04Z"
  labels:
    kubernetes.io/metadata.name: demo
```

```
  name: demo
  resourceVersion: "145537"
  uid: 3f8786e3-1fcb-42e3-a0e3-e2ac54d15001
spec:
  finalizers:
  - kubernetes
```

1 Ajoutez les paires **openshift.io/node-selector** et **<key>:<value>** appropriées.

2. Ajoutez des étiquettes à un nœud en utilisant un ensemble de machines de calcul ou en éditant le nœud directement :

   - Utilisez un objet **MachineSet** pour ajouter des étiquettes aux nœuds gérés par l'ensemble de machines de calcul lors de la création d'un nœud :

     a. Exécutez la commande suivante pour ajouter des étiquettes à un objet **MachineSet**:

     ```
     $ oc patch MachineSet <name> --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"<key>="
     <value>","<key>"="<value>"}}]'  -n openshift-machine-api
     ```

     Par exemple :

     ```
     $ oc patch MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c --type='json' -
     p='[{"op":"add","path":"/spec/template/spec/metadata/labels", "value":{"type":"user-
     node","region":"east"}}]'  -n openshift-machine-api
     ```

     ASTUCE

     Vous pouvez également appliquer le YAML suivant pour ajouter des étiquettes à un ensemble de machines de calcul :

     ```
     apiVersion: machine.openshift.io/v1beta1
     kind: MachineSet
     metadata:
       name: <machineset>
       namespace: openshift-machine-api
     spec:
       template:
         spec:
           metadata:
             labels:
               region: "east"
               type: "user-node"
     ```

     b. Vérifiez que les étiquettes sont ajoutées à l'objet **MachineSet** en utilisant la commande **oc edit**:
     Par exemple :

     ```
     $ oc edit MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
     ```

     Exemple de sortie

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
...
spec:
...
  template:
    metadata:
...
    spec:
      metadata:
        labels:
          region: east
          type: user-node
```

c. Redéployer les nœuds associés à cet ensemble de machines de calcul :
Par exemple :

```
$ oc scale --replicas=0 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

```
$ oc scale --replicas=1 MachineSet ci-ln-l8nry52-f76d1-hl7m7-worker-c -n openshift-machine-api
```

d. Lorsque les nœuds sont prêts et disponibles, vérifiez que l'étiquette a été ajoutée aux nœuds à l'aide de la commande **oc get**:

```
$ oc get nodes -l <key>=<value>
```

Par exemple :

```
$ oc get nodes -l type=user-node,region=east
```

Exemple de sortie

```
NAME                                        STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-c-vmqzp   Ready    worker   61s   v1.25.0
```

- Ajouter des étiquettes directement à un nœud :

a. Modifiez l'objet **Node** pour ajouter des étiquettes :

```
$ oc label <resource> <name> <key>=<value>
```

Par exemple, pour étiqueter un nœud :

```
$ oc label nodes ci-ln-l8nry52-f76d1-hl7m7-worker-c-tgq49 type=user-node region=east
```

ASTUCE

Vous pouvez également appliquer le langage YAML suivant pour ajouter des étiquettes à un nœud :

```
kind: Node
apiVersion: v1
metadata:
  name: <node_name>
  labels:
    type: "user-node"
    region: "east"
```

b. Vérifiez que les étiquettes sont ajoutées à l'objet **Node** à l'aide de la commande **oc get**:

```
$ oc get nodes -l <key>=<value>
```

Par exemple :

```
$ oc get nodes -l type=user-node,region=east
```

Exemple de sortie

```
NAME                                STATUS   ROLES    AGE   VERSION
ci-ln-l8nry52-f76d1-hl7m7-worker-b-tgq49   Ready    worker   17m   v1.25.0
```

Ressources supplémentaires

- Créer un projet avec un sélecteur de nœuds et une tolérance

3.8. CONTRÔLE DU PLACEMENT DES PODS À L'AIDE DE CONTRAINTES D'ÉTALEMENT DE LA TOPOLOGIE DES PODS

Vous pouvez utiliser les contraintes d'étalement de la topologie des pods pour contrôler le placement de vos pods sur des nœuds, des zones, des régions ou d'autres domaines topologiques définis par l'utilisateur.

3.8.1. A propos des contraintes d'étalement de la topologie des pods

L'utilisation de *pod topology spread constraint* permet de contrôler finement la distribution des pods dans les domaines de défaillance afin d'obtenir une haute disponibilité et une utilisation plus efficace des ressources.

Les administrateurs d'OpenShift Container Platform peuvent étiqueter les nœuds pour fournir des informations topologiques, telles que des régions, des zones, des nœuds ou d'autres domaines définis par l'utilisateur. Une fois ces étiquettes définies sur les nœuds, les utilisateurs peuvent alors définir des contraintes d'étalement de la topologie des pods pour contrôler le placement des pods dans ces domaines topologiques.

Vous spécifiez les pods à regrouper, les domaines topologiques dans lesquels ils sont répartis et l'inclinaison acceptable. Seuls les pods situés dans le même espace de noms sont mis en correspondance et regroupés lorsqu'ils sont répartis en raison d'une contrainte.

3.8.2. Configuration des contraintes d'étalement de la topologie des pods

Les étapes suivantes montrent comment configurer les contraintes d'étalement de la topologie des pods pour distribuer les pods qui correspondent aux étiquettes spécifiées en fonction de leur zone.

Vous pouvez spécifier plusieurs contraintes d'étalement de la topologie des pods, mais vous devez vous assurer qu'elles n'entrent pas en conflit les unes avec les autres. Toutes les contraintes d'étalement de la topologie du pod doivent être satisfaites pour qu'un pod soit placé.

Conditions préalables

- Un administrateur de cluster a ajouté les étiquettes requises aux nœuds.

Procédure

1. Créez une spécification **Pod** et spécifiez une contrainte d'étalement de la topologie du pod :

Exemple de fichier pod-spec.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1 (1)
    topologyKey: topology.kubernetes.io/zone (2)
    whenUnsatisfiable: DoNotSchedule (3)
    labelSelector: (4)
      matchLabels:
        foo: bar (5)
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
```

(1) La différence maximale en nombre de pods entre deux domaines topologiques. La valeur par défaut est **1**, et vous ne pouvez pas spécifier une valeur de **0**.

(2) Clé de l'étiquette d'un nœud. Les nœuds ayant cette clé et une valeur identique sont considérés comme faisant partie de la même topologie.

(3) Comment traiter un pod s'il ne satisfait pas à la contrainte d'étalement. La valeur par défaut est **DoNotSchedule**, ce qui indique au planificateur de ne pas planifier le module. La valeur **ScheduleAnyway** permet de planifier le module, mais l'ordonnanceur donne la priorité au respect de la contrainte de dispersion afin de ne pas aggraver le déséquilibre de la grappe.

(4) Les gousses qui correspondent à ce sélecteur d'étiquette sont comptées et reconnues comme un groupe lorsqu'elles sont étalées pour satisfaire à la contrainte. Veillez à spécifier un sélecteur d'étiquette, sinon aucune cosse ne pourra être prise en compte.

(5) Veillez à ce que cette spécification **Pod** définisse également ses étiquettes en fonction de ce sélecteur d'étiquettes si vous voulez qu'elle soit comptée correctement à l'avenir.

2. Créer la capsule :

```
$ oc create -f pod-spec.yaml
```

3.8.3. Exemple de contraintes de répartition de la topologie des pods

Les exemples suivants illustrent les configurations de contraintes de propagation de la topologie des pods.

3.8.3.1. Exemple de contrainte de diffusion d'une topologie de pod unique

Cet exemple de spécification **Pod** définit une contrainte d'étalement de la topologie du pod. Elle s'applique aux pods étiquetés **foo:bar**, se répartit entre les zones, spécifie une inclinaison de **1** et ne planifie pas le pod s'il ne répond pas à ces exigences.

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
```

3.8.3.2. Exemple de contraintes d'étalement de la topologie des pods multiples

Cet exemple de spécification **Pod** définit deux contraintes d'étalement de la topologie des pods. Les deux correspondent aux pods étiquetés **foo:bar**, spécifient une inclinaison de **1**, et ne planifient pas le pod s'il ne répond pas à ces exigences.

La première contrainte distribue les nacelles en fonction d'une étiquette définie par l'utilisateur **node** et la deuxième contrainte distribue les nacelles en fonction d'une étiquette définie par l'utilisateur **rack**. Les deux contraintes doivent être respectées pour que le module soit programmé.

```
kind: Pod
apiVersion: v1
metadata:
  name: my-pod-2
  labels:
    foo: bar
spec:
  topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: node
    whenUnsatisfiable: DoNotSchedule
```

```
    labelSelector:
      matchLabels:
        foo: bar
  - maxSkew: 1
    topologyKey: rack
    whenUnsatisfiable: DoNotSchedule
    labelSelector:
      matchLabels:
        foo: bar
  containers:
  - image: "docker.io/ocpqe/hello-pod"
    name: hello-pod
```

3.8.4. Ressources supplémentaires

- Comprendre comment mettre à jour les étiquettes sur les nœuds

3.9. ÉVICTION DES PODS À L'AIDE DE L'ORDONNANCEUR

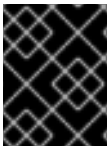
Alors que le planificateur est utilisé pour déterminer le nœud le plus approprié pour héberger un nouveau pod, le déscheduler peut être utilisé pour évincer un pod en cours d'exécution afin qu'il puisse être reprogrammé sur un nœud plus approprié.

3.9.1. À propos du déschedulateur

Vous pouvez utiliser l'ordonnanceur pour expulser les pods sur la base de stratégies spécifiques afin que les pods puissent être replanifiés sur des nœuds plus appropriés.

Vous pouvez bénéficier de la désimplantation des pods en cours d'exécution dans des situations telles que les suivantes :

- Les nœuds sont sous-utilisés ou surutilisés.
- Les exigences relatives aux pods et aux affinités entre nœuds, telles que les taches ou les étiquettes, ont changé et les décisions d'ordonnancement initiales ne sont plus appropriées pour certains nœuds.
- En cas de défaillance d'un nœud, les pods doivent être déplacés.
- De nouveaux nœuds sont ajoutés aux grappes.
- Les pods ont été redémarrés trop souvent.



IMPORTANT

L'ordonnanceur ne planifie pas le remplacement des pods expulsés. Le planificateur effectue automatiquement cette tâche pour les pods évincés.

Lorsque l'ordonnanceur décide d'expulser des pods d'un nœud, il utilise le mécanisme général suivant :

- Les pods des espaces de noms **openshift-*** et **kube-system** ne sont jamais expulsés.
- Les pods critiques dont la valeur de **priorityClassName** est **system-cluster-critical** ou **system-node-critical** ne sont jamais expulsés.

- Les pods statiques, en miroir ou autonomes qui ne font pas partie d'un contrôleur de réplication, d'un ensemble de réplicas, d'un déploiement ou d'un travail ne sont jamais expulsés car ces pods ne seront pas recréés.
- Les pods associés aux ensembles de démons ne sont jamais expulsés.
- Les pods disposant d'un stockage local ne sont jamais expulsés.
- Les pods "best effort" sont éliminés avant les pods "burstable" et "guaranteed".
- Tous les types de pods ayant l'annotation **descheduler.alpha.kubernetes.io/evict** sont éligibles à l'expulsion. Cette annotation est utilisée pour passer outre les contrôles qui empêchent l'expulsion, et l'utilisateur peut choisir le pod qui sera expulsé. Les utilisateurs doivent savoir comment et si le pod sera recréé.
- Les pods soumis au budget de perturbation des pods (PDB) ne sont pas expulsés si la désynchronisation viole leur budget de perturbation des pods (PDB). Les pods sont expulsés en utilisant la sous-ressource d'expulsion pour gérer le PDB.

3.9.2. Profils du déschedulateur

Les profils suivants sont disponibles :

AffinityAndTaints

Ce profil expulse les pods qui violent l'anti-affinité inter-pods, l'affinité des nœuds et les taches des nœuds.
Il permet de mettre en œuvre les stratégies suivantes :

- **RemovePodsViolatingInterPodAntiAffinity**: élimine les pods qui violent l'anti-affinité inter-pods.
- **RemovePodsViolatingNodeAffinity**: supprime les pods qui ne respectent pas l'affinité des nœuds.
- **RemovePodsViolatingNodeTaints**: supprime les pods qui violent **NoSchedule** taints sur les nœuds.
  Les pods dont le type d'affinité de nœud est **requiredDuringSchedulingIgnoredDuringExecution** sont supprimés.

TopologyAndDuplicates

Ce profil expulse les pods afin de répartir uniformément les pods similaires ou les pods du même domaine topologique entre les nœuds.
Il permet de mettre en œuvre les stratégies suivantes :

- **RemovePodsViolatingTopologySpreadConstraint**il détecte les domaines topologiques déséquilibrés et tente d'expulser les pods des domaines les plus vastes lorsque les contraintes de **DoNotSchedule** sont violées.
- **RemoveDuplicates**: garantit qu'il n'y a qu'un seul pod associé à un ensemble de répliques, à un contrôleur de réplication, à un déploiement ou à un travail s'exécutant sur le même nœud. S'il y en a plus, ces pods dupliqués sont évincés pour une meilleure distribution des pods dans un cluster.

LifecycleAndUtilization

Ce profil permet d'expulser les pods qui fonctionnent depuis longtemps et d'équilibrer l'utilisation des ressources entre les nœuds.
Il permet de mettre en œuvre les stratégies suivantes :

- **RemovePodsHavingTooManyRestarts**: supprime les pods dont les conteneurs ont été redémarrés trop souvent.
  Pods où la somme des redémarrages de tous les conteneurs (y compris les conteneurs d'initialisation) est supérieure à 100.

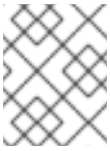
- **LowNodeUtilization**le système de gestion des pods : trouve les nœuds qui sont sous-utilisés et expulse les pods, si possible, des nœuds surutilisés dans l'espoir que la recréation des pods expulsés sera programmée sur ces nœuds sous-utilisés.
  Un nœud est considéré comme sous-utilisé si son utilisation est inférieure à 20 ou à tous les seuils (CPU, mémoire et nombre de pods).

  Un nœud est considéré comme surutilisé si son utilisation est supérieure à 50 ou à l'un des seuils (CPU, mémoire et nombre de pods).

- **PodLifeTime**: évince les nacelles trop anciennes.
  Par défaut, les pods datant de plus de 24 heures sont supprimés. Vous pouvez personnaliser la valeur de la durée de vie des pods.

SoftTopologyAndDuplicates

Ce profil est le même que celui de **TopologyAndDuplicates**, sauf que les pods ayant des contraintes topologiques douces, comme **whenUnsatisfiable: ScheduleAnyway**, sont également pris en compte pour l'expulsion.



NOTE

N'activez pas à la fois **SoftTopologyAndDuplicates** et **TopologyAndDuplicates**. L'activation des deux entraîne un conflit.

EvictPodsWithLocalStorage

Ce profil permet aux pods disposant d'un stockage local d'être éligibles à l'éviction.

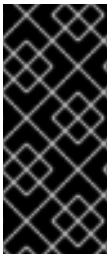
EvictPodsWithPVC

Ce profil permet aux pods ayant des réclamations de volumes persistants d'être éligibles à l'éviction. Si vous utilisez **Kubernetes NFS Subdir External Provisioner**, vous devez ajouter un espace de noms exclu pour l'espace de noms où le provisionneur est installé.

3.9.3. Installation du déschedulateur

Le descheduler n'est pas disponible par défaut. Pour l'activer, vous devez installer Kube Descheduler Operator depuis OperatorHub et activer un ou plusieurs profils de descheduler.

Par défaut, le descheduler fonctionne en mode prédictif, ce qui signifie qu'il ne fait que simuler les évictions de pods. Vous devez changer le mode en mode automatique pour que le descheduler effectue les évictions de pods.



IMPORTANT

Si vous avez activé les plans de contrôle hébergés dans votre cluster, définissez un seuil de priorité personnalisé pour réduire le risque d'éviction des pods dans les espaces de noms des plans de contrôle hébergés. Définissez le nom de la classe de seuil de priorité sur **hypershift-control-plane**, car elle a la valeur de priorité la plus basse (**100000000**) des classes de priorité du plan de contrôle hébergé.

Conditions préalables

- Privilèges d'administrateur de cluster.
- Accès à la console web d'OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Créer l'espace de noms requis pour l'opérateur Kube Descheduler.
   a. Naviguez jusqu'à **Administration → Namespaces** et cliquez sur **Create Namespace**.
   b. Entrez **openshift-kube-descheduler-operator** dans le champ **Name**, entrez **openshift.io/cluster-monitoring=true** dans le champ **Labels** pour activer les métriques du déscheduler, et cliquez sur **Create**.
3. Installez l'opérateur Kube Descheduler.
   a. Naviguez jusqu'à **Operators → OperatorHub**.
   b. Tapez **Kube Descheduler Operator** dans le champ de filtre.
   c. Sélectionnez le site **Kube Descheduler Operator** et cliquez sur **Install**.
   d. Sur la page **Install Operator**, sélectionnez **A specific namespace on the cluster**. Sélectionnez **openshift-kube-descheduler-operator** dans le menu déroulant.
   e. Ajustez les valeurs de **Update Channel** et **Approval Strategy** aux valeurs souhaitées.
   f. Cliquez sur **Install**.
4. Créer une instance de déscheduler.
   a. Dans la page **Operators → Installed Operators**, cliquez sur **Kube Descheduler Operator**.
   b. Sélectionnez l'onglet **Kube Descheduler** et cliquez sur **Create KubeDescheduler**.
   c. Modifiez les paramètres si nécessaire.
      i. Pour expulser des pods au lieu de simuler les expulsions, remplacez le champ **Mode** par **Automatic**.
      ii. Développez la section **Profiles** pour sélectionner un ou plusieurs profils à activer. Le profil **AffinityAndTaints** est activé par défaut. Cliquez sur **Add Profile** pour sélectionner d'autres profils.

NOTE

N'activez pas à la fois **TopologyAndDuplicates** et **SoftTopologyAndDuplicates**. L'activation des deux entraîne un conflit.

iii. Optionnel : Développez la section **Profile Customizations** pour définir des configurations optionnelles pour le déscheduler.

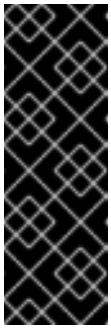
- Définissez une valeur personnalisée de durée de vie des pods pour le profil **LifecycleAndUtilization**. Utilisez le champ **podLifetime** pour définir une valeur numérique et une unité valide (**s**, **m**, ou **h**). La durée de vie par défaut est de 24 heures (**24h**).

- Définissez un seuil de priorité personnalisé pour que les pods soient pris en compte pour l'expulsion uniquement si leur priorité est inférieure à un niveau de priorité spécifié. Utilisez le champ **thresholdPriority** pour définir un seuil de priorité numérique ou utilisez le champ **thresholdPriorityClassName** pour spécifier un certain nom de classe de priorité.

NOTE

Ne spécifiez pas à la fois **thresholdPriority** et **thresholdPriorityClassName** pour le déscheduler.

- Définissez des espaces de noms spécifiques à exclure ou à inclure dans les opérations du déscheduler. Développez le champ **namespaces** et ajoutez des espaces de noms à la liste **excluded** ou **included**. Vous ne pouvez définir qu'une liste d'espaces de noms à exclure ou une liste d'espaces de noms à inclure. Notez que les espaces de noms protégés (**openshift-***, **kube-system**, **hypershift**) sont exclus par défaut.



IMPORTANT

La stratégie **LowNodeUtilization** ne prend pas en charge l'exclusion d'espaces de noms. Si le profil **LifecycleAndUtilization** est défini, ce qui active la stratégie **LowNodeUtilization**, aucun espace de noms n'est exclu, même les espaces de noms protégés. Pour éviter les expulsions des espaces de noms protégés lorsque la stratégie **LowNodeUtilization** est activée, définissez le nom de la classe de priorité sur **system-cluster-critical** ou **system-node-critical**.

- Expérimental : Définir les seuils de sous-utilisation et de surutilisation pour la stratégie **LowNodeUtilization**. Utilisez le champ **devLowNodeUtilizationThresholds** pour définir l'une des valeurs suivantes :

  - **Low**: 10 % sous-utilisés et 30 % surutilisés
  - **Medium**20% de sous-utilisation et 50% de sur-utilisation (par défaut)
  - **High**: 40 % sous-utilisés et 70 % surutilisés

NOTE

Ce paramètre est expérimental et ne doit pas être utilisé dans un environnement de production.

iv. En option : Utilisez le champ **Descheduling Interval Seconds** pour modifier le nombre de secondes entre les exécutions du descheduler. La valeur par défaut est **3600** secondes.

d. Cliquez sur **Create**.

Vous pouvez également configurer les profils et les paramètres du descheduler ultérieurement en utilisant le CLI OpenShift (**oc**). Si vous n'avez pas ajusté les profils lors de la création de l'instance de descheduler depuis la console web, le profil **AffinityAndTaints** est activé par défaut.

3.9.4. Configuration des profils de désordre

Vous pouvez configurer les profils utilisés par le descheduler pour évincer les pods.

Conditions préalables

- Privilèges de l'administrateur du cluster

Procédure

1. Modifiez l'objet **KubeDescheduler**:

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. Spécifiez un ou plusieurs profils dans la section **spec.profiles**.

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  logLevel: Normal
  managementState: Managed
  operatorLogLevel: Normal
  mode: Predictive                                  1
  profileCustomizations:
    namespaces:                                     2
      excluded:
      - my-namespace
    podLifetime: 48h                                3
    thresholdPriorityClassName: my-priority-class-name 4
  profiles:                                         5
  - AffinityAndTaints
  - TopologyAndDuplicates                           6
  - LifecycleAndUtilization
  - EvictPodsWithLocalStorage
  - EvictPodsWithPVC
```

1 Facultatif : Par défaut, le descheduler n'expulse pas les pods. Pour évincer les modules, définissez **mode** sur **Automatic**.

2 Facultatif : Définissez une liste d'espaces de noms créés par l'utilisateur à inclure ou à exclure des opérations de déscheduler. Utilisez **excluded** pour définir une liste d'espaces

IMPORTANT

La stratégie **LowNodeUtilization** ne prend pas en charge l'exclusion d'espaces de noms. Si le profil **LifecycleAndUtilization** est défini, ce qui active la stratégie **LowNodeUtilization**, aucun espace de noms n'est exclu, même les espaces de noms protégés. Pour éviter les expulsions des espaces de noms protégés lorsque la stratégie **LowNodeUtilization** est activée, définissez le nom de la classe de priorité sur **system-cluster-critical** ou **system-node-critical**.

3 Facultatif : Activez une valeur personnalisée de durée de vie du pod pour le profil **LifecycleAndUtilization**. Les unités valides sont **s**, **m**, ou **h**. La durée de vie du pod par défaut est de 24 heures.

4 Facultatif : Spécifiez un seuil de priorité pour que les pods soient pris en compte pour l'expulsion uniquement si leur priorité est inférieure au niveau spécifié. Utilisez le champ **thresholdPriority** pour définir un seuil de priorité numérique (par exemple, **10000**) ou utilisez le champ **thresholdPriorityClassName** pour spécifier un certain nom de classe de priorité (par exemple, **my-priority-class-name**). Si vous spécifiez un nom de classe de priorité, il doit déjà exister ou le descheduler lancera une erreur. Ne définissez pas à la fois **thresholdPriority** et **thresholdPriorityClassName**.

5 Ajouter un ou plusieurs profils à activer. Profils disponibles : **AffinityAndTaints**, **TopologyAndDuplicates**, **LifecycleAndUtilization**, **SoftTopologyAndDuplicates**, **EvictPodsWithLocalStorage**, et **EvictPodsWithPVC**.

6 N'activez pas à la fois **TopologyAndDuplicates** et **SoftTopologyAndDuplicates**. L'activation des deux entraîne un conflit.

Vous pouvez activer plusieurs profils ; l'ordre dans lequel les profils sont spécifiés n'est pas important.

3. Enregistrez le fichier pour appliquer les modifications.

3.9.5. Configuration de l'intervalle de déschedulation

Vous pouvez configurer le temps qui s'écoule entre deux exécutions du Descheduler. La valeur par défaut est de 3600 secondes (une heure).

Conditions préalables

- Privilèges de l'administrateur du cluster

Procédure

1. Modifiez l'objet **KubeDescheduler**:

```
$ oc edit kubedeschedulers.operator.openshift.io cluster -n openshift-kube-descheduler-operator
```

2. Mettez à jour le champ **deschedulingIntervalSeconds** avec la valeur souhaitée :

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600 1
...
```

1 Définit le nombre de secondes entre les exécutions du planificateur. Une valeur de **0** dans ce champ permet d'exécuter le descheduler une fois et de le quitter.

3. Enregistrez le fichier pour appliquer les modifications.

3.9.6. Désinstallation du déscheduler

Vous pouvez supprimer le descheduler de votre cluster en supprimant l'instance du descheduler et en désinstallant Kube Descheduler Operator. Cette procédure nettoie également l'espace de noms **KubeDescheduler** CRD et **openshift-kube-descheduler-operator**.

Conditions préalables

- Privilèges d'administrateur de cluster.
- Accès à la console web d'OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Supprime l'instance du déscheduler.
   a. Sur la page **Operators → Installed Operators**, cliquez sur **Kube Descheduler Operator**.
   b. Sélectionnez l'onglet **Kube Descheduler**.
   c. Cliquez sur le menu Options à côté de l'entrée **cluster** et sélectionnez **Delete KubeDescheduler**.
   d. Dans la boîte de dialogue de confirmation, cliquez sur **Delete**.
3. Désinstaller l'opérateur Kube Descheduler.
   a. Naviguez jusqu'à **Operators → Installed Operators**.
   b. Cliquez sur le menu Options à côté de l'entrée **Kube Descheduler Operator** et sélectionnez **Uninstall Operator**.
   c. Dans la boîte de dialogue de confirmation, cliquez sur **Uninstall**.
4. Supprimer l'espace de noms **openshift-kube-descheduler-operator**.

a. Naviguez jusqu'à **Administration → Namespaces**.

b. Saisissez **openshift-kube-descheduler-operator** dans le champ de filtre.

c. Cliquez sur le menu Options  à côté de l'entrée **openshift-kube-descheduler-operator** et sélectionnez **Delete Namespace**.

d. Dans la boîte de dialogue de confirmation, saisissez **openshift-kube-descheduler-operator** et cliquez sur **Delete**.

5. Supprimer le CRD **KubeDescheduler**.

a. Naviguez jusqu'à **Administration → Custom Resource Definitions**

b. Saisissez **KubeDescheduler** dans le champ de filtre.

c. Cliquez sur le menu Options  à côté de l'entrée **KubeDescheduler** et sélectionnez **Delete CustomResourceDefinition**.

d. Dans la boîte de dialogue de confirmation, cliquez sur **Delete**.

3.10. ORDONNANCEUR SECONDAIRE

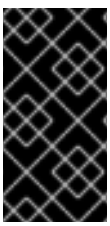
3.10.1. Vue d'ensemble de l'ordonnanceur secondaire

Vous pouvez installer l'opérateur de planification secondaire pour exécuter un planificateur secondaire personnalisé parallèlement au planificateur par défaut afin de planifier les pods.

3.10.1.1. À propos de l'opérateur de planification secondaire

L'opérateur de planificateur secondaire pour Red Hat OpenShift permet de déployer un planificateur secondaire personnalisé dans OpenShift Container Platform. Le planificateur secondaire s'exécute en même temps que le planificateur par défaut pour planifier les pods. Les configurations des pods peuvent spécifier le planificateur à utiliser.

Le planificateur personnalisé doit avoir le binaire **/bin/kube-scheduler** et être basé sur le cadre de planification Kubernetes.



IMPORTANT

Vous pouvez utiliser l'opérateur de planificateur secondaire pour déployer un planificateur secondaire personnalisé dans OpenShift Container Platform, mais Red Hat ne prend pas directement en charge la fonctionnalité du planificateur secondaire personnalisé.

L'opérateur d'ordonnancement secondaire crée les rôles par défaut et les liaisons de rôles nécessaires à l'ordonnateur secondaire. Vous pouvez spécifier les plugins de planification à activer ou à désactiver en configurant la ressource **KubeSchedulerConfiguration** pour l'ordonnanceur secondaire.

3.10.2. Notes de publication de Secondary Scheduler Operator pour Red Hat OpenShift

L'opérateur de planificateur secondaire pour Red Hat OpenShift vous permet de déployer un planificateur secondaire personnalisé dans votre cluster OpenShift Container Platform.

Ces notes de version suivent le développement de l'opérateur d'ordonnancement secondaire pour Red Hat OpenShift.

Pour plus d'informations, voir À propos de l'opérateur de planification secondaire .

3.10.2.1. Notes de version pour Secondary Scheduler Operator pour Red Hat OpenShift 1.1.0

Publié : 2022-9-1

L'avis suivant est disponible pour le Secondary Scheduler Operator pour Red Hat OpenShift 1.1.0 :

- RHSA-2022:6152

3.10.2.1.1. Nouvelles fonctionnalités et améliorations

- La configuration du contexte de sécurité de l'opérateur de planification secondaire a été mise à jour pour se conformer à l'application de l'admission à la sécurité des pods .

3.10.2.1.2. Problèmes connus

- Actuellement, vous ne pouvez pas déployer de ressources supplémentaires, telles que des cartes de configuration, des CRD ou des stratégies RBAC par l'intermédiaire de l'opérateur de planification secondaire. Toutes les ressources autres que les rôles et les liaisons de rôles requises par votre planificateur secondaire personnalisé doivent être appliquées en externe. (BZ#2071684)

3.10.3. Ordonnancement de pods à l'aide d'un ordonnanceur secondaire

Vous pouvez exécuter un planificateur secondaire personnalisé dans OpenShift Container Platform en installant l'opérateur de planificateur secondaire, en déployant le planificateur secondaire et en définissant le planificateur secondaire dans la définition du pod.

3.10.3.1. Installation de l'opérateur de planification secondaire

Vous pouvez utiliser la console web pour installer l'opérateur de planification secondaire pour Red Hat OpenShift.

Conditions préalables

- Vous avez accès au cluster avec les privilèges **cluster-admin**.
- Vous avez accès à la console web de OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Créez l'espace de noms requis pour l'opérateur de planification secondaire pour Red Hat OpenShift.
    a. Naviguez jusqu'à **Administration → Namespaces** et cliquez sur **Create Namespace**.

b. Saisissez **openshift-secondary-scheduler-operator** dans le champ **Name** et cliquez sur **Create**.

3. Installez l'opérateur de planification secondaire pour Red Hat OpenShift.

a. Naviguez jusqu'à **Operators → OperatorHub**.

b. Saisissez **Secondary Scheduler Operator for Red Hat OpenShift**dans le champ de filtre.

c. Sélectionnez le site **Secondary Scheduler Operator for Red Hat OpenShift**et cliquez sur **Install**.

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

i. Le site **Update channel** est défini sur **stable**, ce qui installe la dernière version stable du Secondary Scheduler Operator pour Red Hat OpenShift.

ii. Sélectionnez **A specific namespace on the cluster**et sélectionnez **openshift-secondary-scheduler-operator** dans le menu déroulant.

iii. Sélectionnez une stratégie **Update approval**.

- La stratégie **Automatic** permet à Operator Lifecycle Manager (OLM) de mettre automatiquement à jour l'opérateur lorsqu'une nouvelle version est disponible.
- La stratégie **Manual** exige qu'un utilisateur disposant des informations d'identification appropriées approuve la mise à jour de l'opérateur.

iv. Cliquez sur **Install**.

Vérification

1. Naviguez jusqu'à **Operators → Installed Operators**.
2. Vérifiez que **Secondary Scheduler Operator for Red Hat OpenShift**est répertorié avec **Status** de **Succeeded**.

3.10.3.2. Déploiement d'un planificateur secondaire

Après avoir installé l'opérateur de planification secondaire, vous pouvez déployer un planificateur secondaire.

Conditions préalables

- Vous avez accès au cluster avec les privilèges **cluster-admin**.
- Vous avez accès à la console web de OpenShift Container Platform.
- L'opérateur de planificateur secondaire pour Red Hat OpenShift est installé.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Créer une carte de configuration pour contenir la configuration de l'ordonnanceur secondaire.

a. Naviguez jusqu'à **Workloads → ConfigMaps**.

b. Cliquez sur **Create ConfigMap**.

c. Dans l'éditeur YAML, entrez la définition de la carte de configuration qui contient la configuration nécessaire de **KubeSchedulerConfiguration**. Par exemple :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "secondary-scheduler-config"                    1
  namespace: "openshift-secondary-scheduler-operator"   2
data:
  "config.yaml": |
    apiVersion: kubescheduler.config.k8s.io/v1beta3
    kind: KubeSchedulerConfiguration                    3
    leaderElection:
      leaderElect: false
    profiles:
      - schedulerName: secondary-scheduler              4
        plugins:                                        5
          score:
            disabled:
              - name: NodeResourcesBalancedAllocation
              - name: NodeResourcesLeastAllocated
```

1. Le nom de la carte de configuration. Il est utilisé dans le champ **Scheduler Config** lors de la création du CR **SecondaryScheduler**.
2. La carte de configuration doit être créée dans l'espace de noms **openshift-secondary-scheduler-operator**.
3. La ressource **KubeSchedulerConfiguration** pour l'ordonnanceur secondaire. Pour plus d'informations, voir **KubeSchedulerConfiguration** dans la documentation de l'API Kubernetes.
4. Nom de l'ordonnanceur secondaire. Les pods dont le champ **spec.schedulerName** contient cette valeur sont planifiés avec cet ordonnanceur secondaire.
5. Les plugins à activer ou désactiver pour l'ordonnanceur secondaire. Pour une liste des plugins d'ordonnancement par défaut, voir Scheduling plugins dans la documentation Kubernetes.

d. Cliquez sur **Create**.

3. Créer le CR **SecondaryScheduler**:

a. Naviguez jusqu'à **Operators → Installed Operators**.

b. Sélectionnez **Secondary Scheduler Operator for Red Hat OpenShift**

c. Sélectionnez l'onglet **Secondary Scheduler** et cliquez sur **Create SecondaryScheduler**.

d. La valeur par défaut du champ **Name** est **cluster**; ne modifiez pas ce nom.

e. Le champ **Scheduler Config** a pour valeur par défaut **secondary-scheduler-config**. Assurez-vous que cette valeur correspond au nom de la carte de configuration créée plus tôt dans cette procédure.

f. Dans le champ **Scheduler Image**, saisissez le nom de l'image de votre programmateur personnalisé.

IMPORTANT

Red Hat ne prend pas directement en charge la fonctionnalité de votre planificateur secondaire personnalisé.

g. Cliquez sur **Create**.

3.10.3.3. Programmation d'un pod à l'aide de l'ordonnanceur secondaire

Pour planifier un pod à l'aide du planificateur secondaire, définissez le champ **schedulerName** dans la définition du pod.

Conditions préalables

- Vous avez accès au cluster avec les privilèges **cluster-admin**.
- Vous avez accès à la console web de OpenShift Container Platform.
- L'opérateur de planificateur secondaire pour Red Hat OpenShift est installé.
- Un planificateur secondaire est configuré.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Navigate to **Workloads → Pods**.
3. Cliquez sur **Create Pod**.
4. Dans l'éditeur YAML, entrez la configuration de pod souhaitée et ajoutez le champ **schedulerName**:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
      ports:
        - containerPort: 80
  schedulerName: secondary-scheduler (1)
```

(1) Le champ **schedulerName** doit correspondre au nom défini dans la carte de configuration lorsque vous avez configuré l'ordonnanceur secondaire.

5. Cliquez sur **Create**.

Vérification

1. Connectez-vous à l'interface CLI d'OpenShift.
2. Décrivez le pod à l'aide de la commande suivante :

```
$ oc describe pod nginx -n default
```

Exemple de sortie

```
Name:         nginx
Namespace:    default
Priority:     0
Node:         ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp/10.0.128.3
...
Events:
  Type    Reason          Age   From                  Message
  ----    ------          ----  ----                  -------
  Normal  Scheduled       12s   secondary-scheduler  Successfully assigned default/nginx to
ci-ln-t0w4r1k-72292-xkqs4-worker-b-xqkxp
...
```

3. Dans le tableau des événements, recherchez l'événement dont le message est similaire à **Successfully assigned <namespace>/<pod_name> to <node_name>**.
4. Dans la colonne "From", vérifiez que l'événement a été généré par le planificateur secondaire et non par le planificateur par défaut.

NOTE

Vous pouvez également consulter les journaux de pods de **secondary-scheduler-*** dans **openshift-secondary-scheduler-namespace** pour vérifier que le pod a été planifié par le planificateur secondaire.

3.10.4. Désinstallation de l'opérateur de planification secondaire

Vous pouvez supprimer l'opérateur Secondary Scheduler Operator for Red Hat OpenShift de OpenShift Container Platform en désinstallant l'opérateur et en supprimant ses ressources associées.

3.10.4.1. Désinstallation de l'opérateur de planification secondaire

Vous pouvez désinstaller l'opérateur de planification secondaire pour Red Hat OpenShift à l'aide de la console web.

Conditions préalables

- Vous avez accès au cluster avec les privilèges **cluster-admin**.
- Vous avez accès à la console web de OpenShift Container Platform.
- L'opérateur de planificateur secondaire pour Red Hat OpenShift est installé.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Désinstallez l'opérateur de planification secondaire pour Red Hat OpenShift Operator.
   a. Naviguez jusqu'à **Operators → Installed Operators**.
   b. Cliquez sur le menu Options  à côté de l'entrée **Secondary Scheduler Operator** et cliquez sur **Uninstall Operator**.
   c. Dans la boîte de dialogue de confirmation, cliquez sur **Uninstall**.

3.10.4.2. Suppression des ressources de l'opérateur de planification secondaire

En option, après avoir désinstallé l'opérateur de planificateur secondaire pour Red Hat OpenShift, vous pouvez supprimer ses ressources connexes de votre cluster.

Conditions préalables

- Vous avez accès au cluster avec les privilèges **cluster-admin**.
- Vous avez accès à la console web de OpenShift Container Platform.

Procédure

1. Connectez-vous à la console web de OpenShift Container Platform.
2. Supprimez les CRD qui ont été installés par l'opérateur de planification secondaire :
   a. Naviguez jusqu'à **Administration → CustomResourceDefinitions**.
   b. Saisissez **SecondaryScheduler** dans le champ **Name** pour filtrer les CRD.
   c. Cliquez sur le menu Options  à côté du CRD **SecondaryScheduler** et sélectionnez **Delete Custom Resource Definition**
3. Supprimer l'espace de noms **openshift-secondary-scheduler-operator**.
   a. Naviguez jusqu'à **Administration → Namespaces**.
   b. Cliquez sur le menu Options  à côté de **openshift-secondary-scheduler-operator** et sélectionnez **Delete Namespace**.
   c. Dans la boîte de dialogue de confirmation, saisissez **openshift-secondary-scheduler-operator** dans le champ et cliquez sur **Delete**.

CHAPITRE 4. UTILISATION DES JOBS ET DES DAEMONSETS

4.1. EXÉCUTION AUTOMATIQUE DES TÂCHES D'ARRIÈRE-PLAN SUR LES NŒUDS À L'AIDE D'ENSEMBLES DE DÉMONS

En tant qu'administrateur, vous pouvez créer et utiliser des ensembles de démons pour exécuter des répliques d'un pod sur des nœuds spécifiques ou sur tous les nœuds d'un cluster OpenShift Container Platform.

Un ensemble de démons garantit que tous les nœuds (ou certains d'entre eux) exécutent une copie d'un module. Au fur et à mesure que des nœuds sont ajoutés au cluster, des pods sont ajoutés au cluster. Lorsque des nœuds sont supprimés du cluster, ces pods sont supprimés par le biais du garbage collection. La suppression d'un ensemble de démons nettoie les modules qu'il a créés.

Vous pouvez utiliser des ensembles de démons pour créer un stockage partagé, exécuter un pod de journalisation sur chaque nœud de votre cluster ou déployer un agent de surveillance sur chaque nœud.

Pour des raisons de sécurité, les administrateurs de clusters et les administrateurs de projets peuvent créer des jeux de démons.

Pour plus d'informations sur les ensembles de démons, voir la documentation Kubernetes.

IMPORTANT

La planification du jeu de démons est incompatible avec le sélecteur de nœuds par défaut du projet. Si vous ne le désactivez pas, l'ensemble de démons est restreint par la fusion avec le sélecteur de nœuds par défaut. Il en résulte des recréations fréquentes de pods sur les nœuds qui n'ont pas été sélectionnés par le sélecteur de nœuds fusionné, ce qui entraîne une charge indésirable sur le cluster.

4.1.1. Planifié par le planificateur par défaut

Un ensemble de démons garantit que tous les nœuds éligibles exécutent une copie d'un pod. Normalement, le nœud sur lequel un pod s'exécute est sélectionné par le planificateur Kubernetes. Cependant, auparavant, les pods daemon set sont créés et planifiés par le contrôleur daemon set. Cela pose les problèmes suivants :

- Comportement incohérent des pods : Les pods normaux qui attendent d'être planifiés sont créés et se trouvent dans l'état Pending, mais les pods daemon set ne sont pas créés dans l'état **Pending**. Cette situation est source de confusion pour l'utilisateur.

- La préemption des pods est gérée par l'ordonnanceur par défaut. Lorsque la préemption est activée, le contrôleur de l'ensemble des démons prend des décisions d'ordonnancement sans tenir compte de la priorité et de la préemption des pods.

La fonctionnalité **ScheduleDaemonSetPods**, activée par défaut dans OpenShift Container Platform, vous permet de planifier des ensembles de démons en utilisant le planificateur par défaut au lieu du contrôleur d'ensembles de démons, en ajoutant le terme **NodeAffinity** aux pods d'ensembles de démons, au lieu du terme **spec.nodeName**. L'ordonnanceur par défaut est alors utilisé pour lier le pod à l'hôte cible. Si l'affinité de nœud du pod de l'ensemble de démons existe déjà, elle est remplacée. Le contrôleur de l'ensemble de démons n'effectue ces opérations que lors de la création ou de la modification des modules de l'ensemble de démons, et aucune modification n'est apportée à l'adresse **spec.template** de l'ensemble de démons.

```
nodeAffinity:
  requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchFields:
      - key: metadata.name
        operator: In
        values:
        - target-host-name
```

En outre, une tolérance **node.kubernetes.io/unschedulable:NoSchedule** est ajoutée automatiquement aux pods de l'ensemble des démons. L'ordonnanceur par défaut ignore les nœuds non ordonnançables lors de l'ordonnancement des pods du jeu de démons.

4.1.2. Création de jeux de démons

Lors de la création d'ensembles de démons, le champ **nodeSelector** est utilisé pour indiquer les nœuds sur lesquels l'ensemble de démons doit déployer des répliques.

Conditions préalables

- Avant de commencer à utiliser les ensembles de démons, désactivez le sélecteur de nœuds par défaut pour l'ensemble du projet dans votre espace de noms, en définissant l'annotation de l'espace de noms **openshift.io/node-selector** comme une chaîne vide :

```
$ oc patch namespace myproject -p \
    '{"metadata": {"annotations": {"openshift.io/node-selector": ""}}}'
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour désactiver le sélecteur de nœuds par défaut du projet pour un espace de noms :

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    openshift.io/node-selector: ''
```

- Si vous créez un nouveau projet, écrasez le sélecteur de nœuds par défaut :

```
$ oc adm new-project <name> --node-selector=""
```

Procédure

Pour créer un ensemble de démons :

1. Définir le fichier yaml du daemon :

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: hello-daemonset
```

```
spec:
 selector:
    matchLabels:
      name: hello-daemonset (1)
 template:
  metadata:
    labels:
      name: hello-daemonset (2)
  spec:
    nodeSelector: (3)
      role: worker
    containers:
    - image: openshift/hello-openshift
      imagePullPolicy: Always
      name: registry
      ports:
      - containerPort: 80
        protocol: TCP
      resources: {}
      terminationMessagePath: /dev/termination-log
    serviceAccount: default
    terminationGracePeriodSeconds: 10
```

(1) Le sélecteur d'étiquettes qui détermine quels pods appartiennent à l'ensemble de démons.

(2) Le sélecteur d'étiquette du modèle de pod. Il doit correspondre au sélecteur d'étiquette ci-dessus.

(3) Le sélecteur de nœud qui détermine sur quels nœuds les répliques de pods doivent être déployées. Un label correspondant doit être présent sur le nœud.

2. Créer l'objet daemon set :

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

3. Vérifier que les pods ont été créés et que chaque nœud dispose d'une réplique de pod :

   a. Trouver les pods du daemonset :

   ```
   $ oc get pods
   ```

   Exemple de sortie

   ```
   hello-daemonset-cx6md   1/1       Running   0          2m
   hello-daemonset-e3md9   1/1       Running   0          2m
   ```

   b. Affichez les modules pour vérifier qu'ils ont bien été placés sur le nœud :

   ```
   $ oc describe pod/hello-daemonset-cx6md|grep Node
   ```

   Exemple de sortie

   ```
   Node:        openshift-node01.hostname.com/10.14.20.134
   ```

```
$ oc describe pod/hello-daemonset-e3md9|grep Node
```

Exemple de sortie

```
Node:        openshift-node02.hostname.com/10.14.20.137
```

IMPORTANT

- Si vous mettez à jour un modèle de pod de daemon set, les répliques de pod existantes ne sont pas affectées.
- Si vous supprimez un ensemble de démons et que vous en créez un nouveau avec un modèle différent mais le même sélecteur d'étiquettes, il reconnaît les réplicas de pods existants comme ayant des étiquettes correspondantes et ne les met donc pas à jour ou ne crée pas de nouveaux réplicas en dépit d'une incohérence dans le modèle de pod.
- Si vous modifiez les étiquettes des nœuds, l'ensemble de démons ajoute des modules aux nœuds qui correspondent aux nouvelles étiquettes et supprime les modules des nœuds qui ne correspondent pas aux nouvelles étiquettes.

Pour mettre à jour un ensemble de démons, il faut forcer la création de nouvelles répliques de pods en supprimant les anciennes répliques ou les nœuds.

4.2. EXÉCUTER DES TÂCHES DANS DES PODS À L'AIDE DE JOBS

Un *job* exécute une tâche dans votre cluster OpenShift Container Platform.

Un job suit la progression globale d'une tâche et met à jour son statut avec des informations sur les pods actifs, réussis et échoués. La suppression d'un job nettoiera toutes les répliques de pods qu'il a créées. Les tâches font partie de l'API Kubernetes, qui peut être gérée avec des commandes **oc** comme d'autres types d'objets.

Exemple de cahier des charges

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1    (1)
  completions: 1    (2)
  activeDeadlineSeconds: 1800 (3)
  backoffLimit: 6   (4)
  template:         (5)
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure    (6)
```

1. Les répliques de pods qu'un travail doit exécuter en parallèle.
2. Pour qu'un travail soit considéré comme achevé, il faut que les pods soient terminés avec succès.
3. Durée maximale d'exécution du travail.
4. Nombre de tentatives pour un travail.
5. Le modèle du module créé par le contrôleur.
6. La politique de redémarrage du pod.

Voir la documentation de Kubernetes pour plus d'informations sur les travaux.

4.2.1. Comprendre les jobs et les cron jobs

Un travail permet de suivre la progression globale d'une tâche et de mettre à jour son état avec des informations sur les modules actifs, réussis et échoués. La suppression d'un job nettoie tous les pods qu'il a créés. Les tâches font partie de l'API Kubernetes, qui peut être gérée avec des commandes **oc** comme d'autres types d'objets.

Il existe deux types de ressources possibles qui permettent de créer des objets run-once dans OpenShift Container Platform :

Emploi

Un travail régulier est un objet à exécution unique qui crée une tâche et s'assure qu'elle se termine.

Il existe trois principaux types de tâches susceptibles d'être exécutées en tant que travaux :

- Travaux non parallèles :
  - Un travail qui ne démarre qu'un seul module, à moins que le module n'échoue.
  - Le travail est terminé dès que le pod se termine avec succès.
- Travaux parallèles avec un nombre fixe d'achèvements :
  - un travail qui démarre plusieurs pods.
  - Le travail représente la tâche globale et est terminé lorsqu'il y a un pod réussi pour chaque valeur comprise entre **1** et la valeur **completions**.
- Travaux parallèles avec une file d'attente :
  - Un travail avec plusieurs processus de travail en parallèle dans un pod donné.
  - OpenShift Container Platform coordonne les pods pour déterminer ce sur quoi chacun doit travailler ou utiliser un service de file d'attente externe.
  - Chaque pod est capable de déterminer indépendamment si tous les pods pairs sont complets et si le travail est terminé.
  - Lorsqu'un pod de la tâche se termine avec succès, aucun nouveau pod n'est créé.
  - Lorsqu'au moins un pod s'est terminé avec succès et que tous les pods sont terminés, le travail est terminé avec succès.

- Lorsqu'un module est sorti avec succès, aucun autre module ne doit être en train de travailler sur cette tâche ou d'écrire une sortie. Les modules doivent tous être en train de se terminer.

Pour plus d'informations sur l'utilisation des différents types de tâches, voir Job Patterns dans la documentation Kubernetes.

Travail Cron

Une tâche peut être programmée pour être exécutée plusieurs fois, à l'aide d'une tâche cron.

Un *cron job* s'appuie sur un travail normal en vous permettant de spécifier comment le travail doit être exécuté. Les tâches Cron font partie de l'API Kubernetes, qui peut être gérée avec des commandes **oc** comme d'autres types d'objets.

Les tâches Cron sont utiles pour créer des tâches périodiques et récurrentes, comme l'exécution de sauvegardes ou l'envoi d'e-mails. Les tâches cron peuvent également planifier des tâches individuelles à un moment précis, par exemple si vous souhaitez planifier une tâche pendant une période de faible activité. Un travail cron crée un objet **Job** basé sur le fuseau horaire configuré sur le nœud du plan de contrôle qui exécute le contrôleur de travail cron.

AVERTISSEMENT

Un travail cron crée un objet **Job** environ une fois par heure d'exécution de sa programmation, mais il peut arriver qu'il ne crée pas de travail ou que deux travaux soient créés. Par conséquent, les tâches doivent être idempotentes et vous devez configurer des limites d'historique.

4.2.1.1. Comprendre comment créer des emplois

Les deux types de ressources nécessitent une configuration de travail qui se compose des éléments clés suivants :

- Un modèle de pod, qui décrit le pod créé par OpenShift Container Platform.
- Le paramètre **parallelism**, qui indique combien de pods fonctionnant en parallèle à un moment donné doivent exécuter un travail.
  - Pour les travaux non parallèles, laissez le paramètre non défini. Si la valeur n'est pas définie, la valeur par défaut est **1**.
- Le paramètre **completions**, qui spécifie le nombre d'achèvements de pods réussis nécessaires pour terminer un travail.
  - Pour les travaux non parallèles, laissez le paramètre non défini. Si la valeur n'est pas définie, la valeur par défaut est **1**.
  - Pour les travaux parallèles avec un nombre d'achèvements fixe, indiquez une valeur.
  - Pour les travaux parallèles avec une file d'attente, laissez la valeur non définie. Lorsqu'il n'est pas défini, il prend par défaut la valeur **parallelism**.

4.2.1.2. Comprendre comment fixer une durée maximale pour les travaux

Lors de la définition d'un travail, vous pouvez définir sa durée maximale en définissant le champ **activeDeadlineSeconds**. Ce champ est spécifié en secondes et n'est pas défini par défaut. S'il n'est pas défini, aucune durée maximale n'est imposée.

La durée maximale est calculée à partir du moment où un premier module est programmé dans le système et définit la durée pendant laquelle un travail peut être actif. Elle permet de suivre la durée totale d'une exécution. Après avoir atteint le délai spécifié, le travail est terminé par OpenShift Container Platform.

4.2.1.3. Comprendre comment mettre en place une politique de reprise des travaux en cas de défaillance d'un pod

Un travail peut être considéré comme échoué après un certain nombre de tentatives en raison d'une erreur logique dans la configuration ou d'autres raisons similaires. Les modules défaillants associés au travail sont recréés par le contrôleur avec un délai exponentiel (**10s**, **20s**, **40s**...) plafonné à six minutes. La limite est réinitialisée si aucun nouveau pod échoué n'apparaît entre les vérifications du contrôleur.

Le paramètre **spec.backoffLimit** permet de définir le nombre de tentatives pour un travail.

4.2.1.4. Comprendre comment configurer une tâche cron pour supprimer les artefacts

Les tâches Cron peuvent laisser des ressources artéfactuelles telles que des tâches ou des pods. En tant qu'utilisateur, il est important de configurer les limites de l'historique afin que les anciens travaux et leurs pods soient correctement nettoyés. Il y a deux champs dans la spécification du job cron qui sont responsables de cela :

- **.spec.successfulJobsHistoryLimit**. Nombre de travaux terminés avec succès à conserver (3 par défaut).
- **.spec.failedJobsHistoryLimit**. Nombre d'échecs de travaux terminés à conserver (1 par défaut).

ASTUCE

- Supprimez les tâches cron dont vous n'avez plus besoin :

  ```
  oc delete cronjob/<cron_job_name>
  ```

  Cela leur évite de générer des artefacts inutiles.

- Vous pouvez suspendre les exécutions ultérieures en fixant la valeur de **spec.suspend** à true. Toutes les exécutions suivantes sont suspendues jusqu'à ce que vous réinitialisiez **false**.

4.2.1.5. Limites connues

La politique de redémarrage de la spécification des tâches ne s'applique qu'à *pods*, et non à *job controller*. Cependant, le contrôleur de tâches est codé en dur pour continuer à relancer les tâches jusqu'à ce qu'elles soient terminées.

Ainsi, **restartPolicy: Never** ou **--restart=Never** a le même comportement que **restartPolicy: OnFailure** ou **--restart=OnFailure**, c'est-à-dire que lorsqu'un travail échoue, il est redémarré automatiquement jusqu'à ce qu'il réussisse (ou qu'il soit écarté manuellement). La politique définit uniquement le sous-système qui effectue le redémarrage.

Avec la politique **Never**, c'est *job controller* qui effectue le redémarrage. À chaque tentative, le contrôleur de tâches incrémente le nombre d'échecs dans l'état de la tâche et crée de nouveaux modules. Cela signifie qu'à chaque tentative échouée, le nombre de pods augmente.

Avec la politique **OnFailure**, *kubelet* effectue le redémarrage. Chaque tentative n'incrémente pas le nombre d'échecs dans le statut du travail. De plus, kubelet réessayera les tâches qui ont échoué en démarrant des pods sur les mêmes nœuds.

4.2.2. Créer des emplois

Vous créez un job dans OpenShift Container Platform en créant un objet job.

Procédure

Créer un emploi :

1. Créez un fichier YAML similaire au suivant :

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  parallelism: 1    (1)
  completions: 1    (2)
  activeDeadlineSeconds: 1800 (3)
  backoffLimit: 6   (4)
  template:         (5)
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure    (6)
```

(1) Facultatif : Spécifiez le nombre de répliques de pods qu'un travail doit exécuter en parallèle ; la valeur par défaut est **1**.

- Pour les travaux non parallèles, laissez le paramètre non défini. Si la valeur n'est pas définie, la valeur par défaut est **1**.

(2) Facultatif : Indiquez le nombre de pods terminés avec succès pour qu'un travail soit considéré comme terminé.

- Pour les travaux non parallèles, laissez le paramètre non défini. Si la valeur n'est pas définie, la valeur par défaut est **1**.
- Pour les travaux parallèles avec un nombre d'achèvements fixe, indiquez le nombre d'achèvements.
- Pour les travaux parallèles avec une file d'attente, laissez la valeur non définie. Lorsqu'il n'est pas défini, il prend par défaut la valeur **parallelism**.

3. Facultatif : Indiquez la durée maximale d'exécution du travail.
4. Facultatif : Indiquez le nombre de tentatives pour un travail. La valeur par défaut de ce champ est de six.
5. Spécifiez le modèle du module créé par le contrôleur.
6. Spécifiez la politique de redémarrage du module :
   - **Never**. Ne pas relancer le travail.
   - **OnFailure**. Ne redémarrez le travail qu'en cas d'échec.
   - **Always**. Il faut toujours redémarrer le travail.
     Pour plus de détails sur la façon dont OpenShift Container Platform utilise la politique de redémarrage avec les conteneurs en panne, voir les États d'exemple dans la documentation Kubernetes.

2. Créer l'emploi :

```
oc create -f <nom-de-fichier>.yaml
```

NOTE

Vous pouvez également créer et lancer un travail à partir d'une seule commande en utilisant **oc create job**. La commande suivante crée et lance un travail similaire à celui spécifié dans l'exemple précédent :

```
$ oc create job pi --image=perl -- perl -Mbignum=bpi -wle 'print bpi(2000)'
```

4.2.3. Création de tâches cron

Vous créez un job cron dans OpenShift Container Platform en créant un objet job.

Procédure

Pour créer une tâche cron :

1. Créez un fichier YAML similaire au suivant :

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: pi
spec:
  schedule: "*/1 * * * *"                  1
  timeZone: Etc/UTC                        2
  concurrencyPolicy: "Replace"             3
  startingDeadlineSeconds: 200             4
  suspend: true                            5
  successfulJobsHistoryLimit: 3            6
  failedJobsHistoryLimit: 1                7
  jobTemplate:                             8
```

```
spec:
  template:
    metadata:
      labels:                    9
        parent: "cronjobpi"
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: OnFailure   10
```

1. Planification de la tâche spécifiée au format cron. Dans cet exemple, le travail sera exécuté toutes les minutes.

2. Un fuseau horaire facultatif pour la programmation. Voir Liste des fuseaux horaires de la base de données tz pour les options valides. S'il n'est pas spécifié, le gestionnaire de contrôleur Kubernetes interprète la planification par rapport à son fuseau horaire local. Ce paramètre est proposé en tant qu'aperçu technologique.

3. Une politique de concurrence optionnelle, spécifiant comment traiter les tâches concurrentes au sein d'une tâche cron. Seule l'une des politiques de concurrence suivantes peut être spécifiée. Si elle n'est pas spécifiée, elle autorise par défaut les exécutions simultanées.

   - **Allow** permet aux tâches cron de s'exécuter simultanément.
   - **Forbid** interdit les exécutions simultanées, en sautant l'exécution suivante si la précédente n'est pas encore terminée.
   - **Replace** annule le travail en cours et le remplace par un nouveau.

4. Délai facultatif (en secondes) pour lancer le travail s'il dépasse l'heure prévue pour une raison quelconque. Les exécutions manquées seront considérées comme des échecs. Si ce délai n'est pas spécifié, il n'y a pas de date limite.

5. Un drapeau optionnel permettant de suspendre une tâche cron. S'il vaut **true**, toutes les exécutions suivantes seront suspendues.

6. Nombre de travaux terminés avec succès à conserver (3 par défaut).

7. Nombre d'échecs de travaux terminés à conserver (valeur par défaut : 1).

8. Modèle d'emploi. Ce modèle est similaire à l'exemple d'emploi.

9. Définit une étiquette pour les travaux créés par ce travail cron.

10. La politique de redémarrage du module. Elle ne s'applique pas au contrôleur de tâches.

NOTE

Les champs **.spec.successfulJobsHistoryLimit** et **.spec.failedJobsHistoryLimit** sont facultatifs. Ils indiquent le nombre de tâches terminées et échouées à conserver. Par défaut, ils sont respectivement fixés à **3** et **1**. Définir une limite à **0** correspond à ne conserver aucun des travaux du type correspondant une fois qu'ils sont terminés.

2. Créez la tâche cron :

```
oc create -f <nom-de-fichier>.yaml
```

NOTE

Vous pouvez également créer et lancer une tâche cron à partir d'une seule commande en utilisant **oc create cronjob**. La commande suivante crée et lance une tâche cron similaire à celle spécifiée dans l'exemple précédent :

```
$ oc create cronjob pi --image=perl --schedule='*/1 * * * *' -- perl -Mbignum=bpi -wle
'print bpi(2000)'
```

Avec **oc create cronjob**, l'option **--schedule** accepte les programmes au format cron.

CHAPITRE 5. TRAVAILLER AVEC DES NŒUDS

5.1. AFFICHER ET LISTER LES NŒUDS DE VOTRE CLUSTER OPENSHIFT CONTAINER PLATFORM

Vous pouvez dresser la liste de tous les nœuds de votre cluster afin d'obtenir des informations telles que l'état, l'âge, l'utilisation de la mémoire et des détails sur les nœuds.

Lorsque vous effectuez des opérations de gestion de nœuds, l'interface CLI interagit avec des objets de nœuds qui sont des représentations d'hôtes de nœuds réels. Le maître utilise les informations des objets de nœuds pour valider les nœuds à l'aide de contrôles de santé.

5.1.1. A propos de la liste de tous les nœuds d'une grappe

Vous pouvez obtenir des informations détaillées sur les nœuds du cluster.

- La commande suivante permet de dresser la liste de tous les nœuds :

```
$ oc get nodes
```

L'exemple suivant est celui d'un cluster dont les nœuds sont sains :

```
$ oc get nodes
```

Exemple de sortie

```
NAME                 STATUS    ROLES     AGE       VERSION
master.example.com   Ready     master    7h        v1.25.0
node1.example.com    Ready     worker    7h        v1.25.0
node2.example.com    Ready     worker    7h        v1.25.0
```

L'exemple suivant est celui d'un cluster avec un nœud en mauvaise santé :

```
$ oc get nodes
```

Exemple de sortie

```
NAME                 STATUS                     ROLES     AGE       VERSION
master.example.com   Ready                      master    7h        v1.25.0
node1.example.com    NotReady,SchedulingDisabled worker    7h        v1.25.0
node2.example.com    Ready                      worker    7h        v1.25.0
```

Les conditions qui déclenchent l'état **NotReady** sont présentées plus loin dans cette section.

- L'option **-o wide** fournit des informations supplémentaires sur les nœuds.

```
$ oc get nodes -o wide
```

Exemple de sortie

```
NAME              STATUS   ROLES    AGE    VERSION   INTERNAL-IP    EXTERNAL-IP
```

```
OS-IMAGE                                KERNEL-VERSION           CONTAINER-
RUNTIME
master.example.com Ready   master  171m  v1.25.0  10.0.129.108  <none>       Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)  4.18.0-240.15.1.el8_3.x86_64
cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev
node1.example.com  Ready   worker  72m   v1.25.0  10.0.129.222  <none>       Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)  4.18.0-240.15.1.el8_3.x86_64
cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev
node2.example.com  Ready   worker  164m  v1.25.0  10.0.142.150  <none>       Red Hat
Enterprise Linux CoreOS 48.83.202103210901-0 (Ootpa)  4.18.0-240.15.1.el8_3.x86_64
cri-o://1.25.0-30.rhaos4.10.gitf2f339d.el8-dev
```

- La commande suivante répertorie les informations relatives à un seul nœud :

```
oc get node <node> $ oc get node <node>
```

Par exemple :

```
$ oc get node node1.example.com
```

Exemple de sortie

```
NAME                STATUS   ROLES    AGE     VERSION
node1.example.com   Ready    worker   7h      v1.25.0
```

- La commande suivante fournit des informations plus détaillées sur un nœud spécifique, y compris la raison de l'état actuel :

```
$ oc describe node <node>
```

Par exemple :

```
$ oc describe node node1.example.com
```

Exemple de sortie

```
Name:               node1.example.com 1
Roles:              worker 2
Labels:             beta.kubernetes.io/arch=amd64 3
                    beta.kubernetes.io/instance-type=m4.large
                    beta.kubernetes.io/os=linux
                    failure-domain.beta.kubernetes.io/region=us-east-2
                    failure-domain.beta.kubernetes.io/zone=us-east-2a
                    kubernetes.io/hostname=ip-10-0-140-16
                    node-role.kubernetes.io/worker=
Annotations:        cluster.k8s.io/machine: openshift-machine-api/ahardin-worker-us-east-2a-
q5dzc 4
                    machineconfiguration.openshift.io/currentConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/desiredConfig: worker-
309c228e8b3a92e2235edd544c62fea8
                    machineconfiguration.openshift.io/state: Done
                    volumes.kubernetes.io/controller-managed-attach-detach: true
```

```
CreationTimestamp:  Wed, 13 Feb 2019 11:05:57 -0500
Taints:             <none>  5
Unschedulable:      false
Conditions:  6
  Type             Status  LastHeartbeatTime                 LastTransitionTime                Reason
Message
  ----             ------  -----------------                 ------------------                ------
  -------
  OutOfDisk        False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientDisk     kubelet has sufficient disk space available
  MemoryPressure   False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57
-0500   KubeletHasSufficientMemory   kubelet has sufficient memory available
  DiskPressure     False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasNoDiskPressure     kubelet has no disk pressure
  PIDPressure      False   Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:05:57 -
0500   KubeletHasSufficientPID      kubelet has sufficient PID available
  Ready            True    Wed, 13 Feb 2019 15:09:42 -0500   Wed, 13 Feb 2019 11:07:09 -0500
KubeletReady                 kubelet is posting ready status
Addresses:  7
  InternalIP:   10.0.140.16
  InternalDNS:  ip-10-0-140-16.us-east-2.compute.internal
  Hostname:     ip-10-0-140-16.us-east-2.compute.internal
Capacity:  8
 attachable-volumes-aws-ebs:  39
 cpu:                         2
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      8172516Ki
 pods:                        250
Allocatable:
 attachable-volumes-aws-ebs:  39
 cpu:                         1500m
 hugepages-1Gi:               0
 hugepages-2Mi:               0
 memory:                      7558116Ki
 pods:                        250
System Info:  9
 Machine ID:                         63787c9534c24fde9a0cde35c13f1f66
 System UUID:                        EC22BF97-A006-4A58-6AF8-0A38DEEA122A
 Boot ID:                            f24ad37d-2594-46b4-8830-7f7555918325
 Kernel Version:                     3.10.0-957.5.1.el7.x86_64
 OS Image:                           Red Hat Enterprise Linux CoreOS 410.8.20190520.0 (Ootpa)
 Operating System:                   linux
 Architecture:                       amd64
 Container Runtime Version:          cri-o://1.25.0-0.6.dev.rhaos4.3.git9ad059b.el8-rc2
 Kubelet Version:                    v1.25.0
 Kube-Proxy Version:                 v1.25.0
PodCIDR:                             10.128.4.0/24
ProviderID:                          aws:///us-east-2a/i-04e87b31dc6b3e171
Non-terminated Pods:                 (12 in total)  10
  Namespace                          Name                                CPU Requests  CPU Limits
Memory Requests  Memory Limits
  ---------                          ----                                ------------  ----------  ---------------  -------
------
  openshift-cluster-node-tuning-operator  tuned-hdl5q                    0 (0%)        0 (0%)      0
(0%)          0 (0%)
```

```
  openshift-dns                       dns-default-l69zr                  0 (0%)       0 (0%)      0 (0%)
0 (0%)
  openshift-image-registry            node-ca-9hmcg                      0 (0%)       0 (0%)      0
(0%)          0 (0%)
  openshift-ingress                   router-default-76455c45c-c5ptv     0 (0%)       0 (0%)      0
(0%)          0 (0%)
  openshift-machine-config-operator   machine-config-daemon-cvqw9         20m (1%)     0
(0%)      50Mi (0%)       0 (0%)
  openshift-marketplace               community-operators-f67fh          0 (0%)       0 (0%)
0 (0%)          0 (0%)
  openshift-monitoring                alertmanager-main-0                50m (3%)     50m (3%)
210Mi (2%)      10Mi (0%)
  openshift-monitoring                node-exporter-l7q8d                10m (0%)     20m (1%)
20Mi (0%)       40Mi (0%)
  openshift-monitoring                prometheus-adapter-75d769c874-hvb85  0 (0%)       0
(0%)      0 (0%)          0 (0%)
  openshift-multus                    multus-kw8w5                       0 (0%)       0 (0%)      0 (0%)
0 (0%)
  openshift-sdn                       ovs-t4dsn                          100m (6%)    0 (0%)      300Mi
(4%)      0 (0%)
  openshift-sdn                       sdn-g79hg                          100m (6%)    0 (0%)      200Mi
(2%)      0 (0%)
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  Resource                    Requests     Limits
  --------                    --------     ------
  cpu                         380m (25%)   270m (18%)
  memory                      880Mi (11%)  250Mi (3%)
  attachable-volumes-aws-ebs  0            0
Events:                                                                                          (11)
  Type     Reason                   Age                From                   Message
  ----     ------                   ----               ----                   -------
  Normal   NodeHasSufficientPID     6d (x5 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientPID
  Normal   NodeAllocatableEnforced  6d                 kubelet, m01.example.com  Updated Node
Allocatable limit across pods
  Normal   NodeHasSufficientMemory  6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientMemory
  Normal   NodeHasNoDiskPressure    6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasNoDiskPressure
  Normal   NodeHasSufficientDisk    6d (x6 over 6d)    kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientDisk
  Normal   NodeHasSufficientPID     6d                 kubelet, m01.example.com  Node
m01.example.com status is now: NodeHasSufficientPID
  Normal   Starting                 6d                 kubelet, m01.example.com  Starting kubelet.
  ...
```

1. Le nom du nœud.
2. Le rôle du nœud, soit **master** ou **worker**.
3. Les étiquettes appliquées au nœud.
4. Les annotations appliquées au nœud.
5. Les taches appliquées au nœud.

6. Les conditions et l'état du nœud. La strophe **conditions** énumère les états **Ready**, **PIDPressure**, **PIDPressure**, **MemoryPressure**, **DiskPressure** et **OutOfDisk**. Ces
7. L'adresse IP et le nom d'hôte du nœud.
8. Les ressources pods et les ressources allouables.
9. Informations sur l'hôte du nœud.
10. Les pods sur le nœud.
11. Les événements signalés par le nœud.

Parmi les informations affichées pour les nœuds, les conditions suivantes apparaissent dans la sortie des commandes présentées dans cette section :

Tableau 5.1. Conditions du nœud

Condition	Description
Ready	Si **true**, le nœud est sain et prêt à accepter des pods. S**false**, le nœud n'est pas sain et n'accepte pas de modules. Si **unknown**, le contrôleur de nœud n'a pas reçu de battement de cœur du nœud depuis **node-monitor-grace-period** (la valeur par défaut est de 40 secondes).
DiskPressure	Si **true**, la capacité du disque est faible.
MemoryPressure	Si **true**, la mémoire du nœud est faible.
PIDPressure	Si **true**, il y a trop de processus sur le nœud.
OutOfDisk	Si **true**, le nœud ne dispose pas de suffisamment d'espace libre pour ajouter de nouveaux modules.
NetworkUnavailable	Si **true**, le réseau du nœud n'est pas correctement configuré.
NotReady	Si **true**, l'un des composants sous-jacents, comme l'exécution du conteneur ou le réseau, rencontre des problèmes ou n'est pas encore configuré.
SchedulingDisabled	Les pods ne peuvent pas être planifiés pour être placés sur le nœud.

5.1.2. Lister les pods sur un nœud de votre cluster

Vous pouvez répertorier tous les pods sur un nœud spécifique.

Procédure

- Pour dresser la liste de tous les pods ou d'une sélection de pods sur un ou plusieurs nœuds :

```
$ oc describe node <node1> <node2>
```

Par exemple :

```
$ oc describe node ip-10-0-128-218.ec2.internal
```

- Pour répertorier tous les pods ou certains pods sur les nœuds sélectionnés :

```
oc describe --selector=<node_selector>
```

```
$ oc describe node  --selector=kubernetes.io/os
```

Ou bien :

```
oc describe -l=<pod_selector> $ oc describe -l=<pod_selector>
```

```
$ oc describe node -l node-role.kubernetes.io/worker
```

- Pour lister tous les pods sur un nœud spécifique, y compris les pods terminés :

```
oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename> $ oc get pod --all-namespaces --field-selector=spec.nodeName=<nodename>
```

5.1.3. Affichage des statistiques d'utilisation de la mémoire et de l'unité centrale sur vos nœuds

Vous pouvez afficher les statistiques d'utilisation des nœuds, qui fournissent les environnements d'exécution des conteneurs. Ces statistiques d'utilisation comprennent la consommation de CPU, de mémoire et de stockage.

Conditions préalables

- Vous devez avoir l'autorisation **cluster-reader** pour voir les statistiques d'utilisation.
- Metrics doit être installé pour afficher les statistiques d'utilisation.

Procédure

- Pour consulter les statistiques d'utilisation :

```
$ oc adm top nodes
```

Exemple de sortie

```
NAME                                         CPU(cores)   CPU%      MEMORY(bytes)   MEMORY%
ip-10-0-12-143.ec2.compute.internal   1503m        100%      4533Mi          61%
ip-10-0-132-16.ec2.compute.internal   76m          5%        1391Mi          18%
ip-10-0-140-137.ec2.compute.internal  398m         26%       2473Mi          33%
ip-10-0-142-44.ec2.compute.internal   656m         43%       6119Mi          82%
ip-10-0-146-165.ec2.compute.internal  188m         12%       3367Mi          45%
ip-10-0-19-62.ec2.compute.internal    896m         59%       5754Mi          77%
ip-10-0-44-193.ec2.compute.internal   632m         42%       5349Mi          72%
```

- Pour afficher les statistiques d'utilisation des nœuds avec étiquettes :

```
$ oc adm top node --selector=''
```

Vous devez choisir le sélecteur (requête d'étiquette) sur lequel filtrer. Prend en charge **=**, **==**, et **!=**.

5.2. TRAVAILLER AVEC DES NŒUDS

En tant qu'administrateur, vous pouvez effectuer un certain nombre de tâches pour rendre vos grappes plus efficaces.

5.2.1. Comprendre comment évacuer les pods sur les nœuds

L'évacuation des pods permet de migrer tous les pods ou certains pods d'un ou de plusieurs nœuds donnés.

Vous ne pouvez évacuer que les modules soutenus par un contrôleur de réplication. Le contrôleur de réplication crée de nouveaux pods sur d'autres nœuds et supprime les pods existants sur le(s) nœud(s) spécifié(s).

Les pods nus, c'est-à-dire ceux qui ne sont pas soutenus par un contrôleur de réplication, ne sont pas affectés par défaut. Vous pouvez évacuer un sous-ensemble de modules en spécifiant un sélecteur de modules. Les sélecteurs de modules sont basés sur des étiquettes, de sorte que tous les modules portant l'étiquette spécifiée seront évacués.

Procédure

1. Marquer les nœuds inséparables avant d'effectuer l'évacuation des nacelles.

   a. Marquer le nœud comme non ordonnançable :

   ```
   oc adm cordon <node1> $ oc adm cordon <node1>
   ```

   Exemple de sortie

   ```
   node/<node1> cordonné
   ```

   b. Vérifiez que l'état du nœud est bien **Ready,SchedulingDisabled**:

   ```
   oc get node <node1> $ oc get node <node1>
   ```

   Exemple de sortie

   ```
   NAME        STATUS                     ROLES    AGE     VERSION
   <node1>     Ready,SchedulingDisabled   worker   1d      v1.25.0
   ```

2. Évacuez les nacelles en utilisant l'une des méthodes suivantes :

   - Évacuer tous les pods ou certains pods sur un ou plusieurs nœuds :

   ```
   $ oc adm drain <node1> <node2> [--pod-selector=<pod_selector>]
   ```

- Forcez la suppression des pods nus à l'aide de l'option **--force**. Lorsque l'option est définie sur **true**, la suppression se poursuit même si des pods ne sont pas gérés par un contrôleur de réplication, un ensemble de réplicas, un job, un ensemble de démons ou un ensemble avec état :

```
$ oc adm drain <node1> <node2> --force=true
```

- Définir une période de temps en secondes pour que chaque pod se termine gracieusement, utiliser **--grace-period**. Si elle est négative, la valeur par défaut spécifiée dans le pod sera utilisée :

```
$ oc adm drain <node1> <node2> --grace-period=-1
```

- Ignorer les pods gérés par des ensembles de démons utilisant l'indicateur **--ignore-daemonsets** avec la valeur **true**:

```
$ oc adm drain <node1> <node2> --ignore-daemonsets=true
```

- Fixe le temps d'attente avant d'abandonner en utilisant le drapeau **--timeout**. Une valeur de **0** définit une durée infinie :

```
$ oc adm drain <node1> <node2> --timeout=5s
```

- Supprimez les pods même s'il existe des pods utilisant des volumes **emptyDir** en définissant l'indicateur **--delete-emptydir-data** sur **true**. Les données locales sont supprimées lorsque le nœud est vidé :

```
$ oc adm drain <node1> <node2> --delete-emptydir-data=true
```

- Lister les objets qui seront migrés sans effectuer l'évacuation, en utilisant l'option **--dry-run** réglée sur **true**:

```
$ oc adm drain <node1> <node2>  --dry-run=true
```

  Au lieu de spécifier des noms de nœuds spécifiques (par exemple, **<node1> <node2>**), vous pouvez utiliser l'option **--selector=<node_selector>** pour évacuer les pods sur des nœuds sélectionnés.

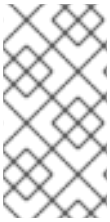
3. Marquer le nœud comme planifiable lorsqu'il est terminé.

```
oc adm uncordon <node1> $ oc adm uncordon <node1>
```

5.2.2. Comprendre comment mettre à jour les étiquettes sur les nœuds

Vous pouvez mettre à jour n'importe quelle étiquette d'un nœud.

Les étiquettes de nœuds ne sont pas conservées après la suppression d'un nœud, même si le nœud est sauvegardé par une machine.



NOTE

Toute modification apportée à un objet **MachineSet** n'est pas appliquée aux machines existantes appartenant à l'ensemble de machines de calcul. Par exemple, les étiquettes modifiées ou ajoutées à un objet **MachineSet** existant ne sont pas propagées aux machines et nœuds existants associés à l'ensemble de machines de calcul.

- La commande suivante permet d'ajouter ou de mettre à jour des étiquettes sur un nœud :

```
$ oc label node <node> <key_1>=<value_1> ... <clé_n>=<valeur_n>
```

Par exemple :

```
$ oc label nodes webconsole-7f7f6 unhealthy=true
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour appliquer l'étiquette :

```
kind: Node
apiVersion: v1
metadata:
  name: webconsole-7f7f6
  labels:
    unhealthy: 'true'
```

- La commande suivante met à jour tous les pods de l'espace de noms :

```
$ oc label pods --all <key_1>=<value_1>
```

Par exemple :

```
$ oc label pods --all status=unhealthy
```

5.2.3. Comprendre comment marquer les nœuds comme non planifiables ou planifiables

Par défaut, les nœuds sains avec un statut **Ready** sont marqués comme planifiables, ce qui signifie que vous pouvez placer de nouveaux pods sur le nœud. Le fait de marquer manuellement un nœud comme non planifiable empêche la planification de nouveaux modules sur ce nœud. Les pods existants sur le nœud ne sont pas affectés.

- La commande suivante marque un ou plusieurs nœuds comme non ordonnançables :

Exemple de sortie

```
$ oc adm cordon <node>
```

Par exemple :

```
$ oc adm cordon node1.example.com
```

Exemple de sortie

```
node/node1.example.com cordoned

NAME                LABELS                                        STATUS
node1.example.com   kubernetes.io/hostname=node1.example.com
Ready,SchedulingDisabled
```

- La commande suivante marque un ou plusieurs nœuds non ordonnançables comme ordonnançables :

```
oc adm uncordon <node1> $ oc adm uncordon <node1>
```

Au lieu de spécifier des noms de nœuds spécifiques (par exemple, **<node>**), vous pouvez utiliser l'option **--selector=<node_selector>** pour marquer les nœuds sélectionnés comme ordonnançables ou non ordonnançables.

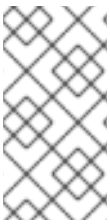
5.2.4. Suppression de nœuds

5.2.4.1. Suppression de nœuds d'une grappe

Lorsque vous supprimez un nœud à l'aide de la CLI, l'objet nœud est supprimé dans Kubernetes, mais les pods qui existent sur le nœud ne sont pas supprimés. Tous les pods nus qui ne sont pas soutenus par un contrôleur de réplication deviennent inaccessibles à OpenShift Container Platform. Les pods soutenus par des contrôleurs de réplication sont replanifiés sur d'autres nœuds disponibles. Vous devez supprimer les pods de manifeste locaux.

Procédure

Pour supprimer un nœud du cluster OpenShift Container Platform, modifiez l'objet **MachineSet** approprié :



NOTE

Si vous exécutez un cluster sur du métal nu, vous ne pouvez pas supprimer un nœud en modifiant les objets **MachineSet**. Les ensembles de machines de calcul ne sont disponibles que lorsqu'un cluster est intégré à un fournisseur de cloud. Au lieu de cela, vous devez déprogrammer et vidanger le nœud avant de le supprimer manuellement.

1. Affichez les ensembles de machines de calcul qui se trouvent dans la grappe :

```
$ oc get machinesets -n openshift-machine-api
```

Les ensembles de machines de calcul sont répertoriés sous la forme <clusterid>-worker-<aws-region-az>.

2. Mettre à l'échelle l'ensemble des machines de calcul :

```
$ oc scale --replicas=2 machineset <machineset> -n openshift-machine-api
```

Ou bien :

```
$ oc edit machineset <machineset> -n openshift-machine-api
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour mettre à l'échelle l'ensemble des machines de calcul :

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineSet
metadata:
  name: <machineset>
  namespace: openshift-machine-api
spec:
  replicas: 2
```

Ressources supplémentaires

- Pour plus d'informations sur la mise à l'échelle de votre cluster à l'aide d'un MachineSet, voir Mise à l'échelle manuelle d'un MachineSet .

5.2.4.2. Suppression de nœuds d'un cluster bare metal

Lorsque vous supprimez un nœud à l'aide de la CLI, l'objet nœud est supprimé dans Kubernetes, mais les pods qui existent sur le nœud ne sont pas supprimés. Tous les pods nus qui ne sont pas soutenus par un contrôleur de réplication deviennent inaccessibles à OpenShift Container Platform. Les pods soutenus par des contrôleurs de réplication sont replanifiés sur d'autres nœuds disponibles. Vous devez supprimer les pods de manifeste locaux.

Procédure

Supprimez un nœud d'un cluster OpenShift Container Platform fonctionnant sur du métal nu en effectuant les étapes suivantes :

1. Marquer le nœud comme non ordonnançable :

```
$ oc adm cordon <node_name>
```

2. Drainer tous les pods sur le nœud :

```
oc adm drain <node_name> --force=true
```

Cette étape peut échouer si le nœud est hors ligne ou ne répond pas. Même si le nœud ne répond pas, il est possible qu'il exécute toujours une charge de travail qui écrit dans le stockage partagé. Pour éviter toute corruption de données, mettez le matériel physique hors tension avant de poursuivre.

3. Supprimer le nœud de la grappe :

```
oc delete node <node_name> $ oc delete node <node_name>
```

Bien que l'objet nœud soit désormais supprimé du cluster, il peut toujours rejoindre le cluster après un redémarrage ou si le service kubelet est redémarré. Pour supprimer définitivement le nœud et toutes ses données, vous devez le déclasser.

4. Si vous avez mis le matériel physique hors tension, remettez-le sous tension pour que le nœud puisse rejoindre le cluster.

5.3. GESTION DES NŒUDS

OpenShift Container Platform utilise une ressource personnalisée KubeletConfig (CR) pour gérer la configuration des nœuds. En créant une instance d'un objet **KubeletConfig**, une configuration de machine gérée est créée pour remplacer les paramètres du nœud.

NOTE

Logging in to remote machines for the purpose of changing their configuration is not supported.

5.3.1. Modification des nœuds

Pour apporter des modifications à la configuration d'un cluster ou d'un pool de machines, vous devez créer une définition de ressource personnalisée (CRD) ou un objet **kubeletConfig**. OpenShift Container Platform utilise le Machine Config Controller pour surveiller les changements introduits par le CRD afin d'appliquer les changements au cluster.

NOTE

Comme les champs d'un objet **kubeletConfig** sont transmis directement au kubelet par Kubernetes en amont, la validation de ces champs est gérée directement par le kubelet lui-même. Veuillez vous référer à la documentation Kubernetes pertinente pour connaître les valeurs valides de ces champs. Des valeurs invalides dans l'objet **kubeletConfig** peuvent rendre les nœuds de cluster inutilisables.

Procédure

1. Obtenez l'étiquette associée au CRD statique, Machine Config Pool, pour le type de nœud que vous souhaitez configurer. Effectuez l'une des étapes suivantes :

   a. Vérifier les étiquettes actuelles du pool de configuration de la machine souhaitée.
   Par exemple :

   ```
   $ oc get machineconfigpool --show-labels
   ```

   Exemple de sortie

   ```
   NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
   LABELS
   master   rendered-master-e05b81f5ca4db1d249a1bf32f9ec24fd   True      False
   False    operator.machineconfiguration.openshift.io/required-for-upgrade=
   worker   rendered-worker-f50e78e1bc06d8e82327763145bfcf62   True      False
   False
   ```

   b. Ajoutez une étiquette personnalisée au pool de configuration de la machine souhaitée.
   Par exemple :

   ```
   $ oc label machineconfigpool worker custom-kubelet=enabled
   ```

2. Créez une ressource personnalisée (CR) **kubeletconfig** pour votre changement de configuration.
   Par exemple :

Exemple de configuration pour un CR custom-config

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: enabled (2)
  kubeletConfig: (3)
    podsPerCore: 10
    maxPods: 250
    systemReserved:
      cpu: 2000m
      memory: 1Gi
```

1. Attribuer un nom au CR.
2. Spécifiez l'étiquette pour appliquer le changement de configuration, il s'agit de l'étiquette que vous avez ajoutée au pool de configuration de la machine.
3. Indiquez la ou les nouvelles valeurs à modifier.

3. Créer l'objet CR.

```
$ oc create -f <nom-de-fichier>
```

Par exemple :

```
$ oc create -f master-kube-config.yaml
```

La plupart des options de configuration de Kubelet peuvent être définies par l'utilisateur. Les options suivantes ne peuvent pas être écrasées :

- CgroupDriver
- ClusterDNS
- Domaine de regroupement
- StaticPodPath

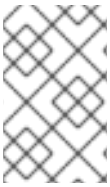
NOTE

Si un seul nœud contient plus de 50 images, la planification des pods peut être déséquilibrée entre les nœuds. En effet, la liste des images sur un nœud est réduite à 50 par défaut. Vous pouvez désactiver la limite d'images en modifiant l'objet **KubeletConfig** et en définissant la valeur de **nodeStatusMaxImages** à **-1**.

5.3.2. Configuration des nœuds du plan de contrôle comme planifiables

Vous pouvez configurer les nœuds du plan de contrôle pour qu'ils soient programmables, ce qui signifie que les nouveaux pods sont autorisés à être placés sur les nœuds maîtres. Par défaut, les nœuds du plan de contrôle ne sont pas programmables.

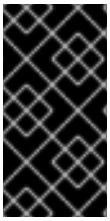
Vous pouvez faire en sorte que les maîtres soient programmables, mais vous devez conserver les nœuds de travail.



NOTE

Vous pouvez déployer OpenShift Container Platform sans nœuds de travail sur un cluster bare metal. Dans ce cas, les nœuds du plan de contrôle sont marqués comme planifiables par défaut.

Vous pouvez autoriser ou non les nœuds du plan de contrôle à être planifiables en configurant le champ **mastersSchedulable**.



IMPORTANT

Lorsque vous configurez les nœuds du plan de contrôle pour qu'ils soient planifiables au lieu d'être non planifiables par défaut, des abonnements supplémentaires sont nécessaires. En effet, les nœuds du plan de contrôle deviennent alors des nœuds de travail.

Procédure

1. Modifier la ressource **schedulers.config.openshift.io**.

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

2. Configurez le champ **mastersSchedulable**.

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: "2019-09-10T03:04:05Z"
  generation: 1
  name: cluster
  resourceVersion: "433"
  selfLink: /apis/config.openshift.io/v1/schedulers/cluster
  uid: a636d30a-d377-11e9-88d4-0a60097bee62
spec:
  mastersSchedulable: false (1)
status: {}
```

(1) La valeur **true** permet aux nœuds du plan de contrôle d'être programmables ou la valeur **false** interdit aux nœuds du plan de contrôle d'être programmables.

3. Enregistrez le fichier pour appliquer les modifications.

5.3.3. Définition des booléens SELinux

OpenShift Container Platform vous permet d'activer et de désactiver un booléen SELinux sur un nœud Red Hat Enterprise Linux CoreOS (RHCOS). La procédure suivante explique comment modifier les

booléens SELinux sur les nœuds à l'aide de l'opérateur de configuration de machine (MCO). Cette procédure utilise **container_manage_cgroup** comme exemple de booléen. Vous pouvez modifier cette valeur pour obtenir le booléen dont vous avez besoin.

Conditions préalables

- Vous avez installé le CLI OpenShift (oc).

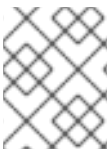
Procédure

1. Créez un nouveau fichier YAML avec un objet **MachineConfig**, comme dans l'exemple suivant :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Set SELinux booleans
          Before=kubelet.service

          [Service]
          Type=oneshot
          ExecStart=/sbin/setsebool container_manage_cgroup=on
          RemainAfterExit=true

          [Install]
          WantedBy=multi-user.target graphical.target
        enabled: true
        name: setsebool.service
```

2. Créez le nouvel objet **MachineConfig** en exécutant la commande suivante :

```
$ oc create -f 99-worker-setsebool.yaml
```



NOTE

L'application de toute modification à l'objet **MachineConfig** entraîne un redémarrage en douceur de tous les nœuds concernés après l'application de la modification.

5.3.4. Ajout d'arguments de noyau aux nœuds

Dans certains cas particuliers, vous pouvez ajouter des arguments de noyau à un ensemble de nœuds de votre cluster. Cela ne doit être fait qu'avec prudence et en comprenant bien les implications des arguments que vous définissez.

AVERTISSEMENT

Une mauvaise utilisation des arguments du noyau peut rendre vos systèmes non amorçables.

Voici quelques exemples d'arguments de noyau que vous pouvez définir :

- **enforcing=0**: Configure Security Enhanced Linux (SELinux) pour qu'il fonctionne en mode permissif. En mode permissif, le système agit comme si SELinux appliquait la politique de sécurité chargée, notamment en étiquetant les objets et en émettant des entrées de refus d'accès dans les journaux, mais il ne refuse en fait aucune opération. Bien qu'il ne soit pas pris en charge par les systèmes de production, le mode permissif peut s'avérer utile pour le débogage.

- **nosmt**: Désactive le multithreading symétrique (SMT) dans le noyau. Le multithreading permet d'avoir plusieurs threads logiques pour chaque unité centrale. Vous pouvez envisager d'utiliser **nosmt** dans les environnements multi-locataires afin de réduire les risques d'attaques croisées. En désactivant le SMT, vous choisissez essentiellement la sécurité au détriment des performances.

- **systemd.unified_cgroup_hierarchy**: Active le groupe de contrôle Linux version 2 (cgroup v2). cgroup v2 est la prochaine version du groupe de contrôle du noyau et offre de nombreuses améliorations.

IMPORTANT

OpenShift Container Platform cgroups version 2 support is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

Voir Kernel.org kernel parameters pour une liste et une description des arguments du noyau.

Dans la procédure suivante, vous créez un objet **MachineConfig** qui identifie :

- Ensemble de machines auxquelles vous souhaitez ajouter l'argument du noyau. Dans ce cas, il s'agit des machines ayant un rôle de travailleur.

- Arguments du noyau qui sont ajoutés à la fin des arguments du noyau existants.

- Une étiquette qui indique à quel endroit de la liste des configurations de machines la modification est appliquée.

Conditions préalables

- Disposer de privilèges administratifs sur un cluster OpenShift Container Platform opérationnel.

Procédure

1. Listez les objets **MachineConfig** existants pour votre cluster OpenShift Container Platform afin de déterminer comment étiqueter votre machine config :

```
$ oc get MachineConfig
```

Exemple de sortie

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                 3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                                 3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

2. Créer un fichier objet **MachineConfig** qui identifie l'argument du noyau (par exemple, **05-worker-kernelarg-selinuxpermissive.yaml**)

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 05-worker-kernelarg-selinuxpermissive 2
spec:
  kernelArguments:
    - enforcing=0 3
```

1. Applique le nouvel argument du noyau uniquement aux nœuds de travail.
2. Nommé pour identifier sa place dans les configurations de la machine (05) et ce qu'il fait (ajoute un argument au noyau pour configurer le mode permissif de SELinux).

3 Identifie l'argument exact du noyau comme **enforcing=0**.

3. Créer la nouvelle configuration de la machine :

```
$ oc create -f 05-worker-kernelarg-selinuxpermissive.yaml
```

4. Vérifiez les configurations de la machine pour voir si le nouveau a été ajouté :

```
$ oc get MachineConfig
```

Exemple de sortie

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                      52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
05-worker-kernelarg-selinuxpermissive                                             3.2.0             105s
99-master-generated-registries                         52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                                     3.2.0             40m
99-worker-generated-registries                         52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                                     3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
```

5. Vérifier les nœuds :

```
$ oc get nodes
```

Exemple de sortie

```
NAME                         STATUS                     ROLES    AGE   VERSION
ip-10-0-136-161.ec2.internal   Ready                      worker   28m   v1.25.0
ip-10-0-136-243.ec2.internal   Ready                      master   34m   v1.25.0
ip-10-0-141-105.ec2.internal   Ready,SchedulingDisabled   worker   28m   v1.25.0
ip-10-0-142-249.ec2.internal   Ready                      master   34m   v1.25.0
ip-10-0-153-11.ec2.internal    Ready                      worker   28m   v1.25.0
ip-10-0-153-150.ec2.internal   Ready                      master   34m   v1.25.0
```

Vous pouvez voir que la planification sur chaque nœud de travailleur est désactivée pendant que la modification est appliquée.

6. Vérifiez que l'argument du noyau a fonctionné en vous rendant sur l'un des nœuds de travail et en listant les arguments de la ligne de commande du noyau (dans **/proc/cmdline** sur l'hôte) :

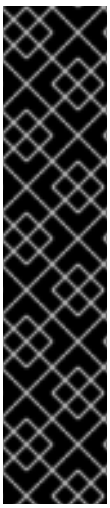
```
$ oc debug node/ip-10-0-141-105.ec2.internal
```

Exemple de sortie

```
Starting pod/ip-10-0-141-105ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# cat /host/proc/cmdline
BOOT_IMAGE=/ostree/rhcos-... console=tty0 console=ttyS0,115200n8
rootflags=defaults,prjquota rw root=UUID=fd0... ostree=/ostree/boot.0/rhcos/16...
coreos.oem.id=qemu coreos.oem.id=ec2 ignition.platform.id=ec2 enforcing=0

sh-4.2# exit
```

Vous devriez voir l'argument **enforcing=0** ajouté aux autres arguments du noyau.

5.3.5. Activation de l'utilisation de la mémoire d'échange sur les nœuds



IMPORTANT

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

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

Vous pouvez activer l'utilisation de la mémoire d'échange pour les charges de travail d'OpenShift Container Platform sur une base par nœud.

AVERTISSEMENT

L'activation de la mémoire d'échange peut avoir un impact négatif sur les performances de la charge de travail et sur la gestion des ressources manquantes. N'activez pas la mémoire d'échange sur les nœuds du plan de contrôle.

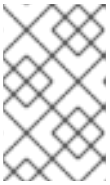
Pour activer la mémoire tampon, créez une ressource personnalisée (CR) **kubeletconfig** afin de définir le paramètre **swapbehavior**. Vous pouvez définir une mémoire d'échange limitée ou illimitée :

- Limité : Utilisez la valeur **LimitedSwap** pour limiter la quantité de mémoire d'échange que les charges de travail peuvent utiliser. Toutes les charges de travail sur le nœud qui ne sont pas gérées par OpenShift Container Platform peuvent toujours utiliser la mémoire d'échange. Le comportement de **LimitedSwap** dépend de l'exécution du nœud avec les groupes de contrôle Linux version 1 (cgroups v1) ou version 2 (cgroup v2):
  - cgroup v1 : Les charges de travail d'OpenShift Container Platform peuvent utiliser n'importe quelle combinaison de mémoire et de swap, jusqu'à la limite de mémoire du pod, si elle est définie.
  - cgroup v2 : Les charges de travail d'OpenShift Container Platform ne peuvent pas utiliser de mémoire d'échange.
- Illimité : Utilisez la valeur **UnlimitedSwap** pour permettre aux charges de travail d'utiliser autant de mémoire d'échange qu'elles le souhaitent, jusqu'à la limite du système.

Comme le kubelet ne démarrera pas en présence de mémoire d'échange sans cette configuration, vous devez activer la mémoire d'échange dans OpenShift Container Platform avant d'activer la mémoire d'échange sur les nœuds. S'il n'y a pas de mémoire d'échange sur un nœud, l'activation de la mémoire d'échange dans OpenShift Container Platform n'a aucun effet.

Conditions préalables

- Vous disposez d'un cluster OpenShift Container Platform en cours d'exécution qui utilise la version 4.10 ou une version ultérieure.
- You are logged in to the cluster as a user with administrative privileges.
- Vous avez activé le jeu de fonctionnalités **TechPreviewNoUpgrade** sur le cluster (voir *Nodes → Working with clusters → Enabling features using feature gates*).



NOTE

L'activation de l'ensemble de fonctionnalités **TechPreviewNoUpgrade** ne peut être annulée et empêche les mises à jour mineures de la version. Ces jeux de fonctionnalités ne sont pas recommandés sur les clusters de production.

- Si le cgroup v2 est activé sur un nœud, vous devez activer la comptabilité de swap sur le nœud, en définissant l'argument du noyau **swapaccount=1**.

Procédure

1. Appliquez une étiquette personnalisée au pool de configuration de la machine dans lequel vous souhaitez autoriser la mémoire d'échange.

```
$ oc label machineconfigpool worker kubelet-swap=enabled
```

2. Créer une ressource personnalisée (CR) pour activer et configurer les paramètres d'échange.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
```

```
  name: swap-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      kubelet-swap: enabled
  kubeletConfig:
    failSwapOn: false (1)
    memorySwap:
      swapBehavior: LimitedSwap (2)
```

1. La valeur **false** permet d'activer l'utilisation de la mémoire d'échange sur les nœuds associés. La valeur **true** désactive l'utilisation de la mémoire d'échange.
2. Spécifier le comportement de la mémoire d'échange. S'il n'est pas spécifié, la valeur par défaut est **LimitedSwap**.

3. Activer la mémoire tampon sur les machines.

5.3.6. Migration des nœuds du plan de contrôle d'un hôte RHOSP à un autre

Vous pouvez exécuter un script qui déplace un nœud de plan de contrôle d'un nœud Red Hat OpenStack Platform (RHOSP) à un autre.

Conditions préalables

- La variable d'environnement **OS_CLOUD** fait référence à une entrée **clouds** qui contient des informations d'identification administratives dans un fichier **clouds.yaml**.
- La variable d'environnement **KUBECONFIG** fait référence à une configuration qui contient les identifiants administratifs de OpenShift Container Platform.

Procédure

- À partir d'une ligne de commande, exécutez le script suivant :

```
#!/usr/bin/env bash

set -Eeuo pipefail

if [ $# -lt 1 ]; then
 echo "Usage: '$0 node_name'"
 exit 64
fi

# Check for admin OpenStack credentials
openstack server list --all-projects >/dev/null || { >&2 echo "The script needs OpenStack admin credentials. Exiting"; exit 77; }

# Check for admin OpenShift credentials
oc adm top node >/dev/null || { >&2 echo "The script needs OpenShift admin credentials. Exiting"; exit 77; }

set -x
```

```
declare -r node_name="$1"
declare server_id
server_id="$(openstack server list --all-projects -f value -c ID -c Name | grep "$node_name" | cut -d' '
-f1)"
readonly server_id

# Drain the node
oc adm cordon "$node_name"
oc adm drain "$node_name" --delete-emptydir-data --ignore-daemonsets --force

# Power off the server
oc debug "node/${node_name}" -- chroot /host shutdown -h 1

# Verify the server is shut off
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5; done

# Migrate the node
openstack server migrate --wait "$server_id"

# Resize the VM
openstack server resize confirm "$server_id"

# Wait for the resize confirm to finish
until openstack server show "$server_id" -f value -c status | grep -q 'SHUTOFF'; do sleep 5; done

# Restart the VM
openstack server start "$server_id"

# Wait for the node to show up as Ready:
until oc get node "$node_name" | grep -q "^${node_name}[[:space:]]\+Ready"; do sleep 5; done

# Uncordon the node
oc adm uncordon "$node_name"

# Wait for cluster operators to stabilize
until oc get co -o go-template='statuses: {{ range .items }}{{ range .status.conditions }}{{ if eq .type
"Degraded" }}{{ if ne .status "False" }}DEGRADED{{ end }}{{ else if eq .type "Progressing"}}{{ if ne
.status "False" }}PROGRESSING{{ end }}{{ else if eq .type "Available"}}{{ if ne .status "True"
}}NOTAVAILABLE{{ end }}{{ end }}{{ end }}{{ end }}' | grep -qv '\
(DEGRADED\|PROGRESSING\|NOTAVAILABLE\)'; do sleep 5; done
```

Si le script aboutit, la machine du plan de contrôle est migrée vers un nouveau nœud RHOSP.

5.4. GÉRER LE NOMBRE MAXIMUM DE PODS PAR NŒUD

Dans OpenShift Container Platform, vous pouvez configurer le nombre de pods qui peuvent s'exécuter sur un nœud en fonction du nombre de cœurs de processeur sur le nœud, d'une limite stricte ou des deux. Si vous utilisez les deux options, la moins élevée des deux limite le nombre de pods sur un nœud.

Le dépassement de ces valeurs peut entraîner

- Augmentation de l'utilisation du processeur par OpenShift Container Platform.
- Lenteur de la programmation des pods.

- Scénarios potentiels de dépassement de mémoire, en fonction de la quantité de mémoire dans le nœud.
- Epuisement du pool d'adresses IP.
- Surcharge des ressources, entraînant de mauvaises performances pour les applications utilisateur.

NOTE

Un pod qui contient un seul conteneur utilise en réalité deux conteneurs. Le deuxième conteneur met en place le réseau avant le démarrage du conteneur proprement dit. Par conséquent, un nœud exécutant 10 pods a en réalité 20 conteneurs en cours d'exécution.

Le paramètre **podsPerCore** limite le nombre de modules que le nœud peut exécuter en fonction du nombre de cœurs de processeur du nœud. Par exemple, si **podsPerCore** est défini sur **10** sur un nœud avec 4 cœurs de processeur, le nombre maximum de modules autorisés sur le nœud est de 40.

Le paramètre **maxPods** limite le nombre de pods que le nœud peut exécuter à une valeur fixe, quelles que soient les propriétés du nœud.

5.4.1. Configurer le nombre maximum de pods par nœud

Deux paramètres contrôlent le nombre maximal de modules qui peuvent être planifiés sur un nœud : **podsPerCore** et **maxPods**. Si vous utilisez les deux options, la moins élevée des deux limite le nombre de modules sur un nœud.

Par exemple, si **podsPerCore** est défini sur **10** sur un nœud avec 4 cœurs de processeur, le nombre maximum de pods autorisé sur le nœud sera de 40.

Conditions préalables

1. Obtenez l'étiquette associée au CRD statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante :

```
oc edit machineconfigpool <name> $ oc edit machineconfigpool <name>
```

Par exemple :

```
$ oc edit machineconfigpool worker
```

Exemple de sortie

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" (1)
  name: worker
```

(1) L'étiquette apparaît sous Étiquettes.

ASTUCE

Si l'étiquette n'est pas présente, ajoutez une paire clé/valeur comme par exemple :

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration.

Exemple de configuration pour un CR max-pods

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-max-pods (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    podsPerCore: 10 (3)
    maxPods: 250 (4)
```

(1) Attribuer un nom au CR.

(2) Spécifiez l'étiquette du pool de configuration de la machine.

(3) Indiquez le nombre de modules que le nœud peut exécuter en fonction du nombre de cœurs de processeur du nœud.

(4) Spécifie le nombre de pods que le nœud peut exécuter à une valeur fixe, indépendamment des propriétés du nœud.

NOTE

Le fait de régler **podsPerCore** sur **0** désactive cette limite.

Dans l'exemple ci-dessus, la valeur par défaut pour **podsPerCore** est **10** et la valeur par défaut pour **maxPods** est **250**. Cela signifie qu'à moins que le nœud ne dispose de 25 cœurs ou plus, par défaut, **podsPerCore** sera le facteur limitant.

2. Exécutez la commande suivante pour créer le CR :

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

Vérification

1. Lister les CRDs **MachineConfigPool** pour voir si le changement est appliqué. La colonne **UPDATING** indique **True** si la modification est prise en compte par le contrôleur de configuration de la machine :

```
$ oc get machineconfigpools
```

Exemple de sortie

```
NAME     CONFIG                     UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     False      False
worker   worker-8cecd1236b33ee3f8a5e   False     True       False
```

Une fois la modification effectuée, la colonne **UPDATED** indique **True**.

```
$ oc get machineconfigpools
```

Exemple de sortie

```
NAME     CONFIG                     UPDATED   UPDATING   DEGRADED
master   master-9cc2c72f205e103bb534   False     True       False
worker   worker-8cecd1236b33ee3f8a5e   True      False      False
```

5.5. UTILISATION DE L'OPÉRATEUR NODE TUNING

Découvrez l'opérateur d'optimisation des nœuds et la manière dont vous pouvez l'utiliser pour gérer l'optimisation au niveau des nœuds en orchestrant le démon d'optimisation.

L'opérateur d'optimisation des nœuds vous aide à gérer l'optimisation au niveau des nœuds en orchestrant le démon TuneD et à obtenir des performances à faible latence en utilisant le contrôleur de profil de performance. La majorité des applications à hautes performances nécessitent un certain niveau de réglage du noyau. Le Node Tuning Operator offre une interface de gestion unifiée aux utilisateurs de sysctls au niveau des nœuds et plus de flexibilité pour ajouter des réglages personnalisés en fonction des besoins de l'utilisateur.

L'opérateur gère le démon TuneD conteneurisé pour OpenShift Container Platform en tant qu'ensemble de démons Kubernetes. Il s'assure que la spécification de réglage personnalisé est transmise à tous les démons TuneD conteneurisés s'exécutant dans le cluster dans le format que les démons comprennent. Les démons s'exécutent sur tous les nœuds du cluster, un par nœud.

Les paramètres de niveau nœud appliqués par le démon TuneD conteneurisé sont annulés lors d'un événement qui déclenche un changement de profil ou lorsque le démon TuneD conteneurisé se termine de manière élégante en recevant et en gérant un signal de fin.

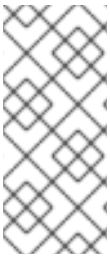
L'opérateur de réglage des nœuds utilise le contrôleur de profil de performance pour mettre en œuvre un réglage automatique afin d'obtenir des performances de faible latence pour les applications OpenShift Container Platform. L'administrateur du cluster configure un profil de performance pour définir des paramètres au niveau du nœud, tels que les suivants :

- Mise à jour du noyau vers kernel-rt.
- Choix des unités centrales de traitement pour l'entretien ménager.
- Choix des unités centrales pour l'exécution des charges de travail.

NOTE

Actuellement, la désactivation de l'équilibrage de la charge du CPU n'est pas prise en charge par cgroup v2. Par conséquent, il se peut que vous n'obteniez pas le comportement souhaité des profils de performance si vous avez activé cgroup v2. L'activation de cgroup v2 n'est pas recommandée si vous utilisez des profils de performance.

L'opérateur Node Tuning fait partie de l'installation standard d'OpenShift Container Platform à partir de la version 4.1.



NOTE

Dans les versions antérieures d'OpenShift Container Platform, l'opérateur Performance Addon était utilisé pour mettre en œuvre un réglage automatique afin d'obtenir des performances de faible latence pour les applications OpenShift. Dans OpenShift Container Platform 4.11 et les versions ultérieures, cette fonctionnalité fait partie de l'opérateur Node Tuning.

5.5.1. Accéder à un exemple de spécification de l'opérateur Node Tuning

Cette procédure permet d'accéder à un exemple de spécification de l'opérateur de réglage des nœuds.

Procédure

- Exécutez la commande suivante pour accéder à un exemple de spécification de l'opérateur Node Tuning :

```
$ oc get Tuned/default -o yaml -n openshift-cluster-node-tuning-operator
```

Le CR par défaut est destiné à fournir un réglage standard au niveau du nœud pour la plateforme OpenShift Container Platform et il ne peut être modifié que pour définir l'état de gestion de l'opérateur. Toute autre modification personnalisée de la CR par défaut sera écrasée par l'opérateur. Pour un réglage personnalisé, créez vos propres CR réglés. Les CR nouvellement créés seront combinés avec le CR par défaut et les réglages personnalisés appliqués aux nœuds d'OpenShift Container Platform en fonction des étiquettes de nœuds ou de pods et des priorités de profil.

AVERTISSEMENT

Bien que dans certaines situations, la prise en charge des étiquettes de pods puisse être un moyen pratique de fournir automatiquement les réglages nécessaires, cette pratique est déconseillée et fortement déconseillée, en particulier dans les clusters à grande échelle. Le CR Tuned par défaut est livré sans correspondance d'étiquettes de pods. Si un profil personnalisé est créé avec la correspondance des étiquettes de pods, alors la fonctionnalité sera activée à ce moment-là. La fonctionnalité d'étiquetage de pods sera obsolète dans les versions futures de l'opérateur de tuning de nœuds.

5.5.2. Spécification de réglage personnalisé

La ressource personnalisée (CR) de l'opérateur comporte deux sections principales. La première section, **profile:**, est une liste de profils TuneD et de leurs noms. La seconde, **recommend:**, définit la logique de sélection des profils.

Plusieurs spécifications de réglage personnalisées peuvent coexister en tant que CR multiples dans l'espace de noms de l'opérateur. L'existence de nouveaux CR ou la suppression d'anciens CR est détectée par l'Opérateur. Toutes les spécifications de réglage personnalisées existantes sont fusionnées et les objets appropriés pour les démons TuneD conteneurisés sont mis à jour.

Management state

L'état de gestion de l'opérateur est défini en ajustant le CR accordé par défaut. Par défaut, l'opérateur est en état de gestion et le champ **spec.managementState** n'est pas présent dans le CR accordé par défaut. Les valeurs valides pour l'état de gestion de l'opérateur sont les suivantes :

- Géré : l'opérateur met à jour ses opérandes au fur et à mesure que les ressources de configuration sont mises à jour
- Non géré : l'opérateur ignore les changements apportés aux ressources de configuration
- Retiré : l'opérateur retire ses opérandes et les ressources qu'il a fournies

Profile data

La section **profile:** dresse la liste des profils TuneD et de leurs noms.

```
profile:
- name: tuned_profile_1
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_1 profile

    [sysctl]
    net.ipv4.ip_forward=1
    # ... other sysctl's or other TuneD daemon plugins supported by the containerized TuneD

# ...

- name: tuned_profile_n
  data: |
    # TuneD profile specification
    [main]
    summary=Description of tuned_profile_n profile

    # tuned_profile_n profile settings
```

Recommended profiles

La logique de sélection de **profile:** est définie par la section **recommend:** du CR. La section **recommend:** est une liste d'éléments permettant de recommander les profils sur la base d'un critère de sélection.

```
recommend:
<recommend-item-1>
# ...
```

```
<recommend-item-n>
```

Les différents éléments de la liste :

```
- machineConfigLabels: 1
    <mcLabels> 2
  match: 3
    <match> 4
  priority: <priority> 5
  profile: <tuned_profile_name> 6
  operand: 7
    debug: <bool> 8
    tunedConfig:
      reapply_sysctl: <bool> 9
```

1. En option.
2. Un dictionnaire d'étiquettes clé/valeur **MachineConfig**. Les clés doivent être uniques.
3. En cas d'omission, la correspondance des profils est présumée, sauf si un profil ayant une priorité plus élevée correspond en premier ou si **machineConfigLabels** est défini.
4. Une liste facultative.
5. Ordre de priorité des profils. Les chiffres les plus bas signifient une priorité plus élevée (**0** est la priorité la plus élevée).
6. Un profil TuneD à appliquer sur un match. Par exemple **tuned_profile_1**.
7. Configuration facultative de l'opérande.
8. Active ou désactive le débogage du démon TuneD. Les options sont **true** pour on ou **false** pour off. La valeur par défaut est **false**.
9. Active ou désactive la fonctionnalité **reapply_sysctl** pour le démon TuneD. Les options sont **true** pour on et **false** pour off.

<match> est une liste optionnelle définie récursivement comme suit :

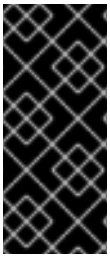
```
- label: <label_name> 1
  value: <label_value> 2
  type: <label_type> 3
    <match> 4
```

1. Nom de l'étiquette du nœud ou du pod.
2. Valeur facultative de l'étiquette du nœud ou du pod. Si elle est omise, la présence de **<label_name>** suffit à établir une correspondance.
3. Type d'objet facultatif (**node** ou **pod**). En cas d'omission, **node** est considéré comme tel.
4. Une liste facultative **<match>**.

Si **<match>** n'est pas omis, toutes les sections imbriquées **<match>** doivent également être évaluées à **true**. Sinon, **false** est supposé et le profil avec la section **<match>** correspondante ne sera pas appliqué ou recommandé. Par conséquent, l'imbrication (sections **<match>** enfant) fonctionne comme un opérateur logique ET. Inversement, si un élément de la liste **<match>** correspond, toute la liste **<match>** est évaluée à **true**. La liste agit donc comme un opérateur logique OU.

Si **machineConfigLabels** est défini, la correspondance basée sur le pool de configuration de la machine est activée pour l'élément de liste **recommend:** donné. **<mcLabels>** spécifie les étiquettes d'une configuration de la machine. La configuration de la machine est créée automatiquement pour appliquer les paramètres de l'hôte, tels que les paramètres de démarrage du noyau, pour le profil **<tuned_profile_name>**. Il s'agit de trouver tous les pools de configuration de machine dont le sélecteur de configuration de machine correspond à **<mcLabels>** et de définir le profil **<tuned_profile_name>** sur tous les nœuds auxquels sont attribués les pools de configuration de machine trouvés. Pour cibler les nœuds qui ont à la fois un rôle de maître et de travailleur, vous devez utiliser le rôle de maître.

Les éléments de la liste **match** et **machineConfigLabels** sont reliés par l'opérateur logique OR. L'élément **match** est évalué en premier, en court-circuit. Par conséquent, s'il est évalué à **true**, l'élément **machineConfigLabels** n'est pas pris en compte.



IMPORTANT

Lors de l'utilisation de la correspondance basée sur le pool de configuration machine, il est conseillé de regrouper les nœuds ayant la même configuration matérielle dans le même pool de configuration machine. Si cette pratique n'est pas respectée, les opérandes TuneD peuvent calculer des paramètres de noyau contradictoires pour deux nœuds ou plus partageant le même pool de configuration de machine.

Exemple : correspondance basée sur l'étiquette d'un nœud ou d'un pod

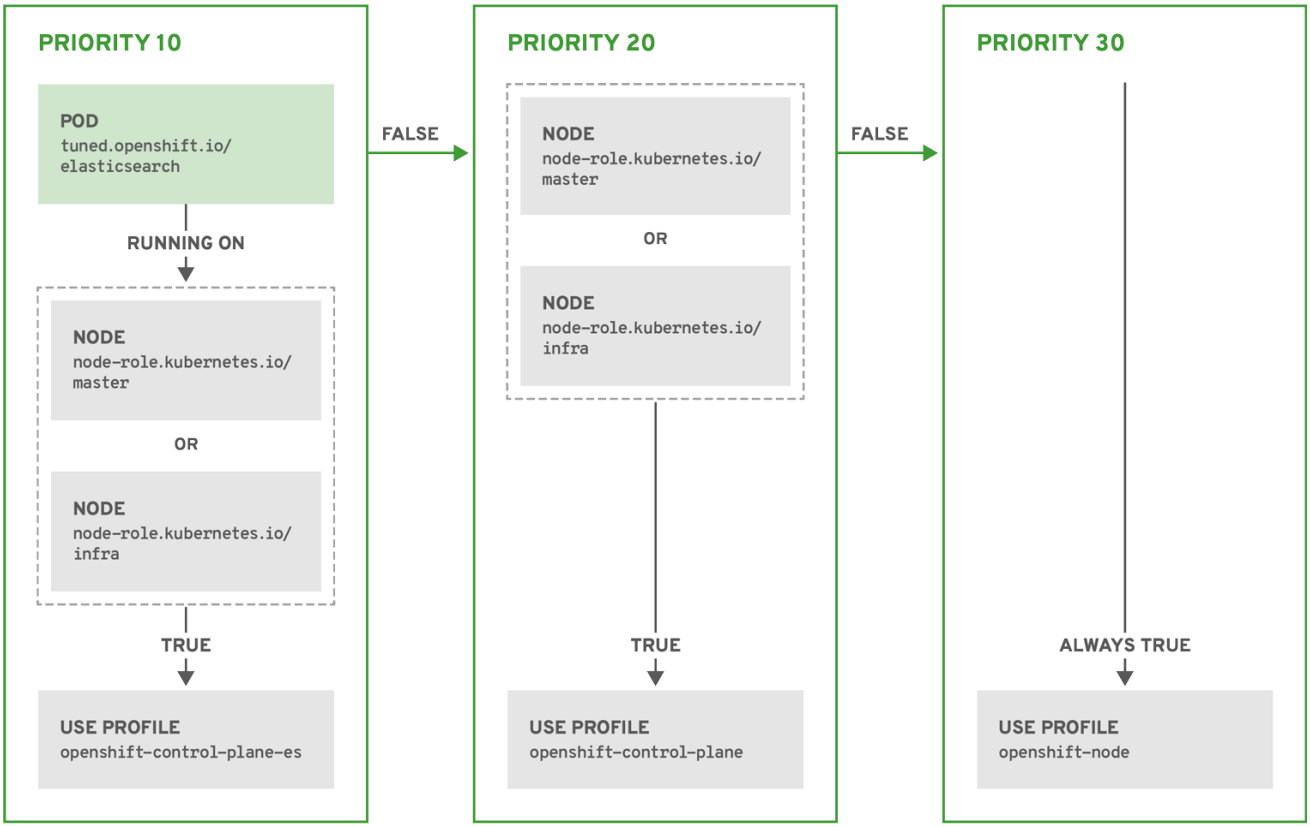
```
- match:
  - label: tuned.openshift.io/elasticsearch
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
    type: pod
  priority: 10
  profile: openshift-control-plane-es
- match:
  - label: node-role.kubernetes.io/master
  - label: node-role.kubernetes.io/infra
  priority: 20
  profile: openshift-control-plane
- priority: 30
  profile: openshift-node
```

Le CR ci-dessus est traduit pour le démon TuneD conteneurisé dans son fichier **recommend.conf** en fonction des priorités du profil. Le profil ayant la priorité la plus élevée (**10**) est **openshift-control-plane-es** et, par conséquent, il est considéré en premier. Le démon TuneD conteneurisé fonctionnant sur un nœud donné vérifie s'il existe un pod fonctionnant sur le même nœud avec l'étiquette **tuned.openshift.io/elasticsearch** définie. Si ce n'est pas le cas, toute la section **<match>** est évaluée comme **false**. S'il existe un pod avec le label, pour que la section **<match>** soit évaluée comme **true**, le label du nœud doit également être **node-role.kubernetes.io/master** ou **node-role.kubernetes.io/infra**.

Si les étiquettes du profil ayant la priorité **10** correspondent, le profil **openshift-control-plane-es** est appliqué et aucun autre profil n'est pris en considération. Si la combinaison d'étiquettes nœud/pod ne

correspond pas, le deuxième profil le plus prioritaire (**openshift-control-plane**) est pris en compte. Ce profil est appliqué si le pod TuneD conteneurisé fonctionne sur un nœud avec les étiquettes **node-role.kubernetes.io/master** ou **node-role.kubernetes.io/infra**.

Enfin, le profil **openshift-node** a la priorité la plus basse de **30**. Il ne contient pas la section **<match>** et, par conséquent, correspondra toujours. Il sert de profil fourre-tout pour définir le profil **openshift-node** si aucun autre profil ayant une priorité plus élevée ne correspond à un nœud donné.



Exemple : correspondance basée sur le pool de configuration de la machine

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: openshift-node-custom
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=Custom OpenShift node profile with an additional kernel parameter
      include=openshift-node
      [bootloader]
      cmdline_openshift_node_custom=+skew_tick=1
    name: openshift-node-custom

  recommend:
  - machineConfigLabels:
```

```
    machineconfiguration.openshift.io/role: "worker-custom"
  priority: 20
  profile: openshift-node-custom
```

Pour minimiser les redémarrages de nœuds, il faut étiqueter les nœuds cibles avec une étiquette que le sélecteur de nœuds du pool de configuration de la machine fera correspondre, puis créer le Tuned CR ci-dessus et enfin créer le pool de configuration de la machine personnalisé lui-même.

Cloud provider-specific TuneD profiles

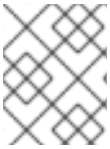
Avec cette fonctionnalité, tous les nœuds spécifiques à un fournisseur de Cloud peuvent commodément se voir attribuer un profil TuneD spécifiquement adapté à un fournisseur de Cloud donné sur un cluster OpenShift Container Platform. Cela peut être accompli sans ajouter d'étiquettes de nœuds supplémentaires ou regrouper les nœuds dans des pools de configuration de machines.

Cette fonctionnalité tire parti des valeurs de l'objet de nœud **spec.providerID** sous la forme de **<cloud-provider>://<cloud-provider-specific-id>** et écrit le fichier **/var/lib/tuned/provider** avec la valeur **<cloud-provider>** dans les conteneurs d'opérandes NTO. Le contenu de ce fichier est ensuite utilisé par TuneD pour charger le profil **provider-<cloud-provider>** s'il existe.

Le profil **openshift** dont les profils **openshift-control-plane** et **openshift-node** héritent des paramètres est maintenant mis à jour pour utiliser cette fonctionnalité grâce à l'utilisation du chargement conditionnel de profil. Ni NTO ni TuneD ne fournissent actuellement de profils spécifiques aux fournisseurs de Cloud. Cependant, il est possible de créer un profil personnalisé **provider-<cloud-provider>** qui sera appliqué à tous les nœuds de cluster spécifiques au fournisseur de cloud.

Exemple de profil de fournisseur GCE Cloud

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: provider-gce
  namespace: openshift-cluster-node-tuning-operator
spec:
  profile:
  - data: |
      [main]
      summary=GCE Cloud provider-specific profile
      # Your tuning for GCE Cloud provider goes here.
    name: provider-gce
```



NOTE

En raison de l'héritage des profils, tout paramètre spécifié dans le profil **provider-<cloud-provider>** sera remplacé par le profil **openshift** et ses profils enfants.

5.5.3. Profils par défaut définis sur un cluster

Les profils par défaut définis sur un cluster sont les suivants.

```
apiVersion: tuned.openshift.io/v1
kind: Tuned
metadata:
  name: default
  namespace: openshift-cluster-node-tuning-operator
```

```
spec:
  profile:
  - data: |
      [main]
      summary=Optimize systems running OpenShift (provider specific parent profile)
      include=-provider-${f:exec:cat:/var/lib/tuned/provider},openshift
    name: openshift
  recommend:
  - profile: openshift-control-plane
    priority: 30
    match:
    - label: node-role.kubernetes.io/master
    - label: node-role.kubernetes.io/infra
  - profile: openshift-node
    priority: 40
```

Depuis OpenShift Container Platform 4.9, tous les profils OpenShift TuneD sont livrés avec le package TuneD. Vous pouvez utiliser la commande **oc exec** pour voir le contenu de ces profils :

```
$ oc exec $tuned_pod -n openshift-cluster-node-tuning-operator -- find /usr/lib/tuned/openshift{,-control-plane,-node} -name tuned.conf -exec grep -H ^ {} \;
```

5.5.4. Plugins de démon TuneD pris en charge

À l'exception de la section **[main]**, les plugins TuneD suivants sont pris en charge lors de l'utilisation des profils personnalisés définis dans la section **profile:** du CR Tuned :

- audio
- cpu
- disque
- eeepc_she
- modules
- montures
- net
- planificateur
- scsi_host
- selinux
- sysctl
- sysfs
- uSB
- vidéo
- vm

- chargeur de démarrage

Certains de ces plugins offrent une fonctionnalité d'accord dynamique qui n'est pas prise en charge. Les plugins TuneD suivants ne sont actuellement pas pris en charge :

- scénario
- systemd

AVERTISSEMENT

Le plugin TuneD bootloader est actuellement pris en charge sur les nœuds de travail Red Hat Enterprise Linux CoreOS (RHCOS) 8.x. Pour les nœuds de travail Red Hat Enterprise Linux (RHEL) 7.x, le plugin de chargeur de démarrage TuneD n'est actuellement pas pris en charge.

Voir Plugins TuneD disponibles et Démarrer avec TuneD pour plus d'informations.

5.6. ASSAINISSEMENT, CLÔTURES ET ENTRETIEN

5.6.1. À propos de l'assainissement des nœuds, des clôtures et de l'entretien

Le matériel est imparfait et les logiciels contiennent des bogues. Lorsque des défaillances au niveau des nœuds, telles que le blocage du noyau ou des contrôleurs d'interface réseau (NIC), surviennent, le travail demandé à la grappe ne diminue pas et les charges de travail des nœuds concernés doivent être redémarrées quelque part. Cependant, certaines charges de travail, telles que les volumes ReadWriteOnce (RWO) et les StatefulSets, peuvent nécessiter une sémantique "at-most-one".

Les défaillances affectant ces charges de travail risquent d'entraîner la perte ou la corruption de données, voire les deux. Il est important de veiller à ce que le nœud atteigne un état sûr (**fencing**) avant d'entamer la reprise de la charge de travail (**remediation**) et, idéalement, la reprise du nœud.

Il n'est pas toujours pratique de dépendre de l'intervention de l'administrateur pour confirmer l'état réel des nœuds et des charges de travail. Pour faciliter cette intervention, OpenShift Container Platform fournit plusieurs composants pour l'automatisation de la détection des défaillances, de la clôture et de la remédiation.

5.6.1.1. Remédiation autonome des nœuds

Le Self Node Remediation Operator est un opérateur complémentaire d'OpenShift Container Platform qui met en œuvre un système externe de clôture et de remédiation qui redémarre les nœuds malsains et supprime les ressources, telles que les Pods et les VolumeAttachments. Le redémarrage garantit que les charges de travail sont clôturées et la suppression des ressources accélère la reprogrammation des charges de travail affectées. Contrairement à d'autres systèmes externes, Self Node Remediation ne nécessite aucune interface de gestion, comme, par exemple, Intelligent Platform Management Interface (IPMI) ou une API pour le provisionnement des nœuds.

L'auto-remédiation des nœuds peut être utilisée par les systèmes de détection des défaillances, comme le bilan de santé de la machine ou le bilan de santé du nœud.

5.6.1.2. Bilan de santé de la machine

Machine Health Check utilise un système intégré de détection des défaillances, de clôture et de remédiation d'OpenShift Container Platform, qui surveille l'état des machines et les conditions des nœuds. Les bilans de santé des machines peuvent être configurés pour déclencher des systèmes de clôture et de remédiation externes, tels que Self Node Remediation.

5.6.1.3. Bilan de santé du nœud

L'opérateur Node Health Check est un opérateur complémentaire d'OpenShift Container Platform qui met en œuvre un système de détection des défaillances qui surveille l'état des nœuds. Il ne dispose pas d'un système de clôture ou de remédiation intégré et doit donc être configuré avec un système externe qui fournit de telles fonctionnalités. Par défaut, il est configuré pour utiliser le système Self Node Remediation.

5.6.1.4. Maintenance des nœuds

Les administrateurs sont confrontés à des situations où ils doivent interrompre la grappe, par exemple pour remplacer un disque, de la mémoire vive ou une carte d'interface réseau.

Avant cette maintenance, les nœuds concernés doivent être isolés et vidés. Lorsqu'un nœud est isolé, il n'est pas possible de programmer de nouvelles charges de travail sur ce nœud. Lorsqu'un nœud est vidé, pour éviter ou minimiser les temps d'arrêt, les charges de travail sur le nœud affecté sont transférées vers d'autres nœuds.

Bien que cette maintenance puisse être réalisée à l'aide d'outils de ligne de commande, l'opérateur de maintenance de nœuds offre une approche déclarative pour y parvenir en utilisant une ressource personnalisée. Lorsqu'une telle ressource existe pour un nœud, l'opérateur cordonne et draine le nœud jusqu'à ce que la ressource soit supprimée.

5.6.2. Utilisation de l'auto-remédiation des nœuds

Vous pouvez utiliser l'opérateur Self Node Remediation pour redémarrer automatiquement les nœuds en mauvais état. Cette stratégie de remédiation minimise les temps d'arrêt pour les applications avec état et les volumes ReadWriteOnce (RWO), et rétablit la capacité de calcul en cas de défaillances transitoires.

5.6.2.1. À propos de l'opérateur d'assainissement autonome des nœuds

L'opérateur Self Node Remediation s'exécute sur les nœuds de la grappe et redémarre les nœuds identifiés comme étant en mauvaise santé. L'opérateur utilise le contrôleur **MachineHealthCheck** ou **NodeHealthCheck** pour détecter l'état d'un nœud dans la grappe. Lorsqu'un nœud est identifié comme étant en mauvaise santé, la ressource **MachineHealthCheck** ou **NodeHealthCheck** crée la ressource personnalisée (CR) **SelfNodeRemediation**, qui déclenche l'opérateur Self Node Remediation.

Le CR **SelfNodeRemediation** ressemble au fichier YAML suivant :

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediation
metadata:
  name: selfnoderemediation-sample
  namespace: openshift-operators
spec:
status:
  lastError: <last_error_message> 1
```

1 Affiche la dernière erreur survenue pendant la remédiation. Lorsque la remédiation réussit ou qu'aucune erreur ne se produit, le champ reste vide.

L'opérateur Self Node Remediation minimise les temps d'arrêt des applications avec état et rétablit la capacité de calcul en cas de défaillance transitoire. Vous pouvez utiliser cet opérateur quelle que soit l'interface de gestion (IPMI ou API pour le provisionnement d'un nœud) et quel que soit le type d'installation du cluster (infrastructure provisionnée par l'installateur ou par l'utilisateur).

5.6.2.1.1. À propos des dispositifs de surveillance

Les dispositifs de surveillance peuvent être l'un des suivants :

- Dispositifs matériels alimentés de manière indépendante
- Dispositifs matériels qui partagent l'énergie avec les hôtes qu'ils contrôlent
- Dispositifs virtuels mis en œuvre dans un logiciel, ou **softdog**

Les dispositifs matériels de surveillance (watchdog) et **softdog** disposent respectivement d'une minuterie électronique ou logicielle. Ces dispositifs de surveillance sont utilisés pour garantir que la machine entre dans un état sûr lorsqu'une condition d'erreur est détectée. Le cluster doit réinitialiser à plusieurs reprises la minuterie du chien de garde pour prouver qu'il est dans un état sain. Cette minuterie peut s'écouler en raison de conditions d'erreur, telles que des blocages, des pannes de CPU et des pertes d'accès au réseau ou au disque. Si le délai expire, le dispositif de chien de garde suppose qu'une erreur s'est produite et déclenche une réinitialisation forcée du nœud.

Les dispositifs de surveillance matériels sont plus fiables que les dispositifs **softdog**.

5.6.2.1.1.1. Comprendre le comportement de l'opérateur de remédiation des nœuds autonomes avec les dispositifs de surveillance

L'opérateur de remédiation du nœud autonome détermine la stratégie de remédiation en fonction des dispositifs de surveillance présents.

Si un dispositif de surveillance matériel est configuré et disponible, l'opérateur l'utilise pour la remédiation. Si un dispositif de surveillance matériel n'est pas configuré, l'opérateur active et utilise un dispositif **softdog** pour la remédiation.

Si aucun dispositif de surveillance n'est pris en charge, que ce soit par le système ou par la configuration, l'opérateur remédie aux nœuds en utilisant le redémarrage du logiciel.

Ressources supplémentaires

Configuration d'un chien de garde

5.6.2.2. Clôture du plan de contrôle

Dans les versions précédentes, vous pouviez activer les fonctions Self Node Remediation et Node Health Check sur les nœuds de travail. En cas de défaillance d'un nœud, vous pouvez désormais suivre des stratégies de remédiation sur les nœuds du plan de contrôle.

L'auto-assainissement d'un nœud se produit dans deux scénarios principaux.

- Connectivité au serveur API

- Dans ce scénario, le nœud du plan de contrôle à assainir n'est pas isolé. Il peut être directement connecté au serveur API ou indirectement connecté au serveur API par l'intermédiaire de nœuds de travail ou de nœuds du plan de contrôle, qui sont directement connectés au serveur API.

- En cas de connectivité avec le serveur API, le nœud du plan de contrôle ne fait l'objet d'une remédiation que si l'opérateur du bilan de santé du nœud a créé une ressource personnalisée (CR) **SelfNodeRemediation** pour le nœud.

- Pas de connectivité avec le serveur API

  - Dans ce scénario, le nœud du plan de contrôle à assainir est isolé du serveur API. Le nœud ne peut pas se connecter directement ou indirectement au serveur API.

  - Lorsqu'il n'y a pas de connectivité avec le serveur API, le nœud du plan de contrôle sera remédié comme indiqué dans les étapes suivantes :

    - Vérifier l'état du nœud du plan de contrôle auprès de la majorité des nœuds homologues. Si la majorité des nœuds homologues ne peut être jointe, le nœud sera analysé plus en détail.

      - Autodiagnostic de l'état du nœud du plan de contrôle

        - Si l'autodiagnostic est réussi, aucune action n'est entreprise.

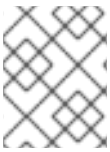
        - Si l'autodiagnostic a échoué, le nœud sera clôturé et remédié.

        - Les autodiagnostics actuellement pris en charge sont la vérification de l'état du service **kubelet** et la vérification de la disponibilité des points d'extrémité à l'aide de la configuration **opt in**.

    - Si le nœud n'a pas réussi à communiquer avec la plupart de ses homologues, vérifiez la connectivité du nœud de plan de contrôle avec d'autres nœuds de plan de contrôle. Si le nœud peut communiquer avec n'importe quel autre homologue du plan de contrôle, aucune mesure ne sera prise. Dans le cas contraire, le nœud sera clôturé et remédié.

5.6.2.3. Installation du Self Node Remediation Operator à l'aide de la console web

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer le Self Node Remediation Operator.



NOTE

L'opérateur du bilan de santé du nœud installe également l'opérateur de remédiation du nœud lui-même en tant que fournisseur de remédiation par défaut.

Conditions préalables

- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Dans la console web d'OpenShift Container Platform, naviguez vers **Operators → OperatorHub**.

2. Recherchez l'opérateur d'assainissement autonome dans la liste des opérateurs disponibles, puis cliquez sur **Install**.

3. Conservez la sélection par défaut de **Installation mode** et **namespace** pour vous assurer que l'opérateur est installé dans l'espace de noms **openshift-operators**.

4. Cliquez sur **Install**.

Vérification

Pour confirmer que l'installation a réussi :

1. Naviguez jusqu'à la page **Operators → Installed Operators**.

2. Vérifiez que l'Opérateur est installé dans l'espace de noms **openshift-operators** et que son statut est **Succeeded**.

Si l'opérateur n'est pas installé correctement :

1. Naviguez jusqu'à la page **Operators → Installed Operators** et vérifiez que la colonne **Status** ne contient pas d'erreurs ou de défaillances.

2. Naviguez jusqu'à la page **Workloads → Pods** et vérifiez les journaux de tous les pods du projet **self-node-remediation-controller-manager** qui signalent des problèmes.

5.6.2.4. Installation du Self Node Remediation Operator à l'aide de la CLI

Vous pouvez utiliser l'OpenShift CLI (**oc**) pour installer le Self Node Remediation Operator.

Vous pouvez installer le Self Node Remediation Operator dans votre propre espace de noms ou dans l'espace de noms **openshift-operators**.

Pour installer l'opérateur dans votre propre espace de noms, suivez les étapes de la procédure.

Pour installer l'opérateur dans l'espace de noms **openshift-operators**, passez à l'étape 3 de la procédure car les étapes de création d'une nouvelle ressource personnalisée (CR) **Namespace** et d'une CR **OperatorGroup** ne sont pas nécessaires.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créer une ressource personnalisée (CR) **Namespace** pour l'opérateur de remédiation de nœud autonome :

   a. Définissez le CR **Namespace** et enregistrez le fichier YAML, par exemple **self-node-remediation-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: self-node-remediation
```

b. Pour créer le CR **Namespace**, exécutez la commande suivante :

```
$ oc create -f self-node-remediation-namespace.yaml
```

2. Créer un CR **OperatorGroup**:

a. Définissez le CR **OperatorGroup** et enregistrez le fichier YAML, par exemple **self-node-remediation-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: self-node-remediation-operator
  namespace: self-node-remediation
```

b. Pour créer le CR **OperatorGroup**, exécutez la commande suivante :

```
$ oc create -f self-node-remediation-operator-group.yaml
```

3. Créer un CR **Subscription**:

a. Définissez le CR **Subscription** et enregistrez le fichier YAML, par exemple **self-node-remediation-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
   name: self-node-remediation-operator
   namespace: self-node-remediation (1)
spec:
   channel: stable
   installPlanApproval: Manual (2)
   name: self-node-remediation-operator
   source: redhat-operators
   sourceNamespace: openshift-marketplace
   package: self-node-remediation
```

(1) Indiquez le site **Namespace** où vous souhaitez installer l'opérateur d'assainissement de l'auto-nœud. Pour installer l'opérateur d'auto-assainissement de nœud dans l'espace de noms **openshift-operators**, spécifiez **openshift-operators** dans le CR **Subscription**.

(2) Définissez la stratégie d'approbation sur Manuel au cas où la version spécifiée serait 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 départ ne puisse terminer l'installation.

b. Pour créer le CR **Subscription**, exécutez la commande suivante :

```
$ oc create -f self-node-remediation-subscription.yaml
```

Vérification

1. Vérifiez que l'installation a réussi en inspectant la ressource CSV :

```
$ oc get csv -n self-node-remediation
```

Exemple de sortie

```
NAME                                  DISPLAY                          VERSION   REPLACES   PHASE
self-node-remediation.v.0.4.0         Self Node Remediation Operator   v.0.4.0
Succeeded
```

2. Vérifiez que l'opérateur de remédiation de nœuds autonomes est opérationnel :

```
$ oc get deploy -n self-node-remediation
```

Exemple de sortie

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
self-node-remediation-controller-manager   1/1     1            1           28h
```

3. Vérifier que l'opérateur d'assainissement autonome a créé la CR **SelfNodeRemediationConfig**:

```
$ oc get selfnoderemediationconfig -n self-node-remediation
```

Exemple de sortie

```
NAME                          AGE
self-node-remediation-config   28h
```

4. Vérifiez que chaque pod de remédiation de nœud autonome est planifié et en cours d'exécution sur chaque nœud de travail :

```
$ oc get daemonset -n self-node-remediation
```

Exemple de sortie

```
NAME                   DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR  AGE
self-node-remediation-ds  3        3        3      3           3          <none>        28h
```

NOTE

Cette commande n'est pas prise en charge pour les nœuds du plan de contrôle.

5.6.2.5. Configuration de l'opérateur de remédiation du nœud autonome

L'opérateur de remédiation du nœud autonome crée la CR **SelfNodeRemediationConfig** et la définition de ressource personnalisée (CRD) **SelfNodeRemediationTemplate**.

5.6.2.5.1. Comprendre la configuration de l'opérateur de remédiation du nœud autonome

L'opérateur d'assainissement autonome crée la CR **SelfNodeRemediationConfig**

L'opérateur d'assainissement de l'auto-nœud crée le CR **SelfNodeRemediationConfig** avec le nom **self-node-remediation-config**. Le CR est créé dans l'espace de noms de l'opérateur d'assainissement de l'auto-nœud.

Un changement dans le CR **SelfNodeRemediationConfig** recrée le jeu de démons Self Node Remediation.

Le CR **SelfNodeRemediationConfig** ressemble au fichier YAML suivant :

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationConfig
metadata:
  name: self-node-remediation-config
  namespace: openshift-operators
spec:
  safeTimeToAssumeNodeRebootedSeconds: 180 (1)
  watchdogFilePath: /dev/watchdog (2)
  isSoftwareRebootEnabled: true (3)
  apiServerTimeout: 15s (4)
  apiCheckInterval: 5s (5)
  maxApiErrorThreshold: 3 (6)
  peerApiServerTimeout: 5s (7)
  peerDialTimeout: 5s (8)
  peerRequestTimeout: 5s (9)
  peerUpdateInterval: 15m (10)
```

(1) Indiquez le délai d'attente pour l'homologue survivant, après lequel l'opérateur peut supposer qu'un nœud malsain a été redémarré. L'opérateur calcule automatiquement la limite inférieure de cette valeur. Toutefois, si différents nœuds ont des délais d'attente différents, vous devez remplacer cette valeur par une valeur plus élevée.

(2) Indiquez le chemin d'accès au fichier du dispositif de surveillance dans les nœuds. Si vous saisissez un chemin d'accès incorrect au dispositif de surveillance, l'opérateur de remédiation des nœuds détecte automatiquement le chemin d'accès au dispositif de surveillance.

Si un dispositif de surveillance n'est pas disponible, le CR **SelfNodeRemediationConfig** utilise un redémarrage logiciel.

(3) Indiquez si vous souhaitez activer le redémarrage du logiciel des nœuds malades. Par défaut, la valeur de **isSoftwareRebootEnabled** est fixée à **true**. Pour désactiver le redémarrage du logiciel, fixez la valeur du paramètre à **false**.

(4) Spécifiez le délai d'attente pour vérifier la connectivité avec chaque serveur API. Lorsque ce délai est écoulé, l'opérateur lance la remédiation. Le délai d'attente doit être supérieur ou égal à 10 millisecondes.

(5) Spécifiez la fréquence de vérification de la connectivité avec chaque serveur API. Le délai d'attente doit être supérieur ou égal à 1 seconde.

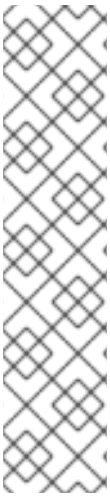
(6) Spécifier une valeur seuil. Une fois ce seuil atteint, le nœud commence à contacter ses homologues. La valeur du seuil doit être supérieure ou égale à 1 seconde.

(7) Spécifiez la durée du délai d'attente pour que l'homologue se connecte au serveur API. La durée du délai d'attente doit être supérieure ou égale à 10 millisecondes.

8 Spécifiez la durée du délai d'attente pour l'établissement de la connexion avec l'homologue. La durée du délai doit être supérieure ou égale à 10 millisecondes.

9 Spécifiez la durée du délai d'attente pour obtenir une réponse de l'homologue. La durée du délai doit être supérieure ou égale à 10 millisecondes.

10 Spécifiez la fréquence de mise à jour des informations sur l'homologue, telles que l'adresse IP. Le délai d'attente doit être supérieur ou égal à 10 secondes.



NOTE

Vous pouvez modifier la CR **self-node-remediation-config** créée par l'opérateur d'assainissement autonome. Cependant, lorsque vous essayez de créer un nouveau CR pour l'opérateur d'assainissement autonome, le message suivant s'affiche dans les journaux :

```
controllers.SelfNodeRemediationConfig
ignoring selfnoderemediationconfig CRs that are not named 'self-node-remediation-config'
or not in the namespace of the operator:
'openshift-operators' {"selfnoderemediationconfig":
"openshift-operators/selfnoderemediationconfig-copy"}
```

5.6.2.5.2. Comprendre la configuration du modèle d'auto-assainissement des nœuds

L'opérateur d'assainissement autonome des nœuds crée également la définition de ressource personnalisée (CRD) **SelfNodeRemediationTemplate**. Ce CRD définit la stratégie de remédiation pour les nœuds. Les stratégies de remédiation suivantes sont disponibles :

ResourceDeletion

Cette stratégie de remédiation supprime les pods et les attachements de volume associés sur le nœud plutôt que sur l'objet nœud. Cette stratégie permet de récupérer les charges de travail plus rapidement. **ResourceDeletion** est la stratégie de remédiation par défaut.

NodeDeletion

Cette stratégie de remédiation est obsolète et sera supprimée dans une prochaine version. Dans la version actuelle, la stratégie **ResourceDeletion** est utilisée même si la stratégie **NodeDeletion** est sélectionnée.

L'opérateur de remédiation du nœud autonome crée le CR **SelfNodeRemediationTemplate** pour la stratégie **self-node-remediation-resource-deletion-template**, que la stratégie de remédiation **ResourceDeletion** utilise.

Le CR **SelfNodeRemediationTemplate** ressemble au fichier YAML suivant :

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  creationTimestamp: "2022-03-02T08:02:40Z"
  name: self-node-remediation-<remediation_object>-deletion-template 1
  namespace: openshift-operators
spec:
```

```
template:
  spec:
    remediationStrategy: <remediation_strategy> 2
```

1 Spécifie le type de modèle de remédiation basé sur la stratégie de remédiation. Remplacez **<remediation_object>** par **resource** ou **node**; par exemple, **self-node-remediation-resource-deletion-template**.

2 Spécifie la stratégie de remédiation. La stratégie de remédiation est **ResourceDeletion**.

5.6.2.6. Dépannage de l'opérateur d'auto-assainissement des nœuds

5.6.2.6.1. Dépannage général

Enjeu

Vous souhaitez résoudre les problèmes liés à l'opérateur de remédiation des nœuds autonomes.

Résolution

Vérifier les journaux de l'opérateur.

5.6.2.6.2. Vérification du jeu de démons

Enjeu

Le Self Node Remediation Operator est installé mais le jeu de démons n'est pas disponible.

Résolution

Vérifiez que les journaux de l'opérateur ne contiennent pas d'erreurs ou d'avertissements.

5.6.2.6.3. Remédiation infructueuse

Enjeu

Un nœud malsain n'a pas été remédié.

Résolution

Vérifiez que le CR **SelfNodeRemediation** a été créé en exécutant la commande suivante :

```
$ oc get snr -A
```

Si le contrôleur **MachineHealthCheck** n'a pas créé la CR **SelfNodeRemediation** lorsque le nœud est devenu malsain, vérifiez les journaux du contrôleur **MachineHealthCheck**. En outre, assurez-vous que le CR **MachineHealthCheck** inclut les spécifications requises pour utiliser le modèle de remédiation.

Si le CR **SelfNodeRemediation** a été créé, assurez-vous que son nom correspond au nœud malsain ou à l'objet machine.

5.6.2.6.4. L'ensemble de démons et d'autres ressources de l'opérateur de remédiation des nœuds autonomes existent même après la désinstallation de l'opérateur

Enjeu

Les ressources de l'opérateur de remédiation du nœud autonome, telles que le jeu de démons, la CR de configuration et la CR du modèle de remédiation, existent même après la désinstallation de l'opérateur.

Résolution

Pour supprimer les ressources de l'opérateur de remédiation du nœud autonome, supprimez les ressources en exécutant les commandes suivantes pour chaque type de ressource :

```
$ oc delete ds <self-node-remediation-ds> -n <namespace>
```

```
oc delete snrc <self-node-remediation-config> -n <namespace>
```

```
oc delete snrt <self-node-remediation-template> -n <namespace>
```

5.6.2.7. Collecte de données sur l'opérateur de remédiation de l'auto-nœud

Pour collecter des informations de débogage sur l'opérateur d'assainissement autonome des nœuds, utilisez l'outil **must-gather**. Pour plus d'informations sur l'image **must-gather** de l'opérateur d'assainissement autonome des nœuds, voir Collecte de données sur des fonctionnalités spécifiques .

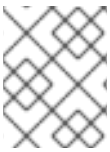
5.6.2.8. Ressources supplémentaires

- Utilisation d'Operator Lifecycle Manager sur des réseaux restreints .
- Suppression d'opérateurs d'une grappe

5.6.3. Remédier aux nœuds avec les bilans de santé des machines

Les contrôles de santé des machines réparent automatiquement les machines en mauvais état dans un pool de machines particulier.

5.6.3.1. À propos des contrôles de santé des machines



NOTE

Vous ne pouvez appliquer un contrôle de l'état des machines qu'aux machines du plan de contrôle des clusters qui utilisent des jeux de machines du plan de contrôle.

Pour surveiller l'état des machines, créez une ressource afin de définir la configuration d'un contrôleur. Définissez une condition à vérifier, telle que le maintien de l'état **NotReady** pendant cinq minutes ou l'affichage d'une condition permanente dans le détecteur de problèmes de nœuds, ainsi qu'une étiquette pour l'ensemble des machines à surveiller.

Le contrôleur qui observe une ressource **MachineHealthCheck** vérifie la condition définie. Si une machine échoue au contrôle de santé, elle est automatiquement supprimée et une autre est créée pour la remplacer. Lorsqu'une machine est supprimée, un événement **machine deleted** s'affiche.

Pour limiter l'impact perturbateur de la suppression des machines, le contrôleur ne draine et ne supprime qu'un seul nœud à la fois. S'il y a plus de machines malsaines que le seuil **maxUnhealthy** ne le permet dans le groupe de machines ciblées, la remédiation s'arrête et permet donc une intervention manuelle.

NOTE

Les délais d'attente doivent être étudiés avec soin, en tenant compte de la charge de travail et des besoins.

- Les délais d'attente prolongés peuvent entraîner de longues périodes d'indisponibilité de la charge de travail sur la machine en état d'insalubrité.
- Des délais trop courts peuvent entraîner une boucle de remédiation. Par exemple, le délai de vérification de l'état de **NotReady** doit être suffisamment long pour permettre à la machine de terminer le processus de démarrage.

Pour arrêter le contrôle, retirez la ressource.

5.6.3.1.1. Limitations lors du déploiement des contrôles de santé des machines

Il y a des limites à prendre en compte avant de déployer un bilan de santé machine :

- Seules les machines appartenant à un jeu de machines sont remédiées par un bilan de santé de la machine.
- Si le nœud d'une machine est supprimé du cluster, un contrôle de santé de la machine considère que la machine n'est pas en bonne santé et y remédie immédiatement.
- Si le nœud correspondant à une machine ne rejoint pas le cluster après le **nodeStartupTimeout**, la machine est remédiée.
- Une machine est remédiée immédiatement si la phase de ressource **Machine** est **Failed**.

5.6.3.2. Configurer les contrôles de santé des machines pour utiliser l'opérateur de remédiation de nœuds autonomes (Self Node Remediation Operator)

Utilisez la procédure suivante pour configurer les contrôles de santé des machines du plan de travail ou du plan de contrôle afin d'utiliser l'opérateur de remédiation du nœud autonome en tant que fournisseur de remédiation.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créer un CR **SelfNodeRemediationTemplate**:
   a. Définir le CR **SelfNodeRemediationTemplate**:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  namespace: openshift-machine-api
  name: selfnoderemediationtemplate-sample
spec:
```

```
  template:
    spec:
      remediationStrategy: ResourceDeletion (1)
```

(1) Spécifie la stratégie de remédiation. La stratégie par défaut est **ResourceDeletion**.

b. Pour créer le CR **SelfNodeRemediationTemplate**, exécutez la commande suivante :

```
oc create -f <snrt-name>.yaml
```

2. Créez ou mettez à jour le CR **MachineHealthCheck** pour qu'il pointe vers le CR **SelfNodeRemediationTemplate**:

a. Définir ou mettre à jour le CR **MachineHealthCheck**:

```
apiVersion: machine.openshift.io/v1beta1
kind: MachineHealthCheck
metadata:
  name: machine-health-check
  namespace: openshift-machine-api
spec:
  selector:
    matchLabels: (1)
      machine.openshift.io/cluster-api-machine-role: "worker"
      machine.openshift.io/cluster-api-machine-type: "worker"
  unhealthyConditions:
  - type:    "Ready"
    timeout: "300s"
    status: "False"
  - type:    "Ready"
    timeout: "300s"
    status: "Unknown"
  maxUnhealthy: "40%"
  nodeStartupTimeout: "10m"
  remediationTemplate: (2)
    kind: SelfNodeRemediationTemplate
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    name: selfnoderemediationtemplate-sample
```

(1) Indique si le contrôle de l'état de la machine concerne les nœuds **worker** ou **control-plane**. L'étiquette peut également être définie par l'utilisateur.

(2) Spécifie les détails du modèle de remédiation.

b. Pour créer un CR **MachineHealthCheck**, exécutez la commande suivante :

```
oc create -f <mhc-name>.yaml
```

c. Pour mettre à jour un CR **MachineHealthCheck**, exécutez la commande suivante :

```
$ oc apply -f <mhc-name>.yaml
```

5.6.4. Remédier aux nœuds avec les contrôles de santé des nœuds

Vous pouvez utiliser l'opérateur de contrôle de l'état des nœuds pour identifier les nœuds en mauvais état. L'opérateur utilise l'opérateur d'auto-remédiation des nœuds pour remédier aux nœuds malsains.

Ressources supplémentaires

Remédiation des nœuds avec l'opérateur de remédiation des nœuds autonomes

5.6.4.1. À propos de l'opérateur du bilan de santé du nœud

L'opérateur Node Health Check détecte l'état de santé des nœuds d'une grappe. Le contrôleur **NodeHealthCheck** crée la ressource personnalisée (CR) **NodeHealthCheck**, qui définit un ensemble de critères et de seuils permettant de déterminer l'état d'un nœud.

L'opérateur du bilan de santé du nœud installe également l'opérateur de remédiation du nœud lui-même en tant que fournisseur de remédiation par défaut.

Lorsque l'opérateur de contrôle de l'état des nœuds détecte un nœud malsain, il crée un CR de remédiation qui déclenche le fournisseur de remédiation. Par exemple, le contrôleur crée la CR **SelfNodeRemediation**, qui déclenche l'opérateur de remédiation de nœud autonome pour remédier au nœud malsain.

Le CR **NodeHealthCheck** ressemble au fichier YAML suivant :

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-sample
spec:
  minHealthy: 51% 1
  pauseRequests: 2
    - <pause-test-cluster>
  remediationTemplate: 3
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    name: self-node-remediation-resource-deletion-template
    namespace: openshift-operators
    kind: SelfNodeRemediationTemplate
  selector: 4
    matchExpressions:
      - key: node-role.kubernetes.io/worker
        operator: Exists
  unhealthyConditions: 5
    - type: Ready
      status: "False"
      duration: 300s 6
    - type: Ready
      status: Unknown
      duration: 300s 7
```

1 Spécifie le nombre de nœuds sains (en pourcentage ou en nombre) requis pour qu'un fournisseur de remédiation remédie simultanément aux nœuds du pool ciblé. Si le nombre de nœuds sains est égal ou supérieur à la limite définie par **minHealthy**, la remédiation a lieu. La valeur par défaut est 51 %.

2 Empêche le démarrage de toute nouvelle remédiation, tout en permettant aux remédiations en cours de persister. La valeur par défaut est vide. Cependant, vous pouvez saisir un tableau de

NOTE

Au cours du processus de mise à niveau, les nœuds de la grappe peuvent devenir temporairement indisponibles et être identifiés comme malsains. Dans le cas des nœuds de travail, lorsque l'opérateur détecte que la grappe est en cours de mise à niveau, il cesse de remédier aux nouveaux nœuds malsains afin d'éviter que ces nœuds ne redémarrent.

3 Spécifie un modèle de remédiation à partir du fournisseur de remédiation. Par exemple, de l'opérateur de remédiation Self Node.

4 Spécifie une adresse **selector** qui correspond aux étiquettes ou aux expressions que vous souhaitez vérifier. La valeur par défaut est empty, qui sélectionne tous les nœuds.

5 Spécifie une liste des conditions qui déterminent si un nœud est considéré comme malsain.

6 7 Spécifie la durée du délai d'attente pour une condition de nœud. Si une condition est remplie pendant la durée du délai, le nœud sera remédié. Les délais d'attente prolongés peuvent entraîner de longues périodes d'indisponibilité pour une charge de travail sur un nœud malsain.

5.6.4.1.1. Comprendre le flux de travail de l'opérateur du bilan de santé du nœud

Lorsqu'un nœud est identifié comme étant en mauvaise santé, l'opérateur de contrôle de la santé des nœuds vérifie combien d'autres nœuds sont en mauvaise santé. Si le nombre de nœuds sains dépasse la quantité spécifiée dans le champ **minHealthy** du CR **NodeHealthCheck**, le contrôleur crée un CR de remédiation à partir des détails fournis dans le modèle de remédiation externe par le fournisseur de remédiation. Après la remédiation, le kubelet met à jour l'état de santé du nœud.

Lorsque le nœud devient sain, le contrôleur supprime le modèle de remédiation externe.

5.6.4.1.2. Comment les contrôles de santé des nœuds évitent les conflits avec les contrôles de santé des machines

Lorsque des contrôles de santé des nœuds et des contrôles de santé des machines sont déployés, le contrôle de santé des nœuds évite tout conflit avec le contrôle de santé des machines.

NOTE

OpenShift Container Platform déploie **machine-api-termination-handler** comme ressource par défaut **MachineHealthCheck**.

La liste suivante résume le comportement du système lorsque les contrôles de santé des nœuds et des machines sont déployés :

- Si seul le contrôle de l'état de la machine par défaut existe, le contrôle de l'état des nœuds continue d'identifier les nœuds en mauvaise santé. Toutefois, le bilan de santé des nœuds ignore les nœuds malsains en état d'arrêt. Le contrôle de l'état de santé de la machine par défaut traite les nœuds malsains dont l'état est en voie d'achèvement.

  Exemple de message du journal

  -

```
INFO MHCChecker ignoring unhealthy Node, it is terminating and will be handled by MHC
{"NodeName": "node-1.example.com"}
```

- Si le contrôle de l'état de la machine par défaut est modifié (par exemple, **unhealthyConditions** est **Ready**) ou si des contrôles de l'état de la machine supplémentaires sont créés, le contrôle de l'état du nœud est désactivé.

  Exemple de message du journal

```
INFO controllers.NodeHealthCheck disabling NHC in order to avoid conflict with custom
MHCs configured in the cluster {"NodeHealthCheck": "/nhc-worker-default"}
```

- Lorsque, à nouveau, seul le bilan de santé par défaut de la machine existe, le bilan de santé du nœud est réactivé.

  Exemple de message du journal

```
INFO controllers.NodeHealthCheck re-enabling NHC, no conflicting MHC configured in the
cluster {"NodeHealthCheck": "/nhc-worker-default"}
```

5.6.4.2. Clôture du plan de contrôle

Dans les versions précédentes, vous pouviez activer les fonctions Self Node Remediation et Node Health Check sur les nœuds de travail. En cas de défaillance d'un nœud, vous pouvez désormais suivre des stratégies de remédiation sur les nœuds du plan de contrôle.

N'utilisez pas le même **NodeHealthCheck** CR pour les nœuds de travail et les nœuds de plan de contrôle. Le regroupement de nœuds de travail et de nœuds de plan de contrôle peut entraîner une évaluation incorrecte du nombre minimal de nœuds sains et provoquer des mesures correctives inattendues ou manquantes. Cela est dû à la manière dont l'opérateur du bilan de santé des nœuds gère les nœuds de plan de contrôle. Vous devez regrouper les nœuds de plan de contrôle dans leur propre groupe et les nœuds de travail dans leur propre groupe. Si nécessaire, vous pouvez également créer plusieurs groupes de nœuds de travail.

Considérations relatives aux stratégies de remédiation :

- Évitez les configurations du bilan de santé des nœuds qui impliquent plusieurs configurations chevauchant les mêmes nœuds, car elles peuvent entraîner un comportement inattendu. Cette suggestion s'applique aux nœuds du plan de travail et du plan de contrôle.

- L'opérateur de contrôle de l'état des nœuds met en œuvre une limitation codée en dur consistant à remédier à un maximum d'un nœud de plan de contrôle à la fois. Plusieurs nœuds de plan de contrôle ne doivent pas être assainis en même temps.

5.6.4.3. Installation du Node Health Check Operator à l'aide de la console web

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer l'opérateur Node Health Check.

Conditions préalables

- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Dans la console web d'OpenShift Container Platform, naviguez vers **Operators → OperatorHub**.
2. Recherchez l'opérateur du bilan de santé du nœud, puis cliquez sur **Install**.
3. Conservez la sélection par défaut de **Installation mode** et **namespace** pour vous assurer que l'opérateur sera installé dans l'espace de noms **openshift-operators**.
4. Assurez-vous que l'adresse **Console plug-in** est réglée sur **Enable**.
5. Cliquez sur **Install**.

Vérification

Pour confirmer que l'installation a réussi :

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Vérifiez que l'Opérateur est installé dans l'espace de noms **openshift-operators** et que son statut est **Succeeded**.

Si l'opérateur n'est pas installé correctement :

1. Naviguez jusqu'à la page **Operators → Installed Operators** et vérifiez que la colonne **Status** ne contient pas d'erreurs ou de défaillances.
2. Naviguez jusqu'à la page **Workloads → Pods** et vérifiez les journaux de tous les pods du projet **openshift-operators** qui signalent des problèmes.

5.6.4.4. Installation de l'opérateur de contrôle de santé des nœuds à l'aide de la CLI

Vous pouvez utiliser le CLI OpenShift (**oc**) pour installer l'opérateur Node Health Check.

Pour installer l'opérateur dans votre propre espace de noms, suivez les étapes de la procédure.

Pour installer l'opérateur dans l'espace de noms **openshift-operators**, passez à l'étape 3 de la procédure car les étapes de création d'une nouvelle ressource personnalisée (CR) **Namespace** et d'une CR **OperatorGroup** ne sont pas nécessaires.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créer une ressource personnalisée (CR) **Namespace** pour l'opérateur du bilan de santé du nœud :
   a. Définissez le CR **Namespace** et enregistrez le fichier YAML, par exemple **node-health-check-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: node-health-check
```

b. Pour créer le CR **Namespace**, exécutez la commande suivante :

```
$ oc create -f node-health-check-namespace.yaml
```

2. Créer un CR **OperatorGroup**:

a. Définissez le CR **OperatorGroup** et enregistrez le fichier YAML, par exemple **node-health-check-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-health-check-operator
  namespace: node-health-check
```

b. Pour créer le CR **OperatorGroup**, exécutez la commande suivante :

```
$ oc create -f node-health-check-operator-group.yaml
```

3. Créer un CR **Subscription**:

a. Définissez le CR **Subscription** et enregistrez le fichier YAML, par exemple **node-health-check-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
   name: node-health-check-operator
   namespace: node-health-check (1)
spec:
   channel: stable (2)
   installPlanApproval: Manual (3)
   name: node-healthcheck-operator
   source: redhat-operators
   sourceNamespace: openshift-marketplace
   package: node-healthcheck-operator
```

(1) Indiquez le site **Namespace** où vous souhaitez installer l'opérateur de contrôle de santé des nœuds. Pour installer l'opérateur de contrôle de santé des nœuds dans l'espace de noms **openshift-operators**, indiquez **openshift-operators** dans le CR **Subscription**.

(2) Indiquez le nom du canal pour votre abonnement. Pour passer à la dernière version du Node Health Check Operator, vous devez modifier manuellement le nom du canal de votre abonnement de **candidate** à **stable**.

(3) Définissez la stratégie d'approbation sur Manuel au cas où la version spécifiée serait 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 départ ne puisse terminer l'installation.

b. Pour créer le CR **Subscription**, exécutez la commande suivante :

```
$ oc create -f node-health-check-subscription.yaml
```

Vérification

1. Vérifiez que l'installation a réussi en inspectant la ressource CSV :

```
$ oc get csv -n openshift-operators
```

Exemple de sortie

```
NAME                                    DISPLAY                     VERSION  REPLACES PHASE
node-healthcheck-operator.v0.2.0. Node Health Check Operator  0.2.0            Succeeded
```

2. Vérifiez que l'opérateur du bilan de santé du nœud est opérationnel :

```
$ oc get deploy -n openshift-operators
```

Exemple de sortie

```
NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
node-health-check-operator-controller-manager  1/1     1            1          10d
```

5.6.4.5. Création d'un bilan de santé d'un nœud

À l'aide de la console web, vous pouvez créer un bilan de santé des nœuds afin d'identifier les nœuds en mauvaise santé et de spécifier le type et la stratégie de remédiation pour y remédier.

Procédure

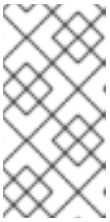
1. Dans la perspective **Administrator** de la console web OpenShift Container Platform, cliquez sur **Compute → NodeHealthChecks → CreateNodeHealthCheck**.

2. Indiquez si vous souhaitez configurer le contrôle de santé du nœud à l'aide de **Form view** ou de **YAML view**.

3. Saisissez une adresse **Name** pour le contrôle de santé du nœud. Le nom doit être composé de minuscules, de caractères alphanumériques, de "-" ou de ".", et doit commencer et se terminer par un caractère alphanumérique.

4. Spécifiez le type **Remediator** et **Self node remediation** ou **Other**. L'option Self node remediation fait partie de l'opérateur Self Node Remediation qui est installé avec l'opérateur Node Health Check. La sélection de **Other** nécessite la saisie de **API version**, **Kind**, **Name** et **Namespace**, qui pointent ensuite vers la ressource de modèle de remédiation d'un remédiateur.

5. Effectuez une sélection sur **Nodes** en spécifiant les étiquettes des nœuds que vous souhaitez assainir. La sélection correspond aux étiquettes que vous souhaitez vérifier. Si plusieurs étiquettes sont spécifiées, les nœuds doivent contenir chacune d'entre elles. La valeur par défaut est empty, ce qui permet de sélectionner à la fois les nœuds du plan de travail et les nœuds du plan de contrôle.



NOTE

Lors de la création d'un contrôle de l'état d'un nœud à l'aide de l'opérateur d'auto-remédiation des nœuds, vous devez sélectionner **node-role.kubernetes.io/worker** ou **node-role.kubernetes.io/control-plane** comme valeur.

6. Spécifiez le nombre minimum de nœuds sains, sous la forme d'un pourcentage ou d'un nombre, requis pour qu'un site **NodeHealthCheck** remédie aux nœuds du pool ciblé. Si le nombre de nœuds sains est égal ou supérieur à la limite définie par **Min healthy**, la remédiation a lieu. La valeur par défaut est 51 %.

7. Spécifiez une liste de **Unhealthy conditions** qui, si un nœud y répond, détermine si le nœud est considéré comme malsain et nécessite une remédiation. Vous pouvez spécifier les types **Type**, **Status** et **Duration**. Vous pouvez également créer votre propre type personnalisé.

8. Cliquez sur **Create** pour créer le bilan de santé du nœud.

Vérification

- Accédez à la page **Compute → NodeHealthCheck** et vérifiez que le contrôle de santé du nœud correspondant est répertorié et que son état est affiché. Une fois créés, les contrôles de santé des nœuds peuvent être interrompus, modifiés et supprimés.

5.6.4.6. Collecte de données sur l'opérateur du bilan de santé du nœud

Pour collecter des informations de débogage sur le Node Health Check Operator, utilisez l'outil **must-gather**. Pour plus d'informations sur l'image **must-gather** de l'opérateur de contrôle de santé des nœuds, voir Collecte de données sur des fonctionnalités spécifiques .

5.6.4.7. Ressources supplémentaires

- Changer le canal de mise à jour d'un opérateur
- Utilisation d'Operator Lifecycle Manager sur des réseaux restreints .

5.6.5. Mise en mode maintenance des nœuds avec l'opérateur de maintenance des nœuds

Vous pouvez utiliser l'opérateur de maintenance des nœuds pour placer les nœuds en mode maintenance à l'aide de l'utilitaire **oc adm** ou des ressources personnalisées (CR) **NodeMaintenance**.

5.6.5.1. À propos de l'opérateur de maintenance des nœuds

L'opérateur de maintenance des nœuds surveille les CR **NodeMaintenance** nouveaux ou supprimés. Lorsqu'un nouveau CR **NodeMaintenance** est détecté, aucune nouvelle charge de travail n'est programmée et le nœud est isolé du reste du cluster. Tous les pods qui peuvent être expulsés le sont du nœud. Lorsqu'un CR **NodeMaintenance** est supprimé, le nœud référencé dans le CR est rendu disponible pour de nouvelles charges de travail.

NOTE

L'utilisation d'un CR **NodeMaintenance** pour les tâches de maintenance des nœuds permet d'obtenir les mêmes résultats que les commandes **oc adm cordon** et **oc adm drain** à l'aide du traitement CR standard de OpenShift Container Platform.

5.6.5.2. Installation de l'opérateur de maintenance des nœuds

Vous pouvez installer l'opérateur de maintenance de nœuds à l'aide de la console Web ou de la CLI OpenShift (**oc**).

NOTE

Si la version 4.10 ou moins d'OpenShift Virtualization est installée dans votre cluster, elle inclut une version obsolète de l'opérateur de maintenance Node.

5.6.5.2.1. Installation de l'opérateur de maintenance de nœuds à l'aide de la console web

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer l'opérateur de maintenance Node.

Conditions préalables

- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Dans la console web d'OpenShift Container Platform, naviguez vers **Operators → OperatorHub**.
2. Recherchez l'opérateur de maintenance du nœud, puis cliquez sur **Install**.
3. Conservez la sélection par défaut de **Installation mode** et **namespace** pour vous assurer que l'opérateur sera installé dans l'espace de noms **openshift-operators**.
4. Cliquez sur **Install**.

Vérification

Pour confirmer que l'installation a réussi :

1. Naviguez jusqu'à la page **Operators → Installed Operators**.
2. Vérifiez que l'Opérateur est installé dans l'espace de noms **openshift-operators** et que son statut est **Succeeded**.

Si l'opérateur n'est pas installé correctement :

1. Naviguez jusqu'à la page **Operators → Installed Operators** et vérifiez que la colonne **Status** ne contient pas d'erreurs ou de défaillances.
2. Naviguez jusqu'à la page **Operators → Installed Operators → Node Maintenance Operator → Details**, et vérifiez que la section **Conditions** ne contient pas d'erreurs avant la création du pod.
3. Naviguez jusqu'à la page **Workloads → Pods**, recherchez le pod **Node Maintenance Operator** dans l'espace de noms installé et vérifiez les journaux dans l'onglet **Logs**.

5.6.5.2.2. Installation de l'opérateur de maintenance de nœuds à l'aide de la CLI

Vous pouvez utiliser le CLI OpenShift (**oc**) pour installer l'opérateur de maintenance Node.

Vous pouvez installer l'opérateur de maintenance de nœuds dans votre propre espace de noms ou dans l'espace de noms **openshift-operators**.

Pour installer l'opérateur dans votre propre espace de noms, suivez les étapes de la procédure.

Pour installer l'opérateur dans l'espace de noms **openshift-operators**, passez à l'étape 3 de la procédure car les étapes de création d'une nouvelle ressource personnalisée (CR) **Namespace** et d'une CR **OperatorGroup** ne sont pas nécessaires.

Conditions préalables

- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créer un CR **Namespace** pour l'opérateur de maintenance des nœuds :
   a. Définissez le CR **Namespace** et enregistrez le fichier YAML, par exemple **node-maintenance-namespace.yaml**:

```
apiVersion: v1
kind: Namespace
metadata:
  name: nmo-test
```

   b. Pour créer le CR **Namespace**, exécutez la commande suivante :

```
$ oc create -f node-maintenance-namespace.yaml
```

2. Créer un CR **OperatorGroup**:
   a. Définissez le CR **OperatorGroup** et enregistrez le fichier YAML, par exemple **node-maintenance-operator-group.yaml**:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: node-maintenance-operator
  namespace: nmo-test
```

   b. Pour créer le CR **OperatorGroup**, exécutez la commande suivante :

```
$ oc create -f node-maintenance-operator-group.yaml
```

3. Créer un CR **Subscription**:
   a. Définissez le CR **Subscription** et enregistrez le fichier YAML, par exemple **node-maintenance-subscription.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: node-maintenance-operator
  namespace: nmo-test 1
spec:
  channel: stable
  InstallPlaneApproval: Automatic
  name: node-maintenance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  StartingCSV: node-maintenance-operator.v4.12.0
```

1 Spécifiez le site **Namespace** où vous souhaitez installer l'opérateur de maintenance de nœuds.

IMPORTANT

Pour installer l'opérateur de maintenance de nœuds dans l'espace de noms **openshift-operators**, spécifiez **openshift-operators** dans le CR **Subscription**.

b. Pour créer le CR **Subscription**, exécutez la commande suivante :

```
$ oc create -f node-maintenance-subscription.yaml
```

Vérification

1. Vérifiez que l'installation a réussi en inspectant la ressource CSV :

```
$ oc get csv -n openshift-operators
```

Exemple de sortie

```
NAME                                DISPLAY                    VERSION   REPLACES  PHASE
node-maintenance-operator.v4.12     Node Maintenance Operator  4.12                Succeeded
```

2. Vérifiez que l'opérateur de maintenance des nœuds est en cours d'exécution :

```
$ oc get deploy -n openshift-operators
```

Exemple de sortie

```
NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
node-maintenance-operator-controller-manager  1/1     1            1           10d
```

L'opérateur de maintenance de nœuds est pris en charge dans un environnement de réseau restreint. Pour plus d'informations, voir Utilisation d'Operator Lifecycle Manager sur des réseaux restreints .

5.6.5.3. Mise en mode maintenance d'un nœud

Vous pouvez placer un nœud en mode maintenance à partir de la console Web ou de la CLI en utilisant **NodeMaintenance** CR.

5.6.5.3.1. Mise en mode maintenance d'un nœud à l'aide de la console web

Pour mettre un nœud en mode maintenance, vous pouvez créer une ressource personnalisée (CR) **NodeMaintenance** à l'aide de la console web.

Conditions préalables

- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.
- Installez l'opérateur de maintenance du nœud à partir du site **OperatorHub**.

Procédure

1. Depuis la perspective **Administrator** dans la console web, naviguez vers **Operators → Installed Operators**.
2. Sélectionnez l'opérateur de maintenance du nœud dans la liste des opérateurs.
3. Dans l'onglet **Node Maintenance**, cliquez sur **Create NodeMaintenance**.
4. Dans la page **Create NodeMaintenance**, sélectionnez **Form view** ou **YAML view** pour configurer le CR **NodeMaintenance**.
5. Pour appliquer la CR **NodeMaintenance** que vous avez configurée, cliquez sur **Create**.

Vérification

Dans l'onglet **Node Maintenance**, inspectez la colonne **Status** et vérifiez que son statut est **Succeeded**.

5.6.5.3.2. Mise en mode maintenance d'un nœud à l'aide de la CLI

Vous pouvez mettre un nœud en mode maintenance avec une ressource personnalisée (CR) **NodeMaintenance**. Lorsque vous appliquez une CR **NodeMaintenance**, tous les pods autorisés sont expulsés et le nœud devient inutilisable. Les pods expulsés sont mis en file d'attente pour être déplacés vers un autre nœud du cluster.

Conditions préalables

- Install the OpenShift Container Platform CLI **oc**.
- Connectez-vous au cluster en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

1. Créez le document **NodeMaintenance** CR suivant et enregistrez le fichier sous **nodemaintenance-cr.yaml**:

```
apiVersion: nodemaintenance.medik8s.io/v1beta1
kind: NodeMaintenance
metadata:
  name: nodemaintenance-cr 1
```

```
spec:
  nodeName: node-1.example.com 2
  reason: "NIC replacement" 3
```

1. Le nom du CR de maintenance du nœud.
2. Le nom du nœud à mettre en mode maintenance.
3. Description en texte clair du motif de la maintenance.

2. Appliquez le CR de maintenance des nœuds en exécutant la commande suivante :

```
$ oc apply -f nodemaintenance-cr.yaml
```

Vérification

1. Vérifiez l'état d'avancement de la tâche de maintenance en exécutant la commande suivante :

```
$ oc describe node <node-name>
```

où **<node-name>** est le nom de votre nœud ; par exemple, **node-1.example.com**

2. Vérifier la sortie de l'exemple :

```
Events:
  Type     Reason                   Age                From     Message
  ----     ------                   ----               ----     -------
  Normal   NodeNotSchedulable       61m                kubelet  Node node-1.example.com
status is now: NodeNotSchedulable
```

5.6.5.3.3. Vérification de l'état des tâches de maintenance des nœuds en cours de CR

Vous pouvez vérifier l'état des tâches **NodeMaintenance** CR en cours.

Conditions préalables

- Install the OpenShift Container Platform CLI **oc**.
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

- Vérifiez l'état des tâches de maintenance des nœuds en cours, par exemple l'objet **NodeMaintenance** CR ou **nm**, en exécutant la commande suivante :

```
$ oc get nm -o yaml
```

Exemple de sortie

```
apiVersion: v1
items:
- apiVersion: nodemaintenance.medik8s.io/v1beta1
```

```
  kind: NodeMaintenance
  metadata:
...
  spec:
    nodeName: node-1.example.com
    reason: Node maintenance
  status:
    drainProgress: 100 (1)
    evictionPods: 3 (2)
    lastError: "Last failure message" (3)
    lastUpdate: "2022-06-23T11:43:18Z" (4)
    phase: Succeeded
    totalpods: 5 (5)
...
```

1. Le pourcentage d'achèvement de la vidange du nœud.
2. Le nombre de nacelles dont l'expulsion est prévue.
3. La dernière erreur d'expulsion, le cas échéant.
4. La dernière fois que le statut a été mis à jour.
5. Nombre total de pods avant que le nœud ne passe en mode maintenance.

5.6.5.4. Reprise d'un nœud en mode maintenance

Vous pouvez reprendre un nœud depuis le mode de maintenance à partir de la console Web ou de l'interface CLI en utilisant **NodeMaintenance** CR. La reprise d'un nœud le fait sortir du mode de maintenance et le rend à nouveau planifiable.

5.6.5.4.1. Reprise d'un nœud en mode maintenance à l'aide de la console web

Pour reprendre un nœud en mode maintenance, vous pouvez supprimer une ressource personnalisée (CR) **NodeMaintenance** à l'aide de la console web.

Conditions préalables

- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.
- Installez l'opérateur de maintenance du nœud à partir du site **OperatorHub**.

Procédure

1. Depuis la perspective **Administrator** dans la console web, naviguez vers **Operators → Installed Operators**.
2. Sélectionnez l'opérateur de maintenance du nœud dans la liste des opérateurs.
3. Dans l'onglet **Node Maintenance**, sélectionnez le CR **NodeMaintenance** que vous souhaitez supprimer.

4. Cliquez sur le menu Options  à l'extrémité du nœud et sélectionnez **Delete NodeMaintenance**.

Vérification

1. Dans la console OpenShift Container Platform, cliquez sur **Compute → Nodes**.
2. Inspectez la colonne **Status** du nœud pour lequel vous avez supprimé le CR **NodeMaintenance** et vérifiez que son statut est **Ready**.

5.6.5.4.2. Reprise d'un nœud en mode maintenance à l'aide de la CLI

Vous pouvez reprendre un nœud en mode maintenance qui a été initié avec un CR **NodeMaintenance** en supprimant le CR **NodeMaintenance**.

Conditions préalables

- Install the OpenShift Container Platform CLI **oc**.
- Connectez-vous au cluster en tant qu'utilisateur disposant des privilèges **cluster-admin**.

Procédure

- Lorsque votre tâche de maintenance du nœud est terminée, supprimez le CR actif **NodeMaintenance**:

```
$ oc delete -f nodemaintenance-cr.yaml
```

Exemple de sortie

```
nodemaintenance.nodemaintenance.medik8s.io "maintenance-example" deleted
```

Vérification

1. Vérifiez l'état d'avancement de la tâche de maintenance en exécutant la commande suivante :

```
$ oc describe node <node-name>
```

où **<node-name>** est le nom de votre nœud ; par exemple, **node-1.example.com**

2. Vérifier la sortie de l'exemple :

```
Events:
  Type     Reason             Age                From     Message
  ----     ------             ----               ----     -------
  Normal   NodeSchedulable    2m                 kubelet  Node node-1.example.com status
is now: NodeSchedulable
```

5.6.5.5. Travailler avec des nœuds nus

Pour les clusters avec nœuds bare-metal, vous pouvez placer un nœud en mode maintenance et reprendre un nœud depuis le mode maintenance en utilisant le contrôle de la console web **Actions**.

NOTE

Les grappes dotées de nœuds nus peuvent également placer un nœud en mode de maintenance et reprendre un nœud en mode de maintenance à l'aide de la console web et de l'interface de ligne de commande (CLI), comme indiqué. Ces méthodes, qui font appel à la console web **Actions**, ne s'appliquent qu'aux clusters bare-metal.

5.6.5.5.1. Maintenance des nœuds nus

Lorsque vous déployez OpenShift Container Platform sur une infrastructure bare-metal, vous devez prendre en compte des considérations supplémentaires par rapport au déploiement sur une infrastructure cloud. Contrairement aux environnements cloud, où les nœuds de cluster sont considérés comme éphémères, le reprovisionnement d'un nœud bare-metal nécessite beaucoup plus de temps et d'efforts pour les tâches de maintenance.

Lorsqu'un nœud bare-metal tombe en panne à cause d'une erreur du noyau ou d'une défaillance matérielle de la carte NIC, les charges de travail sur le nœud en panne doivent être redémarrées sur un autre nœud de la grappe pendant que le nœud en panne est réparé ou remplacé. Le mode de maintenance des nœuds permet aux administrateurs de grappes d'éteindre les nœuds avec élégance, de déplacer les charges de travail vers d'autres parties de la grappe et de s'assurer que les charges de travail ne sont pas interrompues. Des informations détaillées sur la progression et l'état des nœuds sont fournies pendant la maintenance.

5.6.5.5.2. Passage d'un nœud bare-metal en mode maintenance

Mettre un nœud bare-metal en mode maintenance à l'aide du menu Options qui se trouve sur chaque nœud dans la liste **Compute → Nodes**, ou en utilisant la commande **Actions** de l'écran **Node Details**.

Procédure

1. Dans la perspective **Administrator** de la console web, cliquez sur **Compute → Nodes**.
2. Vous pouvez définir le nœud à gérer à partir de cet écran, ce qui facilite l'exécution d'actions sur plusieurs nœuds, ou à partir de l'écran **Node Details**, où vous pouvez afficher des détails complets sur le nœud sélectionné :
   - Cliquez sur le menu Options à l'extrémité du nœud et sélectionnez **Start Maintenance**.
   - Cliquez sur le nom du nœud pour ouvrir l'écran **Node Details** et cliquez sur **Actions → Start Maintenance**.
3. Cliquez sur **Start Maintenance** dans la fenêtre de confirmation.

Le nœud n'est plus planifiable. S'il avait des machines virtuelles avec la stratégie d'éviction **LiveMigration**, il les migrera en direct. Tous les autres pods et machines virtuelles sur le nœud sont supprimés et recréés sur un autre nœud.

Vérification

- Naviguez jusqu'à la page **Compute → Nodes** et vérifiez que le nœud correspondant a le statut **Under maintenance**.

5.6.5.5.3. Reprise d'un nœud bare-metal depuis le mode maintenance

Reprendre un nœud bare-metal depuis le mode maintenance à l'aide du menu Options  qui se trouve sur chaque nœud dans la liste **Compute → Nodes**, ou en utilisant la commande **Actions** de l'écran **Node Details**.

Procédure

1. Dans la perspective **Administrator** de la console web, cliquez sur **Compute → Nodes**.
2. Vous pouvez reprendre le nœud à partir de cet écran, ce qui facilite l'exécution d'actions sur plusieurs nœuds, ou à partir de l'écran **Node Details**, où vous pouvez afficher des détails complets sur le nœud sélectionné :
   - Cliquez sur le menu Options  à l'extrémité du nœud et sélectionnez **Stop Maintenance**.
   - Cliquez sur le nom du nœud pour ouvrir l'écran **Node Details** et cliquez sur **Actions → Stop Maintenance**.
3. Cliquez sur **Stop Maintenance** dans la fenêtre de confirmation.

Le nœud devient planifiable. Si des instances de machines virtuelles fonctionnaient sur le nœud avant la maintenance, elles ne migreront pas automatiquement vers ce nœud.

Vérification

- Naviguez jusqu'à la page **Compute → Nodes** et vérifiez que le nœud correspondant a le statut **Ready**.

5.6.5.6. Collecte de données sur l'opérateur de maintenance du nœud

Pour collecter des informations de débogage sur l'opérateur de maintenance de nœuds, utilisez l'outil **must-gather**. Pour plus d'informations sur l'image **must-gather** de l'opérateur de maintenance de nœuds, voir Collecte de données sur des fonctionnalités spécifiques .

5.6.5.7. Ressources supplémentaires

- Collecte de données sur votre cluster
- Comprendre comment évacuer les pods sur les nœuds
- Comprendre comment marquer les nœuds comme non planifiables ou planifiables

5.7. COMPRENDRE LE REDÉMARRAGE DES NŒUDS

Pour redémarrer un nœud sans provoquer de panne pour les applications fonctionnant sur la plate-

forme, il est important d'évacuer d'abord les pods. Pour les pods qui sont rendus hautement disponibles par le niveau de routage, il n'y a rien d'autre à faire. Pour les autres modules qui ont besoin de stockage, généralement des bases de données, il est essentiel de s'assurer qu'ils peuvent continuer à fonctionner même si l'un d'entre eux est temporairement hors ligne. Bien que la mise en œuvre de la résilience pour les pods avec état soit différente pour chaque application, dans tous les cas, il est important de configurer l'ordonnanceur pour utiliser l'anti-affinité des nœuds afin de s'assurer que les pods sont correctement répartis sur les nœuds disponibles.

Un autre défi consiste à gérer les nœuds qui gèrent une infrastructure critique telle que le routeur ou le registre. Le même processus d'évacuation des nœuds s'applique, bien qu'il soit important de comprendre certains cas limites.

5.7.1. À propos du redémarrage des nœuds exécutant une infrastructure critique

Lors du redémarrage des nœuds qui hébergent des composants d'infrastructure critiques d'OpenShift Container Platform, tels que les pods de routeur, les pods de registre et les pods de surveillance, assurez-vous qu'il y a au moins trois nœuds disponibles pour exécuter ces composants.

Le scénario suivant montre comment des interruptions de service peuvent se produire avec des applications exécutées sur OpenShift Container Platform lorsque seuls deux nœuds sont disponibles :

- Le nœud A est déclaré inséparable et toutes les nacelles sont évacuées.
- Le pod de registre fonctionnant sur ce nœud est maintenant redéployé sur le nœud B. Le nœud B exécute maintenant les deux pods de registre.
- Le nœud B est maintenant considéré comme non planifiable et est évacué.
- Le service exposant les deux points d'extrémité de pods sur le nœud B perd tous les points d'extrémité, pendant une brève période, jusqu'à ce qu'ils soient redéployés sur le nœud A.

Lorsque trois nœuds sont utilisés pour les composants d'infrastructure, ce processus n'entraîne pas d'interruption de service. Toutefois, en raison de la planification des pods, le dernier nœud évacué et remis en rotation n'a pas de pod de registre. L'un des autres nœuds dispose de deux modules de registre. Pour planifier le troisième module de registre sur le dernier nœud, utilisez l'anti-affinité de module pour empêcher l'ordonnanceur de placer deux modules de registre sur le même nœud.

Informations complémentaires

- Pour plus d'informations sur l'anti-affinité des pods, voir Placement des pods par rapport à d'autres pods à l'aide de règles d'affinité et d'anti-affinité.

5.7.2. Redémarrage d'un nœud à l'aide d'un pod anti-affinité

L'anti-affinité des pods est légèrement différente de l'anti-affinité des nœuds. L'anti-affinité de nœud peut être violée s'il n'y a pas d'autres emplacements appropriés pour déployer un pod. L'anti-affinité des pods peut être définie comme requise ou préférée.

Ainsi, si seuls deux nœuds d'infrastructure sont disponibles et que l'un d'entre eux est redémarré, le pod de registre d'images de conteneurs ne peut pas s'exécuter sur l'autre nœud. **oc get pods** signale le pod comme étant non prêt jusqu'à ce qu'un nœud approprié soit disponible. Une fois qu'un nœud est disponible et que tous les pods sont de nouveau prêts, le nœud suivant peut être redémarré.

Procédure

Pour redémarrer un nœud en utilisant l'anti-affinité des pods :

1. Modifiez la spécification du nœud pour configurer l'anti-affinité du pod :

```
apiVersion: v1
kind: Pod
metadata:
  name: with-pod-antiaffinity
spec:
  affinity:
    podAntiAffinity: 1
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 100 3
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: registry 4
              operator: In 5
              values:
              - default
          topologyKey: kubernetes.io/hostname
```

1 Stanza pour configurer l'anti-affinité du pod.

2 Définit une règle préférentielle.

3 Spécifie un poids pour une règle préférentielle. Le nœud ayant le poids le plus élevé est privilégié.

4 Description de l'étiquette du pod qui détermine quand la règle anti-affinité s'applique. Spécifiez une clé et une valeur pour l'étiquette.

5 L'opérateur représente la relation entre l'étiquette de la capsule existante et l'ensemble des valeurs des paramètres **matchExpression** dans la spécification de la nouvelle capsule. Il peut s'agir de **In**, **NotIn**, **Exists** ou **DoesNotExist**.

Cet exemple suppose que le pod du registre d'images de conteneurs a une étiquette de **registry=default**. L'anti-affinité de pod peut utiliser n'importe quelle expression de correspondance de Kubernetes.

2. Activer le prédicat de l'ordonnanceur **MatchInterPodAffinity** dans le fichier de stratégie d'ordonnancement.

3. Effectuer un redémarrage gracieux du nœud.

5.7.3. Comprendre comment redémarrer les nœuds utilisant des routeurs

Dans la plupart des cas, un pod exécutant un routeur OpenShift Container Platform expose un port hôte.

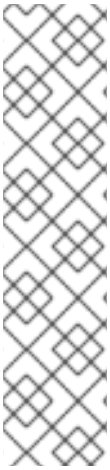
Le prédicat de l'ordonnanceur **PodFitsPorts** garantit qu'aucun pod de routeur utilisant le même port ne peut s'exécuter sur le même nœud, et que l'anti-affinité des pods est réalisée. Si les routeurs s'appuient sur le basculement IP pour la haute disponibilité, il n'y a rien d'autre à faire.

Pour les router pods qui dépendent d'un service externe tel que AWS Elastic Load Balancing pour la haute disponibilité, il est de la responsabilité de ce service de réagir aux redémarrages des router pods.

Dans de rares cas, un router pod peut ne pas avoir de port hôte configuré. Dans ce cas, il est important de suivre la procédure de redémarrage recommandée pour les nœuds d'infrastructure.

5.7.4. Redémarrer un nœud avec élégance

Avant de redémarrer un nœud, il est recommandé de sauvegarder les données etcd afin d'éviter toute perte de données sur le nœud.



NOTE

Pour les clusters OpenShift à un seul nœud qui nécessitent que les utilisateurs exécutent la commande **oc login** plutôt que d'avoir les certificats dans le fichier **kubeconfig** pour gérer le cluster, les commandes **oc adm** peuvent ne pas être disponibles après le cordon et la vidange du nœud. Cela est dû au fait que le pod **openshift-oauth-apiserver** n'est pas en cours d'exécution en raison du cordon. Vous pouvez utiliser SSH pour accéder aux nœuds comme indiqué dans la procédure suivante.

Dans un cluster OpenShift à un seul nœud, les pods ne peuvent pas être reprogrammés lors du cordonage et de la vidange. Cependant, cela donne aux pods, en particulier à vos pods de charge de travail, le temps de s'arrêter correctement et de libérer les ressources associées.

Procédure

Pour effectuer un redémarrage gracieux d'un nœud :

1. Marquer le nœud comme non ordonnançable :

```
oc adm cordon <node1> $ oc adm cordon <node1>
```

2. Drainer le nœud pour supprimer tous les pods en cours d'exécution :

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force
```

Il se peut que vous receviez des erreurs indiquant que les pods associés à des budgets de perturbation de pods personnalisés (PDB) ne peuvent pas être expulsés.

Exemple d'erreur

```
error when evicting pods/"rails-postgresql-example-1-72v2w" -n "rails" (will retry after 5s):
Cannot evict pod as it would violate the pod's disruption budget.
```

Dans ce cas, exécutez à nouveau la commande drain, en ajoutant le drapeau **disable-eviction**, ce qui permet de contourner les contrôles PDB :

```
$ oc adm drain <node1> --ignore-daemonsets --delete-emptydir-data --force --disable-
eviction
```

3. Accéder au nœud en mode débogage :

```
$ oc debug node/<node1>
```

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

```
$ chroot /host
```

5. Redémarrer le nœud :

```
$ systemctl reboot
```

En un instant, le nœud entre dans l'état **NotReady**.

NOTE

Avec certains clusters OpenShift à un seul nœud, les commandes **oc** peuvent ne pas être disponibles après avoir cordonné et drainé le nœud parce que le pod **openshift-oauth-apiserver** n'est pas en cours d'exécution. Vous pouvez utiliser SSH pour vous connecter au nœud et effectuer le redémarrage.

```
$ ssh core@<master-node>.<cluster_name>.<base_domain>
```

```
$ sudo systemctl reboot
```

6. Une fois le redémarrage terminé, marquez le nœud comme planifiable en exécutant la commande suivante :

```
oc adm uncordon <node1> $ oc adm uncordon <node1>
```

NOTE

Avec certains clusters OpenShift à nœud unique, les commandes **oc** peuvent ne pas être disponibles après avoir cordonné et drainé le nœud parce que le pod **openshift-oauth-apiserver** n'est pas en cours d'exécution. Vous pouvez utiliser SSH pour vous connecter au nœud et le déconnecter.

```
$ ssh core@<target_node>
```

```
$ sudo oc adm uncordon <node> --kubeconfig /etc/kubernetes/static-pod-resources/kube-apiserver-certs/secrets/node-kubeconfigs/localhost.kubeconfig
```

7. Vérifiez que le nœud est prêt :

```
oc get node <node1> $ oc get node <node1>
```

Exemple de sortie

```
NAME    STATUS  ROLES    AGE     VERSION
<node1> Ready   worker   6d22h   v1.18.3+b0068a8
```

Informations complémentaires

Pour plus d'informations sur la sauvegarde des données etcd, voir Sauvegarde des données etcd.

5.8. LIBÉRER LES RESSOURCES DES NŒUDS À L'AIDE DU RAMASSAGE DES ORDURES

En tant qu'administrateur, vous pouvez utiliser OpenShift Container Platform pour vous assurer que vos nœuds fonctionnent efficacement en libérant des ressources grâce au garbage collection.

Le nœud OpenShift Container Platform effectue deux types de collecte de déchets :

- Collecte des déchets des conteneurs : Supprime les conteneurs terminés.
- Ramassage des images : Supprime les images non référencées par les pods en cours d'exécution.

5.8.1. Comprendre comment les conteneurs terminés sont supprimés par le ramasse-miettes (garbage collection)

Le ramassage des ordures du conteneur peut être effectué à l'aide de seuils d'éviction.

Lorsque des seuils d'éviction sont définis pour le ramassage des ordures, le nœud tente de conserver tout conteneur pour tout module accessible à partir de l'API. Si le module a été supprimé, les conteneurs le seront également. Les conteneurs sont conservés tant que le module n'est pas supprimé et que le seuil d'éviction n'est pas atteint. Si le nœud est soumis à une pression de disque, il supprimera les conteneurs et leurs journaux ne seront plus accessibles à l'aide de **oc logs**.

- **eviction-soft** - Un seuil d'expulsion souple associe un seuil d'expulsion à un délai de grâce spécifié par l'administrateur.
- **eviction-hard** - Un seuil d'éviction dur n'a pas de période de grâce, et s'il est observé, OpenShift Container Platform prend des mesures immédiates.

Le tableau suivant énumère les seuils d'éviction :

Tableau 5.2. Variables pour configurer le ramassage des ordures du conteneur

État du nœud	Signal d'expulsion	Description
MémoirePression	**memory.available**	Mémoire disponible sur le nœud.
Pression du disque	• **nodefs.available** • **nodefs.inodesFree** • **imagefs.available** • **imagefs.inodesFree**	L'espace disque ou les inodes disponibles sur le système de fichiers racine du nœud, **nodefs**, ou sur le système de fichiers image, **imagefs**.

NOTE

Pour **evictionHard**, vous devez spécifier tous ces paramètres. Si vous ne les spécifiez pas tous, seuls les paramètres spécifiés seront appliqués et le ramassage des ordures ne fonctionnera pas correctement.

Si un nœud oscille au-dessus et au-dessous d'un seuil d'éviction souple, mais sans dépasser le délai de grâce qui lui est associé, le nœud correspondant oscille constamment entre **true** et **false**. En conséquence, l'ordonnanceur pourrait prendre de mauvaises décisions en matière d'ordonnancement.

Pour se protéger contre cette oscillation, utilisez le drapeau **eviction-pressure-transition-period** pour contrôler la durée pendant laquelle OpenShift Container Platform doit attendre avant de sortir d'une condition de pression. OpenShift Container Platform ne définira pas un seuil d'éviction comme étant atteint pour la condition de pression spécifiée pendant la période spécifiée avant de basculer la condition sur false.

5.8.2. Comprendre comment les images sont supprimées par le ramassage des ordures

La collecte d'images s'appuie sur l'utilisation du disque telle qu'elle est rapportée par **cAdvisor** sur le nœud pour décider quelles images doivent être supprimées du nœud.

La politique de collecte des images est basée sur deux conditions :

- Le pourcentage d'utilisation du disque (exprimé sous forme d'un nombre entier) qui déclenche le ramassage des images. La valeur par défaut est **85**.
- Pourcentage d'utilisation du disque (exprimé sous forme d'un nombre entier) que le ramasse-miettes tente de libérer. La valeur par défaut est **80**.

Pour le ramassage des images, vous pouvez modifier l'une des variables suivantes à l'aide d'une ressource personnalisée.

Tableau 5.3. Variables pour configurer le ramassage des images

Paramètres	Description
imageMinimumGCAge	L'âge minimum d'une image inutilisée avant qu'elle ne soit supprimée par le ramasse-miettes. La valeur par défaut est **2m**.
imageGCHighThresholdPercent	Le pourcentage d'utilisation du disque, exprimé sous la forme d'un entier, qui déclenche le ramassage des images. La valeur par défaut est **85**.
imageGCLowThresholdPercent	Le pourcentage d'utilisation du disque, exprimé sous forme d'un nombre entier, que le ramasse-miettes tente de libérer. La valeur par défaut est **80**.

Deux listes d'images sont récupérées à chaque passage de l'éboueur :

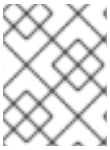
1. Liste des images en cours d'exécution dans au moins un module.
2. Liste des images disponibles sur un hôte.

Au fur et à mesure que de nouveaux conteneurs sont lancés, de nouvelles images apparaissent. Toutes les images sont marquées d'un horodatage. Si l'image est en cours d'exécution (première liste ci-dessus) ou nouvellement détectée (deuxième liste ci-dessus), elle est marquée avec l'heure actuelle. Les autres images sont déjà marquées lors des tours précédents. Toutes les images sont ensuite triées en fonction de l'horodatage.

Une fois la collecte commencée, les images les plus anciennes sont supprimées en premier jusqu'à ce que le critère d'arrêt soit rempli.

5.8.3. Configuration du ramassage des ordures pour les conteneurs et les images

En tant qu'administrateur, vous pouvez configurer la façon dont OpenShift Container Platform effectue la collecte des ordures en créant un objet **kubeletConfig** pour chaque pool de configuration de machine.



NOTE

OpenShift Container Platform ne prend en charge qu'un seul objet **kubeletConfig** pour chaque pool de configuration de machine.

Vous pouvez configurer une combinaison des éléments suivants :

- Expulsion douce pour les conteneurs
- Expulsion dure pour les conteneurs
- Expulsion pour des images

Conditions préalables

1. Obtenez l'étiquette associée au CRD statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante :

```
oc edit machineconfigpool <name> $ oc edit machineconfigpool <name>
```

Par exemple :

```
$ oc edit machineconfigpool worker
```

Exemple de sortie

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```

1 L'étiquette apparaît sous Étiquettes.

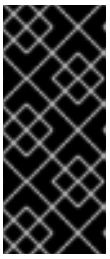
ASTUCE

Si l'étiquette n'est pas présente, ajoutez une paire clé/valeur comme par exemple :

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration.



IMPORTANT

S'il n'y a qu'un seul système de fichiers, ou si **/var/lib/kubelet** et **/var/lib/containers/** se trouvent dans le même système de fichiers, ce sont les paramètres ayant les valeurs les plus élevées qui déclenchent les expulsions, car ils sont respectés en premier. C'est le système de fichiers qui déclenche l'expulsion.

Exemple de configuration pour un conteneur de collecte de déchets CR :

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: worker-kubeconfig (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    evictionSoft: (3)
      memory.available: "500Mi" (4)
      nodefs.available: "10%"
      nodefs.inodesFree: "5%"
      imagefs.available: "15%"
      imagefs.inodesFree: "10%"
    evictionSoftGracePeriod: (5)
      memory.available: "1m30s"
      nodefs.available: "1m30s"
      nodefs.inodesFree: "1m30s"
      imagefs.available: "1m30s"
      imagefs.inodesFree: "1m30s"
    evictionHard: (6)
      memory.available: "200Mi"
      nodefs.available: "5%"
      nodefs.inodesFree: "4%"
      imagefs.available: "10%"
      imagefs.inodesFree: "5%"
    evictionPressureTransitionPeriod: 0s (7)
    imageMinimumGCAge: 5m (8)
    imageGCHighThresholdPercent: 80 (9)
    imageGCLowThresholdPercent: 75 (10)
```

(1) Nom de l'objet.

(2) Spécifiez l'étiquette du pool de configuration de la machine.

(3) Type d'expulsion : **evictionSoft** ou **evictionHard**.

(4) Seuils d'expulsion basés sur un signal de déclenchement d'expulsion spécifique.

(5) Délais de grâce pour l'expulsion douce. Ce paramètre ne s'applique pas à **eviction-hard**.

(6) Seuils d'éviction basés sur un signal de déclenchement d'éviction spécifique. Pour **evictionHard**, vous devez spécifier tous ces paramètres. Si vous ne les spécifiez pas tous, seuls les paramètres spécifiés seront appliqués et le ramassage des ordures ne

seuls les paramètres spécifiés seront appliqués et le ramassage des ordures ne fonctionnera pas correctement.

7 Durée d'attente avant de sortir d'une condition de pression d'expulsion.

8 L'âge minimum d'une image inutilisée avant qu'elle ne soit supprimée par le ramasse-miettes.

9 Pourcentage d'utilisation du disque (exprimé sous forme d'un nombre entier) qui déclenche le ramassage des images.

10 Pourcentage de l'utilisation du disque (exprimé sous forme d'un entier) que le système de collecte des images tente de libérer.

2. Exécutez la commande suivante pour créer le CR :

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

Par exemple :

```
$ oc create -f gc-container.yaml
```

Exemple de sortie

```
kubeletconfig.machineconfiguration.openshift.io/gc-container created
```

Vérification

1. Vérifiez que le ramassage des ordures est actif en entrant la commande suivante. Le pool de configuration de la machine que vous avez spécifié dans la ressource personnalisée apparaît avec **UPDATING** comme 'true' jusqu'à ce que le changement soit complètement implémenté :

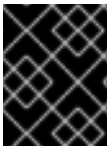
```
$ oc get machineconfigpool
```

Exemple de sortie

```
NAME     CONFIG                                UPDATED   UPDATING
master   rendered-master-546383f80705bd5aeaba93   True      False
worker   rendered-worker-b4c51bb33ccaae6fc4a6a5   False     True
```

5.9. ALLOCATION DE RESSOURCES POUR LES NŒUDS D'UN CLUSTER OPENSHIFT CONTAINER PLATFORM

Pour assurer une planification plus fiable et minimiser le surengagement des ressources du nœud, réservez une partie des ressources CPU et mémoire aux composants sous-jacents du nœud, tels que **kubelet** et **kube-proxy**, et aux autres composants du système, tels que **sshd** et **NetworkManager**. En spécifiant les ressources à réserver, vous fournissez au planificateur davantage d'informations sur les ressources de CPU et de mémoire restantes qu'un nœud peut utiliser pour les pods. Vous pouvez permettre à OpenShift Container Platform de déterminer automatiquement les ressources optimales de mémoire et de CPU **system-reserved** pour vos nœuds ou vous pouvez déterminer et définir manuellement les meilleures ressources pour vos nœuds.



IMPORTANT

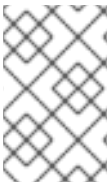
Pour définir manuellement les valeurs des ressources, vous devez utiliser un CR de configuration de kubelet. Vous ne pouvez pas utiliser un CR de configuration de machine.

5.9.1. Comprendre comment allouer des ressources aux nœuds

Les ressources de CPU et de mémoire réservées aux composants de nœuds dans OpenShift Container Platform sont basées sur deux paramètres de nœuds :

Paramètres	Description
kube-reserved	Ce paramètre n'est pas utilisé avec OpenShift Container Platform. Ajoutez les ressources CPU et mémoire que vous avez prévu de réserver au paramètre **system-reserved**.
system-reserved	Ce paramètre identifie les ressources à réserver pour les composants du nœud et les composants du système, tels que CRI-O et Kubelet. Les paramètres par défaut dépendent des versions d'OpenShift Container Platform et de Machine Config Operator. Confirmez le paramètre par défaut **systemReserved** sur le référentiel **machine-config-operator**.

Si un indicateur n'est pas défini, les valeurs par défaut sont utilisées. Si aucun indicateur n'est défini, la ressource allouée correspond à la capacité du nœud telle qu'elle était avant l'introduction des ressources allouables.



NOTE

Les unités centrales spécifiquement réservées à l'aide du paramètre **reservedSystemCPUs** ne sont pas disponibles pour une allocation à l'aide des paramètres **kube-reserved** ou **system-reserved**.

5.9.1.1. Comment OpenShift Container Platform calcule les ressources allouées

La quantité allouée d'une ressource est calculée sur la base de la formule suivante :

```
[Allocatable] = [Node Capacity] - [system-reserved] - [Hard-Eviction-Thresholds]
```

NOTE

L'exclusion de **Hard-Eviction-Thresholds** de **Allocatable** améliore la fiabilité du système car la valeur de **Allocatable** est appliquée aux pods au niveau du nœud.

Si **Allocatable** est négatif, il est fixé à **0**.

Chaque nœud indique les ressources système utilisées par l'exécution du conteneur et le kubelet. Pour simplifier la configuration du paramètre **system-reserved**, affichez l'utilisation des ressources pour le nœud en utilisant l'API de résumé du nœud. Le résumé du nœud est disponible à l'adresse **/api/v1/nodes/<node>/proxy/stats/summary**.

5.9.1.2. Comment les nœuds appliquent les contraintes de ressources

Le nœud peut limiter la quantité totale de ressources que les modules peuvent consommer en fonction de la valeur d'allocation configurée. Cette fonction améliore considérablement la fiabilité du nœud en empêchant les modules d'utiliser les ressources de CPU et de mémoire dont ont besoin les services système tels que le moteur d'exécution du conteneur et l'agent du nœud. Pour améliorer la fiabilité du nœud, les administrateurs doivent réserver des ressources en fonction d'un objectif d'utilisation des ressources.

Le nœud impose des contraintes de ressources en utilisant une nouvelle hiérarchie de cgroupes qui assure la qualité du service. Tous les pods sont lancés dans une hiérarchie de cgroup dédiée, séparée des démons du système.

Les administrateurs doivent traiter les démons système de la même manière que les pods qui ont une qualité de service garantie. Les démons système peuvent éclater au sein de leurs groupes de contrôle et ce comportement doit être géré dans le cadre des déploiements de clusters. Réservez des ressources de CPU et de mémoire aux démons système en spécifiant la quantité de ressources de CPU et de mémoire dans **system-reserved**.

L'application des limites de **system-reserved** peut empêcher les services système critiques de recevoir des ressources de CPU et de mémoire. Par conséquent, un service système critique peut être interrompu par le "out-of-memory killer". Il est recommandé d'appliquer **system-reserved** uniquement si vous avez profilé les nœuds de manière exhaustive afin de déterminer des estimations précises et si vous êtes certain que les services système critiques peuvent se rétablir si l'un des processus de ce groupe est interrompu par le tueur de mémoire.

5.9.1.3. Comprendre les seuils d'expulsion

Si un nœud subit une pression de mémoire, cela peut avoir un impact sur l'ensemble du nœud et sur tous les pods s'exécutant sur le nœud. Par exemple, un démon système qui utilise plus que la quantité de mémoire qui lui est réservée peut déclencher un événement de sortie de mémoire. Pour éviter ou réduire la probabilité d'événements de sortie de mémoire du système, le nœud fournit une gestion des ressources manquantes.

Vous pouvez réserver de la mémoire en utilisant l'option **--eviction-hard**. Le nœud tente d'expulser les modules chaque fois que la disponibilité de la mémoire sur le nœud tombe en dessous de la valeur absolue ou du pourcentage. Si les démons système n'existent pas sur un nœud, les modules sont limités à la mémoire **capacity - eviction-hard**. Pour cette raison, les ressources mises de côté en tant que tampon pour l'expulsion avant d'atteindre les conditions d'épuisement de la mémoire ne sont pas disponibles pour les modules.

L'exemple suivant illustre l'impact du nœud allouable pour la mémoire :

- La capacité des nœuds est de **32Gi**
- --réservé au système est **3Gi**
- --eviction-hard est fixé à **100Mi**.

Pour ce nœud, la valeur allouable effective du nœud est **28.9Gi**. Si le nœud et les composants du système utilisent toutes leurs réservations, la mémoire disponible pour les pods est **28.9Gi**, et le kubelet évince les pods lorsqu'elle dépasse ce seuil.

Si vous imposez l'allocation de nœuds, **28.9Gi**, avec des cgroups de premier niveau, les pods ne peuvent jamais dépasser **28.9Gi**. Les expulsions ne sont pas effectuées à moins que les démons du système ne consomment plus de **3.1Gi** de mémoire.

Si les démons du système n'utilisent pas toute leur réservation, dans l'exemple ci-dessus, les pods devraient faire face à des destructions OOM memcg de leur cgroup limitant avant que les évictions de nœuds ne démarrent. Pour mieux appliquer la QoS dans cette situation, le nœud applique les seuils d'éviction durs au cgroup de niveau supérieur pour tous les pods devant être **Node Allocatable Eviction Hard Thresholds**.

Si les démons du système n'utilisent pas toute leur réserve, le nœud expulse les modules dès qu'ils consomment plus de **28.9Gi** de mémoire. Si l'éviction n'a pas lieu à temps, un pod sera tué par OOM si les pods consomment **29Gi** de mémoire.

5.9.1.4. Comment l'ordonnanceur détermine la disponibilité des ressources

L'ordonnanceur utilise la valeur de **node.Status.Allocatable** au lieu de **node.Status.Capacity** pour décider si un nœud sera candidat à l'ordonnancement de pods.

Par défaut, le nœud indique que la capacité de sa machine est entièrement planifiable par le cluster.

5.9.2. Attribution automatique de ressources aux nœuds

OpenShift Container Platform peut déterminer automatiquement les ressources optimales **system-reserved** CPU et mémoire pour les nœuds associés à un pool de configuration machine spécifique et mettre à jour les nœuds avec ces valeurs lorsque les nœuds démarrent. Par défaut, le CPU de **system-reserved** est **500m** et la mémoire de **system-reserved** est **1Gi**.

Pour déterminer et allouer automatiquement les ressources **system-reserved** sur les nœuds, créez une ressource personnalisée (CR) **KubeletConfig** pour définir le paramètre **autoSizingReserved: true**. Un script sur chaque nœud calcule les valeurs optimales pour les ressources réservées respectives sur la base de la capacité de CPU et de mémoire installée sur chaque nœud. Le script tient compte du fait qu'une augmentation de la capacité nécessite une augmentation correspondante des ressources réservées.

La détermination automatique des paramètres **system-reserved** optimaux garantit l'efficacité de votre cluster et prévient les défaillances de nœuds dues à la pénurie de ressources des composants du système, tels que CRI-O et kubelet, sans que vous ayez besoin de calculer et de mettre à jour manuellement les valeurs.

Cette fonction est désactivée par défaut.

Conditions préalables

1. Obtenez l'étiquette associée à l'objet statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante :

   ```
   oc edit machineconfigpool <name> $ oc edit machineconfigpool <name>
   ```

   Par exemple :

   ```
   $ oc edit machineconfigpool worker
   ```

   Exemple de sortie

   ```
   apiVersion: machineconfiguration.openshift.io/v1
   kind: MachineConfigPool
   metadata:
   ```

```
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" (1)
  name: worker
 ...
```

(1) L'étiquette apparaît sous **Labels**.

ASTUCE

Si l'étiquette n'est pas présente, ajoutez une paire clé/valeur comme par exemple :

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration :

   Exemple de configuration pour un CR d'allocation de ressources

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: dynamic-node (1)
spec:
  autoSizingReserved: true (2)
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (3)
```

   (1) Attribuer un nom au CR.

   (2) Ajoutez le paramètre **autoSizingReserved** défini sur **true** pour permettre à OpenShift Container Platform de déterminer et d'allouer automatiquement les ressources **system-reserved** sur les nœuds associés à l'étiquette spécifiée. Pour désactiver l'allocation automatique sur ces nœuds, définissez ce paramètre à **false**.

   (3) Spécifiez l'étiquette du pool de configuration de la machine.

   L'exemple précédent active l'allocation automatique des ressources sur tous les nœuds de travail. OpenShift Container Platform draine les nœuds, applique la configuration kubelet et redémarre les nœuds.

2. Créez le CR en entrant la commande suivante :

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

Vérification

1. Connectez-vous à un nœud que vous avez configuré en entrant la commande suivante :

```
oc debug node/<node_name>
```

2. Définir **/host** comme répertoire racine dans l'interpréteur de commandes de débogage :

```
# chroot /host
```

3. Consulter le fichier **/etc/node-sizing.env**:

Exemple de sortie

```
SYSTEM_RESERVED_MEMORY=3Gi
SYSTEM_RESERVED_CPU=0.08
```

Le kubelet utilise les valeurs de **system-reserved** dans le fichier **/etc/node-sizing.env**. Dans l'exemple précédent, les nœuds de travail se voient attribuer **0.08** CPU et 3 Gi de mémoire. L'apparition des valeurs optimales peut prendre plusieurs minutes.

5.9.3. Attribution manuelle de ressources aux nœuds

OpenShift Container Platform prend en charge les types de ressources CPU et mémoire pour l'allocation. Le type de ressource **ephemeral-resource** est également pris en charge. Pour le type **cpu**, vous spécifiez la quantité de ressources en unités de cœurs, telles que **200m**, **0.5**, ou **1**. Pour **memory** et **ephemeral-storage**, vous spécifiez la quantité de ressources en unités d'octets, comme **200Ki**, **50Mi** ou **5Gi**. Par défaut, l'unité centrale **system-reserved** est **500m** et la mémoire **system-reserved** est **1Gi**.

En tant qu'administrateur, vous pouvez définir ces valeurs en utilisant une ressource personnalisée (CR) de configuration kubelet par le biais d'un ensemble de paires **<resource_type>=<resource_quantity>** (par exemple, **cpu=200m,memory=512Mi**).

IMPORTANT

Vous devez utiliser un CR de configuration de kubelet pour définir manuellement les valeurs des ressources. Vous ne pouvez pas utiliser un CR de configuration de machine.

Pour plus de détails sur les valeurs recommandées pour **system-reserved**, voir les valeurs recommandées pour les réserves du système.

Conditions préalables

1. Obtenez l'étiquette associée au CRD statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante :

```
oc edit machineconfigpool <name> $ oc edit machineconfigpool <name>
```

Par exemple :

```
$ oc edit machineconfigpool worker
```

Exemple de sortie

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
```

```
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" (1)
  name: worker
```

(1) L'étiquette apparaît sous Étiquettes.

ASTUCE

Si l'étiquette n'est pas présente, ajoutez une paire clé/valeur comme par exemple :

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration.

   Exemple de configuration pour un CR d'allocation de ressources

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-allocatable (1)
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (2)
  kubeletConfig:
    systemReserved: (3)
      cpu: 1000m
      memory: 1Gi
```

   (1) Attribuer un nom au CR.

   (2) Spécifiez l'étiquette du pool de configuration de la machine.

   (3) Spécifiez les ressources à réserver pour les composants du nœud et du système.

2. Exécutez la commande suivante pour créer le CR :

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

5.10. ATTRIBUTION D'UNITÉS CENTRALES SPÉCIFIQUES AUX NŒUDS D'UN CLUSTER

Lorsque vous utilisez la stratégie statique du gestionnaire de CPU , vous pouvez réserver des CPU spécifiques à l'usage de nœuds spécifiques dans votre cluster. Par exemple, sur un système doté de 24 CPU, vous pouvez réserver les CPU numérotés de 0 à 3 pour le plan de contrôle, ce qui permet aux

nœuds de calcul d'utiliser les CPU 4 à 23.

5.10.1. Réserver des CPU pour les nœuds

Pour définir explicitement une liste d'unités centrales réservées à des nœuds spécifiques, créez une ressource personnalisée (CR) **KubeletConfig** pour définir le paramètre **reservedSystemCPUs**. Cette liste remplace les unités centrales qui peuvent être réservées à l'aide des paramètres **systemReserved** et **kubeReserved**.

Procédure

1. Obtenez l'étiquette associée au pool de configuration de la machine (MCP) pour le type de nœud que vous souhaitez configurer :

```
oc describe machineconfigpool <name> $ oc describe machineconfigpool <name>
```

Par exemple :

```
$ oc describe machineconfigpool worker
```

Exemple de sortie

```
Name:         worker
Namespace:
Labels:       machineconfiguration.openshift.io/mco-built-in=
              pools.operator.machineconfiguration.openshift.io/worker= 1
Annotations:  <none>
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfigPool
...
```

1 Obtenir le label MCP.

2. Créer un fichier YAML pour le CR **KubeletConfig**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-reserved-cpus 1
spec:
  kubeletConfig:
    reservedSystemCPUs: "0,1,2,3" 2
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 3
```

1 Spécifiez un nom pour le CR.

2 Spécifiez les ID de cœur des CPU que vous souhaitez réserver pour les nœuds associés au MCP.

3 Spécifier l'étiquette à partir du MCP.

3. Créer l'objet CR :

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

Ressources supplémentaires

- Pour plus d'informations sur les paramètres **systemReserved** et **kubeReserved**, voir Allocation de ressources pour les nœuds dans un cluster OpenShift Container Platform.

5.11. ACTIVATION DES PROFILS DE SÉCURITÉ TLS POUR LE KUBELET

Vous pouvez utiliser un profil de sécurité TLS (Transport Layer Security) pour définir les codes TLS requis par le kubelet lorsqu'il agit en tant que serveur HTTP. Le kubelet utilise son serveur HTTP/GRPC pour communiquer avec le serveur API Kubernetes, qui envoie des commandes aux pods, collecte des journaux et exécute des commandes exec sur les pods par l'intermédiaire du kubelet.

Un profil de sécurité TLS définit les algorithmes de chiffrement TLS que le serveur API Kubernetes doit utiliser lors de la connexion avec le kubelet pour protéger la communication entre le kubelet et le serveur API Kubernetes.

NOTE

Par défaut, lorsque le kubelet agit en tant que client avec le serveur API de Kubernetes, il négocie automatiquement les paramètres TLS avec le serveur API.

5.11.1. Comprendre les profils de sécurité TLS

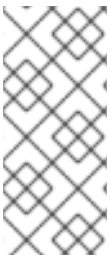
Vous pouvez utiliser un profil de sécurité TLS (Transport Layer Security) pour définir les algorithmes TLS requis par les différents composants d'OpenShift Container Platform. Les profils de sécurité TLS d'OpenShift Container Platform sont basés sur les configurations recommandées par Mozilla .

Vous pouvez spécifier l'un des profils de sécurité TLS suivants pour chaque composant :

Tableau 5.4. Profils de sécurité TLS

Profile	Description
Old	Ce profil est destiné à être utilisé avec des clients ou des bibliothèques anciens. Il est basé sur l'ancienne configuration recommandée pour la rétrocompatibilité. Le profil **Old** nécessite une version TLS minimale de 1.0. **NOTE** Pour le contrôleur d'entrée, la version minimale de TLS passe de 1.0 à 1.1.

Profile	Description
Intermediate	Ce profil est la configuration recommandée pour la majorité des clients. Il s'agit du profil de sécurité TLS par défaut pour le contrôleur d'entrée, le kubelet et le plan de contrôle. Le profil est basé sur la configuration recommandée pour la compatibilité intermédiaire. Le profil **Intermediate** nécessite une version TLS minimale de 1.2.
Modern	Ce profil est destiné à être utilisé avec des clients modernes qui n'ont pas besoin de rétrocompatibilité. Ce profil est basé sur la configuration recommandée pour la compatibilité moderne. Le profil **Modern** nécessite une version TLS minimale de 1.3.
Custom	Ce profil permet de définir la version de TLS et les algorithmes de chiffrement à utiliser. **AVERTISSEMENT** Soyez prudent lorsque vous utilisez un profil **Custom**, car des configurations non valides peuvent causer des problèmes.



NOTE

Lorsque l'on utilise l'un des types de profil prédéfinis, la configuration effective du profil est susceptible d'être modifiée entre les versions. Par exemple, si l'on spécifie l'utilisation du profil intermédiaire déployé dans la version X.Y.Z, une mise à niveau vers la version X.Y.Z 1 peut entraîner l'application d'une nouvelle configuration de profil, ce qui se traduit par un déploiement.

5.11.2. Configuration du profil de sécurité TLS pour le kubelet

Pour configurer un profil de sécurité TLS pour le kubelet lorsqu'il agit en tant que serveur HTTP, créez une ressource personnalisée (CR) **KubeletConfig** pour spécifier un profil de sécurité TLS prédéfini ou personnalisé pour des nœuds spécifiques. Si aucun profil de sécurité TLS n'est configuré, le profil de sécurité TLS par défaut est **Intermediate**.

Exemple de CR KubeletConfig qui configure le profil de sécurité TLS Old sur les nœuds de travail

```
apiVersion: config.openshift.io/v1
kind: KubeletConfig
 ...
spec:
  tlsSecurityProfile:
    old: {}
```

```
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
```

Vous pouvez voir les codes et la version TLS minimale du profil de sécurité TLS configuré dans le fichier **kubelet.conf** sur un nœud configuré.

Conditions préalables

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

Procédure

1. Créez un CR **KubeletConfig** pour configurer le profil de sécurité TLS :

Exemple de CR KubeletConfig pour un profil Custom

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" (4)
```

(1) Spécifiez le type de profil de sécurité TLS (**Old**, **Intermediate**, ou **Custom**). La valeur par défaut est **Intermediate**.

(2) Spécifiez le champ approprié pour le type sélectionné :

- **old: {}**
- **intermediate: {}**
- **custom:**

(3) Pour le type **custom**, spécifiez une liste de chiffrements TLS et la version TLS minimale acceptée.

(4) Facultatif : Indiquez l'étiquette du pool de configuration de la machine pour les nœuds auxquels vous souhaitez appliquer le profil de sécurité TLS.

2. Créer l'objet **KubeletConfig**:

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

En fonction du nombre de nœuds de travail dans la grappe, attendez que les nœuds configurés soient redémarrés un par un.

Vérification

Pour vérifier que le profil est défini, effectuez les étapes suivantes une fois que les nœuds sont dans l'état **Ready**:

1. Démarrer une session de débogage pour un nœud configuré :

```
oc debug node/<node_name>
```

2. Définir /**host** comme répertoire racine dans l'interpréteur de commandes de débogage :

```
sh-4.4# chroot /host
```

3. Consulter le fichier **kubelet.conf**:

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

Exemple de sortie

```
kind: KubeletConfiguration
apiVersion: kubelet.config.k8s.io/v1beta1
 ...
  "tlsCipherSuites": [
    "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
    "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
  ],
  "tlsMinVersion": "VersionTLS12",
```

5.12. MACHINE CONFIG DAEMON METRICS

Le Machine Config Daemon fait partie du Machine Config Operator. Il s'exécute sur chaque nœud de la grappe. Le Machine Config Daemon gère les modifications et les mises à jour de la configuration sur chacun des nœuds.

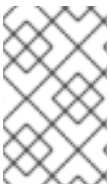
5.12.1. Machine Config Daemon metrics

Depuis OpenShift Container Platform 4.3, le Machine Config Daemon fournit un ensemble de métriques. Ces métriques sont accessibles à l'aide de la pile Prometheus Cluster Monitoring.

Le tableau suivant décrit cet ensemble de mesures.

NOTE

Les mesures marquées par * dans les colonnes *Name* et **Description** représentent des erreurs graves qui peuvent entraîner des problèmes de performances. Ces problèmes peuvent empêcher les mises à jour et les mises à niveau.



NOTE

Bien que certaines entrées contiennent des commandes permettant d'obtenir des journaux spécifiques, l'ensemble le plus complet de journaux est disponible à l'aide de la commande **oc adm must-gather**.

Tableau 5.5. Mesures MCO

Nom	Format	Description	Notes
mcd_host_os_and_version	**[]string{"os", "version"}**	Indique le système d'exploitation sur lequel MCD fonctionne, par exemple RHCOS ou RHEL. Dans le cas de RHCOS, la version est indiquée.	
mcd_drain_err*		Enregistre les erreurs reçues lors de l'échec de la vidange. *	Alors que les vidanges peuvent nécessiter plusieurs tentatives pour réussir, les vidanges terminales qui échouent empêchent les mises à jour de se poursuivre. La métrique **drain_time**, qui indique le temps qu'a pris la vidange, peut aider à résoudre le problème. Pour plus d'informations, consultez les journaux en exécutant le programme : **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

Nom	Format	Description	Notes
mcd_pivot_err*	**[]string{"err", "node", "pivot_target"}**	Enregistre les erreurs rencontrées pendant le pivot. *	Les erreurs de pivot peuvent empêcher la mise à niveau du système d'exploitation. Pour plus d'informations, exécutez la commande suivante pour accéder au nœud et consulter tous ses journaux : **$ oc debug node/<node> — chroot /host journalctl -u pivot.service** Vous pouvez également exécuter cette commande pour ne voir que les journaux du conteneur **machine-config-daemon**: **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_state	**[]string{"state", "reason"}**	État du démon de configuration de la machine pour le nœud indiqué. Les états possibles sont : "Terminé", "En cours" et "Dégradé". Dans le cas de "Dégradé", la raison est incluse.	Pour plus d'informations, consultez les journaux en exécutant le programme : **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_kubelet_state*		Journalise les échecs de santé des kubelets. *	Si le nombre d'échecs dépasse 2, le seuil d'erreur est dépassé. Cela indique un problème possible avec la santé du kubelet. Pour plus d'informations, exécutez la commande suivante pour accéder au nœud et consulter tous ses journaux : **$ oc debug node/<node> — chroot /host journalctl -u kubelet**

Nom	Format	Description	Notes
mcd_reboot_err*	**[]string{"message", "err", "node"}**	Enregistre les échecs de redémarrage et les erreurs correspondantes. *	Ce champ devrait être vide, ce qui indique que le redémarrage s'est bien déroulé. Pour plus d'informations, consultez les journaux en exécutant le programme : **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**
mcd_update_state	**[]string{"config", "err"}**	Enregistre le succès ou l'échec des mises à jour de la configuration et les erreurs correspondantes.	La valeur attendue est **rendered-master/rendered-worker-XXXX**. Si la mise à jour échoue, une erreur est présente. Pour plus d'informations, consultez les journaux en exécutant le programme : **$ oc logs -f -n openshift-machine-config-operator machine-config-daemon-<hash> -c machine-config-daemon**

Ressources supplémentaires

- Voir l'aperçu de la surveillance .
- Voir la documentation sur la collecte de données sur votre cluster .

5.13. CRÉATION DE NŒUDS D'INFRASTRUCTURE

IMPORTANT

Vous ne pouvez utiliser les fonctionnalités avancées de gestion et de mise à l'échelle des machines que dans les clusters où l'API Machine est opérationnelle. Les clusters dont l'infrastructure est fournie par l'utilisateur nécessitent une validation et une configuration supplémentaires pour utiliser l'API Machine.

Les clusters avec le type de plateforme d'infrastructure **none** ne peuvent pas utiliser l'API Machine. Cette limitation s'applique même si les machines de calcul attachées au cluster sont installées sur une plateforme qui prend en charge cette fonctionnalité. Ce paramètre ne peut pas être modifié après l'installation.

Pour afficher le type de plateforme de votre cluster, exécutez la commande suivante :

```
$ oc get infrastructure cluster -o jsonpath='{.status.platform}'
```

Vous pouvez utiliser les jeux de machines d'infrastructure pour créer des machines qui hébergent uniquement des composants d'infrastructure, tels que le routeur par défaut, le registre intégré d'images de conteneurs et les composants pour les métriques et la surveillance des clusters. Ces machines d'infrastructure ne sont pas comptabilisées dans le nombre total d'abonnements requis pour faire fonctionner l'environnement.

Dans un déploiement de production, il est recommandé de déployer au moins trois ensembles de machines pour contenir les composants de l'infrastructure. OpenShift Logging et Red Hat OpenShift Service Mesh déploient tous deux Elasticsearch, qui nécessite l'installation de trois instances sur différents nœuds. Chacun de ces nœuds peut être déployé dans différentes zones de disponibilité pour une haute disponibilité. Cette configuration nécessite trois ensembles de machines différents, un pour chaque zone de disponibilité. Dans les régions Azure globales qui ne disposent pas de plusieurs zones de disponibilité, vous pouvez utiliser des ensembles de machines pour garantir une haute disponibilité.

5.13.1. Composants de l'infrastructure OpenShift Container Platform

Les charges de travail d'infrastructure suivantes ne donnent pas lieu à des abonnements de travailleurs OpenShift Container Platform :

- Les services du plan de contrôle de Kubernetes et d'OpenShift Container Platform qui s'exécutent sur des maîtres
- Le routeur par défaut
- Le registre intégré des images de conteneurs
- Le contrôleur d'entrée basé sur HAProxy
- Le service de collecte ou de surveillance des données de la grappe, y compris les composants permettant de surveiller les projets définis par l'utilisateur
- Journalisation agrégée des clusters
- Courtiers en services
- Red Hat Quay
- Red Hat OpenShift Data Foundation
- Red Hat Advanced Cluster Manager

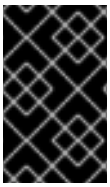
- Red Hat Advanced Cluster Security pour Kubernetes
- Red Hat OpenShift GitOps
- Red Hat OpenShift Pipelines

Tout nœud qui exécute un autre conteneur, pod ou composant est un nœud de travail que votre abonnement doit couvrir.

Pour plus d'informations sur les nœuds d'infrastructure et les composants qui peuvent être exécutés sur les nœuds d'infrastructure, consultez la section "Red Hat OpenShift control plane and infrastructure nodes" dans le document OpenShift sizing and subscription guide for enterprise Kubernetes (Guide de dimensionnement et d'abonnement pour Kubernetes d'entreprise).

Pour créer un nœud d'infrastructure, vous pouvez utiliser un jeu de machines , étiqueter le nœud ou utiliser un pool de configuration de machines .

5.13.1.1. Création d'un nœud d'infrastructure



IMPORTANT

Voir Création de jeux de machines d'infrastructure pour les environnements d'infrastructure fournis par l'installateur ou pour tout cluster dont les nœuds du plan de contrôle sont gérés par l'API des machines.

Les exigences du cluster imposent le provisionnement de l'infrastructure, également appelée **infra** nodes. Le programme d'installation ne fournit des provisions que pour le plan de contrôle et les nœuds de travail. Les nœuds de travail peuvent être désignés comme nœuds d'infrastructure ou nœuds d'application, également appelés **app**, par le biais de l'étiquetage.

Procédure

1. Ajoutez une étiquette au nœud de travailleur que vous voulez utiliser comme nœud d'application :

```
$ oc label node <node-name> node-role.kubernetes.io/app=""
```

2. Ajoutez une étiquette aux nœuds de travailleur que vous souhaitez utiliser comme nœuds d'infrastructure :

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

3. Vérifiez si les nœuds concernés ont désormais les rôles **infra** et **app**:

```
$ oc get nodes
```

4. Créer un sélecteur de nœuds par défaut pour l'ensemble du cluster. Le sélecteur de nœuds par défaut est appliqué aux modules créés dans tous les espaces de noms. Cela crée une intersection avec tous les sélecteurs de nœuds existants sur un pod, ce qui contraint davantage le sélecteur du pod.

IMPORTANT

Si la clé du sélecteur de nœuds par défaut est en conflit avec la clé de l'étiquette d'un pod, le sélecteur de nœuds par défaut n'est pas appliqué.

Cependant, ne définissez pas un sélecteur de nœud par défaut qui pourrait rendre un module non ordonnançable. Par exemple, si le sélecteur de nœud par défaut est défini sur un rôle de nœud spécifique, tel que **node-role.kubernetes.io/infra=""**, alors que l'étiquette d'un module est définie sur un rôle de nœud différent, tel que **node-role.kubernetes.io/master=""**, le module risque de ne plus être ordonnançable. C'est pourquoi il convient d'être prudent lorsque l'on définit le sélecteur de nœuds par défaut sur des rôles de nœuds spécifiques.

Vous pouvez également utiliser un sélecteur de nœud de projet pour éviter les conflits de clés de sélecteur de nœud à l'échelle du cluster.

a. Modifiez l'objet **Scheduler**:

```
$ oc edit scheduler cluster
```

b. Ajoutez le champ **defaultNodeSelector** avec le sélecteur de nœud approprié :

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
...
spec:
  defaultNodeSelector: topology.kubernetes.io/region=us-east-1 1
...
```

1 Cet exemple de sélecteur de nœuds déploie par défaut les pods sur les nœuds de la région **us-east-1**.

c. Enregistrez le fichier pour appliquer les modifications.

Vous pouvez maintenant déplacer les ressources d'infrastructure vers les nœuds **infra** nouvellement étiquetés.

Ressources supplémentaires

- Déplacement de ressources vers des ensembles de machines d'infrastructure

CHAPITRE 6. TRAVAILLER AVEC DES CONTENEURS

6.1. COMPRENDRE LES CONTENEURS

Les unités de base des applications OpenShift Container Platform sont appelées *containers*. Les technologies de conteneurs Linux sont des mécanismes légers permettant d'isoler les processus en cours d'exécution afin qu'ils ne puissent interagir qu'avec les ressources qui leur sont attribuées.

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

Le noyau Linux intègre depuis des années des fonctionnalités pour les technologies de conteneurs. OpenShift Container Platform et Kubernetes ajoutent la possibilité d'orchestrer des conteneurs sur des installations multi-hôtes.

6.1.1. A propos des conteneurs et de la mémoire du noyau RHEL

En raison du comportement de Red Hat Enterprise Linux (RHEL), un conteneur sur un nœud avec une utilisation élevée du CPU peut sembler consommer plus de mémoire que prévu. La consommation de mémoire plus élevée peut être causée par **kmem_cache** dans le noyau RHEL. Le noyau RHEL crée un site **kmem_cache** pour chaque cgroup. Pour plus de performances, le site **kmem_cache** contient un site **cpu_cache** et un cache de nœud pour tous les nœuds NUMA. Ces caches consomment tous de la mémoire du noyau.

La quantité de mémoire stockée dans ces caches est proportionnelle au nombre de CPU que le système utilise. Par conséquent, un nombre plus élevé de CPU se traduit par une plus grande quantité de mémoire du noyau conservée dans ces caches. Des quantités plus importantes de mémoire du noyau dans ces caches peuvent amener les conteneurs OpenShift Container Platform à dépasser les limites de mémoire configurées, ce qui entraîne la destruction du conteneur.

Pour éviter de perdre des conteneurs en raison de problèmes de mémoire du noyau, assurez-vous que les conteneurs demandent suffisamment de mémoire. Vous pouvez utiliser la formule suivante pour estimer la quantité de mémoire consommée par la commande **kmem_cache**, où **nproc** est le nombre d'unités de traitement disponibles indiqué par la commande **nproc**. La limite inférieure des demandes de conteneurs doit correspondre à cette valeur plus les besoins en mémoire des conteneurs :

```
$(nproc) X 1/2 MiB
```

6.1.2. À propos du moteur de conteneurs et de l'exécution des conteneurs

Un *container engine* est un logiciel qui traite les demandes des utilisateurs, y compris les options de la ligne de commande et les extractions d'images. Le moteur de conteneurs utilise un *container runtime*, également appelé *lower-level container runtime*, pour exécuter et gérer les composants nécessaires au déploiement et au fonctionnement des conteneurs. Vous n'aurez probablement pas besoin d'interagir avec le moteur de conteneur ou le runtime de conteneur.

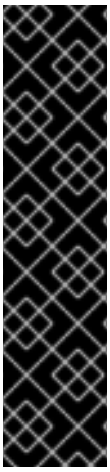
NOTE

La documentation d'OpenShift Container Platform utilise le terme *container runtime* pour se référer à l'exécution de conteneur de niveau inférieur. D'autres documentations peuvent faire référence au moteur de conteneurs en tant que runtime de conteneurs.

OpenShift Container Platform utilise CRI-O comme moteur de conteneur et runC ou crun comme runtime de conteneur. L'exécution de conteneur par défaut est runC. Les deux moteurs d'exécution de conteneurs respectent les spécifications de l'Open Container Initiative (OCI) en matière de moteurs d'exécution.

CRI-O est une implémentation de moteur de conteneur natif de Kubernetes qui s'intègre étroitement avec le système d'exploitation pour fournir une expérience Kubernetes efficace et optimisée. Le moteur de conteneurs CRI-O s'exécute en tant que service systemd sur chaque nœud de cluster OpenShift Container Platform.

runC, développé par Docker et maintenu par l'Open Container Project, est un runtime de conteneur léger et portable écrit en Go. crun, développé par Red Hat, est un runtime de conteneur rapide et à faible mémoire entièrement écrit en C. Depuis OpenShift Container Platform 4.12, vous pouvez choisir entre les deux.



IMPORTANT

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

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

crun présente plusieurs améliorations par rapport à runC, notamment

- Binaire de petite taille
- Traitement plus rapide
- Une empreinte mémoire plus faible

runC présente certains avantages par rapport à crun, notamment

- Le système d'exécution de conteneurs OCI le plus populaire.
- Une plus grande ancienneté dans la production.
- Durée d'exécution du conteneur par défaut de CRI-O.

Vous pouvez passer d'un conteneur à l'autre si nécessaire.

Pour plus d'informations sur la définition de la durée d'exécution du conteneur à utiliser, voir Création d'un CR **ContainerRuntimeConfig** pour modifier les paramètres CRI-O.

6.2. UTILISATION DE CONTENEURS D'INITIATION POUR EFFECTUER DES TÂCHES AVANT LE DÉPLOIEMENT D'UN MODULE

OpenShift Container Platform fournit *init containers*, qui sont des conteneurs spécialisés qui s'exécutent avant les conteneurs d'application et peuvent contenir des utilitaires ou des scripts d'installation qui ne sont pas présents dans une image d'application.

6.2.1. Comprendre les conteneurs Init

Vous pouvez utiliser une ressource Init Container pour effectuer des tâches avant que le reste d'un pod ne soit déployé.

Un pod peut avoir des Init Containers en plus des conteneurs d'application. Les conteneurs Init permettent de réorganiser les scripts d'installation et le code de liaison.

Un conteneur Init peut :

- Contenir et exécuter des utilitaires qu'il n'est pas souhaitable d'inclure dans l'image du conteneur d'applications pour des raisons de sécurité.
- Contiennent des utilitaires ou du code personnalisé pour l'installation qui n'est pas présent dans l'image de l'application. Par exemple, il n'est pas nécessaire de créer une image à partir d'une autre image simplement pour utiliser un outil comme sed, awk, python ou dig pendant l'installation.
- Utiliser les espaces de noms Linux pour qu'ils aient des vues du système de fichiers différentes de celles des conteneurs d'applications, comme l'accès à des secrets auxquels les conteneurs d'applications ne peuvent pas accéder.

Chaque Init Container doit s'achever avec succès avant que le suivant ne soit lancé. Les conteneurs d'initialisation constituent donc un moyen simple de bloquer ou de retarder le démarrage des conteneurs d'applications jusqu'à ce qu'un certain nombre de conditions préalables soient remplies.

Par exemple, voici quelques façons d'utiliser les conteneurs Init :

- Attendez qu'un service soit créé à l'aide d'une commande shell comme :

```
for i in {1..100}; do sleep 1; if dig myservice; then exit 0; fi; done; exit 1
```

- Enregistrez ce pod auprès d'un serveur distant à partir de l'API descendante avec une commande comme :

```
$ curl -X POST
http://$MANAGEMENT_SERVICE_HOST:$MANAGEMENT_SERVICE_PORT/register -d
‘instance=$()&ip=$()’
```

- Attendez un peu avant de lancer l'application Container à l'aide d'une commande telle que **sleep 60**.
- Cloner un dépôt git dans un volume.
- Placez des valeurs dans un fichier de configuration et exécutez un outil de modèle pour générer dynamiquement un fichier de configuration pour le conteneur de l'application principale. Par exemple, placez la valeur POD_IP dans une configuration et générez le fichier de configuration de l'application principale à l'aide de Jinja.

Voir la documentation de Kubernetes pour plus d'informations.

6.2.2. Création de conteneurs Init

L'exemple suivant décrit un pod simple qui possède deux conteneurs d'initialisation (Init Containers). Le premier attend **myservice** et le second **mydb**. Une fois les deux conteneurs terminés, le pod commence.

Procédure

1. Créer un fichier YAML pour le conteneur d'initiation :

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-myservice
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'until getent hosts myservice; do echo waiting for myservice; sleep 2;
done;']
  - name: init-mydb
    image: registry.access.redhat.com/ubi8/ubi:latest
    command: ['sh', '-c', 'until getent hosts mydb; do echo waiting for mydb; sleep 2; done;']
```

2. Créer un fichier YAML pour le service **myservice**.

```
kind: Service
apiVersion: v1
metadata:
  name: myservice
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

3. Créer un fichier YAML pour le service **mydb**.

```
kind: Service
apiVersion: v1
metadata:
  name: mydb
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9377
```

4. Exécutez la commande suivante pour créer le site **myapp-pod**:

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

Exemple de sortie

```
pod/myapp-pod created
```

5. Visualiser l'état du pod :

```
$ oc get pods
```

Exemple de sortie

```
NAME                     READY     STATUS            RESTARTS   AGE
myapp-pod                0/1       Init:0/2          0          5s
```

Notez que le statut du pod indique qu'il est en attente

6. Exécutez les commandes suivantes pour créer les services :

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

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

7. Visualiser l'état du pod :

```
$ oc get pods
```

Exemple de sortie

```
NAME                     READY     STATUS            RESTARTS   AGE
myapp-pod                1/1       Running           0          2m
```

6.3. UTILISATION DE VOLUMES POUR CONSERVER LES DONNÉES DES CONTENEURS

Les fichiers d'un conteneur sont éphémères. En tant que tels, lorsqu'un conteneur tombe en panne ou s'arrête, les données sont perdues. Vous pouvez utiliser *volumes* pour conserver les données utilisées par les conteneurs d'un module. Un volume est un répertoire, accessible aux conteneurs d'un module, dans lequel les données sont stockées pendant toute la durée de vie du module.

6.3.1. Comprendre les volumes

Les volumes sont des systèmes de fichiers montés disponibles pour les pods et leurs conteneurs qui peuvent être soutenus par un certain nombre de points d'extrémité de stockage locaux à l'hôte ou attachés au réseau. Les conteneurs ne sont pas persistants par défaut ; au redémarrage, leur contenu est effacé.

Pour s'assurer que le système de fichiers sur le volume ne contient pas d'erreurs et, si des erreurs sont présentes, pour les réparer si possible, OpenShift Container Platform invoque l'utilitaire **fsck** avant l'utilitaire **mount**. Cela se produit lors de l'ajout d'un volume ou de la mise à jour d'un volume existant.

Le type de volume le plus simple est **emptyDir**, qui est un répertoire temporaire sur une seule machine. Les administrateurs peuvent également vous permettre de demander un volume persistant qui est automatiquement attaché à vos pods.

NOTE

emptyDir peut être limité par un quota basé sur le FSGroup du pod, si le paramètre FSGroup est activé par votre administrateur de cluster.

6.3.2. Travailler avec des volumes en utilisant le CLI de OpenShift Container Platform

Vous pouvez utiliser la commande CLI **oc set volume** pour ajouter et supprimer des volumes et des montages de volumes pour tout objet ayant un modèle de pod, comme les contrôleurs de réplication ou les configurations de déploiement. Vous pouvez également dresser la liste des volumes dans les pods ou tout objet ayant un modèle de pod.

La commande **oc set volume** utilise la syntaxe générale suivante :

```
$ oc set volume <object_selection> <operation> <mandatory_parameters> <options>
```

Sélection d'objets

Spécifiez l'un des éléments suivants pour le paramètre **object_selection** de la commande **oc set volume**:

Tableau 6.1. Sélection d'objets

Syntaxe	Description	Exemple :
<object_type> <name>	Sélectionne ***<name>*** du type ***<object_type>***.	**deploymentConfig registry**
<object_type>/<name>	Sélectionne ***<name>*** du type ***<object_type>***.	**deploymentConfig/registry**
<object_type>*--selector=*<object_label_selector>	Sélectionne les ressources de type ***<object_type>*** correspondant au sélecteur d'étiquette donné.	**deploymentConfig--selector="name=registry"**
<object_type> **--all**	Sélectionne toutes les ressources de type ***<object_type>***.	**deploymentConfig --all**
-f ou **--filename=*<file_name>***	Nom de fichier, répertoire ou URL du fichier à utiliser pour modifier la ressource.	**-f registry-deployment-config.json**

Fonctionnement

Spécifiez **--add** ou **--remove** pour le paramètre **operation** de la commande **oc set volume**.

Paramètres obligatoires

Les paramètres obligatoires sont spécifiques à l'opération sélectionnée et sont abordés dans les sections suivantes.

Options

Toutes les options sont spécifiques à l'opération sélectionnée et sont discutées dans les sections suivantes.

6.3.3. Liste des volumes et des montages de volumes dans un pod

Vous pouvez répertorier les volumes et les montages de volumes dans les pods ou les modèles de pods :

Procédure

Pour dresser la liste des volumes :

```
oc set volume <object_type>/<name> [options]
```

Liste des options supportées par le volume :

Option	Description	Défaut
--name	Nom du volume.	
-c, --containers	Sélectionner les conteneurs par leur nom. Il peut également prendre un caractère générique **'*'** qui correspond à n'importe quel caractère.	**'*'**

Par exemple :

- Pour dresser la liste de tous les volumes du pod **p1**:

```
$ oc set volume pod/p1
```

- Pour lister les volumes **v1** définis dans toutes les configurations de déploiement :

```
$ oc set volume dc --all --name=v1
```

6.3.4. Ajouter des volumes à un module

Vous pouvez ajouter des volumes et des montages de volumes à un module.

Procédure

Pour ajouter un volume, un montage de volume ou les deux à des modèles de pods :

```
oc set volume <object_type>/<name> --add [options] $ oc set volume <object_type>/<name> --add [options]
```

Tableau 6.2. Options prises en charge pour l'ajout de volumes

Option	Description	Défaut
--name	Nom du volume.	Généré automatiquement s'il n'est pas spécifié.
-t, --type	Nom de la source du volume. Valeurs soutenues : **emptyDir**, **hostPath**, **secret**, **configmap**, **persistentVolumeClaim** ou **projected**.	**emptyDir**
-c, --containers	Sélectionner les conteneurs par leur nom. Il peut également prendre un caractère générique **'*'** qui correspond à n'importe quel caractère.	**'*'**
-m, --mount-path	Chemin de montage à l'intérieur des conteneurs sélectionnés. Ne montez pas sur la racine du conteneur, /, ni sur un chemin identique dans l'hôte et le conteneur. Cela peut corrompre votre système hôte si le conteneur est suffisamment privilégié, comme les fichiers de l'hôte **/dev/pts**. Vous pouvez monter l'hôte en toute sécurité en utilisant **/host**.	
--path	Chemin d'accès à l'hôte. Paramètre obligatoire pour **--type=hostPath**. Ne montez pas sur la racine du conteneur, /, ou sur un chemin identique sur l'hôte et le conteneur. Cela peut corrompre votre système hôte si le conteneur est suffisamment privilégié, comme les fichiers **/dev/pts** de l'hôte. Vous pouvez monter l'hôte en toute sécurité en utilisant **/host**.	
--secret-name	Nom du secret. Paramètre obligatoire pour **--type=secret**.	
--configmap-name	Nom de la carte de configuration. Paramètre obligatoire pour **--type=configmap**.	

Option	Description	Défaut
--claim-name	Nom de la revendication de volume persistant. Paramètre obligatoire pour **--type=persistentVolumeClaim**.	
--source	Détails de la source de volume sous forme de chaîne JSON. Recommandé si la source de volume souhaitée n'est pas prise en charge par **--type**.	
-o, --output	Affiche les objets modifiés au lieu de les mettre à jour sur le serveur. Valeurs prises en charge : **json**, **yaml**.	
--output-version	Affiche les objets modifiés avec la version donnée.	**api-version**

Par exemple :

- Pour ajouter une nouvelle source de volume **emptyDir** à l'objet **registry DeploymentConfig** :

```
$ oc set volume dc/registry --add
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter le volume :

Exemple 6.1. Exemple de configuration de déploiement avec un volume ajouté

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: registry
  namespace: registry
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes: 1
        - name: volume-pppsw
          emptyDir: {}
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
```

1 Ajouter la source du volume **emptyDir**.

- Pour ajouter le volume **v1** avec le secret **secret1** pour le contrôleur de réplication **r1** et le monter dans les conteneurs à l'adresse suivante ***/data***:

```
$ oc set volume rc/r1 --add --name=v1 --type=secret --secret-name='secret1' --mount-
path=/data
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter le volume :

Exemple 6.2. Exemple de contrôleur de réplication avec volume et secret ajoutés

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes: 1
        - name: v1
          secret:
            secretName: secret1
            defaultMode: 420
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          volumeMounts: 2
            - name: v1
              mountPath: /data
```

1 Ajouter le volume et le secret.
2 Ajouter le chemin de montage du conteneur.

- Pour ajouter le volume persistant existant **v1** avec le nom de revendication **pvc1** à la configuration de déploiement ***dc.json*** sur le disque, monter le volume sur le conteneur **c1** à l'adresse ***/data***et mettez à jour l'objet **DeploymentConfig** sur le serveur :

```
$ oc set volume -f dc.json --add --name=v1 --type=persistentVolumeClaim \
  --claim-name=pvc1 --mount-path=/data --containers=c1
```

ASTUCE

Vous pouvez également appliquer le code YAML suivant pour ajouter le volume :

Exemple 6.3. Exemple de configuration de déploiement avec ajout d'un volume persistant

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v1 (1)
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: (2)
            - name: v1
              mountPath: /data
```

1. Ajouter la revendication de volume persistant nommée `pvc1.
2. Ajouter le chemin de montage du conteneur.

- Pour ajouter un volume **v1** basé sur le dépôt Git **https://github.com/namespace1/project1** avec la révision **5125c45f9f563** pour tous les contrôleurs de réplication :

```
$ oc set volume rc --all --add --name=v1 \
  --source='{"gitRepo": {
              "repository": "https://github.com/namespace1/project1",
              "revision": "5125c45f9f563"
            }}'
```

6.3.5. Mise à jour des volumes et des montages de volumes dans un pod

Vous pouvez modifier les volumes et les montages de volumes dans un pod.

Procédure

Mise à jour des volumes existants à l'aide de l'option **--overwrite**:

```
oc set volume <object_type>/<name> --add --overwrite [options] $ oc set volume
<object_type>/<name> --add --overwrite
```

Par exemple :

- Pour remplacer le volume existant **v1** pour le contrôleur de réplication **r1** par une revendication de volume persistant existante **pvc1**:

```
$ oc set volume rc/r1 --add --overwrite --name=v1 --type=persistentVolumeClaim --claim-
name=pvc1
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour remplacer le volume :

Exemple 6.4. Exemple de contrôleur de réplication avec un volume persistant nommé pvc1

```
kind: ReplicationController
apiVersion: v1
metadata:
  name: example-1
  namespace: example
spec:
  replicas: 0
  selector:
    app: httpd
    deployment: example-1
    deploymentconfig: example
  template:
    metadata:
      labels:
        app: httpd
        deployment: example-1
        deploymentconfig: example
    spec:
      volumes:
        - name: v1 (1)
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts:
            - name: v1
              mountPath: /data
```

(1) Définir la revendication de volume persistant sur **pvc1**.

- Pour modifier le point de montage de l'objet **DeploymentConfig** **d1** en */opt* pour le volume **v1**:

```
$ oc set volume dc/d1 --add --overwrite --name=v1 --mount-path=/opt
```

ASTUCE

Vous pouvez également appliquer le YAML suivant pour modifier le point de montage :

Exemple 6.5. Exemple de configuration de déploiement avec un point de montage défini sur opt.

```
kind: DeploymentConfig
apiVersion: apps.openshift.io/v1
metadata:
  name: example
  namespace: example
spec:
  replicas: 3
  selector:
    app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      volumes:
        - name: volume-pppsw
          emptyDir: {}
        - name: v2
          persistentVolumeClaim:
            claimName: pvc1
        - name: v1
          persistentVolumeClaim:
            claimName: pvc1
      containers:
        - name: httpd
          image: >-
            image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest
          ports:
            - containerPort: 8080
              protocol: TCP
          volumeMounts: 1
            - name: v1
              mountPath: /opt
```

1 Définissez le point de montage à /**opt**.

6.3.6. Suppression de volumes et de montages de volumes d'un pod

Vous pouvez supprimer un volume ou un montage de volume d'un module.

Procédure

Pour supprimer un volume des modèles de pods :

```
oc set volume <object_type>/<name> --remove [options] $ oc set volume <object_type>/<name> --
remove
```

Tableau 6.3. Options prises en charge pour la suppression des volumes

Option	Description	Défaut
--name	Nom du volume.	
-c, --containers	Sélectionner les conteneurs par leur nom. Il peut également prendre un caractère générique **'*'** qui correspond à n'importe quel caractère.	**'*'**
--confirm	Indiquez que vous souhaitez supprimer plusieurs volumes à la fois.	
-o, --output	Affiche les objets modifiés au lieu de les mettre à jour sur le serveur. Valeurs prises en charge : **json**, **yaml**.	
--output-version	Affiche les objets modifiés avec la version donnée.	**api-version**

Par exemple :

- Pour supprimer un volume **v1** de l'objet **DeploymentConfig** d1 :

```
$ oc set volume dc/d1 --remove --name=v1
```

- Démonter le volume **v1** du conteneur **c1** pour l'objet **DeploymentConfig** d1 et supprimer le volume **v1** s'il n'est référencé par aucun conteneur sur **d1**:

```
$ oc set volume dc/d1 --remove --name=v1 --containers=c1
```

- Pour supprimer tous les volumes du contrôleur de réplication **r1**:

```
$ oc set volume rc/r1 --remove --confirm
```

6.3.7. Configurer des volumes pour des utilisations multiples dans un pod

Vous pouvez configurer un volume pour vous permettre de partager un volume pour plusieurs utilisations dans un seul pod en utilisant la propriété **volumeMounts.subPath** pour spécifier une valeur **subPath** à l'intérieur d'un volume au lieu de la racine du volume.

Procédure

1. Pour afficher la liste des fichiers contenus dans le volume, exécutez la commande **oc rsh**:

```
$ oc rsh <pod>
```

Exemple de sortie

```
sh-4.2$ ls /path/to/volume/subpath/mount
example_file1 example_file2 example_file3
```

2. Spécifiez le site **subPath**:

Exemple : Pod spec avec subPath parameter

```
apiVersion: v1
kind: Pod
metadata:
  name: my-site
spec:
    containers:
    - name: mysql
      image: mysql
      volumeMounts:
      - mountPath: /var/lib/mysql
        name: site-data
        subPath: mysql (1)
    - name: php
      image: php
      volumeMounts:
      - mountPath: /var/www/html
        name: site-data
        subPath: html (2)
    volumes:
    - name: site-data
      persistentVolumeClaim:
        claimName: my-site-data
```

(1) Les bases de données sont stockées dans le dossier **mysql**.

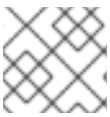
(2) Le contenu HTML est stocké dans le dossier **html**.

6.4. CARTOGRAPHIE DES VOLUMES À L'AIDE DES VOLUMES PROJETÉS

Un site *projected volume* met en correspondance plusieurs sources de volume existantes dans le même répertoire.

Les types de sources de volume suivants peuvent être projetés :

- Secrets
- Cartes de configuration
- API vers le bas



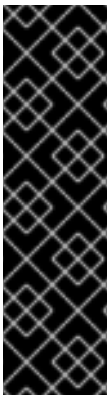
NOTE

Toutes les sources doivent se trouver dans le même espace de noms que le module.

6.4.1. Comprendre les volumes prévus

Les volumes projetés peuvent représenter n'importe quelle combinaison de ces sources de volume dans un seul répertoire, ce qui permet à l'utilisateur de.. :

- remplir automatiquement un seul volume avec les clés de plusieurs secrets, les cartes de configuration et les informations de l'API descendante, afin que je puisse synthétiser un seul répertoire avec diverses sources d'informations ;
- remplir un seul volume avec les clés de plusieurs secrets, les cartes de configuration et les informations de l'API descendante, en spécifiant explicitement les chemins pour chaque élément, de sorte que je puisse avoir un contrôle total sur le contenu de ce volume.



IMPORTANT

Lorsque l'autorisation **RunAsUser** est définie dans le contexte de sécurité d'un pod basé sur Linux, les fichiers projetés ont les autorisations correctes, y compris la propriété de l'utilisateur du conteneur. Toutefois, lorsque l'autorisation équivalente de Windows, **RunAsUsername**, est définie dans un module Windows, le kubelet n'est pas en mesure de définir correctement la propriété des fichiers du volume projeté.

Par conséquent, l'autorisation **RunAsUsername** définie dans le contexte de sécurité d'un pod Windows n'est pas honorée pour les volumes projetés Windows fonctionnant dans OpenShift Container Platform.

Les scénarios généraux suivants montrent comment vous pouvez utiliser les volumes prévisionnels.

Config map, secrets, Downward API.

Les volumes projetés vous permettent de déployer des conteneurs avec des données de configuration comprenant des mots de passe. Une application utilisant ces ressources pourrait déployer Red Hat OpenStack Platform (RHOSP) sur Kubernetes. Les données de configuration peuvent devoir être assemblées différemment selon que les services seront utilisés pour la production ou pour les tests. Si un pod est étiqueté comme production ou test, le sélecteur d'API descendant **metadata.labels** peut être utilisé pour produire les configurations RHOSP correctes.

Config map secrets.

Les volumes projetés vous permettent de déployer des conteneurs contenant des données de configuration et des mots de passe. Par exemple, vous pouvez exécuter une carte de configuration avec des tâches sensibles cryptées qui sont décryptées à l'aide d'un fichier de mots de passe du coffre-fort.

ConfigMap Downward API.

Les volumes projetés vous permettent de générer une configuration comprenant le nom du pod (disponible via le sélecteur **metadata.name**). Cette application peut alors transmettre le nom du pod avec les requêtes afin de déterminer facilement la source sans utiliser le suivi des adresses IP.

Secrets Downward API.

Les volumes projetés vous permettent d'utiliser un secret comme clé publique pour crypter l'espace de noms du pod (disponible via le sélecteur **metadata.namespace**). Cet exemple permet à l'opérateur d'utiliser l'application pour fournir les informations sur l'espace de noms en toute sécurité sans utiliser de transport crypté.

6.4.1.1. Exemple de spécifications d'un pod

Voici des exemples de spécifications **Pod** pour la création de volumes prévisionnels.

Pod avec un secret, une API descendante et une carte de configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts: 1
    - name: all-in-one
      mountPath: "/projected-volume" 2
      readOnly: true 3
  volumes: 4
  - name: all-in-one 5
    projected:
      defaultMode: 0400 6
      sources:
      - secret:
          name: mysecret 7
          items:
            - key: username
              path: my-group/my-username 8
      - downwardAPI: 9
          items:
            - path: "labels"
              fieldRef:
                fieldPath: metadata.labels
            - path: "cpu_limit"
              resourceFieldRef:
                containerName: container-test
                resource: limits.cpu
      - configMap: 10
          name: myconfigmap
          items:
            - key: config
              path: my-group/my-config
              mode: 0777 11
```

1. Ajoutez une section **volumeMounts** pour chaque conteneur qui a besoin du secret.
2. Indiquez le chemin d'accès à un répertoire inutilisé dans lequel le secret apparaîtra.
3. Définir **readOnly** à **true**.
4. Ajouter un bloc **volumes** pour énumérer chaque source de volume projeté.
5. Indiquez un nom quelconque pour le volume.
6. Définir l'autorisation d'exécution sur les fichiers.
7. Ajouter un secret. Saisissez le nom de l'objet secret. Chaque secret que vous souhaitez utiliser doit être répertorié.

8 Indiquez le chemin d'accès au fichier des secrets sous **mountPath**. Ici, le fichier des secrets se trouve dans */projected-volume/my-group/my-username*.

9 Ajouter une source API descendante.

10 Ajouter une source ConfigMap.

11 Régler le mode pour la projection spécifique

NOTE

S'il y a plusieurs conteneurs dans le module, chaque conteneur a besoin d'une section **volumeMounts**, mais une seule section **volumes** est nécessaire.

Pod avec plusieurs secrets dont le mode d'autorisation n'est pas par défaut

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      defaultMode: 0755
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/my-username
      - secret:
          name: mysecret2
          items:
            - key: password
              path: my-group/my-password
              mode: 511
```

NOTE

Le **defaultMode** ne peut être spécifié qu'au niveau de la projection et non pour chaque source de volume. Cependant, comme illustré ci-dessus, vous pouvez explicitement définir le **mode** pour chaque projection individuelle.

6.4.1.2. Considérations sur le cheminement

Collisions Between Keys when Configured Paths are Identical

Si vous configurez des clés avec le même chemin d'accès, la spécification du pod ne sera pas acceptée comme valide. Dans l'exemple suivant, le chemin spécifié pour **mysecret** et **myconfigmap** est le même :

```
apiVersion: v1
kind: Pod
metadata:
  name: volume-test
spec:
  containers:
  - name: container-test
    image: busybox
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:
          name: mysecret
          items:
            - key: username
              path: my-group/data
      - configMap:
          name: myconfigmap
          items:
            - key: config
              path: my-group/data
```

Examinez les situations suivantes relatives aux chemins d'accès aux fichiers de volume.

Collisions Between Keys without Configured Paths

La seule validation en cours d'exécution possible est lorsque tous les chemins sont connus lors de la création du module, comme dans le scénario ci-dessus. Sinon, en cas de conflit, la ressource spécifiée la plus récente écrase tout ce qui la précède (cela vaut également pour les ressources mises à jour après la création du module).

Collisions when One Path is Explicit and the Other is Automatically Projected

En cas de collision due à la correspondance entre un chemin spécifié par l'utilisateur et des données projetées automatiquement, cette dernière ressource écrasera tout ce qui la précède, comme auparavant

6.4.2. Configuration d'un volume projeté pour un pod

Lors de la création de volumes projetés, tenez compte des situations de chemin de fichier de volume décrites à l'adresse *Understanding projected volumes*.

L'exemple suivant montre comment utiliser un volume projeté pour monter une source de volume secret existante. Les étapes peuvent être utilisées pour créer un nom d'utilisateur et un mot de passe secrets à partir de fichiers locaux. Vous créez ensuite un pod qui exécute un conteneur, en utilisant un volume projeté pour monter les secrets dans le même répertoire partagé.

Procédure

Pour utiliser un volume projeté afin de monter une source de volume secrète existante.

1. Créez des fichiers contenant les secrets, en saisissant ce qui suit, en remplaçant le mot de passe et les informations relatives à l'utilisateur, le cas échéant :

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
```

   Les valeurs **user** et **pass** peuvent être n'importe quelle chaîne de caractères valide codée **base64**.

   L'exemple suivant montre **admin** en base64 :

```
$ echo -n "admin" | base64
```

   Exemple de sortie

```
YWRtaW4=
```

   L'exemple suivant montre le mot de passe **1f2d1e2e67df** en base64 :.

```
$ echo -n "1f2d1e2e67df" | base64
```

   Exemple de sortie

```
MWYyZDFlMmU2N2Rm
```

2. Utilisez la commande suivante pour créer les secrets :

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

   Par exemple :

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

   Exemple de sortie

```
secret "mysecret" created
```

3. Vous pouvez vérifier que le secret a été créé à l'aide des commandes suivantes :

```
$ oc get secret <secret-name>
```

   Par exemple :

```
$ oc get secret mysecret
```

Exemple de sortie

```
NAME       TYPE      DATA      AGE
mysecret   Opaque    2         17h
```

```
$ oc get secret <secret-name> -o yaml
```

Par exemple :

```
$ oc get secret mysecret -o yaml
```

```
apiVersion: v1
data:
  pass: MWYyZDFlMmU2N2Rm
  user: YWRtaW4=
kind: Secret
metadata:
  creationTimestamp: 2017-05-30T20:21:38Z
  name: mysecret
  namespace: default
  resourceVersion: "2107"
  selfLink: /api/v1/namespaces/default/secrets/mysecret
  uid: 959e0424-4575-11e7-9f97-fa163e4bd54c
type: Opaque
```

4. Créez un fichier de configuration de pod similaire au suivant qui inclut une section **volumes**:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-projected-volume
spec:
  containers:
  - name: test-projected-volume
    image: busybox
    args:
    - sleep
    - "86400"
    volumeMounts:
    - name: all-in-one
      mountPath: "/projected-volume"
      readOnly: true
  volumes:
  - name: all-in-one
    projected:
      sources:
      - secret:          (1)
          name: user
      - secret:          (2)
          name: pass
```

1 2 Le nom du secret que vous avez créé.

5. Créer le pod à partir du fichier de configuration :

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

Par exemple :

```
$ oc create -f secret-pod.yaml
```

Exemple de sortie

```
pod "test-projected-volume" created
```

6. Vérifiez que le conteneur de pods est en cours d'exécution, puis surveillez les modifications apportées au pod :

```
$ oc get pod <name>
```

Par exemple :

```
$ oc get pod test-projected-volume
```

Le résultat devrait ressembler à ce qui suit :

Exemple de sortie

```
NAME                    READY     STATUS    RESTARTS   AGE
test-projected-volume   1/1       Running   0          14s
```

7. Dans un autre terminal, utilisez la commande **oc exec** pour ouvrir un shell sur le conteneur en cours d'exécution :

```
oc exec -it <pod> <command> $ oc exec -it <pod> <command>
```

Par exemple :

```
$ oc exec -it test-projected-volume -- /bin/sh
```

8. Dans votre shell, vérifiez que le répertoire **projected-volumes** contient les sources projetées :

```
/ # ls
```

Exemple de sortie

```
bin              home             root             tmp
dev              proc             run              usr
etc              projected-volume  sys              var
```

6.5. PERMETTRE AUX CONTENEURS DE CONSOMMER DES OBJETS API

Le site *Downward API* est un mécanisme qui permet aux conteneurs de consommer des informations sur les objets API sans être couplés à OpenShift Container Platform. Ces informations comprennent le nom du pod, l'espace de noms et les valeurs des ressources. Les conteneurs peuvent consommer les informations de l'API descendante à l'aide de variables d'environnement ou d'un plugin de volume.

6.5.1. Exposer les informations sur les pods aux conteneurs à l'aide de l'API descendante

L'API descendante contient des informations telles que le nom du pod, le projet et les valeurs des ressources. Les conteneurs peuvent consommer les informations de l'API descendante à l'aide de variables d'environnement ou d'un plugin de volume.

Les champs du module sont sélectionnés à l'aide du type d'API **FieldRef**. **FieldRef** comporte deux champs :

Field	Description
fieldPath	Le chemin du champ à sélectionner, par rapport au pod.
apiVersion	La version de l'API dans laquelle le sélecteur **fieldPath** doit être interprété.

Actuellement, les sélecteurs valides dans l'API v1 sont les suivants :

Sélecteur	Description
metadata.name	Le nom du module. Ceci est supporté à la fois dans les variables d'environnement et les volumes.
metadata.namespace	L'espace de noms du pod est pris en charge à la fois par les variables d'environnement et les volumes.
metadata.labels	Les étiquettes du pod. Ceci n'est possible que dans les volumes et non dans les variables d'environnement.
metadata.annotations	Les annotations du pod. Ceci n'est possible que dans les volumes et non dans les variables d'environnement.
status.podIP	L'IP du pod. Ceci n'est supporté que dans les variables d'environnement et non dans les volumes.

Le champ **apiVersion**, s'il n'est pas spécifié, prend par défaut la version API du modèle de pod qui l'entoure.

6.5.2. Comprendre comment consommer les valeurs des conteneurs à l'aide de l'API descendante

Les conteneurs peuvent consommer les valeurs de l'API à l'aide de variables d'environnement ou d'un plugin de volume. Selon la méthode choisie, les conteneurs peuvent consommer :

- Nom du pod
- Pod projet/espace de noms
- Annotations sur les pods
- Étiquettes des gousses

Les annotations et les étiquettes sont disponibles en utilisant uniquement un plugin de volume.

6.5.2.1. Consommer des valeurs de conteneurs à l'aide de variables d'environnement

Lorsque vous utilisez les variables d'environnement d'un conteneur, utilisez le champ **valueFrom** du type **EnvVar** (de type **EnvVarSource**) pour spécifier que la valeur de la variable doit provenir d'une source **FieldRef** au lieu de la valeur littérale spécifiée par le champ **value**.

Seuls les attributs constants du pod peuvent être consommés de cette manière, car les variables d'environnement ne peuvent pas être mises à jour une fois qu'un processus est lancé d'une manière qui permette au processus d'être informé que la valeur d'une variable a changé. Les champs pris en charge par les variables d'environnement sont les suivants

- Nom du pod
- Pod projet/espace de noms

Procédure

Pour utiliser des variables d'environnement

1. Créer un fichier **pod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        - name: MY_POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
  restartPolicy: Never
```

2. Créer le pod à partir du fichier **pod.yaml**:

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

3. Vérifiez les journaux du conteneur pour les valeurs **MY_POD_NAME** et **MY_POD_NAMESPACE**:

```
$ oc logs -p dapi-env-test-pod
```

6.5.2.2. Consommer les valeurs d'un conteneur à l'aide d'un plugin de volume

Les conteneurs peuvent consommer les valeurs de l'API à l'aide d'un plugin de volume.

Les conteneurs peuvent consommer :

- Nom du pod
- Pod projet/espace de noms
- Annotations sur les pods
- Étiquettes des gousses

Procédure

Pour utiliser le plugin de volume :

1. Créer un fichier **volume-pod.yaml**:

```
kind: Pod
apiVersion: v1
metadata:
  labels:
    zone: us-east-coast
    cluster: downward-api-test-cluster1
    rack: rack-123
  name: dapi-volume-test-pod
  annotations:
    annotation1: "345"
    annotation2: "456"
spec:
  containers:
    - name: volume-test-container
      image: gcr.io/google_containers/busybox
      command: ["sh", "-c", "cat /tmp/etc/pod_labels /tmp/etc/pod_annotations"]
      volumeMounts:
        - name: podinfo
          mountPath: /tmp/etc
          readOnly: false
  volumes:
  - name: podinfo
    downwardAPI:
      defaultMode: 420
      items:
      - fieldRef:
          fieldPath: metadata.name
```

```
      path: pod_name
    - fieldRef:
        fieldPath: metadata.namespace
      path: pod_namespace
    - fieldRef:
        fieldPath: metadata.labels
      path: pod_labels
    - fieldRef:
        fieldPath: metadata.annotations
      path: pod_annotations
  restartPolicy: Never
```

2. Créer le pod à partir du fichier **volume-pod.yaml**:

```
$ oc create -f volume-pod.yaml
```

3. Consultez les journaux du conteneur et vérifiez la présence des champs configurés :

```
$ oc logs -p dapi-volume-test-pod
```

Exemple de sortie

```
cluster=downward-api-test-cluster1
rack=rack-123
zone=us-east-coast
annotation1=345
annotation2=456
kubernetes.io/config.source=api
```

6.5.3. Comprendre comment consommer les ressources d'un conteneur à l'aide de l'API descendante

Lors de la création de pods, vous pouvez utiliser l'API descendante pour injecter des informations sur les demandes et les limites des ressources informatiques afin que les auteurs d'images et d'applications puissent créer correctement une image pour des environnements spécifiques.

Vous pouvez le faire en utilisant une variable d'environnement ou un plugin de volume.

6.5.3.1. Consommer les ressources d'un conteneur à l'aide de variables d'environnement

Lors de la création de pods, vous pouvez utiliser l'API descendante pour injecter des informations sur les demandes et les limites des ressources informatiques à l'aide de variables d'environnement.

Procédure

Pour utiliser les variables d'environnement :

1. Lors de la création d'une configuration de pod, spécifiez les variables d'environnement qui correspondent au contenu du champ **resources** dans le champ **spec.container** dans le champ

```
....
spec:
  containers:
    - name: test-container
```

```
      image: gcr.io/google_containers/busybox:1.24
      command: [ "/bin/sh", "-c", "env" ]
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      env:
        - name: MY_CPU_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.cpu
        - name: MY_CPU_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.cpu
        - name: MY_MEM_REQUEST
          valueFrom:
            resourceFieldRef:
              resource: requests.memory
        - name: MY_MEM_LIMIT
          valueFrom:
            resourceFieldRef:
              resource: limits.memory
....
```

Si les limites de ressources ne sont pas incluses dans la configuration du conteneur, l'API descendante utilise par défaut les valeurs de CPU et de mémoire allouables du nœud.

2. Créer le pod à partir du fichier ***pod.yaml*** fichier :

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

6.5.3.2. Consommer les ressources d'un conteneur à l'aide d'un plugin de volume

Lors de la création de pods, vous pouvez utiliser l'API Downward pour injecter des informations sur les demandes et les limites des ressources informatiques à l'aide d'un plugin de volume.

Procédure

Pour utiliser le plugin de volume :

1. Lors de la création d'une configuration de pod, utilisez le champ **spec.volumes.downwardAPI.items** pour décrire les ressources souhaitées qui correspondent au champ **spec.resources**:

```
....
spec:
  containers:
    - name: client-container
      image: gcr.io/google_containers/busybox:1.24
      command: ["sh", "-c", "while true; do echo; if [[ -e /etc/cpu_limit ]]; then cat /etc/cpu_limit;
fi; if [[ -e /etc/cpu_request ]]; then cat /etc/cpu_request; fi; if [[ -e /etc/mem_limit ]]; then cat
/etc/mem_limit; fi; if [[ -e /etc/mem_request ]]; then cat /etc/mem_request; fi; sleep 5; done"]
```

```
      resources:
        requests:
          memory: "32Mi"
          cpu: "125m"
        limits:
          memory: "64Mi"
          cpu: "250m"
      volumeMounts:
        - name: podinfo
          mountPath: /etc
          readOnly: false
  volumes:
    - name: podinfo
      downwardAPI:
        items:
          - path: "cpu_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.cpu
          - path: "cpu_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.cpu
          - path: "mem_limit"
            resourceFieldRef:
              containerName: client-container
              resource: limits.memory
          - path: "mem_request"
            resourceFieldRef:
              containerName: client-container
              resource: requests.memory
....
```

Si les limites de ressources ne sont pas incluses dans la configuration du conteneur, l'API descendante utilise par défaut les valeurs de l'unité centrale et de la mémoire allouable du nœud.

2. Créer le pod à partir du fichier ***volume-pod.yaml*** fichier :

```
$ oc create -f volume-pod.yaml
```

6.5.4. Consommer des secrets à l'aide de l'API descendante

Lors de la création de pods, vous pouvez utiliser l'API descendante pour injecter des secrets afin que les auteurs d'images et d'applications puissent créer une image pour des environnements spécifiques.

Procédure

1. Créer un fichier **secret.yaml**:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
data:
```

```
  password: cGFzc3dvcmQ=
  username: ZGV2ZWxvcGVy
type: kubernetes.io/basic-auth
```

2. Créer un objet **Secret** à partir du fichier **secret.yaml**:

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

3. Créez un fichier **pod.yaml** qui fait référence au champ **username** de l'objet **Secret** ci-dessus :

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
  restartPolicy: Never
```

4. Créer le pod à partir du fichier **pod.yaml**:

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

5. Vérifiez les journaux du conteneur pour la valeur **MY_SECRET_USERNAME**:

```
$ oc logs -p dapi-env-test-pod
```

6.5.5. Consommer des cartes de configuration à l'aide de l'API descendante

Lors de la création de pods, vous pouvez utiliser l'API Downward pour injecter des valeurs de carte de configuration afin que les auteurs d'images et d'applications puissent créer une image pour des environnements spécifiques.

Procédure

1. Créer un ***configmap.yaml*** fichier :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: myconfigmap
data:
  mykey: myvalue
```

2. Créer un objet **ConfigMap** à partir du ***configmap.yaml*** fichier :

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

3. Créer un fichier ***pod.yaml*** qui fait référence à l'objet **ConfigMap** ci-dessus :

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_CONFIGMAP_VALUE
          valueFrom:
            configMapKeyRef:
              name: myconfigmap
              key: mykey
  restartPolicy: Always
```

4. Créer le pod à partir du fichier ***pod.yaml*** fichier :

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

5. Vérifiez les journaux du conteneur pour la valeur **MY_CONFIGMAP_VALUE**:

```
$ oc logs -p dapi-env-test-pod
```

6.5.6. Référence aux variables d'environnement

Lors de la création de pods, vous pouvez faire référence à la valeur d'une variable d'environnement précédemment définie en utilisant la syntaxe **$()**. Si la référence à la variable d'environnement ne peut pas être résolue, la valeur sera laissée sous la forme de la chaîne fournie.

Procédure

1. Créer un fichier ***pod.yaml*** qui fait référence à un fichier existant **environment variable**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_EXISTING_ENV
          value: my_value
```

```
      - name: MY_ENV_VAR_REF_ENV
        value: $(MY_EXISTING_ENV)
  restartPolicy: Never
```

2. Créer le pod à partir du fichier ***pod.yaml*** fichier :

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

3. Vérifiez les journaux du conteneur pour la valeur **MY_ENV_VAR_REF_ENV**:

```
$ oc logs -p dapi-env-test-pod
```

6.5.7. Échapper aux références aux variables d'environnement

Lors de la création d'un module, vous pouvez échapper à une référence à une variable d'environnement en utilisant un double signe de dollar. La valeur sera alors remplacée par une version avec un seul signe de dollar de la valeur fournie.

Procédure

1. Créer un fichier ***pod.yaml*** qui fait référence à un fichier existant **environment variable**:

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-env-test-pod
spec:
  containers:
    - name: env-test-container
      image: gcr.io/google_containers/busybox
      command: [ "/bin/sh", "-c", "env" ]
      env:
        - name: MY_NEW_ENV
          value: $$(SOME_OTHER_ENV)
  restartPolicy: Never
```

2. Créer le pod à partir du fichier ***pod.yaml*** fichier :

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

3. Vérifiez les journaux du conteneur pour la valeur **MY_NEW_ENV**:

```
$ oc logs -p dapi-env-test-pod
```

6.6. COPIER DES FICHIERS VERS OU DEPUIS UN CONTENEUR OPENSHIFT CONTAINER PLATFORM

Vous pouvez utiliser l'interface de commande pour copier des fichiers locaux vers ou depuis un répertoire distant dans un conteneur à l'aide de la commande **rsync**.

6.6.1. Comprendre comment copier des fichiers

La commande **oc rsync**, ou synchronisation à distance, est un outil utile pour copier des archives de bases de données vers et depuis vos modules à des fins de sauvegarde et de restauration. Vous pouvez également utiliser **oc rsync** pour copier les modifications du code source dans un pod en cours d'exécution pour le débogage du développement, lorsque le pod en cours d'exécution prend en charge le rechargement à chaud des fichiers source.

```
$ oc rsync <source> <destination> [-c <container>]
```

6.6.1.1. Requirements

Spécification de la source de la copie

L'argument source de la commande **oc rsync** doit pointer vers un répertoire local ou un répertoire pod. Les fichiers individuels ne sont pas pris en charge.

Lors de la spécification d'un répertoire de pods, le nom du répertoire doit être préfixé par le nom du pod :

```
<pod name>:<dir>
```

Si le nom du répertoire se termine par un séparateur de chemin (/), seul le contenu du répertoire est copié vers la destination. Sinon, le répertoire et son contenu sont copiés vers la destination.

Spécification de la destination de la copie

L'argument de destination de la commande **oc rsync** doit pointer vers un répertoire. Si le répertoire n'existe pas, mais que **rsync** est utilisé pour la copie, le répertoire est créé pour vous.

Suppression de fichiers à la destination

L'option **--delete** peut être utilisée pour supprimer tous les fichiers du répertoire distant qui ne se trouvent pas dans le répertoire local.

Synchronisation continue en cas de changement de fichier

L'utilisation de l'option **--watch** permet à la commande de surveiller le chemin d'accès à la source pour toute modification du système de fichiers et de synchroniser les modifications lorsqu'elles se produisent. Avec cet argument, la commande s'exécute indéfiniment.

La synchronisation s'effectue après de courtes périodes de silence afin de s'assurer qu'un système de fichiers en évolution rapide n'entraîne pas d'appels de synchronisation continus.

Lorsque vous utilisez l'option **--watch**, le comportement est effectivement le même que l'invocation manuelle répétée de **oc rsync**, y compris tous les arguments normalement transmis à **oc rsync**. Par conséquent, vous pouvez contrôler le comportement via les mêmes drapeaux que ceux utilisés pour les invocations manuelles de **oc rsync**, tels que **--delete**.

6.6.2. Copier des fichiers vers et depuis des conteneurs

La prise en charge de la copie de fichiers locaux vers ou depuis un conteneur est intégrée à l'interface de gestion.

Conditions préalables

Lorsque vous travaillez avec **oc rsync**, tenez compte des points suivants :

rsync doit être installé

La commande **oc rsync** utilise l'outil local **rsync** s'il est présent sur la machine cliente et le conteneur distant.

Si **rsync** n'est pas trouvé localement ou dans le conteneur distant, une archive **tar** est créée localement et envoyée au conteneur où l'utilitaire **tar** est utilisé pour extraire les fichiers. Si **tar** n'est pas disponible dans le conteneur distant, la copie échoue.

La méthode de copie **tar** n'offre pas les mêmes fonctionnalités que **oc rsync**. Par exemple, **oc rsync** crée le répertoire de destination s'il n'existe pas et n'envoie que les fichiers qui sont différents entre la source et la destination.

NOTE

Sous Windows, le client **cwRsync** doit être installé et ajouté au PATH pour être utilisé avec la commande **oc rsync**.

Procédure

- Pour copier un répertoire local dans un répertoire pod :

```
$ oc rsync <local-dir> <pod-name>:/<remote-dir> -c <container-name>
```

  Par exemple :

```
$ oc rsync /home/user/source devpod1234:/src -c user-container
```

- Pour copier un répertoire pod dans un répertoire local :

```
$ oc rsync devpod1234:/src /home/user/source
```

  Exemple de sortie

```
$ oc rsync devpod1234:/src/status.txt /home/user/
```

6.6.3. Utiliser les fonctions avancées de Rsync

La commande **oc rsync** propose moins d'options de ligne de commande que la commande standard **rsync**. Si vous souhaitez utiliser une option de la ligne de commande **rsync** standard qui n'est pas disponible dans **oc rsync**, par exemple l'option **--exclude-from=FILE**, il est possible d'utiliser l'option **--rsh** (**-e**) ou la variable d'environnement **RSYNC_RSH** de la commande **rsync** standard comme solution de contournement, comme suit :

```
$ rsync --rsh='oc rsh' --exclude-from=FILE SRC POD:DEST
```

ou :

Exporter la variable **RSYNC_RSH**:

```
$ export RSYNC_RSH='oc rsh'
```

Exécutez ensuite la commande rsync :

```
$ rsync --exclude-from=FILE SRC POD:DEST
```

Les deux exemples ci-dessus configurent le logiciel standard **rsync** pour qu'il utilise **oc rsh** comme programme shell distant afin de lui permettre de se connecter au pod distant, et constituent une alternative à l'exécution de **oc rsync**.

6.7. EXÉCUTER DES COMMANDES À DISTANCE DANS UN CONTENEUR OPENSHIFT CONTAINER PLATFORM

Vous pouvez utiliser le CLI pour exécuter des commandes à distance dans un conteneur OpenShift Container Platform.

6.7.1. Exécuter des commandes à distance dans des conteneurs

La prise en charge de l'exécution à distance des commandes de conteneurs est intégrée à la CLI.

Procédure

Pour exécuter une commande dans un conteneur :

```
$ oc exec <pod> [-c <container>] <command> [<arg_1> ... <arg_n>]
```

Par exemple :

```
$ oc exec mypod date
```

Exemple de sortie

```
Thu Apr  9 02:21:53 UTC 2015
```

IMPORTANT

Pour des raisons de sécurité, la commande **oc exec** ne fonctionne pas lors de l'accès à des conteneurs privilégiés, sauf si la commande est exécutée par un utilisateur **cluster-admin**.

6.7.2. Protocole pour initier une commande à distance à partir d'un client

Les clients lancent l'exécution d'une commande à distance dans un conteneur en envoyant une demande au serveur API de Kubernetes :

```
/proxy/nodes/<node_name>/exec/<namespace>/<pod>/<container>?command=<command>
```

Dans l'URL ci-dessus :

- **<node_name>** est le FQDN du nœud.
- **<namespace>** est le projet du pod cible.
- **<pod>** est le nom du module cible.
- **<container>** est le nom du conteneur cible.
- **<command>** est la commande à exécuter.

Par exemple :

```
/proxy/nodes/node123.openshift.com/exec/myns/mypod/mycontainer?command=date
```

En outre, le client peut ajouter des paramètres à la demande pour indiquer si :

- le client doit envoyer des données à la commande du conteneur distant (stdin).
- le terminal du client est un ATS.
- la commande du conteneur distant doit envoyer la sortie de stdout au client.
- la commande du conteneur distant doit envoyer la sortie de stderr au client.

Après avoir envoyé une demande **exec** au serveur API, le client met à niveau la connexion pour qu'elle prenne en charge les flux multiplexés ; la mise en œuvre actuelle utilise **HTTP/2**.

Le client crée un flux pour stdin, stdout et stderr. Pour distinguer les flux, le client définit l'en-tête **streamType** du flux sur l'une des valeurs suivantes : **stdin**, **stdout** ou **stderr**.

Le client ferme tous les flux, la connexion améliorée et la connexion sous-jacente lorsqu'il a terminé la demande d'exécution de la commande à distance.

6.8. UTILISER LA REDIRECTION DE PORT POUR ACCÉDER AUX APPLICATIONS DANS UN CONTENEUR

OpenShift Container Platform prend en charge la redirection des ports vers les pods.

6.8.1. Comprendre le transfert de port

Vous pouvez utiliser l'interface de commande pour transférer un ou plusieurs ports locaux vers un module. Cela vous permet d'écouter localement sur un port donné ou aléatoire, et d'avoir des données transférées vers et depuis des ports donnés dans le pod.

La prise en charge de la redirection de port est intégrée à la CLI :

```
$ oc port-forward <pod> [<local_port> :]<remote_port> [...[<local_port_n> :]<remote_port_n>]
```

La CLI écoute sur chaque port local spécifié par l'utilisateur, en utilisant le protocole décrit ci-dessous.

Les ports peuvent être spécifiés en utilisant les formats suivants :

5000	Le client écoute sur le port 5000 localement et transmet à 5000 dans le pod.
6000:5000	Le client écoute sur le port 6000 localement et transmet à 5000 dans le pod.
:5000 ou **0:5000**	Le client sélectionne un port local libre et le transmet à 5000 dans le pod.

OpenShift Container Platform gère les demandes de transfert de port des clients. À la réception d'une demande, OpenShift Container Platform met à jour la réponse et attend que le client crée des flux de transfert de port. Lorsque OpenShift Container Platform reçoit un nouveau flux, il copie les données

entre le flux et le port du pod.

D'un point de vue architectural, il existe des options de transfert vers le port d'un pod. L'implémentation prise en charge par OpenShift Container Platform invoque **nsenter** directement sur l'hôte du nœud pour entrer dans l'espace de noms réseau du pod, puis invoque **socat** pour copier les données entre le flux et le port du pod. Cependant, une implémentation personnalisée pourrait inclure l'exécution d'un pod *helper* qui exécute ensuite **nsenter** et **socat**, de sorte que ces binaires ne doivent pas être installés sur l'hôte.

6.8.2. Utilisation de la redirection de port

Vous pouvez utiliser le CLI pour transférer un ou plusieurs ports locaux vers un pod.

Procédure

La commande suivante permet d'écouter le port spécifié dans un pod :

```
$ oc port-forward <pod> [<local_port> :]<remote_port> [...[<local_port_n> :]<remote_port_n>]
```

Par exemple :

- Utilisez la commande suivante pour écouter les ports **5000** et **6000** localement et transmettre des données vers et depuis les ports **5000** et **6000** dans le pod :

  ```
  $ oc port-forward <pod> 5000 6000
  ```

  Exemple de sortie

  ```
  Forwarding from 127.0.0.1:5000 -> 5000
  Forwarding from [::1]:5000 -> 5000
  Forwarding from 127.0.0.1:6000 -> 6000
  Forwarding from [::1]:6000 -> 6000
  ```

- Utilisez la commande suivante pour écouter le port **8888** localement et le transférer à **5000** dans le pod :

  ```
  $ oc port-forward <pod> 8888:5000
  ```

  Exemple de sortie

  ```
  Forwarding from 127.0.0.1:8888 -> 5000
  Forwarding from [::1]:8888 -> 5000
  ```

- Utilisez la commande suivante pour écouter sur un port libre localement et transmettre à **5000** dans le pod :

  ```
  $ oc port-forward <pod> :5000
  ```

  Exemple de sortie

  ```
  Forwarding from 127.0.0.1:42390 -> 5000
  Forwarding from [::1]:42390 -> 5000
  ```

Ou bien :

```
$ oc port-forward <pod> 0:5000
```

6.8.3. Protocole pour initier le transfert de port à partir d'un client

Les clients initient le transfert de port vers un pod en émettant une demande au serveur API de Kubernetes :

```
/proxy/nodes/<node_name>/portForward/<namespace>/<pod>
```

Dans l'URL ci-dessus :

- **<node_name>** est le FQDN du nœud.
- **<namespace>** est l'espace de noms du pod cible.
- **<pod>** est le nom du module cible.

Par exemple :

```
/proxy/nodes/node123.openshift.com/portForward/myns/mypod
```

Après avoir envoyé une demande de transfert de port au serveur de l'API, le client met à niveau la connexion pour qu'elle prenne en charge les flux multiplexés Hyptertext Transfer Protocol Version 2 (HTTP/2).

Le client crée un flux avec l'en-tête **port** contenant le port cible dans le pod. Toutes les données écrites dans le flux sont transmises via le kubelet au pod et au port cibles. De même, toutes les données envoyées depuis le module pour cette connexion transférée sont renvoyées vers le même flux dans le client.

Le client ferme tous les flux, la connexion améliorée et la connexion sous-jacente lorsqu'il a terminé la demande de transfert de port.

6.9. UTILISATION DE SYSCTLS DANS LES CONTENEURS

Les paramètres sysctl sont exposés à travers Kubernetes, ce qui permet aux utilisateurs de modifier certains paramètres du noyau au moment de l'exécution. Seuls les sysctls qui sont namespaced peuvent être définis indépendamment sur les pods. Si un sysctl n'est pas namespaced, appelé *node-level*, vous devez utiliser une autre méthode pour définir le sysctl, par exemple en utilisant l'opérateur Node Tuning.

Les sysctls de réseau sont une catégorie spéciale de sysctl. Les sysctls de réseau comprennent

- Les sysctls de l'ensemble du système, par exemple **net.ipv4.ip_local_port_range**, qui sont valables pour toute la mise en réseau. Vous pouvez les définir indépendamment pour chaque module d'un nœud.
- Les sysctls spécifiques à l'interface, par exemple **net.ipv4.conf.IFNAME.accept_local**, qui ne s'appliquent qu'à une interface réseau supplémentaire spécifique pour un pod donné. Vous pouvez les définir indépendamment pour chaque configuration de réseau supplémentaire. Vous les définissez à l'aide d'une configuration sur le site **tuning-cni** après la création des interfaces réseau.

En outre, seuls les sysctls considérés comme *safe* sont inscrits sur la liste blanche par défaut ; vous pouvez activer manuellement d'autres sysctls *unsafe* sur le nœud pour qu'ils soient accessibles à l'utilisateur.

Ressources supplémentaires

- Opérateur de réglage des nœuds

6.9.1. À propos de sysctls

Sous Linux, l'interface sysctl permet à un administrateur de modifier les paramètres du noyau au moment de l'exécution. Les paramètres sont disponibles dans le système de fichiers ***/proc/sys/*** le système de fichiers des processus virtuels. Les paramètres couvrent différents sous-systèmes, tels que :

- noyau (préfixe commun : ***kernel.***)
- la mise en réseau (préfixe commun : ***net.***)
- la mémoire virtuelle (préfixe commun : ***vm.***)
- MDADM (préfixe commun : ***dev.***)

D'autres sous-systèmes sont décrits dans la documentation du noyau. Pour obtenir une liste de tous les paramètres, exécutez :

```
$ sudo sysctl -a
```

6.9.2. Sysctl à espace de noms et à niveau de nœud

Un certain nombre de sysctls se trouvent à l'adresse *namespaced* dans les noyaux Linux. Cela signifie que vous pouvez les définir indépendamment pour chaque pod sur un nœud. Le fait d'être namespaced est une exigence pour que les sysctls soient accessibles dans un contexte de pod au sein de Kubernetes.

Les sysctls suivants sont connus pour être des espaces de noms :

- ***kernel.shm****
- ***kernel.msg****
- ***kernel.sem***
- ***fs.mqueue.****

En outre, la plupart des sysctls du groupe **net.*** sont connus pour leur espace de noms. Leur adoption de l'espace de noms diffère en fonction de la version du noyau et du distributeur.

Les sysctls qui n'ont pas d'espace de noms sont appelés *node-level* et doivent être définis manuellement par l'administrateur du cluster, soit au moyen de la distribution Linux sous-jacente des nœuds, par exemple en modifiant le fichier ***/etc/sysctls.conf*** ou en utilisant un démon avec des conteneurs privilégiés. Vous pouvez utiliser l'opérateur Node Tuning pour définir *node-level* sysctls.

NOTE

Envisager de marquer les nœuds avec des sysctls spéciaux comme tainted. Ne planifiez sur ces nœuds que les pods qui ont besoin de ces paramètres sysctl. Utilisez la fonctionnalité taints and toleration pour marquer les noeuds.

6.9.3. Sysctls sûrs et non sûrs

Les sysctls sont regroupés en *safe* et *unsafe* sysctls.

Pour que les sysctls à l'échelle du système soient considérés comme sûrs, ils doivent avoir un espace de noms. Un sysctl à espace de noms garantit l'isolation entre les espaces de noms et, par conséquent, entre les modules. Si vous définissez un sysctl pour un pod, il ne doit ajouter aucun des éléments suivants :

- Influencer tout autre pod sur le nœud
- Nuisent à la santé du nœud
- Obtenir des ressources de CPU ou de mémoire en dehors des limites de ressources d'un pod

NOTE

L'espacement des noms n'est pas suffisant pour que le sysctl soit considéré comme sûr.

Tout sysctl qui n'est pas ajouté à la liste des sysctls autorisés sur OpenShift Container Platform est considéré comme dangereux pour OpenShift Container Platform.

Les sysctls non sécurisés ne sont pas autorisés par défaut. Pour les sysctls à l'échelle du système, l'administrateur du cluster doit les activer manuellement pour chaque nœud. Les pods dont les sysctls non sécurisés sont désactivés sont planifiés mais ne se lancent pas.

NOTE

Vous ne pouvez pas activer manuellement les sysctls non sécurisés spécifiques à l'interface.

OpenShift Container Platform ajoute les sysctls sûrs suivants à l'échelle du système et spécifiques à l'interface à une liste de sysctls sûrs autorisés :

Tableau 6.4. Sysctls sûrs pour l'ensemble du système

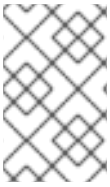
sysctl	Description
kernel.shm_rmid_forced	Lorsqu'il vaut **1**, tous les objets de mémoire partagée dans l'espace de noms IPC actuel sont automatiquement forcés d'utiliser IPC_RMID. Pour plus d'informations, voir shm_rmid_forced.

sysctl	Description
net.ipv4.ip_local_port_range	Définit la plage de ports locaux utilisée par TCP et UDP pour choisir le port local. Le premier chiffre est le premier numéro de port, et le second est le dernier numéro de port local. Si possible, il est préférable que ces numéros aient une parité différente (une valeur paire et une valeur impaire). Ils doivent être supérieurs ou égaux à **ip_unprivileged_port_start**. Les valeurs par défaut sont respectivement **32768** et **60999**. Pour plus d'informations, voir ip_local_port_range.
net.ipv4.tcp_syncookies	Lorsque **net.ipv4.tcp_syncookies** est activé, le noyau traite les paquets TCP SYN normalement jusqu'à ce que la file d'attente des connexions semi-ouvertes soit pleine, moment où la fonctionnalité de cookie SYN entre en jeu. Cette fonctionnalité permet au système de continuer à accepter des connexions valides, même en cas d'attaque par déni de service. Pour plus d'informations, voir tcp_syncookies.
net.ipv4.ping_group_range	Cela limite les sockets de datagramme **ICMP_PROTO** aux utilisateurs de l'intervalle de groupes. La valeur par défaut est **1 0**, ce qui signifie que personne, pas même root, ne peut créer de sockets ping. Pour plus d'informations, voir ping_group_range.
net.ipv4.ip_unprivileged_port_start	Cette valeur définit le premier port non privilégié dans l'espace de noms du réseau. Pour désactiver tous les ports privilégiés, définissez cette valeur à **0**. Les ports privilégiés ne doivent pas chevaucher le port **ip_local_port_range**. Pour plus d'informations, voir ip_unprivileged_port_start.

Tableau 6.5. Sysctls de sécurité spécifiques à l'interface

sysctl	Description
net.ipv4.conf.IFNAME.accept_redirects	Accepter les messages de redirection ICMP IPv4.
net.ipv4.conf.IFNAME.accept_source_route	Accepter les paquets IPv4 avec l'option strict source route (SRR).
net.ipv4.conf.IFNAME.arp_accept	Définir le comportement pour les trames ARP gratuites avec une adresse IPv4 qui n'est pas déjà présente dans la table ARP : • **0** - Ne pas créer de nouvelles entrées dans la table ARP. • **1** - Créer de nouvelles entrées dans la table ARP.
net.ipv4.conf.IFNAME.arp_notify	Définir le mode de notification des changements d'adresse IPv4 et de périphérique.

sysctl	Description
net.ipv4.conf.IFNAME.disable_policy	Désactiver la politique IPSEC (SPD) pour cette interface IPv4.
net.ipv4.conf.IFNAME.secure_redirects	Accepter les messages de redirection ICMP uniquement vers les passerelles figurant dans la liste des passerelles actuelles de l'interface.
net.ipv4.conf.IFNAME.send_redirects	L'option Envoyer des redirections n'est activée que si le nœud agit comme un routeur. En d'autres termes, un hôte ne doit pas envoyer de message de redirection ICMP. Il est utilisé par les routeurs pour informer l'hôte de l'existence d'un meilleur chemin de routage pour une destination donnée.
net.ipv6.conf.IFNAME.accept_ra	Accepter les annonces de routeur IPv6 ; autoconfigurer à l'aide de ces annonces. Il détermine également s'il faut ou non transmettre les sollicitations du routeur. Les sollicitations de routeur ne sont transmises que si le paramètre fonctionnel est l'acceptation des annonces de routeur.
net.ipv6.conf.IFNAME.accept_redirects	Accepter les messages de redirection ICMP IPv6.
net.ipv6.conf.IFNAME.accept_source_route	Accepter les paquets IPv6 avec l'option SRR.
net.ipv6.conf.IFNAME.arp_accept	Définir le comportement pour les trames ARP gratuites avec une adresse IPv6 qui n'est pas déjà présente dans la table ARP : • **0** - Ne pas créer de nouvelles entrées dans la table ARP. • **1** - Créer de nouvelles entrées dans la table ARP.
net.ipv6.conf.IFNAME.arp_notify	Définir le mode de notification des changements d'adresse IPv6 et de périphérique.
net.ipv6.neigh.IFNAME.base_reachable_time_ms	Ce paramètre contrôle la durée de vie de la correspondance entre l'adresse matérielle et l'adresse IP dans la table de voisinage pour IPv6.
net.ipv6.neigh.IFNAME.retrans_time_ms	Définir le délai de retransmission des messages de découverte du voisin.

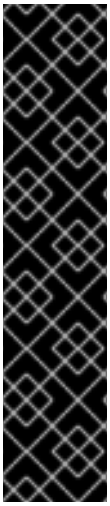


NOTE

Lorsque vous définissez ces valeurs à l'aide du plugin CNI **tuning**, utilisez littéralement la valeur **IFNAME**. Le nom de l'interface est représenté par le jeton **IFNAME** et est remplacé par le nom réel de l'interface au moment de l'exécution.

6.9.4. Mise à jour de la liste des sysctls sûrs spécifiques à l'interface

OpenShift Container Platform inclut une liste prédéfinie d'interfaces sûres spécifiques **sysctls**. Vous pouvez modifier cette liste en mettant à jour le **cni-sysctl-allowlist** dans l'espace de noms **openshift-multus**.



IMPORTANT

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

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

Suivez cette procédure pour modifier la liste prédéfinie des sites sûrs **sysctls**. Cette procédure décrit comment étendre la liste d'autorisations par défaut.

Procédure

1. Affichez la liste prédéfinie existante en exécutant la commande suivante :

```
$ oc get cm -n openshift-multus cni-sysctl-allowlist -oyaml
```

Résultats attendus

```
apiVersion: v1
data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
kind: ConfigMap
metadata:
  annotations:
    kubernetes.io/description: |
      Sysctl allowlist for nodes.
    release.openshift.io/version: 4.12.0-0.nightly-2022-11-16-003434
  creationTimestamp: "2022-11-17T14:09:27Z"
  name: cni-sysctl-allowlist
```

```
  namespace: openshift-multus
  resourceVersion: "2422"
  uid: 96d138a3-160e-4943-90ff-6108fa7c50c3
```

2. Modifiez la liste à l'aide de la commande suivante :

```
$ oc edit cm -n openshift-multus cni-sysctl-allowlist -oyaml
```

Par exemple, pour vous permettre de mettre en œuvre un transfert de chemin inverse plus strict, vous devez ajouter **^net.ipv4.conf.IFNAME.rp_filter$** et **^net.ipv6.conf.IFNAME.rp_filter$** à la liste, comme indiqué ici :

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  allowlist.conf: |-
    ^net.ipv4.conf.IFNAME.accept_redirects$
    ^net.ipv4.conf.IFNAME.accept_source_route$
    ^net.ipv4.conf.IFNAME.arp_accept$
    ^net.ipv4.conf.IFNAME.arp_notify$
    ^net.ipv4.conf.IFNAME.disable_policy$
    ^net.ipv4.conf.IFNAME.secure_redirects$
    ^net.ipv4.conf.IFNAME.send_redirects$
    ^net.ipv4.conf.IFNAME.rp_filter$
    ^net.ipv6.conf.IFNAME.accept_ra$
    ^net.ipv6.conf.IFNAME.accept_redirects$
    ^net.ipv6.conf.IFNAME.accept_source_route$
    ^net.ipv6.conf.IFNAME.arp_accept$
    ^net.ipv6.conf.IFNAME.arp_notify$
    ^net.ipv6.neigh.IFNAME.base_reachable_time_ms$
    ^net.ipv6.neigh.IFNAME.retrans_time_ms$
    ^net.ipv6.conf.IFNAME.rp_filter$
```

3. Enregistrez les modifications dans le fichier et quittez.

NOTE

La suppression de **sysctls** est également possible. Modifiez le fichier, supprimez **sysctl** ou **sysctls**, puis enregistrez les modifications et quittez.

Vérification

Suivez cette procédure pour mettre en œuvre un transfert de chemin inverse plus strict pour IPv4. Pour plus d'informations sur le transfert de chemin inverse, voir Transfert de chemin inverse .

1. Créez une définition de pièce jointe au réseau, telle que **reverse-path-fwd-example.yaml**, avec le contenu suivant :

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tuningnad
```

```
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tuningnad",
    "plugins": [{
      "type": "bridge"
      },
      {
      "type": "tuning",
      "sysctl": {
        "net.ipv4.conf.IFNAME.rp_filter": "1"
       }
    }
  ]
}'
```

2. Appliquez le fichier yaml en exécutant la commande suivante :

```
$ oc apply -f reverse-path-fwd-example.yaml
```

Exemple de sortie

```
networkattachmentdefinition.k8.cni.cncf.io/tuningnad created
```

3. Créez un pod tel que **examplepod.yaml** en utilisant le YAML suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: example
  labels:
    app: httpd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad (1)
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: httpd
      image: 'image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest'
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
```

(1) Spécifiez le nom du site configuré **NetworkAttachmentDefinition**.

4. Appliquez le fichier yaml en exécutant la commande suivante :

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

5. Vérifiez que le module est créé en exécutant la commande suivante :

```
$ oc get pod
```

Exemple de sortie

```
NAME      READY   STATUS    RESTARTS   AGE
example   1/1     Running   0          47s
```

6. Connectez-vous au module en exécutant la commande suivante :

```
$ oc rsh example
```

7. Vérifiez la valeur de l'indicateur sysctl configuré. Par exemple, trouvez la valeur **net.ipv4.conf.net1.rp_filter** en exécutant la commande suivante :

```
sh-4.4# sysctl net.ipv4.conf.net1.rp_filter
```

Résultats attendus

```
net.ipv4.conf.net1.rp_filter = 1
```

Ressources supplémentaires

- Configuration du CNI d'accord
- Documentation sur les réseaux Linux

6.9.5. Démarrer un pod avec des sysctls sûrs

Vous pouvez définir des sysctls sur les pods à l'aide de la page **securityContext** du pod. Le site **securityContext** s'applique à tous les conteneurs du même pod.

Les sysctls sûrs sont autorisés par défaut.

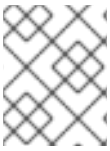
Cet exemple utilise le pod **securityContext** pour définir les sysctls de sécurité suivants :

- **kernel.shm_rmid_forced**
- **net.ipv4.ip_local_port_range**
- **net.ipv4.tcp_syncookies**
- **net.ipv4.ping_group_range**

AVERTISSEMENT

Pour éviter de déstabiliser votre système d'exploitation, ne modifiez les paramètres sysctl qu'après avoir compris leurs effets.

Cette procédure permet de démarrer un pod avec les paramètres sysctl configurés.



NOTE

Dans la plupart des cas, vous modifiez une définition de pod existante et ajoutez la spécification **securityContext**.

Procédure

1. Créez un fichier YAML **sysctl_pod.yaml** qui définit un exemple de pod et ajoutez la spécification **securityContext**, comme le montre l'exemple suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example
  namespace: default
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 (1)
      runAsGroup: 3000 (2)
      allowPrivilegeEscalation: false (3)
      capabilities: (4)
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true (5)
    seccompProfile: (6)
      type: RuntimeDefault
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "1"
    - name: net.ipv4.ip_local_port_range
      value: "32770       60666"
    - name: net.ipv4.tcp_syncookies
      value: "0"
    - name: net.ipv4.ping_group_range
      value: "0           200000000"
```

(1) **runAsUser** contrôle l'identifiant de l'utilisateur avec lequel le conteneur est exécuté.

2. **runAsGroup** contrôle l'identifiant du groupe primaire avec lequel les conteneurs sont exécutés.
3. **allowPrivilegeEscalation** détermine si un pod peut demander à autoriser l'escalade des privilèges. S'il n'est pas spécifié, la valeur par défaut est true. Ce booléen contrôle directement si le drapeau **no_new_privs** est activé sur le processus du conteneur.
4. **capabilities** permettent des actions privilégiées sans donner un accès complet à la racine. Cette politique permet de s'assurer que toutes les capacités sont supprimées du pod.
5. **runAsNonRoot: true** exige que le conteneur fonctionne avec un utilisateur dont l'UID est différent de 0.
6. **RuntimeDefault** active le profil seccomp par défaut pour un pod ou une charge de travail de conteneur.

2. Créez le pod en exécutant la commande suivante :

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

3. Vérifiez que le module est créé en exécutant la commande suivante :

```
$ oc get pod
```

Exemple de sortie

```
NAME              READY   STATUS          RESTARTS   AGE
sysctl-example    1/1     Running         0          14s
```

4. Connectez-vous au module en exécutant la commande suivante :

```
$ oc rsh sysctl-example
```

5. Vérifiez les valeurs des drapeaux sysctl configurés. Par exemple, trouvez la valeur **kernel.shm_rmid_forced** en exécutant la commande suivante :

```
sh-4.4# sysctl kernel.shm_rmid_forced
```

Résultats attendus

```
kernel.shm_rmid_forced = 1
```

6.9.6. Démarrer un pod avec des sysctls non sécurisés

Un pod avec des sysctls non sécurisés ne peut être lancé sur aucun nœud à moins que l'administrateur du cluster n'active explicitement les sysctls non sécurisés pour ce nœud. Comme pour les sysctls au niveau du nœud, utilisez la fonctionnalité de taints et de tolérance ou les étiquettes sur les nœuds pour planifier ces pods sur les bons nœuds.

L'exemple suivant utilise le pod **securityContext** pour définir un sysctl sûr **kernel.shm_rmid_forced** et deux sysctls non sûrs, **net.core.somaxconn** et **kernel.msgmax**. La spécification ne fait aucune distinction entre les sysctls *safe* et *unsafe*.

AVERTISSEMENT

Pour éviter de déstabiliser votre système d'exploitation, ne modifiez les paramètres sysctl qu'après avoir compris leurs effets.

L'exemple suivant illustre ce qui se passe lorsque vous ajoutez des sysctls sûrs et non sûrs à une spécification de pod :

Procédure

1. Créez un fichier YAML **sysctl-example-unsafe.yaml** qui définit un exemple de pod et ajoutez la spécification **securityContext**, comme le montre l'exemple suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-unsafe
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
    sysctls:
    - name: kernel.shm_rmid_forced
      value: "0"
    - name: net.core.somaxconn
      value: "1024"
    - name: kernel.msgmax
      value: "65536"
```

2. Créez le pod à l'aide de la commande suivante :

```
$ oc apply -f sysctl-example-unsafe.yaml
```

3. Vérifiez que le pod est planifié mais qu'il ne se déploie pas parce que les sysctls non sécurisés ne sont pas autorisés pour le nœud à l'aide de la commande suivante :

```
$ oc get pod
```

Exemple de sortie

```
NAME                     READY          STATUS          RESTARTS   AGE
sysctl-example-unsafe    0/1              SysctlForbidden   0          14s
```

6.9.7. Activation des sysctls non sûrs

Un administrateur de cluster peut autoriser certains sysctls non sécurisés dans des situations très particulières telles que des performances élevées ou le réglage d'applications en temps réel.

Si vous souhaitez utiliser des sysctls non sécurisés, un administrateur de cluster doit les activer individuellement pour un type de nœud spécifique. Les sysctls doivent être nommés dans l'espace de noms.

Vous pouvez contrôler davantage les sysctls définis dans les pods en spécifiant des listes de sysctls ou des modèles de sysctl dans le champ **allowedUnsafeSysctls** des contraintes du contexte de sécurité.

- L'option **allowedUnsafeSysctls** permet de répondre à des besoins spécifiques tels que l'optimisation d'applications à haute performance ou en temps réel.

AVERTISSEMENT

En raison de leur nature non sûre, l'utilisation de sysctls non sûrs est à vos propres risques et peut entraîner de graves problèmes, tels qu'un comportement inapproprié des conteneurs, une pénurie de ressources ou la rupture d'un nœud.

Procédure

1. Listez les objets MachineConfig existants pour votre cluster OpenShift Container Platform afin de décider comment étiqueter votre machine config en exécutant la commande suivante :

```
$ oc get machineconfigpool
```

Exemple de sortie

```
NAME     CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-bfb92f0cd1684e54d8e234ab7423cc96   True      False      False
3              3                    3                     0                      42m
worker   rendered-worker-21b6cb9a0f8919c88caf39db80ac1fce   True      False      False
3              3                    3                     0                      42m
```

2. Ajoutez une étiquette au pool de configuration de la machine où les conteneurs avec les sysctls non sécurisés seront exécutés en exécutant la commande suivante :

```
$ oc label machineconfigpool worker custom-kubelet=sysctl
```

3. Créer un fichier YAML **set-sysctl-worker.yaml** qui définit une ressource personnalisée (CR) **KubeletConfig**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-kubelet
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: sysctl 1
  kubeletConfig:
    allowedUnsafeSysctls: 2
      - "kernel.msg*"
      - "net.core.somaxconn"
```

1 Spécifiez l'étiquette du pool de configuration de la machine.

2 Dressez la liste des sysctls non sûrs que vous souhaitez autoriser.

4. Créez l'objet en exécutant la commande suivante :

```
$ oc apply -f set-sysctl-worker.yaml
```

5. Attendez que le Machine Config Operator génère la nouvelle configuration rendue et appliquez-la aux machines en exécutant la commande suivante :

```
$ oc get machineconfigpool worker -w
```

Après quelques minutes, l'état de **UPDATING** passe de Vrai à Faux :

```
NAME    CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
worker   rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800   False     True       False
3            2                 2                   0                    71m
worker   rendered-worker-f1704a00fc6f30d3a7de9a15fd68a800   False     True       False
3            2                 3                   0                    72m
worker   rendered-worker-0188658afe1f3a183ec8c4f14186f4d5   True      False      False
3            3                 3                   0                    72m
```

6. Créez un fichier YAML **sysctl-example-safe-unsafe.yaml** qui définit un exemple de pod et ajoutez la spécification **securityContext**, comme le montre l'exemple suivant :

```
apiVersion: v1
kind: Pod
metadata:
  name: sysctl-example-safe-unsafe
spec:
  containers:
  - name: podexample
    image: centos
    command: ["bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
```

```
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
    securityContext:
      runAsNonRoot: true
      seccompProfile:
        type: RuntimeDefault
      sysctls:
      - name: kernel.shm_rmid_forced
        value: "0"
      - name: net.core.somaxconn
        value: "1024"
      - name: kernel.msgmax
        value: "65536"
```

7. Créez le pod en exécutant la commande suivante :

```
$ oc apply -f sysctl-example-safe-unsafe.yaml
```

Résultats attendus

```
Warning: would violate PodSecurity "restricted:latest": forbidden sysctls
(net.core.somaxconn, kernel.msgmax)
pod/sysctl-example-safe-unsafe created
```

8. Vérifiez que le module est créé en exécutant la commande suivante :

```
$ oc get pod
```

Exemple de sortie

```
NAME                       READY   STATUS    RESTARTS   AGE
sysctl-example-safe-unsafe   1/1     Running   0          19s
```

9. Connectez-vous au module en exécutant la commande suivante :

```
$ oc rsh sysctl-example-safe-unsafe
```

10. Vérifiez les valeurs des drapeaux sysctl configurés. Par exemple, trouvez la valeur **net.core.somaxconn** en exécutant la commande suivante :

```
sh-4.4# sysctl net.core.somaxconn
```

Résultats attendus

```
net.core.somaxconn = 1024
```

Le sysctl non sécurisé est désormais autorisé et la valeur est définie comme indiqué dans la spécification **securityContext** de la spécification pod mise à jour.

6.9.8. Ressources supplémentaires

- Définition des sysctl de réseau au niveau de l'interface

CHAPITRE 7. TRAVAILLER AVEC DES CLUSTERS

7.1. VISUALISATION DES INFORMATIONS SUR LES ÉVÉNEMENTS SYSTÈME DANS UN CLUSTER OPENSHIFT CONTAINER PLATFORM

Les événements dans OpenShift Container Platform sont modélisés sur la base des événements qui se produisent sur les objets API dans un cluster OpenShift Container Platform.

7.1.1. Comprendre les événements

Les événements permettent à OpenShift Container Platform d'enregistrer des informations sur des événements du monde réel de manière agnostique en termes de ressources. Ils permettent également aux développeurs et aux administrateurs de consommer des informations sur les composants du système de manière unifiée.

7.1.2. Visualisation des événements à l'aide de la CLI

Vous pouvez obtenir une liste d'événements dans un projet donné à l'aide de l'interface de programmation.

Procédure

- Pour visualiser les événements d'un projet, utilisez la commande suivante :

```
$ oc get events [-n <projet>] 1
```

1 Le nom du projet.

Par exemple :

```
$ oc get events -n openshift-config
```

Exemple de sortie

```
LAST SEEN   TYPE      REASON                  OBJECT                    MESSAGE
97m         Normal    Scheduled               pod/dapi-env-test-pod     Successfully assigned
openshift-config/dapi-env-test-pod to ip-10-0-171-202.ec2.internal
97m         Normal    Pulling                 pod/dapi-env-test-pod     pulling image
"gcr.io/google_containers/busybox"
97m         Normal    Pulled                  pod/dapi-env-test-pod     Successfully pulled image
"gcr.io/google_containers/busybox"
97m         Normal    Created                 pod/dapi-env-test-pod     Created container
9m5s        Warning   FailedCreatePodSandBox  pod/dapi-volume-test-pod  Failed create
pod sandbox: rpc error: code = Unknown desc = failed to create pod network sandbox
k8s_dapi-volume-test-pod_openshift-config_6bc60c1f-452e-11e9-9140-
0eec59c23068_0(748c7a40db3d08c07fb4f9eba774bd5effe5f0d5090a242432a73eee66ba9e22
): Multus: Err adding pod to network "openshift-sdn": cannot set "openshift-sdn" ifname to
"eth0": no netns: failed to Statfs "/proc/33366/ns/net": no such file or directory
8m31s       Normal    Scheduled               pod/dapi-volume-test-pod  Successfully assigned
openshift-config/dapi-volume-test-pod to ip-10-0-171-202.ec2.internal
```

- Pour visualiser les événements de votre projet à partir de la console OpenShift Container Platform.

  1. Lancez la console OpenShift Container Platform.

  2. Cliquez sur **Home → Events** et sélectionnez votre projet.

  3. Déplacez-vous vers la ressource dont vous voulez voir les événements. Par exemple : **Home → Projects →** <nom-du-projet> **→** <nom-de-la-ressource>.
     De nombreux objets, tels que les pods et les déploiements, ont également leur propre onglet **Events**, qui affiche les événements liés à cet objet.

7.1.3. Liste des événements

Cette section décrit les événements de OpenShift Container Platform.

Tableau 7.1. Événements de configuration

Nom	Description
FailedValidation	Échec de la validation de la configuration du pod.

Tableau 7.2. Événements liés aux conteneurs

Nom	Description
BackOff	Le redémarrage à reculons a fait échouer le conteneur.
Created	Conteneur créé.
Failed	L'opération de traction/création/démarrage a échoué.
Killing	Mise à mort du conteneur.
Started	Le conteneur a démarré.
Preempting	Prédominance d'autres pods.
ExceededGracePeriod	La durée d'exécution du conteneur n'a pas arrêté le pod dans le délai de grâce spécifié.

Tableau 7.3. Événements de santé

Nom	Description
Unhealthy	Le conteneur est malsain.

Tableau 7.4. Evénements liés à l'image

Nom	Description
BackOff	Reculez Ctr Start, tirez sur l'image.
ErrImageNeverPull	Le site **NeverPull Policy** de l'image n'est pas respecté.
Failed	Échec de l'extraction de l'image.
InspectFailed	Échec de l'inspection de l'image.
Pulled	L'image a été extraite avec succès ou l'image du conteneur est déjà présente sur la machine.
Pulling	Tirer l'image.

Tableau 7.5. Événements du gestionnaire d'images

Nom	Description
FreeDiskSpaceFailed	Échec de l'espace disque libre.
InvalidDiskCapacity	La capacité du disque n'est pas valide.

Tableau 7.6. Événements du nœud

Nom	Description
FailedMount	Le montage du volume a échoué.
HostNetworkNotSupported	Le réseau hôte n'est pas pris en charge.
HostPortConflict	Conflit hôte/port.
KubeletSetupFailed	L'installation du Kubelet a échoué.
NilShaper	Forme non définie.
NodeNotReady	Le nœud n'est pas prêt.
NodeNotSchedulable	Le nœud n'est pas programmable.

Nom	Description
NodeReady	Le nœud est prêt.
NodeSchedulable	Le nœud est programmable.
NodeSelectorMismatching	Inadéquation du sélecteur de nœud.
OutOfDisk	Hors disque.
Rebooted	Le nœud a été redémarré.
Starting	Démarrage de kubelet.
FailedAttachVolume	Échec de l'attachement du volume.
FailedDetachVolume	Échec du détachement du volume.
VolumeResizeFailed	Échec de l'extension/réduction du volume.
VolumeResizeSuccessful	Augmentation/réduction du volume avec succès.
FileSystemResizeFailed	Échec de l'extension/réduction du système de fichiers.
FileSystemResizeSuccessful	Le système de fichiers a été étendu/réduit avec succès.
FailedUnMount	Échec du démontage du volume.
FailedMapVolume	Échec du mappage d'un volume.
FailedUnmapDevice	Échec de la mise en forme de l'appareil.
AlreadyMountedVolume	Le volume est déjà monté.
SuccessfulDetachVolume	Le volume est détaché avec succès.

Nom	Description
SuccessfulMountVolume	Le volume est monté avec succès.
SuccessfulUnMountVolume	Le volume a été démonté avec succès.
ContainerGCFailed	Le ramassage des ordures du conteneur a échoué.
ImageGCFailed	Le ramassage des images a échoué.
FailedNodeAllocatableEnforcement	Échec de l'application de la limite de Cgroup réservée au système.
NodeAllocatableEnforced	Limite du groupe C réservée au système.
UnsupportedMountOption	Option de montage non prise en charge.
SandboxChanged	Le bac à sable du pod a changé.
FailedCreatePodSandBox	Échec de la création d'un bac à sable pour pods.
FailedPodSandBoxStatus	Échec de l'état du bac à sable du pod.

Tableau 7.7. Événements du travailleur en pods

Nom	Description
FailedSync	La synchronisation des pods a échoué.

Tableau 7.8. Événements du système

Nom	Description
SystemOOM	Il y a une situation OOM (out of memory) sur le cluster.

Tableau 7.9. Événements en cosse

Nom	Description
FailedKillPod	Échec de l'arrêt d'un pod.
FailedCreatePodContainer	Échec de la création d'un conteneur de pods.
Failed	Échec de la création de répertoires de données de pods.
NetworkNotReady	Le réseau n'est pas prêt.
FailedCreate	Erreur de création : **<error-msg>**.
SuccessfulCreate	Création d'un pod : **<pod-name>**.
FailedDelete	Erreur de suppression : **<error-msg>**.
SuccessfulDelete	Pod supprimé : **<pod-id>**.

Tableau 7.10. Événements AutoScaler Pod Horizontal

Nom	Description
Sélecteur requis	Un sélecteur est nécessaire.
InvalidSelector	Impossible de convertir le sélecteur en un objet sélecteur interne correspondant.
FailedGetObjectMetric	HPA n'a pas été en mesure de calculer le nombre de répliques.
InvalidMetricSourceType	Type de source métrique inconnu.
ValidMetricFound	HPA a pu calculer avec succès le nombre de répliques.
FailedConvertHPA	Échec de la conversion de l'APH donné.
FailedGetScale	Le contrôleur HPA n'a pas pu obtenir l'échelle actuelle de la cible.
SucceededGetScale	Le contrôleur HPA a pu obtenir l'échelle actuelle de la cible.

Nom	Description
FailedComputeMetricsReplicas	N'a pas réussi à calculer le nombre de réplicas souhaité en fonction des métriques listées.
FailedRescale	Nouvelle taille : **<size>**; raison :**<msg>**; erreur :**<error-msg>**.
SuccessfulRescale	Nouvelle taille : **<size>**; raison :**<msg>**.
FailedUpdateStatus	Échec de la mise à jour de l'état.

Tableau 7.11. Événements réseau (openshift-sdn)

Nom	Description
Starting	Démarrer OpenShift SDN.
NetworkFailed	L'interface réseau du pod a été perdue et le pod sera arrêté.

Tableau 7.12. Événements réseau (kube-proxy)

Nom	Description
NeedPods	Le service-port **<serviceName>:<port>** a besoin de pods.

Tableau 7.13. Événements en volume

Nom	Description
FailedBinding	Aucun volume persistant n'est disponible et aucune classe de stockage n'est définie.
VolumeMismatch	La taille du volume ou la classe est différente de ce qui est demandé dans la demande.
VolumeFailedRecycle	Erreur lors de la création d'un module de recyclage.
VolumeRecycled	Se produit lorsque le volume est recyclé.
RecyclerPod	Se produit lorsque le pod est recyclé.
VolumeDelete	Se produit lorsque le volume est supprimé.

Nom	Description
VolumeFailedDelete	Erreur lors de la suppression du volume.
ExternalProvisioning	Se produit lorsque le volume de la demande est approvisionné soit manuellement, soit par un logiciel externe.
ProvisioningFailed	Échec de l'approvisionnement du volume.
ProvisioningCleanupFailed	Erreur de nettoyage du volume provisionné.
ProvisioningSucceeded	Se produit lorsque le volume est approvisionné avec succès.
WaitForFirstConsumer	Retarder la fixation jusqu'à l'ordonnancement du pod.

Tableau 7.14. Crochets du cycle de vie

Nom	Description
FailedPostStartHook	Le gestionnaire a échoué pour le démarrage du pod.
FailedPreStopHook	Le gestionnaire a échoué pour le préarrêt.
UnfinishedPreStopHook	Crochet de préarrêt inachevé.

Tableau 7.15. Déploiements

Nom	Description
DeploymentCancellationFailed	Échec de l'annulation du déploiement.
DeploymentCancelled	Déploiement annulé.
DeploymentCreated	Création d'un nouveau contrôleur de réplication.
IngressIPRangeFull	Pas d'IP d'entrée disponible à allouer au service.

Tableau 7.16. Événements du planificateur

Nom	Description
FailedScheduling	Échec de la programmation du pod : **<pod-namespace>/<pod-name>**. Cet événement est déclenché pour de multiples raisons, par exemple : **AssumePodVolumes** a échoué, la liaison a été rejetée, etc.
Preempted	Par **<preemptor-namespace>/<preemptor-name>** sur le nœud **<node-name>**.
Scheduled	L'attribution de **<pod-name>** à **<node-name>** a été couronnée de succès.

Tableau 7.17. Événements de l'ensemble des démons

Nom	Description
SelectingAll	Cet ensemble de démons sélectionne tous les pods. Un sélecteur non vide est nécessaire.
FailedPlacement	Échec du placement d'un pod sur **<node-name>**.
FailedDaemonPod	Le daemon pod **<pod-name>** a échoué sur le nœud **<node-name>**, il va essayer de le tuer.

Tableau 7.18. Événements du service LoadBalancer

Nom	Description
CreatingLoadBalancerFailed	Erreur lors de la création de l'équilibreur de charge.
DeletingLoadBalancer	Suppression de l'équilibreur de charge.
EnsuringLoadBalancer	Assurer l'équilibre de la charge.
EnsuredLoadBalancer	Équilibreur de charge assuré.
UnAvailableLoadBalancer	Il n'y a pas de nœuds disponibles pour le service **LoadBalancer**.
LoadBalancerSourceRanges	Liste les nouveaux **LoadBalancerSourceRanges**. Par exemple, **<old-source-range> → <new-source-range>**.
LoadbalancerIP	Liste la nouvelle adresse IP. Par exemple, **<old-ip> → <new-ip>**.

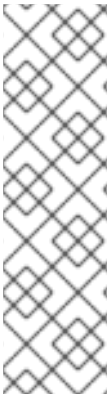
Nom	Description
ExternalIP	Liste des adresses IP externes. Par exemple, **Added: <external-ip>**.
UID	Liste le nouvel UID. Par exemple, **<old-service-uid> → <new-service-uid>**.
ExternalTrafficPolicy	Liste les nouveaux **ExternalTrafficPolicy**. Par exemple,**<old-policy> → <new-policy>**.
HealthCheckNodePort	Liste les nouveaux **HealthCheckNodePort**. Par exemple,**<old-node-port> → new-node-port>**.
UpdatedLoadBalancer	Mise à jour de l'équilibreur de charge avec de nouveaux hôtes.
LoadBalancerUpdateFailed	Erreur de mise à jour de l'équilibreur de charge avec les nouveaux hôtes.
DeletingLoadBalancer	Suppression de l'équilibreur de charge.
DeletingLoadBalancerFailed	Erreur de suppression de l'équilibreur de charge.
DeletedLoadBalancer	L'équilibreur de charge a été supprimé.

7.2. ESTIMATION DU NOMBRE DE PODS QUE PEUVENT CONTENIR LES NŒUDS D'OPENSHIFT CONTAINER PLATFORM

En tant qu'administrateur de cluster, vous pouvez utiliser l'outil de capacité de cluster pour visualiser le nombre de pods qui peuvent être planifiés pour augmenter les ressources actuelles avant qu'elles ne soient épuisées, et pour garantir que tous les pods futurs pourront être planifiés. Cette capacité provient d'un hôte de nœud individuel dans une grappe et comprend le CPU, la mémoire, l'espace disque et d'autres éléments.

7.2.1. Comprendre l'outil de capacité de cluster d'OpenShift Container Platform

L'outil de capacité de la grappe simule une séquence de décisions d'ordonnancement afin de déterminer le nombre d'instances d'un pod d'entrée qui peuvent être ordonnancées sur la grappe avant qu'elle n'épuise ses ressources, ce qui permet d'obtenir une estimation plus précise.



NOTE

La capacité allouable restante est une estimation approximative, car elle ne tient pas compte de toutes les ressources distribuées entre les nœuds. Elle analyse uniquement les ressources restantes et estime la capacité disponible qui est encore consommable en termes de nombre d'instances d'un pod avec des exigences données qui peuvent être planifiées dans un cluster.

En outre, les modules ne peuvent être programmés que sur des ensembles particuliers de nœuds en fonction de leurs critères de sélection et d'affinité. Par conséquent, il peut être difficile d'estimer quels pods restants une grappe peut programmer.

Vous pouvez exécuter l'outil d'analyse de la capacité des clusters en tant qu'utilitaire autonome à partir de la ligne de commande ou en tant que tâche dans un pod au sein d'un cluster OpenShift Container Platform. L'exécution en tant que tâche à l'intérieur d'un pod vous permet de l'exécuter plusieurs fois sans intervention.

7.2.2. Exécution de l'outil de capacité de cluster sur la ligne de commande

Vous pouvez exécuter l'outil OpenShift Container Platform cluster capacity à partir de la ligne de commande pour estimer le nombre de pods qui peuvent être planifiés sur votre cluster.

Conditions préalables

- Exécutez l'outil OpenShift Cluster Capacity Tool , qui est disponible sous forme d'image de conteneur à partir du catalogue de l'écosystème Red Hat.
- Créez un exemple de fichier spec **Pod**, que l'outil utilise pour estimer l'utilisation des ressources. Le fichier **podspec** spécifie ses besoins en ressources sous la forme **limits** ou **requests**. L'outil de calcul de la capacité des clusters prend en compte les besoins en ressources du pod dans son analyse d'estimation.
  Voici un exemple de l'entrée **Pod** spec :

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
      requests:
        cpu: 150m
        memory: 100Mi
```

Procédure

Pour utiliser l'outil de capacité de cluster sur la ligne de commande :

1. À partir du terminal, connectez-vous au Red Hat Registry :

```
$ podman login registry.redhat.io
```

2. Tirez l'image de l'outil de capacité de la grappe :

```
$ podman pull registry.redhat.io/openshift4/ose-cluster-capacity
```

3. Exécutez l'outil de calcul de la capacité des clusters :

```
$ podman run -v $HOME/.kube:/kube:Z -v $(pwd):/cc:Z  ose-cluster-capacity \
/bin/cluster-capacity --kubeconfig /kube/config --podspec /cc/pod-spec.yaml \
--verbose (1)
```

(1) Vous pouvez également ajouter l'option **--verbose** pour obtenir une description détaillée du nombre de pods pouvant être planifiés sur chaque nœud du cluster.

Exemple de sortie

```
small-pod pod requirements:
 - CPU: 150m
 - Memory: 100Mi

The cluster can schedule 88 instance(s) of the pod small-pod.

Termination reason: Unschedulable: 0/5 nodes are available: 2 Insufficient cpu,
3 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't
tolerate.

Pod distribution among nodes:
small-pod
 - 192.168.124.214: 45 instance(s)
 - 192.168.124.120: 43 instance(s)
```

Dans l'exemple ci-dessus, le nombre de pods estimés qui peuvent être programmés sur le cluster est de 88.

7.2.3. Exécution de l'outil de capacité de cluster en tant que tâche à l'intérieur d'un pod

L'exécution de l'outil de capacité de cluster en tant que tâche à l'intérieur d'un pod présente l'avantage de pouvoir être exécutée plusieurs fois sans nécessiter d'intervention de la part de l'utilisateur. L'exécution de l'outil de capacité de cluster en tant que tâche implique l'utilisation d'un objet **ConfigMap**.

Conditions préalables

Téléchargez et installez l'outil de calcul de la capacité des clusters .

Procédure

Pour exécuter l'outil de capacité de cluster :

1. Créer le rôle de cluster :

```
$ cat << EOF| oc create -f -
```

Exemple de sortie

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-capacity-role
rules:
- apiGroups: [""]
  resources: ["pods", "nodes", "persistentvolumeclaims", "persistentvolumes", "services",
"replicationcontrollers"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["apps"]
  resources: ["replicasets", "statefulsets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["policy"]
  resources: ["poddisruptionbudgets"]
  verbs: ["get", "watch", "list"]
- apiGroups: ["storage.k8s.io"]
  resources: ["storageclasses"]
  verbs: ["get", "watch", "list"]
EOF
```

2. Créer le compte de service :

```
$ oc create sa cluster-capacity-sa
```

3. Ajouter le rôle au compte de service :

```
$ oc adm policy add-cluster-role-to-user cluster-capacity-role \
    system:serviceaccount:default:cluster-capacity-sa
```

4. Définir et créer la spécification **Pod**:

```
apiVersion: v1
kind: Pod
metadata:
  name: small-pod
  labels:
    app: guestbook
    tier: frontend
spec:
  containers:
  - name: php-redis
    image: gcr.io/google-samples/gb-frontend:v4
    imagePullPolicy: Always
    resources:
      limits:
        cpu: 150m
        memory: 100Mi
```

```
      requests:
        cpu: 150m
        memory: 100Mi
```

5. L'analyse de la capacité du cluster est montée dans un volume à l'aide d'un objet **ConfigMap** nommé **cluster-capacity-configmap** pour monter le fichier d'entrée pod spec **pod.yaml** dans un volume **test-volume** au chemin **/test-pod**.
   Si vous n'avez pas créé d'objet **ConfigMap**, créez-en un avant de créer le travail :

```
$ oc create configmap cluster-capacity-configmap \
    --from-file=pod.yaml=pod.yaml
```

6. Créez le travail en utilisant l'exemple ci-dessous de fichier de spécification de travail :

```
apiVersion: batch/v1
kind: Job
metadata:
  name: cluster-capacity-job
spec:
  parallelism: 1
  completions: 1
  template:
    metadata:
      name: cluster-capacity-pod
    spec:
       containers:
       - name: cluster-capacity
         image: openshift/origin-cluster-capacity
         imagePullPolicy: "Always"
         volumeMounts:
         - mountPath: /test-pod
           name: test-volume
         env:
         - name: CC_INCLUSTER (1)
           value: "true"
         command:
         - "/bin/sh"
         - "-ec"
         - |
           /bin/cluster-capacity --podspec=/test-pod/pod.yaml --verbose
       restartPolicy: "Never"
       serviceAccountName: cluster-capacity-sa
       volumes:
       - name: test-volume
         configMap:
           name: cluster-capacity-configmap
```

(1) Variable d'environnement obligatoire permettant à l'outil de capacité de cluster de savoir qu'il s'exécute dans un cluster en tant que pod.
La clé **pod.yaml** de l'objet **ConfigMap** est la même que le nom du fichier de spécification **Pod**, bien qu'elle ne soit pas obligatoire. Ainsi, le fichier spec du pod d'entrée est accessible à l'intérieur du pod en tant que **/test-pod/pod.yaml**.

7. Exécutez l'image de capacité de cluster en tant que tâche dans un module :

```
$ oc create -f cluster-capacity-job.yaml
```

8. Consultez les journaux de travail pour connaître le nombre de pods pouvant être planifiés dans le cluster :

```
$ oc logs jobs/cluster-capacity-job
```

Exemple de sortie

```
small-pod pod requirements:
        - CPU: 150m
        - Memory: 100Mi

The cluster can schedule 52 instance(s) of the pod small-pod.

Termination reason: Unschedulable: No nodes are available that match all of the
following predicates:: Insufficient cpu (2).

Pod distribution among nodes:
small-pod
        - 192.168.124.214: 26 instance(s)
        - 192.168.124.120: 26 instance(s)
```

7.3. RESTREINDRE LA CONSOMMATION DES RESSOURCES PAR DES PLAGES DE LIMITES

Par défaut, les conteneurs s'exécutent avec des ressources de calcul non limitées sur un cluster OpenShift Container Platform. Avec les plages de limites, vous pouvez restreindre la consommation de ressources pour des objets spécifiques dans un projet :

- des pods et des conteneurs : Vous pouvez définir des exigences minimales et maximales en matière de CPU et de mémoire pour les pods et leurs conteneurs.
- Flux d'images : Vous pouvez limiter le nombre d'images et de balises dans un objet **ImageStream**.
- Images : Vous pouvez limiter la taille des images qui peuvent être transférées vers un registre interne.
- Réclamations de volumes persistants (PVC) : Vous pouvez limiter la taille des PVC qui peuvent être demandés.

Si un pod ne respecte pas les contraintes imposées par la plage de limites, le pod ne peut pas être créé dans l'espace de noms.

7.3.1. À propos des plages de limites

Une plage de limites, définie par un objet **LimitRange**, restreint la consommation de ressources dans un projet. Dans le projet, vous pouvez définir des limites de ressources spécifiques pour un pod, un conteneur, une image, un flux d'images ou une revendication de volume persistant (PVC).

Toutes les demandes de création et de modification de ressources sont évaluées par rapport à chaque objet **LimitRange** du projet. Si la ressource ne respecte pas l'une des contraintes énumérées, elle est rejetée.

L'illustration suivante montre un objet de plage de limites pour tous les composants : pod, conteneur, image, flux d'images ou PVC. Vous pouvez configurer des limites pour l'un ou l'ensemble de ces composants dans le même objet. Vous créez un objet de plage de limites différent pour chaque projet dans lequel vous souhaitez contrôler les ressources.

Objet de la plage de limites d'un échantillon pour un conteneur

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits"
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default:
        cpu: "300m"
        memory: "200Mi"
      defaultRequest:
        cpu: "200m"
        memory: "100Mi"
      maxLimitRequestRatio:
        cpu: "10"
```

7.3.1.1. Limites des composants

Les exemples suivants montrent les paramètres de la plage de limites pour chaque composant. Les exemples sont divisés pour plus de clarté. Vous pouvez créer un seul objet **LimitRange** pour un ou tous les composants si nécessaire.

7.3.1.1.1. Limites des conteneurs

Une plage de limites vous permet de spécifier le minimum et le maximum de CPU et de mémoire que chaque conteneur d'un pod peut demander pour un projet spécifique. Si un conteneur est créé dans le projet, les demandes de CPU et de mémoire du conteneur dans la spécification **Pod** doivent être conformes aux valeurs définies dans l'objet **LimitRange**. Si ce n'est pas le cas, le module n'est pas créé.

- La demande et la limite du processeur ou de la mémoire du conteneur doivent être supérieures ou égales à la contrainte de ressource **min** pour les conteneurs spécifiés dans l'objet **LimitRange**.

- La demande et la limite du processeur ou de la mémoire du conteneur doivent être inférieures ou égales à la contrainte de ressource **max** pour les conteneurs spécifiés dans l'objet **LimitRange**.
  Si l'objet **LimitRange** définit une unité centrale **max**, il n'est pas nécessaire de définir une valeur d'unité centrale **request** dans la spécification **Pod**. Mais vous devez spécifier une valeur de CPU **limit** qui satisfasse la contrainte de CPU maximale spécifiée dans la plage de limites.

- Le rapport entre les limites des conteneurs et les demandes doit être inférieur ou égal à la valeur **maxLimitRequestRatio** pour les conteneurs, spécifiée dans l'objet **LimitRange**.

Si l'objet **LimitRange** définit une contrainte **maxLimitRequestRatio**, tout nouveau conteneur doit avoir une valeur **request** et une valeur **limit**. OpenShift Container Platform calcule le ratio limite/demande en divisant la valeur **limit** par la valeur **request**. Cette valeur doit être un nombre entier non négatif supérieur à 1.

Par exemple, si un conteneur a **cpu: 500** dans la valeur **limit** et **cpu: 100** dans la valeur **request**, le ratio limite/demande pour **cpu** est **5**. Ce ratio doit être inférieur ou égal à la valeur **maxLimitRequestRatio**.

Si la spécification **Pod** ne précise pas de limite ou de mémoire de ressource de conteneur, les valeurs de CPU et de mémoire de **default** ou **defaultRequest** pour les conteneurs spécifiés dans l'objet de plage de limites sont attribuées au conteneur.

Conteneur LimitRange définition de l'objet

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: "Container"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "100m" 4
        memory: "4Mi" 5
      default:
        cpu: "300m" 6
        memory: "200Mi" 7
      defaultRequest:
        cpu: "200m" 8
        memory: "100Mi" 9
      maxLimitRequestRatio:
        cpu: "10" 10
```

1. Le nom de l'objet LimitRange.
2. La quantité maximale de CPU qu'un conteneur unique dans un pod peut demander.
3. Quantité maximale de mémoire qu'un conteneur unique d'un module peut demander.
4. La quantité minimale de CPU qu'un conteneur unique dans un pod peut demander.
5. Quantité minimale de mémoire qu'un conteneur unique d'un module peut demander.
6. La quantité par défaut de CPU qu'un conteneur peut utiliser si elle n'est pas spécifiée dans la spécification **Pod**.
7. La quantité de mémoire par défaut qu'un conteneur peut utiliser si elle n'est pas spécifiée dans la spécification **Pod**.
8. La quantité par défaut de CPU qu'un conteneur peut demander si elle n'est pas spécifiée dans la spécification **Pod**.

9 La quantité de mémoire par défaut qu'un conteneur peut demander si elle n'est pas spécifiée dans la spécification **Pod**.

10 Rapport maximum entre la limite et la demande pour un conteneur.

7.3.1.1.2. Limites du pod

Une plage de limites vous permet de spécifier les limites minimales et maximales de CPU et de mémoire pour tous les conteneurs d'un pod dans un projet donné. Pour créer un conteneur dans le projet, les demandes de CPU et de mémoire du conteneur dans la spécification **Pod** doivent être conformes aux valeurs définies dans l'objet **LimitRange**. Si ce n'est pas le cas, le module n'est pas créé.

Si la spécification **Pod** ne précise pas de limite ou de mémoire de ressource de conteneur, les valeurs de CPU et de mémoire de **default** ou **defaultRequest** pour les conteneurs spécifiés dans l'objet de plage de limites sont attribuées au conteneur.

Pour tous les conteneurs d'un module, les conditions suivantes doivent être remplies :

- La demande et la limite du processeur ou de la mémoire du conteneur doivent être supérieures ou égales aux contraintes de ressources de **min** pour les pods spécifiés dans l'objet **LimitRange**.
- La demande et la limite du processeur ou de la mémoire du conteneur doivent être inférieures ou égales aux contraintes de ressources de **max** pour les pods spécifiés dans l'objet **LimitRange**.
- Le rapport entre les limites des conteneurs et les demandes doit être inférieur ou égal à la contrainte **maxLimitRequestRatio** spécifiée dans l'objet **LimitRange**.

Pod LimitRange définition de l'objet

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: "Pod"
      max:
        cpu: "2" 2
        memory: "1Gi" 3
      min:
        cpu: "200m" 4
        memory: "6Mi" 5
      maxLimitRequestRatio:
        cpu: "10" 6
```

1 Le nom de l'objet de la plage de limites.

2 La quantité maximale de CPU qu'un pod peut demander pour tous les conteneurs.

3 La quantité maximale de mémoire qu'un pod peut demander pour tous les conteneurs.

4 La quantité minimale de CPU qu'un pod peut demander à tous les conteneurs.

5 La quantité minimale de mémoire qu'un pod peut demander à tous les conteneurs.

6 Rapport maximum entre la limite et la demande pour un conteneur.

7.3.1.1.3. Limites d'images

Un objet **LimitRange** vous permet de spécifier la taille maximale d'une image qui peut être poussée vers un registre d'images OpenShift.

Lorsque l'on pousse des images vers un registre d'images OpenShift, les points suivants doivent être respectés :

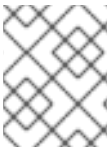
- **max** La taille de l'image doit être inférieure ou égale à la taille des images spécifiée dans l'objet **LimitRange**.

Image LimitRange Définition de l'objet

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: openshift.io/Image
      max:
        storage: 1Gi 2
```

1 Le nom de l'objet **LimitRange**.

2 La taille maximale d'une image qui peut être poussée vers un registre d'images OpenShift.



NOTE

Pour éviter que des blobs dépassant la limite ne soient téléchargés vers le registre, ce dernier doit être configuré pour appliquer des quotas.

AVERTISSEMENT

La taille de l'image n'est pas toujours disponible dans le manifeste d'une image téléchargée. C'est particulièrement le cas pour les images construites avec Docker 1.10 ou plus et poussées vers un registre v2. Si une telle image est tirée avec un ancien daemon Docker, le manifeste de l'image est converti par le registre en schéma v1 sans les informations de taille. Aucune limite de stockage fixée sur les images ne les empêche d'être téléchargées.

La question est en cours de traitement.

7.3.1.1.4. Limites du flux d'images

Un objet **LimitRange** vous permet de spécifier des limites pour les flux d'images.

Pour chaque flux d'images, les conditions suivantes doivent être remplies :

- Le nombre de balises d'image dans une spécification **ImageStream** doit être inférieur ou égal à la contrainte **openshift.io/image-tags** dans l'objet **LimitRange**.
- Le nombre de références uniques aux images dans une spécification **ImageStream** doit être inférieur ou égal à la contrainte **openshift.io/images** dans l'objet limit range.

Définition de l'objet Imagestream LimitRange

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" 1
spec:
  limits:
    - type: openshift.io/ImageStream
      max:
        openshift.io/image-tags: 20 2
        openshift.io/images: 30 3
```

1. Le nom de l'objet **LimitRange**.
2. Nombre maximal de balises d'image uniques dans le paramètre **imagestream.spec.tags** de la spécification du flux d'images.
3. Le nombre maximum de références d'images uniques dans le paramètre **imagestream.status.tags** de la spécification **imagestream**.

La ressource **openshift.io/image-tags** représente des références d'images uniques. Les références possibles sont an **ImageStreamTag**, an **ImageStreamImage** et a **DockerImage**. Les balises peuvent être créées à l'aide des commandes **oc tag** et **oc import-image**. Aucune distinction n'est faite entre les références internes et externes. Cependant, chaque référence unique étiquetée dans une spécification **ImageStream** n'est comptée qu'une seule fois. Cette spécification ne limite en rien les accès à un registre interne d'images de conteneurs, mais elle est utile pour la restriction des balises.

La ressource **openshift.io/images** représente des noms d'images uniques enregistrés dans le statut du flux d'images. Elle permet de restreindre le nombre d'images qui peuvent être poussées vers le registre d'images d'OpenShift. Les références internes et externes ne sont pas distinguées.

7.3.1.1.5. Limites des demandes d'indemnisation en cas de volume persistant

Un objet **LimitRange** vous permet de restreindre le stockage demandé dans une revendication de volume persistant (PVC).

Pour toutes les demandes de volumes persistants d'un projet, les points suivants doivent être respectés :

- La demande de ressources dans une revendication de volume persistant (PVC) doit être supérieure ou égale à la contrainte **min** pour les PVC qui est spécifiée dans l'objet **LimitRange**.
- La demande de ressources dans une revendication de volume persistant (PVC) doit être inférieure ou égale à la contrainte **max** pour les PVC qui est spécifiée dans l'objet **LimitRange**.

PVC LimitRange définition de l'objet

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" (1)
spec:
  limits:
    - type: "PersistentVolumeClaim"
      min:
        storage: "2Gi" (2)
      max:
        storage: "50Gi" (3)
```

(1) Le nom de l'objet **LimitRange**.

(2) La quantité minimale de stockage qui peut être demandée dans une demande de volume persistant.

(3) La quantité maximale de stockage qui peut être demandée dans une demande de volume persistant.

7.3.2. Création d'une plage de limites

Pour appliquer une plage de limites à un projet :

1. Créez un objet **LimitRange** avec les spécifications requises :

```
apiVersion: "v1"
kind: "LimitRange"
metadata:
  name: "resource-limits" (1)
spec:
  limits:
    - type: "Pod" (2)
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "200m"
        memory: "6Mi"
    - type: "Container" (3)
      max:
        cpu: "2"
        memory: "1Gi"
      min:
        cpu: "100m"
        memory: "4Mi"
      default: (4)
        cpu: "300m"
        memory: "200Mi"
      defaultRequest: (5)
        cpu: "200m"
```

```
      memory: "100Mi"
    maxLimitRequestRatio: 6
      cpu: "10"
  - type: openshift.io/Image 7
    max:
      storage: 1Gi
  - type: openshift.io/ImageStream 8
    max:
      openshift.io/image-tags: 20
      openshift.io/images: 30
  - type: "PersistentVolumeClaim" 9
    min:
      storage: "2Gi"
    max:
      storage: "50Gi"
```

1. Spécifiez un nom pour l'objet **LimitRange**.
2. Pour définir les limites d'un module, spécifiez les demandes minimales et maximales de CPU et de mémoire, selon les besoins.
3. Pour définir des limites pour un conteneur, spécifiez les demandes minimales et maximales de CPU et de mémoire selon les besoins.
4. Facultatif. Pour un conteneur, indiquez la quantité par défaut de CPU ou de mémoire qu'un conteneur peut utiliser, si elle n'est pas spécifiée dans la spécification **Pod**.
5. Facultatif. Pour un conteneur, spécifiez la quantité par défaut de CPU ou de mémoire qu'un conteneur peut demander, si elle n'est pas spécifiée dans la spécification **Pod**.
6. Facultatif. Pour un conteneur, spécifier le rapport maximum entre la limite et la demande qui peut être spécifié dans la spécification **Pod**.
7. Pour définir les limites d'un objet Image, définissez la taille maximale d'une image qui peut être poussée vers un registre d'images OpenShift.
8. Pour fixer des limites à un flux d'images, définissez le nombre maximum de balises et de références d'images pouvant figurer dans le fichier objet **ImageStream**, selon les besoins.
9. Pour définir les limites d'une demande de volume persistant, définissez la quantité minimale et maximale de stockage qui peut être demandée.

2. Créer l'objet :

```
oc create -f <limit_range_file> -n <project> $ oc create -f <limit_range_file> 1
```

1. Indiquez le nom du fichier YAML que vous avez créé et le projet dans lequel vous souhaitez que les limites s'appliquent.

7.3.3. Visualisation d'une limite

Vous pouvez visualiser toutes les limites définies dans un projet en naviguant dans la console web jusqu'à la page **Quota** du projet.

Vous pouvez également utiliser l'interface de communication pour afficher les détails de la plage de limites :

1. Obtenir la liste des objets **LimitRange** définis dans le projet. Par exemple, pour un projet appelé **demoproject**:

```
$ oc get limits -n demoproject
```

```
NAME              CREATED AT
resource-limits   2020-07-15T17:14:23Z
```

2. Décrivez l'objet **LimitRange** qui vous intéresse, par exemple la plage de limites **resource-limits**:

```
$ oc describe limits resource-limits -n demoproject
```

```
Name:                   resource-limits
Namespace:              demoproject
Type                    Resource              Min     Max     Default Request Default Limit   Max
Limit/Request Ratio
----                    --------              ---     ---     --------------- -------------   -----------------------
Pod                     cpu                   200m    2       -               -               -
Pod                     memory                6Mi     1Gi     -               -               -
Container               cpu                   100m    2       200m            300m            10
Container               memory                4Mi     1Gi     100Mi           200Mi           -
openshift.io/Image      storage               -       1Gi     -               -               -
openshift.io/ImageStream openshift.io/image   -       12      -               -               -
openshift.io/ImageStream openshift.io/image-tags -    10      -               -               -
PersistentVolumeClaim   storage               -       50Gi    -               -               -
```

7.3.4. Suppression d'une plage de limites

Pour supprimer tout objet **LimitRange** actif afin de ne plus appliquer les limites dans un projet :

1. Exécutez la commande suivante :

```
oc delete limits <limit_name>
```

7.4. CONFIGURATION DE LA MÉMOIRE DES CLUSTERS POUR RÉPONDRE AUX EXIGENCES EN MATIÈRE DE MÉMOIRE DES CONTENEURS ET DE RISQUE

En tant qu'administrateur de cluster, vous pouvez aider vos clusters à fonctionner efficacement en gérant la mémoire des applications :

- Déterminer les besoins en mémoire et en risques d'un composant d'application conteneurisé et configurer les paramètres de mémoire du conteneur en fonction de ces besoins.
- Configurer les moteurs d'exécution des applications conteneurisées (par exemple, OpenJDK) pour qu'ils adhèrent de manière optimale aux paramètres de mémoire configurés du conteneur.
- Diagnostiquer et résoudre les erreurs de mémoire liées à l'utilisation d'un conteneur.

7.4.1. Comprendre la gestion de la mémoire des applications

Il est recommandé de lire entièrement l'aperçu de la façon dont OpenShift Container Platform gère les ressources informatiques avant de poursuivre.

Pour chaque type de ressource (mémoire, CPU, stockage), OpenShift Container Platform permet de placer des valeurs optionnelles **request** et **limit** sur chaque conteneur d'un pod.

Notez les points suivants concernant les demandes de mémoire et les limites de mémoire :

- **Memory request**
  - La valeur de la demande de mémoire, si elle est spécifiée, influence le planificateur de OpenShift Container Platform. Le planificateur prend en compte la demande de mémoire lorsqu'il planifie un conteneur sur un nœud, puis délimite la mémoire demandée sur le nœud choisi pour l'utilisation du conteneur.
  - Si la mémoire d'un nœud est épuisée, OpenShift Container Platform donne la priorité à l'éviction des conteneurs dont l'utilisation de la mémoire dépasse le plus leur demande de mémoire. Dans les cas graves d'épuisement de la mémoire, le tueur OOM du nœud peut sélectionner et tuer un processus dans un conteneur sur la base d'une métrique similaire.
  - L'administrateur du cluster peut attribuer un quota ou des valeurs par défaut pour la valeur de la demande de mémoire.
  - L'administrateur du cluster peut remplacer les valeurs de demande de mémoire spécifiées par un développeur, afin de gérer le surengagement du cluster.
- **Memory limit**
  - La valeur de la limite de mémoire, si elle est spécifiée, fournit une limite stricte à la mémoire qui peut être allouée à tous les processus d'un conteneur.
  - Si la mémoire allouée par tous les processus dans un conteneur dépasse la limite de mémoire, le tueur de nœuds hors mémoire (OOM) sélectionnera et tuera immédiatement un processus dans le conteneur.
  - Si une demande de mémoire et une limite sont spécifiées, la valeur de la limite de mémoire doit être supérieure ou égale à la demande de mémoire.
  - L'administrateur du cluster peut attribuer des quotas ou des valeurs par défaut pour la limite de mémoire.
  - La limite minimale de mémoire est de 12 Mo. Si un conteneur ne démarre pas en raison d'un événement pod **Cannot allocate memory**, la limite de mémoire est trop basse. Augmentez ou supprimez la limite de mémoire. La suppression de la limite permet aux pods de consommer des ressources de nœuds illimitées.

7.4.1.1. Gestion de la stratégie de mémoire des applications

Les étapes pour dimensionner la mémoire des applications sur OpenShift Container Platform sont les suivantes :

1. **Determine expected container memory usage**
   Déterminez l'utilisation moyenne et maximale prévue de la mémoire du conteneur, de manière empirique si nécessaire (par exemple, en effectuant des tests de charge distincts). N'oubliez pas de tenir compte de tous les processus susceptibles de s'exécuter en parallèle dans le

conteneur : par exemple, l'application principale génère-t-elle des scripts auxiliaires ?

2. **Determine risk appetite**
   Déterminer le risque d'éviction. Si l'appétit pour le risque est faible, le conteneur doit demander de la mémoire en fonction de l'utilisation maximale prévue plus un pourcentage de marge de sécurité. Si l'appétit pour le risque est plus élevé, il peut être plus approprié de demander de la mémoire en fonction de l'utilisation moyenne prévue.

3. **Set container memory request**
   Définir la demande de mémoire du conteneur en fonction de ce qui précède. Plus la demande représente précisément l'utilisation de la mémoire de l'application, mieux c'est. Si la demande est trop élevée, l'utilisation du cluster et du quota sera inefficace. Si la demande est trop faible, les risques d'éviction de l'application augmentent.

4. **Set container memory limit, if required**
   Définir la limite de mémoire du conteneur, si nécessaire. Définir une limite a pour effet de tuer immédiatement un processus du conteneur si l'utilisation combinée de la mémoire de tous les processus du conteneur dépasse la limite, et c'est donc une bénédiction mitigée. D'une part, cela peut rendre évident un excès d'utilisation de la mémoire non anticipé ("fail fast") ; d'autre part, cela met fin aux processus de manière abrupte.

   Notez que certains clusters OpenShift Container Platform peuvent exiger qu'une valeur limite soit définie ; certains peuvent surcharger la demande en fonction de la limite ; et certaines images d'application s'appuient sur une valeur limite définie car elle est plus facile à détecter qu'une valeur de demande.

   Si une limite de mémoire est fixée, elle ne doit pas être inférieure au pic prévu d'utilisation de la mémoire du conteneur, plus un pourcentage de marge de sécurité.

5. **Ensure application is tuned**
   Assurez-vous que l'application est adaptée aux demandes configurées et aux valeurs limites, le cas échéant. Cette étape est particulièrement importante pour les applications qui mettent en commun de la mémoire, comme la JVM. Le reste de cette page en traite.

Ressources supplémentaires

- Comprendre les ressources informatiques et les conteneurs

7.4.2. Comprendre les paramètres OpenJDK pour OpenShift Container Platform

Les paramètres par défaut de l'OpenJDK ne fonctionnent pas bien avec les environnements conteneurisés. Par conséquent, certains paramètres supplémentaires de la mémoire Java doivent toujours être fournis lorsque l'OpenJDK est exécuté dans un conteneur.

L'agencement de la mémoire de la JVM est complexe, dépend de la version et sa description détaillée dépasse le cadre de cette documentation. Cependant, comme point de départ pour l'exécution d'OpenJDK dans un conteneur, au moins les trois tâches suivantes liées à la mémoire sont essentielles :

1. Surcharge de la taille maximale du tas de la JVM.

2. Encourager la JVM à libérer la mémoire inutilisée au profit du système d'exploitation, le cas échéant.

3. S'assurer que tous les processus JVM au sein d'un conteneur sont correctement configurés.

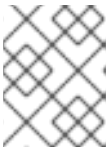
L'optimisation des charges de travail JVM pour l'exécution dans un conteneur dépasse le cadre de cette documentation et peut nécessiter la définition de plusieurs options JVM supplémentaires.

7.4.2.1. Comprendre comment passer outre la taille maximale du tas de la JVM

Pour de nombreuses charges de travail Java, le tas de la JVM est le plus gros consommateur de mémoire. Actuellement, l'OpenJDK autorise par défaut jusqu'à 1/4 (1/**-XX:MaxRAMFraction**) de la mémoire du nœud de calcul à être utilisée pour le tas, que l'OpenJDK soit exécuté dans un conteneur ou non. Il est donc possible d'outrepasser ce comportement à l'adresse **essential**, en particulier si une limite de mémoire est également fixée pour le conteneur.

Il y a au moins deux façons d'y parvenir :

1. Si la limite de mémoire du conteneur est définie et que les options expérimentales sont prises en charge par la JVM, définissez **-XX: UnlockExperimentalVMOptions -XX: UseCGroupMemoryLimitForHeap**.

   NOTE

   L'option **UseCGroupMemoryLimitForHeap** a été supprimée dans le JDK 11. Utilisez **-XX: UseContainerSupport** à la place.

   Cela fixe **-XX:MaxRAM** à la limite de mémoire du conteneur et la taille maximale du tas (**-XX:MaxHeapSize** / **-Xmx**) à 1/**-XX:MaxRAMFraction** (1/4 par défaut).

2. Remplacer directement l'un des éléments suivants : **-XX:MaxRAM**, **-XX:MaxHeapSize** ou **-Xmx**. Cette option implique le codage en dur d'une valeur, mais présente l'avantage de permettre le calcul d'une marge de sécurité.

7.4.2.2. Comprendre comment encourager la JVM à libérer la mémoire inutilisée au profit du système d'exploitation

Par défaut, l'OpenJDK ne renvoie pas agressivement la mémoire inutilisée au système d'exploitation. Cela peut convenir à de nombreuses charges de travail Java conteneurisées, mais les exceptions notables comprennent les charges de travail où des processus actifs supplémentaires coexistent avec une JVM dans un conteneur, que ces processus supplémentaires soient natifs, des JVM supplémentaires ou une combinaison des deux.

Les agents basés sur Java peuvent utiliser les arguments JVM suivants pour encourager la JVM à libérer la mémoire inutilisée au profit du système d'exploitation :

```
-XX:+UseParallelGC
-XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4
-XX:AdaptiveSizePolicyWeight=90.
```

Ces arguments sont destinés à restituer la mémoire du tas au système d'exploitation lorsque la mémoire allouée dépasse 110 % de la mémoire utilisée (**-XX:MaxHeapFreeRatio**), en consacrant jusqu'à 20 % du temps de l'unité centrale au ramasse-miettes (**-XX:GCTimeRatio**). À aucun moment, l'allocation du tas de l'application ne sera inférieure à l'allocation initiale du tas (surchargée par **-XX:InitialHeapSize** / **-Xms**). Des informations complémentaires détaillées sont disponibles sur Tuning Java's footprint in OpenShift (Part 1), Tuning Java's footprint in OpenShift (Part 2) , et sur OpenJDK and Containers.

7.4.2.3. Comprendre comment s'assurer que tous les processus JVM au sein d'un conteneur sont correctement configurés

Si plusieurs JVM sont exécutées dans le même conteneur, il est essentiel de s'assurer qu'elles sont toutes configurées de manière appropriée. Pour de nombreuses charges de travail, il sera nécessaire d'accorder à chaque JVM un pourcentage de budget mémoire, en laissant une marge de sécurité supplémentaire peut-être substantielle.

De nombreux outils Java utilisent différentes variables d'environnement (**JAVA_OPTS**, **GRADLE_OPTS**, etc.) pour configurer leurs JVM et il peut être difficile de s'assurer que les bons paramètres sont transmis à la bonne JVM.

La variable d'environnement **JAVA_TOOL_OPTIONS** est toujours respectée par l'OpenJDK et les valeurs spécifiées dans **JAVA_TOOL_OPTIONS** seront remplacées par d'autres options spécifiées sur la ligne de commande de la JVM. Par défaut, pour s'assurer que ces options sont utilisées par défaut pour toutes les charges de travail JVM exécutées dans l'image de l'agent basé sur Java, l'image de l'agent OpenShift Container Platform Jenkins Maven définit :

```
JAVA_TOOL_OPTIONS="-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true"
```

NOTE

L'option **UseCGroupMemoryLimitForHeap** a été supprimée dans le JDK 11. Utilisez **-XX: UseContainerSupport** à la place.

Cela ne garantit pas que des options supplémentaires ne soient pas nécessaires, mais constitue un point de départ utile.

7.4.3. Recherche de la demande et de la limite de mémoire à l'intérieur d'un pod

Une application souhaitant découvrir dynamiquement sa demande et sa limite de mémoire à l'intérieur d'un pod doit utiliser l'API descendante.

Procédure

1. Configurez le pod pour ajouter les strophes **MEMORY_REQUEST** et **MEMORY_LIMIT**:

```
apiVersion: v1
kind: Pod
metadata:
  name: test
spec:
  containers:
  - name: test
    image: fedora:latest
    command:
    - sleep
    - "3600"
    env:
    - name: MEMORY_REQUEST 1
      valueFrom:
        resourceFieldRef:
          containerName: test
          resource: requests.memory
    - name: MEMORY_LIMIT 2
      valueFrom:
```

```
      resourceFieldRef:
        containerName: test
        resource: limits.memory
  resources:
    requests:
      memory: 384Mi
    limits:
      memory: 512Mi
```

1. Ajoutez ce paragraphe pour découvrir la valeur de la demande de mémoire de l'application.
2. Ajoutez cette strophe pour découvrir la valeur de la limite de mémoire de l'application.

2. Créer la capsule :

```
oc create -f <nom-de-fichier>.yaml
```

3. Accéder au pod à l'aide d'un shell distant :

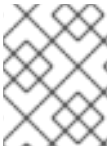
```
$ oc rsh test
```

4. Vérifier que les valeurs demandées ont été appliquées :

```
$ env | grep MEMORY | sort
```

Exemple de sortie

```
MEMORY_LIMIT=536870912
MEMORY_REQUEST=402653184
```



NOTE

La valeur de la limite de mémoire peut également être lue à l'intérieur du conteneur par le fichier **/sys/fs/cgroup/memory/memory.limit_in_bytes**.

7.4.4. Comprendre la politique de mise à mort des OOM

OpenShift Container Platform peut tuer un processus dans un conteneur si l'utilisation totale de la mémoire de tous les processus dans le conteneur dépasse la limite de mémoire, ou dans des cas graves d'épuisement de la mémoire du nœud.

Lorsqu'un processus est tué pour cause de mémoire insuffisante (OOM), il se peut que le conteneur se termine immédiatement. Si le processus PID 1 du conteneur reçoit le message **SIGKILL**, le conteneur se termine immédiatement. Sinon, le comportement du conteneur dépend du comportement des autres processus.

Par exemple, un processus de conteneur s'est terminé avec le code 137, indiquant qu'il avait reçu un signal SIGKILL.

Si le conteneur ne sort pas immédiatement, un OOM kill est détectable comme suit :

1. Accéder au pod à l'aide d'un shell distant :

```
# oc rsh test
```

2. Exécutez la commande suivante pour voir le nombre actuel de morts OOM dans **/sys/fs/cgroup/memory/memory.oom_control**:

```
$ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
oom_kill 0
```

3. Exécutez la commande suivante pour provoquer une mise à mort OOM :

```
$ sed -e '' </dev/zero
```

Exemple de sortie

```
Killed
```

4. Exécutez la commande suivante pour afficher l'état de sortie de la commande **sed**:

```
$ echo $?
```

Exemple de sortie

```
137
```

Le code **137** indique que le processus du conteneur s'est arrêté avec le code 137, indiquant qu'il a reçu un signal SIGKILL.

5. Exécutez la commande suivante pour vérifier que le compteur de mises à mort OOM de **/sys/fs/cgroup/memory/memory.oom_control** s'est incrémenté :

```
$ grep '^oom_kill ' /sys/fs/cgroup/memory/memory.oom_control
oom_kill 1
```

Si un ou plusieurs processus d'un module sont tués par OOM, lorsque le module se termine par la suite, immédiatement ou non, il aura la phase **Failed** et la raison **OOMKilled**. Un module tué par OOM peut être redémarré en fonction de la valeur de **restartPolicy**. S'il n'est pas redémarré, les contrôleurs tels que le contrôleur de réplication remarqueront l'état d'échec du module et créeront un nouveau module pour remplacer l'ancien.

Utilisez la commande suivante pour obtenir l'état du pod :

```
$ oc get pod test
```

Exemple de sortie

```
NAME      READY     STATUS      RESTARTS   AGE
test      0/1       OOMKilled   0          1m
```

- Si le module n'a pas redémarré, exécutez la commande suivante pour visualiser le module :

```
$ oc get pod test -o yaml
```

Exemple de sortie

```
...
status:
  containerStatuses:
  - name: test
    ready: false
    restartCount: 0
    state:
      terminated:
        exitCode: 137
        reason: OOMKilled
  phase: Failed
```

- En cas de redémarrage, exécutez la commande suivante pour afficher le pod :

```
$ oc get pod test -o yaml
```

Exemple de sortie

```
...
status:
  containerStatuses:
  - name: test
    ready: true
    restartCount: 1
    lastState:
      terminated:
        exitCode: 137
        reason: OOMKilled
    state:
      running:
  phase: Running
```

7.4.5. Comprendre l'expulsion d'un pod

OpenShift Container Platform peut évincer un pod de son nœud lorsque la mémoire du nœud est épuisée. En fonction de l'ampleur de l'épuisement de la mémoire, l'expulsion peut être gracieuse ou non. L'expulsion gracieuse implique que le processus principal (PID 1) de chaque conteneur reçoive un signal SIGTERM, puis un peu plus tard un signal SIGKILL si le processus n'est pas déjà sorti. L'éviction non gracieuse implique que le processus principal de chaque conteneur reçoive immédiatement un signal SIGKILL.

Un pod évincé a la phase **Failed** et la raison **Evicted**. Il ne sera pas redémarré, quelle que soit la valeur de **restartPolicy**. Cependant, les contrôleurs tels que le contrôleur de réplication remarqueront l'état d'échec du module et créeront un nouveau module pour remplacer l'ancien.

```
$ oc get pod test
```

Exemple de sortie

```
NAME      READY     STATUS    RESTARTS   AGE
test      0/1       Evicted   0          1m
```

```
$ oc get pod test -o yaml
```

Exemple de sortie

```
...
status:
  message: 'Pod The node was low on resource: [MemoryPressure].'
  phase: Failed
  reason: Evicted
```

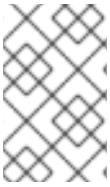
7.5. CONFIGURER VOTRE CLUSTER POUR PLACER DES PODS SUR DES NŒUDS SUR-ENGAGÉS

Dans un état *overcommitted*, la somme des demandes et des limites des ressources de calcul du conteneur dépasse les ressources disponibles sur le système. Par exemple, vous pouvez utiliser le surengagement dans les environnements de développement où un compromis entre les performances garanties et la capacité est acceptable.

Les conteneurs peuvent spécifier des demandes et des limites de ressources de calcul. Les demandes sont utilisées pour planifier votre conteneur et fournissent une garantie de service minimum. Les limites restreignent la quantité de ressources de calcul qui peut être consommée sur votre nœud.

L'ordonnanceur tente d'optimiser l'utilisation des ressources de calcul sur tous les nœuds de votre cluster. Il place les pods sur des nœuds spécifiques, en tenant compte des demandes de ressources de calcul des pods et de la capacité disponible des nœuds.

Les administrateurs d'OpenShift Container Platform peuvent contrôler le niveau de surengagement et gérer la densité des conteneurs sur les nœuds. Vous pouvez configurer le surengagement au niveau du cluster à l'aide de l'opérateur ClusterResourceOverride pour remplacer le rapport entre les demandes et les limites définies sur les conteneurs de développement. En conjonction avec le surengagement des nœuds et les limites et valeurs par défaut de la mémoire et de l'unité centrale du projet , vous pouvez ajuster la limite et la demande de ressources afin d'atteindre le niveau de surengagement souhaité.



NOTE

Dans OpenShift Container Platform, vous devez activer le surengagement au niveau du cluster. Le surengagement au niveau du nœud est activé par défaut. Voir Désactivation du surengagement pour un nœud.

7.5.1. Demandes de ressources et surengagement

Pour chaque ressource informatique, un conteneur peut spécifier une demande de ressource et une limite. Les décisions d'ordonnancement sont prises en fonction de la demande afin de s'assurer qu'un nœud dispose d'une capacité suffisante pour répondre à la valeur demandée. Si un conteneur spécifie des limites, mais omet des demandes, les demandes sont définies par défaut en fonction des limites. Un conteneur ne peut pas dépasser la limite spécifiée sur le nœud.

L'application des limites dépend du type de ressource de calcul. Si un conteneur ne formule aucune demande ou limite, il est programmé sur un nœud sans garantie de ressources. En pratique, le conteneur est capable de consommer autant de ressources spécifiées qu'il est disponible avec la priorité locale la plus basse. Dans les situations où les ressources sont faibles, les conteneurs qui ne formulent aucune demande de ressources bénéficient de la qualité de service la plus faible.

L'ordonnancement est basé sur les ressources demandées, tandis que les quotas et les limites strictes font référence aux limites de ressources, qui peuvent être plus élevées que les ressources demandées. La différence entre la demande et la limite détermine le niveau de surengagement ; par exemple, si un conteneur reçoit une demande de mémoire de 1Gi et une limite de mémoire de 2Gi, il est ordonnancé sur la base de la demande de 1Gi disponible sur le nœud, mais pourrait utiliser jusqu'à 2Gi ; il est donc surengagé à 200%.

7.5.2. Surengagement au niveau du cluster à l'aide de l'opérateur d'annulation des ressources du cluster

Le Cluster Resource Override Operator est un webhook d'admission qui vous permet de contrôler le niveau d'overcommit et de gérer la densité des conteneurs sur tous les nœuds de votre cluster. L'opérateur contrôle la façon dont les nœuds de projets spécifiques peuvent dépasser les limites de mémoire et de CPU définies.

Vous devez installer le Cluster Resource Override Operator à l'aide de la console OpenShift Container Platform ou du CLI, comme indiqué dans les sections suivantes. Lors de l'installation, vous créez une ressource personnalisée (CR) **ClusterResourceOverride**, dans laquelle vous définissez le niveau de surengagement, comme le montre l'exemple suivant :

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster 1
spec:
 podResourceOverride:
  spec:
   memoryRequestToLimitPercent: 50 2
   cpuRequestToLimitPercent: 25 3
   limitCPUToMemoryPercent: 200 4
```

1. Le nom doit être **cluster**.
2. Facultatif. Si une limite de mémoire de conteneur a été spécifiée ou définie par défaut, la demande de mémoire est remplacée par ce pourcentage de la limite, compris entre 1 et 100. La valeur par défaut est 50.
3. Facultatif. Si une limite de CPU pour le conteneur a été spécifiée ou définie par défaut, la demande de CPU est remplacée par ce pourcentage de la limite, compris entre 1 et 100. La valeur par défaut est 25.
4. Facultatif. Si une limite de mémoire de conteneur a été spécifiée ou définie par défaut, la limite de CPU est remplacée par un pourcentage de la limite de mémoire, si elle est spécifiée. La mise à l'échelle de 1Gi de RAM à 100 % équivaut à 1 cœur de CPU. Cette opération est effectuée avant de passer outre la demande de CPU (si elle est configurée). La valeur par défaut est 200.

NOTE

Les dérogations de l'opérateur de dérogations des ressources du cluster n'ont aucun effet si des limites n'ont pas été définies pour les conteneurs. Créez un objet **LimitRange** avec des limites par défaut pour chaque projet ou configurez des limites dans les spécifications **Pod** pour que les dérogations s'appliquent.

Lorsqu'elles sont configurées, les dérogations peuvent être activées par projet en appliquant l'étiquette suivante à l'objet Namespace pour chaque projet :

```
apiVersion: v1
kind: Namespace
metadata:

....

  labels:
    clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true"

....
```

L'opérateur guette le CR **ClusterResourceOverride** et s'assure que le webhook d'admission **ClusterResourceOverride** est installé dans le même espace de noms que l'opérateur.

7.5.2.1. Installation de l'opérateur de remplacement des ressources de cluster à l'aide de la console web

Vous pouvez utiliser la console web d'OpenShift Container Platform pour installer le Cluster Resource Override Operator afin de contrôler l'overcommit dans votre cluster.

Conditions préalables

- L'opérateur d'annulation des ressources de cluster n'a aucun effet si des limites n'ont pas été définies pour les conteneurs. Vous devez spécifier des limites par défaut pour un projet à l'aide d'un objet **LimitRange** ou configurer des limites dans les spécifications **Pod** pour que les surcharges s'appliquent.

Procédure

Pour installer le Cluster Resource Override Operator à l'aide de la console web d'OpenShift Container Platform :

1. Dans la console web d'OpenShift Container Platform, naviguez vers **Home → Projects**
   a. Cliquez sur **Create Project**.
   b. Spécifiez **clusterresourceoverride-operator** comme nom du projet.
   c. Cliquez sur **Create**.
2. Naviguez jusqu'à **Operators → OperatorHub**.
   a. Choisissez **ClusterResourceOverride Operator** dans la liste des opérateurs disponibles et cliquez sur **Install**.
   b. Sur la page **Install Operator**, assurez-vous que **A specific Namespace on the cluster**est sélectionné pour **Installation Mode**.
   c. Assurez-vous que **clusterresourceoverride-operator** est sélectionné pour **Installed Namespace**.
   d. Sélectionnez une adresse **Update Channel** et **Approval Strategy**.
   e. Cliquez sur **Install**.

3. Sur la page **Installed Operators**, cliquez sur **ClusterResourceOverride**.

   a. Sur la page de détails **ClusterResourceOverride Operator**, cliquez sur **Create Instance**.

   b. Sur la page **Create ClusterResourceOverride**, modifiez le modèle YAML pour définir les valeurs de surengagement selon les besoins :

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
```

   (1) Le nom doit être **cluster**.

   (2) Facultatif. Indiquez le pourcentage de dépassement de la limite de mémoire du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 50.

   (3) Facultatif. Spécifiez le pourcentage de dépassement de la limite de CPU du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 25.

   (4) Facultatif. Indiquez le pourcentage qui doit remplacer la limite de mémoire du conteneur, si elle est utilisée. La mise à l'échelle de 1Gi de RAM à 100 % équivaut à 1 cœur de CPU. Ceci est traité avant d'outrepasser la demande de CPU, si elle est configurée. La valeur par défaut est 200.

   c. Cliquez sur **Create**.

4. Vérifier l'état actuel du webhook d'admission en vérifiant l'état de la ressource personnalisée cluster :

   a. Sur la page **ClusterResourceOverride Operator**, cliquez sur **cluster**.

   b. Sur la page **ClusterResourceOverride Details**, cliquez sur **YAML**. La section **mutatingWebhookConfigurationRef** s'affiche lorsque le webhook est appelé.

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadata":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPercent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
```

```
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

....

    mutatingWebhookConfigurationRef: 1
      apiVersion: admissionregistration.k8s.io/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1 Référence au webhook d'admission **ClusterResourceOverride**.

7.5.2.2. Installation de l'opérateur de remplacement des ressources de cluster à l'aide de l'interface de ligne de commande (CLI)

Vous pouvez utiliser le CLI d'OpenShift Container Platform pour installer le Cluster Resource Override Operator afin de contrôler l'overcommit dans votre cluster.

Conditions préalables

- L'opérateur d'annulation des ressources de cluster n'a aucun effet si des limites n'ont pas été définies pour les conteneurs. Vous devez spécifier des limites par défaut pour un projet à l'aide d'un objet **LimitRange** ou configurer des limites dans les spécifications **Pod** pour que les surcharges s'appliquent.

Procédure

Pour installer l'opérateur de remplacement des ressources de cluster à l'aide de l'interface de ligne de commande :

1. Créez un espace de noms pour l'opérateur de remplacement des ressources de cluster :
   a. Créez un fichier YAML de l'objet **Namespace** (par exemple, **cro-namespace.yaml**) pour l'opérateur de remplacement des ressources du cluster :

```
apiVersion: v1
kind: Namespace
metadata:
  name: clusterresourceoverride-operator
```

   b. Créer l'espace de noms :

```
oc create -f <nom-de-fichier>.yaml
```

Par exemple :

```
$ oc create -f cro-namespace.yaml
```

2. Créer un groupe d'opérateurs :

   a. Créez un fichier YAML de l'objet **OperatorGroup** (par exemple, cro-og.yaml) pour l'opérateur de remplacement des ressources de cluster :

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

   b. Créer le groupe d'opérateurs :

```
oc create -f <nom-de-fichier>.yaml
```

   Par exemple :

```
$ oc create -f cro-og.yaml
```

3. Créer un abonnement :

   a. Créez un fichier YAML de l'objet **Subscription** (par exemple, cro-sub.yaml) pour l'opérateur de remplacement des ressources du cluster :

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: clusterresourceoverride
  namespace: clusterresourceoverride-operator
spec:
  channel: "4.12"
  name: clusterresourceoverride
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

   b. Créer l'abonnement :

```
oc create -f <nom-de-fichier>.yaml
```

   Par exemple :

```
$ oc create -f cro-sub.yaml
```

4. Créer un objet ressource personnalisée (CR) **ClusterResourceOverride** dans l'espace de noms **clusterresourceoverride-operator**:

   a. Passage à l'espace de noms **clusterresourceoverride-operator**.

```
$ oc project clusterresourceoverride-operator
```

b. Créez un fichier YAML de l'objet **ClusterResourceOverride** (par exemple, cro-cr.yaml) pour l'opérateur de remplacement des ressources de cluster :

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  name: cluster (1)
spec:
  podResourceOverride:
    spec:
      memoryRequestToLimitPercent: 50 (2)
      cpuRequestToLimitPercent: 25 (3)
      limitCPUToMemoryPercent: 200 (4)
```

(1) Le nom doit être **cluster**.

(2) Facultatif. Indiquez le pourcentage de dépassement de la limite de mémoire du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 50.

(3) Facultatif. Spécifiez le pourcentage de dépassement de la limite de CPU du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 25.

(4) Facultatif. Indiquez le pourcentage qui doit remplacer la limite de mémoire du conteneur, si elle est utilisée. La mise à l'échelle de 1Gi de RAM à 100 % équivaut à 1 cœur de CPU. Ceci est traité avant d'outrepasser la demande de CPU, si elle est configurée. La valeur par défaut est 200.

c. Créer l'objet **ClusterResourceOverride**:

```
oc create -f <nom-de-fichier>.yaml
```

Par exemple :

```
$ oc create -f cro-cr.yaml
```

5. Vérifiez l'état actuel du webhook d'admission en contrôlant l'état de la ressource personnalisée cluster.

```
$ oc get clusterresourceoverride cluster -n clusterresourceoverride-operator -o yaml
```

La section **mutatingWebhookConfigurationRef** s'affiche lorsque le webhook est appelé.

Exemple de sortie

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"operator.autoscaling.openshift.io/v1","kind":"ClusterResourceOverride","metadat
```

```
a":{"annotations":{},"name":"cluster"},"spec":{"podResourceOverride":{"spec":
{"cpuRequestToLimitPercent":25,"limitCPUToMemoryPercent":200,"memoryRequestToLimitPe
rcent":50}}}}
  creationTimestamp: "2019-12-18T22:35:02Z"
  generation: 1
  name: cluster
  resourceVersion: "127622"
  selfLink: /apis/operator.autoscaling.openshift.io/v1/clusterresourceoverrides/cluster
  uid: 978fc959-1717-4bd1-97d0-ae00ee111e8d
spec:
  podResourceOverride:
    spec:
      cpuRequestToLimitPercent: 25
      limitCPUToMemoryPercent: 200
      memoryRequestToLimitPercent: 50
status:

....

    mutatingWebhookConfigurationRef: 1
      apiVersion: admissionregistration.k8s.io/v1beta1
      kind: MutatingWebhookConfiguration
      name: clusterresourceoverrides.admission.autoscaling.openshift.io
      resourceVersion: "127621"
      uid: 98b3b8ae-d5ce-462b-8ab5-a729ea8f38f3

....
```

1 Référence au webhook d'admission **ClusterResourceOverride**.

7.5.2.3. Configuration de l'overcommit au niveau du cluster

L'opérateur d'annulation des ressources de cluster a besoin d'une ressource personnalisée (CR) **ClusterResourceOverride** et d'un libellé pour chaque projet sur lequel vous souhaitez que l'opérateur contrôle le surengagement.

Conditions préalables

- L'opérateur d'annulation des ressources de cluster n'a aucun effet si des limites n'ont pas été définies pour les conteneurs. Vous devez spécifier des limites par défaut pour un projet à l'aide d'un objet **LimitRange** ou configurer des limites dans les spécifications **Pod** pour que les surcharges s'appliquent.

Procédure

Pour modifier l'overcommit au niveau du cluster :

1. Modifier le CR **ClusterResourceOverride**:

```
apiVersion: operator.autoscaling.openshift.io/v1
kind: ClusterResourceOverride
metadata:
    name: cluster
spec:
  podResourceOverride:
```

```
spec:
  memoryRequestToLimitPercent: 50 (1)
  cpuRequestToLimitPercent: 25 (2)
  limitCPUToMemoryPercent: 200 (3)
```

(1) Facultatif. Indiquez le pourcentage de dépassement de la limite de mémoire du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 50.

(2) Facultatif. Spécifiez le pourcentage de dépassement de la limite de CPU du conteneur, s'il est utilisé, entre 1 et 100. La valeur par défaut est 25.

(3) Facultatif. Indiquez le pourcentage qui doit remplacer la limite de mémoire du conteneur, si elle est utilisée. La mise à l'échelle de 1Gi de RAM à 100 % équivaut à 1 cœur de CPU. Ceci est traité avant d'outrepasser la demande de CPU, si elle est configurée. La valeur par défaut est 200.

2. Assurez-vous que l'étiquette suivante a été ajoutée à l'objet Namespace pour chaque projet dans lequel vous souhaitez que l'opérateur de remplacement des ressources de cluster contrôle le surengagement :

```
apiVersion: v1
kind: Namespace
metadata:

 ...

 labels:
   clusterresourceoverrides.admission.autoscaling.openshift.io/enabled: "true" (1)

...
```

(1) Ajoutez cette étiquette à chaque projet.

7.5.3. Sur-engagement au niveau du nœud

Vous pouvez utiliser différents moyens pour contrôler le surengagement sur des nœuds spécifiques, tels que les garanties de qualité de service (QOS), les limites de CPU ou les ressources de réserve. Vous pouvez également désactiver le surengagement pour des nœuds et des projets spécifiques.

7.5.3.1. Comprendre les ressources informatiques et les conteneurs

Le comportement renforcé par le nœud pour les ressources de calcul est spécifique au type de ressource.

7.5.3.1.1. Comprendre les demandes d'unité centrale des conteneurs

Un conteneur se voit garantir la quantité de CPU qu'il demande et peut en outre consommer le surplus de CPU disponible sur le nœud, jusqu'à une limite spécifiée par le conteneur. Si plusieurs conteneurs tentent d'utiliser l'unité centrale excédentaire, le temps d'utilisation de l'unité centrale est réparti en fonction de la quantité d'unité centrale demandée par chaque conteneur.

Par exemple, si un conteneur demande 500 m de temps CPU et qu'un autre conteneur demande 250 m de temps CPU, le temps CPU supplémentaire disponible sur le nœud est réparti entre les conteneurs

dans un rapport de 2:1. Si un conteneur a spécifié une limite, il sera limité pour ne pas utiliser plus de CPU que la limite spécifiée. Les demandes de CPU sont appliquées en utilisant le support des parts CFS dans le noyau Linux. Par défaut, les limites de CPU sont appliquées en utilisant le support des quotas CFS dans le noyau Linux sur un intervalle de mesure de 100ms, bien que cela puisse être désactivé.

7.5.3.1.2. Comprendre les demandes de mémoire des conteneurs

Un conteneur se voit garantir la quantité de mémoire qu'il demande. Un conteneur peut utiliser plus de mémoire que celle demandée, mais une fois qu'il dépasse la quantité demandée, il peut être interrompu en cas de manque de mémoire sur le nœud. Si un conteneur utilise moins de mémoire que la quantité demandée, il ne sera pas interrompu, sauf si des tâches ou des démons du système ont besoin de plus de mémoire que ce qui a été pris en compte dans la réservation des ressources du nœud. Si un conteneur spécifie une limite de mémoire, il est immédiatement interrompu s'il dépasse cette limite.

7.5.3.2. Comprendre le surengagement et les classes de qualité de service

Un nœud est *overcommitted* lorsqu'un pod programmé n'effectue aucune demande ou lorsque la somme des limites de tous les pods sur ce nœud dépasse la capacité disponible de la machine.

Dans un environnement surengagé, il est possible que les pods sur le nœud tentent d'utiliser plus de ressources de calcul que celles disponibles à un moment donné. Dans ce cas, le nœud doit donner la priorité à un module plutôt qu'à un autre. La fonction utilisée pour prendre cette décision est appelée classe de qualité de service (QoS).

Un pod est désigné comme l'une des trois classes de qualité de service, par ordre de priorité décroissant :

Tableau 7.19. Classes de qualité de service

Priorité	Nom de la classe	Description
1 (le plus élevé)	**Guaranteed**	Si des limites et, éventuellement, des demandes sont fixées (et non égales à 0) pour toutes les ressources et qu'elles sont égales, le pod est classé comme **Guaranteed**.
2	**Burstable**	Si des demandes et éventuellement des limites sont définies (différentes de 0) pour toutes les ressources, et qu'elles ne sont pas égales, le pod est classé comme **Burstable**.
3 (le plus bas)	**BestEffort**	Si les demandes et les limites ne sont définies pour aucune des ressources, le pod est classé comme **BestEffort**.

La mémoire étant une ressource incompressible, dans les situations où la mémoire est faible, les conteneurs qui ont la priorité la plus basse sont terminés en premier :

- **Guaranteed** sont considérés comme prioritaires et ne seront interrompus que s'ils dépassent leurs limites ou si le système est sous pression de mémoire et qu'il n'y a pas de conteneurs de moindre priorité pouvant être expulsés.

- **Burstable** les conteneurs soumis à la pression de la mémoire du système sont plus susceptibles d'être interrompus lorsqu'ils dépassent leurs demandes et qu'il n'existe pas d'autres conteneurs **BestEffort**.

- **BestEffort** sont traités avec la priorité la plus basse. Les processus de ces conteneurs sont les premiers à être interrompus si le système manque de mémoire.

7.5.3.2.1. Comprendre comment réserver de la mémoire entre les différents niveaux de qualité de service

Vous pouvez utiliser le paramètre **qos-reserved** pour spécifier un pourcentage de mémoire à réserver par un module dans un niveau de qualité de service particulier. Cette fonctionnalité tente de réserver les ressources demandées afin d'empêcher les pods des classes de qualité de service inférieures d'utiliser les ressources demandées par les pods des classes de qualité de service supérieures.

OpenShift Container Platform utilise le paramètre **qos-reserved** comme suit :

- Une valeur de **qos-reserved=memory=100%** empêchera les classes de qualité de service **Burstable** et **BestEffort** de consommer la mémoire demandée par une classe de qualité de service supérieure. Cela augmente le risque d'induire des OOM sur les charges de travail **BestEffort** et **Burstable** en faveur d'une augmentation des garanties de ressources mémoire pour les charges de travail **Guaranteed** et **Burstable**.

- Une valeur de **qos-reserved=memory=50%** permet aux classes de qualité de service **Burstable** et **BestEffort** de consommer la moitié de la mémoire demandée par une classe de qualité de service supérieure.

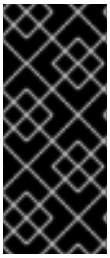
- Une valeur de **qos-reserved=memory=0%** permet aux classes de qualité de service **Burstable** et **BestEffort** de consommer la totalité de la quantité allouable au nœud si elle est disponible, mais augmente le risque qu'une charge de travail **Guaranteed** n'ait pas accès à la mémoire demandée. Cette condition désactive effectivement cette fonctionnalité.

7.5.3.3. Comprendre la mémoire d'échange et le QOS

Vous pouvez désactiver le swap par défaut sur vos nœuds afin de préserver les garanties de qualité de service (QOS). Dans le cas contraire, les ressources physiques d'un nœud peuvent se surinscrire, ce qui affecte les garanties de ressources que le planificateur Kubernetes apporte lors du placement des pods.

Par exemple, si deux modules garantis ont atteint leur limite de mémoire, chaque conteneur peut commencer à utiliser la mémoire d'échange. Finalement, s'il n'y a pas assez d'espace d'échange, les processus dans les modules peuvent être interrompus parce que le système est sursouscrit.

Si l'échange n'est pas désactivé, les nœuds ne reconnaissent pas qu'ils sont confrontés à **MemoryPressure**, ce qui fait que les modules ne reçoivent pas la mémoire qu'ils ont demandée dans leur requête de planification. En conséquence, des modules supplémentaires sont placés sur le nœud, ce qui augmente encore la pression sur la mémoire et, en fin de compte, le risque d'une panne de mémoire du système (OOM).



IMPORTANT

Si la permutation est activée, les seuils d'éviction de la mémoire disponible pour la gestion des ressources ne fonctionneront pas comme prévu. Tirez parti de la gestion des ressources manquantes pour permettre aux pods d'être expulsés d'un nœud lorsqu'il est soumis à une pression de mémoire, et replanifiés sur un autre nœud qui n'est pas soumis à une telle pression.

7.5.3.4. Comprendre le surengagement des nœuds

Dans un environnement surchargé, il est important de configurer correctement votre nœud afin d'obtenir le meilleur comportement possible du système.

Lorsque le nœud démarre, il s'assure que les drapeaux ajustables du noyau pour la gestion de la mémoire sont correctement définis. Le noyau ne devrait jamais échouer dans l'allocation de la mémoire à moins qu'il ne soit à court de mémoire physique.

Pour garantir ce comportement, OpenShift Container Platform configure le noyau pour qu'il surengage toujours de la mémoire en définissant le paramètre **vm.overcommit_memory** sur **1**, ce qui annule le paramètre par défaut du système d'exploitation.

OpenShift Container Platform configure également le noyau pour qu'il ne panique pas lorsqu'il manque de mémoire en définissant le paramètre **vm.panic_on_oom** sur **0**. Un paramètre de 0 indique au noyau d'appeler oom_killer dans une condition de manque de mémoire (OOM), ce qui tue les processus en fonction de leur priorité

Vous pouvez afficher le paramètre actuel en exécutant les commandes suivantes sur vos nœuds :

```
$ sysctl -a |grep commit
```

Exemple de sortie

```
vm.overcommit_memory = 1
```

```
$ sysctl -a |grep panic
```

Exemple de sortie

```
vm.panic_on_oom = 0
```

NOTE

Les drapeaux ci-dessus devraient déjà être activés sur les nœuds, et aucune autre action n'est nécessaire.

Vous pouvez également effectuer les configurations suivantes pour chaque nœud :

- Désactiver ou appliquer les limites de l'unité centrale à l'aide des quotas CFS de l'unité centrale
- Réserver des ressources pour les processus du système
- Réserve de mémoire pour les différents niveaux de qualité de service

7.5.3.5. Désactivation ou application des limites de l'unité centrale à l'aide des quotas CFS de l'unité centrale

Les nœuds appliquent par défaut les limites de CPU spécifiées en utilisant la prise en charge des quotas CFS (Completely Fair Scheduler) dans le noyau Linux.

Si vous désactivez l'application de la limite du CPU, il est important de comprendre l'impact sur votre nœud :

- Si un conteneur a une demande d'unité centrale, la demande continue d'être exécutée par les parts CFS dans le noyau Linux.
- Si un conteneur n'a pas de demande de CPU, mais a une limite de CPU, la demande de CPU est fixée par défaut à la limite de CPU spécifiée, et est appliquée par les parts CFS dans le noyau Linux.
- Si un conteneur a à la fois une demande et une limite de CPU, la demande de CPU est appliquée par les parts CFS dans le noyau Linux, et la limite de CPU n'a pas d'impact sur le nœud.

Conditions préalables

1. Obtenez l'étiquette associée au CRD statique **MachineConfigPool** pour le type de nœud que vous souhaitez configurer en entrant la commande suivante :

```
oc edit machineconfigpool <name> $ oc edit machineconfigpool <name>
```

Par exemple :

```
$ oc edit machineconfigpool worker
```

Exemple de sortie

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  creationTimestamp: "2022-11-16T15:34:25Z"
  generation: 4
  labels:
    pools.operator.machineconfiguration.openshift.io/worker: "" 1
  name: worker
```

1 L'étiquette apparaît sous Étiquettes.

ASTUCE

Si l'étiquette n'est pas présente, ajoutez une paire clé/valeur comme par exemple :

```
$ oc label machineconfigpool worker custom-kubelet=small-pods
```

Procédure

1. Créez une ressource personnalisée (CR) pour votre changement de configuration.

Exemple de configuration pour la désactivation des limites de l'unité centrale

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units 1
spec:
```

```
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 2
  kubeletConfig:
    cpuCfsQuota: 3
      - "true"
```

1. Attribuer un nom au CR.
2. Spécifiez l'étiquette du pool de configuration de la machine.
3. Réglez le paramètre **cpuCfsQuota** sur **true**.

2. Exécutez la commande suivante pour créer le CR :

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

7.5.3.6. Réserver des ressources pour les processus du système

Pour assurer une planification plus fiable et minimiser le surengagement des ressources des nœuds, chaque nœud peut réserver une partie de ses ressources aux démons du système qui doivent être exécutés sur le nœud pour que la grappe fonctionne. Il est notamment recommandé de réserver des ressources incompressibles telles que la mémoire.

Procédure

Pour réserver explicitement des ressources aux processus qui ne sont pas des nœuds, allouez des ressources aux nœuds en spécifiant les ressources disponibles pour l'ordonnancement. Pour plus de détails, voir Allocation de ressources pour les nœuds.

7.5.3.7. Désactivation du surengagement pour un nœud

Lorsqu'il est activé, le surengagement peut être désactivé sur chaque nœud.

Procédure

Pour désactiver le surengagement dans un nœud, exécutez la commande suivante sur ce nœud :

```
$ sysctl -w vm.overcommit_memory=0
```

7.5.4. Limites au niveau du projet

Pour contrôler l'overcommit, vous pouvez définir des plages de limites de ressources par projet, en spécifiant des limites de mémoire et de CPU et des valeurs par défaut pour un projet que l'overcommit ne peut pas dépasser.

Pour plus d'informations sur les limites de ressources au niveau du projet, voir Ressources supplémentaires.

Vous pouvez également désactiver le surengagement pour des projets spécifiques.

7.5.4.1. Désactiver le surengagement pour un projet

Lorsqu'il est activé, le surengagement peut être désactivé par projet. Par exemple, vous pouvez autoriser la configuration des composants d'infrastructure indépendamment du surengagement.

Procédure

Pour désactiver le surengagement dans un projet :

1. Modifier le fichier de l'élément de projet
2. Ajouter l'annotation suivante :

```
quota.openshift.io/cluster-resource-override-enabled: "false"
```

3. Créer l'élément de projet :

```
oc create -f <nom-de-fichier>.yaml
```

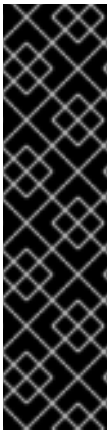
7.5.5. Ressources supplémentaires

- Définition des ressources de déploiement.
- Attribution de ressources aux nœuds.

7.6. ENABLING LINUX CONTROL GROUP VERSION 2 (CGROUP V2)

You can enable Linux control group version 2 (cgroup v2) in your cluster by editing the **node.config** object. Enabling cgroup v2 in OpenShift Container Platform disables all cgroups version 1 controllers and hierarchies in your cluster. cgroup v1 is enabled by default.

cgroup v2 is the next version of the Linux cgroup API. cgroup v2 offers several improvements over cgroup v1, including a unified hierarchy, safer sub-tree delegation, new features such as Pressure Stall Information, and enhanced resource management and isolation.



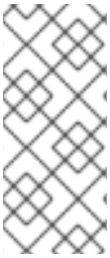
IMPORTANT

OpenShift Container Platform cgroups version 2 support is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

7.6.1. Configuration de Linux cgroup v2

Vous activez le cgroup v2 en modifiant l'objet **node.config**.



NOTE

Actuellement, la désactivation de l'équilibrage de la charge du CPU n'est pas prise en charge par cgroup v2. Par conséquent, il se peut que vous n'obteniez pas le comportement souhaité des profils de performance si vous avez activé cgroup v2. L'activation de cgroup v2 n'est pas recommandée si vous utilisez des profils de performance.

Conditions préalables

- Vous avez un cluster OpenShift Container Platform en cours d'exécution qui utilise la version 4.12 ou une version ultérieure.
- You are logged in to the cluster as a user with administrative privileges.
- Vous avez activé l'ensemble de fonctions **TechPreviewNoUpgrade** en utilisant les portes de fonctions.

Procédure

1. Activer le cgroup v2 sur les nœuds :
   a. Modifiez l'objet **node.config**:

```
$ oc edit nodes.config/cluster
```

   b. Ajouter **spec.cgroupMode: "v2"**:

Exemple d'objet node.config

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  cgroupMode: "v2" (1)
...
```

(1) Active le cgroup v2.

Vérification

1. Vérifiez les configurations de la machine pour voir si les nouvelles configurations de la machine ont été ajoutées :

```
$ oc get mc
```

Exemple de sortie

```
NAME                                               GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
00-worker                                          52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0
33m
01-master-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-master-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-container-runtime                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
01-worker-kubelet                                  52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
97-master-generated-kubelet                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             3m (1)
99-worker-generated-kubelet                              52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             3m
99-master-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-master-ssh                                                       3.2.0             40m
99-worker-generated-registries                           52dd3ba6a9a527fc3ab42afac8d12b693534c8c9
3.2.0             33m
99-worker-ssh                                                       3.2.0             40m
rendered-master-23e785de7587df95a4b517e0647e5ab7
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
rendered-worker-5d596d9293ca3ea80c896a1191735bb1
52dd3ba6a9a527fc3ab42afac8d12b693534c8c9   3.2.0             33m
worker-enable-cgroups-v2                                            3.2.0             10s
```

(1) De nouvelles configurations de machines sont créées, comme prévu.

2. Vérifier que les nouvelles **kernelArguments** ont été ajoutées aux nouvelles configurations des machines :

```
$ oc describe mc <name>
```

Exemple de sortie

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
```

```
  name: 05-worker-kernelarg-selinuxpermissive
spec:
  kernelArguments:
  - systemd_unified_cgroup_hierarchy=1 (1)
  - cgroup_no_v1="all" (2)
  - psi=1 (3)
```

(1) Active cgroup v2 dans systemd.

(2) Désactive cgroups v1.

(3) Active la fonction Linux Pressure Stall Information (PSI).

3. Vérifiez que la planification sur les nœuds est désactivée. Cela indique que la modification est en cours d'application :

```
$ oc get nodes
```

Exemple de sortie

```
NAME                                STATUS                     ROLES    AGE   VERSION
ci-ln-fm1qnwt-72292-99kt6-master-0  Ready                      master   58m   v1.25.0
ci-ln-fm1qnwt-72292-99kt6-master-1  Ready                      master   58m   v1.25.0
ci-ln-fm1qnwt-72292-99kt6-master-2  Ready                      master   58m   v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-a-h5gt4   Ready,SchedulingDisabled   worker   48m
v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-b-7vtmd   Ready                      worker   48m   v1.25.0
ci-ln-fm1qnwt-72292-99kt6-worker-c-rhzkv   Ready                      worker   48m   v1.25.0
```

4. Une fois qu'un nœud est revenu à l'état **Ready**, démarrez une session de débogage pour ce nœud :

```
oc debug node/<node_name>
```

5. Définir **/host** comme répertoire racine dans l'interpréteur de commandes de débogage :

```
sh-4.4# chroot /host
```

6. Vérifiez que le fichier **sys/fs/cgroup/cgroup2fs** est présent sur vos nœuds. Ce fichier est créé par le cgroup v2 :

```
$ stat -c %T -f /sys/fs/cgroup
```

Exemple de sortie

```
cgroup2fs
```

Ressources supplémentaires

- Enabling OpenShift Container Platform features using FeatureGates
- OpenShift Container Platform installation overview

7.7. ACTIVATION DE FONCTIONNALITÉS À L'AIDE DE PORTES DE FONCTIONNALITÉS

En tant qu'administrateur, vous pouvez utiliser des barrières de fonctionnalités pour activer des fonctionnalités qui ne font pas partie de l'ensemble des fonctionnalités par défaut.

7.7.1. Comprendre les "feature gates

Vous pouvez utiliser la ressource personnalisée (CR) **FeatureGate** pour activer des ensembles de fonctionnalités spécifiques dans votre cluster. Un ensemble de fonctionnalités est une collection de fonctionnalités d'OpenShift Container Platform qui ne sont pas activées par défaut.

Vous pouvez activer l'ensemble des fonctions suivantes en utilisant le CR **FeatureGate**:

- **TechPreviewNoUpgrade**. Cet ensemble de fonctionnalités est un sous-ensemble des fonctionnalités de l'aperçu technologique actuel. Cet ensemble de fonctionnalités vous permet d'activer ces fonctionnalités d'aperçu technologique sur des clusters de test, où vous pouvez les tester entièrement, tout en laissant les fonctionnalités désactivées sur les clusters de production.

  AVERTISSEMENT

  L'activation de l'ensemble de fonctionnalités **TechPreviewNoUpgrade** sur votre cluster ne peut pas être annulée et empêche les mises à jour mineures de la version. Vous ne devez pas activer ce jeu de fonctionnalités sur les clusters de production.

  Les fonctionnalités suivantes de l'aperçu technologique sont activées par cet ensemble de fonctionnalités :

  - Migration automatique CSI. Permet la migration automatique des plugins de volume in-tree pris en charge vers leurs pilotes CSI (Container Storage Interface) équivalents. Pris en charge pour :
    - Fichier Azure (**CSIMigrationAzureFile**)
    - VMware vSphere (**CSIMigrationvSphere**)
  - Shared Resources CSI Driver and Build CSI Volumes in OpenShift Builds. Active l'interface de stockage de conteneurs (CSI). (**CSIDriverSharedResource**)
  - Volumes CSI. Active la prise en charge des volumes CSI pour le système de construction de la plateforme OpenShift Container. (**BuildCSIVolumes**)
  - Mémoire d'échange sur les nœuds. Active l'utilisation de la mémoire d'échange pour les charges de travail de la plateforme OpenShift Container sur une base par nœud. (**NodeSwap**)
  - cgroups v2. Active cgroup v2, la prochaine version de l'API cgroup de Linux. (**CGroupsV2**)
  - crun. Active l'exécution du conteneur crun. (**Crun**)

- Insights Operator. Active l'opérateur Insights, qui rassemble les données de configuration d'OpenShift Container Platform et les envoie à Red Hat. (**InsightsConfigAPI**)
- Fournisseurs de clouds externes. Permet la prise en charge des fournisseurs de cloud externes pour les clusters sur vSphere, AWS, Azure et GCP. La prise en charge d'OpenStack est de type GA. (**ExternalCloudProvider**)
- Contraintes d'étalement de la topologie du pod. Active le paramètre **matchLabelKeys** pour les contraintes de topologie des pods. Le paramètre est une liste de clés d'étiquettes de pods permettant de sélectionner les pods sur lesquels l'étalement sera calculé. (**MatchLabelKeysInPodTopologySpread**)
- Application de l'admission à la sécurité des pods. Active l'application restreinte de l'admission à la sécurité des pods. Au lieu d'enregistrer un simple avertissement, les pods sont rejetés s'ils ne respectent pas les normes de sécurité des pods. (**OpenShiftPodSecurityAdmission**)

NOTE

L'application restreinte de l'admission à la sécurité des pods n'est activée que si vous activez l'ensemble de fonctionnalités **TechPreviewNoUpgrade** après l'installation de votre cluster OpenShift Container Platform. Elle n'est pas activée si vous activez l'ensemble de fonctionnalités **TechPreviewNoUpgrade** pendant l'installation du cluster.

Pour plus d'informations sur les fonctions activées par le portail de fonctions **TechPreviewNoUpgrade**, voir les rubriques suivantes :

- Volumes éphémères en ligne CSI
- Migration automatique CSI
- Utilisation de l'interface de stockage des conteneurs (CSI)
- Volumes de construction source-image (S2I) et volumes de construction Docker
- Échange de mémoire sur les nœuds
- Gérer les machines avec l'API Cluster
- Activation du groupe de contrôle Linux version 2 (cgroup v2)
- À propos du moteur de conteneurs et de l'exécution des conteneurs
- Using Insights Operator
- Contrôle du placement des pods à l'aide de contraintes d'étalement de la topologie des pods
- Pod Security Admission dans la documentation Kubernetes et Understanding and managing pod security admission

7.7.2. Activation des jeux de fonctionnalités à l'installation

Vous pouvez activer les jeux de fonctionnalités pour tous les nœuds de la grappe en modifiant le fichier **install-config.yaml** avant de déployer la grappe.

Conditions préalables

- You have an **install-config.yaml** file.

Procédure

1. Utilisez le paramètre **featureSet** pour spécifier le nom de l'ensemble de fonctions que vous souhaitez activer, par exemple **TechPreviewNoUpgrade**:

AVERTISSEMENT

L'activation de l'ensemble de fonctionnalités **TechPreviewNoUpgrade** sur votre cluster ne peut pas être annulée et empêche les mises à jour mineures de la version. Vous ne devez pas activer ce jeu de fonctionnalités sur les clusters de production.

Exemple de fichier install-config.yaml avec un ensemble de fonctionnalités activées

```
compute:
- hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 500
        type: io1
      metadataService:
        authentication: Optional
      type: c5.4xlarge
      zones:
      - us-west-2c
  replicas: 3
featureSet: TechPreviewNoUpgrade
```

2. Enregistrez le fichier et faites-y référence lorsque vous utilisez le programme d'installation pour déployer le cluster.

Vérification

Vous pouvez vérifier que les portes de fonctionnalités sont activées en consultant le fichier **kubelet.conf** d'un nœud après le retour des nœuds à l'état prêt.

1. Depuis la perspective **Administrator** dans la console web, naviguez vers **Compute → Nodes**.
2. Sélectionnez un nœud.
3. Dans la page **Node details**, cliquez sur **Terminal**.
4. Dans la fenêtre du terminal, changez votre répertoire racine en /**host**:

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

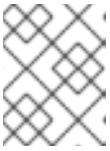
5. Consulter le fichier **kubelet.conf**:

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

Exemple de sortie

```
...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
...
```

Les fonctionnalités répertoriées à l'adresse **true** sont activées sur votre cluster.



NOTE

Les fonctionnalités listées varient en fonction de la version d'OpenShift Container Platform.

7.7.3. Activation des jeux de fonctionnalités à l'aide de la console web

Vous pouvez utiliser la console Web d'OpenShift Container Platform pour activer les jeux de fonctionnalités pour tous les nœuds d'un cluster en modifiant la ressource personnalisée (CR) **FeatureGate**.

Procédure

Pour activer les jeux de fonctionnalités :

1. Dans la console web d'OpenShift Container Platform, passez à la page **Administration → Custom Resource Definitions**.
2. Sur la page **Custom Resource Definitions**, cliquez sur **FeatureGate**.
3. Sur la page **Custom Resource Definition Details**, cliquez sur l'onglet **Instances**.
4. Cliquez sur le portail de fonctionnalités **cluster**, puis sur l'onglet **YAML**.
5. Modifier l'instance **cluster** pour ajouter des ensembles de fonctionnalités spécifiques :

AVERTISSEMENT

L'activation de l'ensemble de fonctionnalités **TechPreviewNoUpgrade** sur votre cluster ne peut pas être annulée et empêche les mises à jour mineures de la version. Vous ne devez pas activer ce jeu de fonctionnalités sur les clusters de production.

Exemple de ressource personnalisée Feature Gate

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster (1)
....

spec:
  featureSet: TechPreviewNoUpgrade (2)
```

(1) Le nom du CR **FeatureGate** doit être **cluster**.

(2) Ajoutez l'ensemble de fonctions que vous souhaitez activer :

- **TechPreviewNoUpgrade** permet d'activer des fonctionnalités spécifiques de l'aperçu technologique.

Après avoir enregistré les modifications, de nouvelles configurations de machines sont créées, les pools de configurations de machines sont mis à jour et la planification sur chaque nœud est désactivée pendant l'application de la modification.

Vérification

Vous pouvez vérifier que les portes de fonctionnalités sont activées en consultant le fichier **kubelet.conf** d'un nœud après le retour des nœuds à l'état prêt.

1. Depuis la perspective **Administrator** dans la console web, naviguez vers **Compute → Nodes**.
2. Sélectionnez un nœud.
3. Dans la page **Node details**, cliquez sur **Terminal**.
4. Dans la fenêtre du terminal, changez votre répertoire racine en /**host**:

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

5. Consulter le fichier **kubelet.conf**:

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

Exemple de sortie

```
 ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
 ...
```

Les fonctionnalités répertoriées à l'adresse **true** sont activées sur votre cluster.

NOTE

Les fonctionnalités listées varient en fonction de la version d'OpenShift Container Platform.

7.7.4. Activation des jeux de fonctionnalités à l'aide de l'interface de ligne de commande

Vous pouvez utiliser la CLI OpenShift (**oc**) pour activer les jeux de fonctionnalités pour tous les nœuds d'un cluster en modifiant la ressource personnalisée (CR) **FeatureGate**.

Conditions préalables

- Vous avez installé l'OpenShift CLI (**oc**).

Procédure

Pour activer les jeux de fonctionnalités :

1. Modifier le CR **FeatureGate** nommé **cluster**:

```
$ oc edit featuregate cluster
```

AVERTISSEMENT

L'activation de l'ensemble de fonctionnalités **TechPreviewNoUpgrade** sur votre cluster ne peut pas être annulée et empêche les mises à jour mineures de la version. Vous ne devez pas activer ce jeu de fonctionnalités sur les clusters de production.

Exemple de ressource personnalisée FeatureGate

```
apiVersion: config.openshift.io/v1
kind: FeatureGate
metadata:
  name: cluster 1
spec:
  featureSet: TechPreviewNoUpgrade 2
```

1. Le nom du CR **FeatureGate** doit être **cluster**.
2. Ajoutez l'ensemble de fonctions que vous souhaitez activer :
   - **TechPreviewNoUpgrade** permet d'activer des fonctionnalités spécifiques de l'aperçu technologique.

Après avoir enregistré les modifications, de nouvelles configurations de machines sont créées, les pools de configurations de machines sont mis à jour et la planification sur chaque nœud est désactivée pendant l'application de la modification.

Vérification

Vous pouvez vérifier que les portes de fonctionnalités sont activées en consultant le fichier **kubelet.conf** d'un nœud après le retour des nœuds à l'état prêt.

1. Depuis la perspective **Administrator** dans la console web, naviguez vers **Compute → Nodes**.
2. Sélectionnez un nœud.
3. Dans la page **Node details**, cliquez sur **Terminal**.
4. Dans la fenêtre du terminal, changez votre répertoire racine en /**host**:

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

5. Consulter le fichier **kubelet.conf**:

```
sh-4.2# cat /etc/kubernetes/kubelet.conf
```

Exemple de sortie

```
 ...
featureGates:
  InsightsOperatorPullingSCA: true,
  LegacyNodeRoleBehavior: false
 ...
```

Les fonctionnalités répertoriées à l'adresse **true** sont activées sur votre cluster.

NOTE

Les fonctionnalités listées varient en fonction de la version d'OpenShift Container Platform.

7.8. AMÉLIORATION DE LA STABILITÉ DES GRAPPES DANS LES ENVIRONNEMENTS À FORTE LATENCE À L'AIDE DE PROFILS DE LATENCE DES TRAVAILLEURS

Tous les nœuds envoient des battements de cœur à l'opérateur du contrôleur Kubernetes (kube controller) dans le cluster OpenShift Container Platform toutes les 10 secondes, par défaut. Si le cluster ne reçoit pas de battements de cœur d'un nœud, OpenShift Container Platform répond à l'aide de plusieurs mécanismes par défaut.

Par exemple, si l'opérateur du gestionnaire de contrôleur Kubernetes perd le contact avec un nœud après une période configurée :

1. Le contrôleur de nœuds sur le plan de contrôle met à jour l'état du nœud à **Unhealthy** et marque l'état du nœud **Ready** comme **Unknown**.
2. En réponse, l'ordonnanceur arrête de programmer des pods sur ce nœud.
3. Le contrôleur de nœuds sur site ajoute au nœud une tare **node.kubernetes.io/unreachable** avec un effet **NoExecute** et planifie l'éviction de tous les pods du nœud après cinq minutes, par défaut.

Ce comportement peut poser des problèmes si votre réseau est sujet à des problèmes de latence, en particulier si vous avez des nœuds à la périphérie du réseau. Dans certains cas, l'opérateur du gestionnaire de contrôleur Kubernetes peut ne pas recevoir de mise à jour d'un nœud sain en raison de la latence du réseau. L'opérateur du gestionnaire de contrôleur Kubernetes expulserait alors les pods du nœud, même si celui-ci est sain. Pour éviter ce problème, vous pouvez utiliser *worker latency profiles* pour ajuster la fréquence à laquelle le kubelet et le Kubernetes Controller Manager Operator attendent les mises à jour d'état avant d'agir. Ces ajustements permettent de s'assurer que votre cluster fonctionne correctement dans le cas où la latence du réseau entre le plan de contrôle et les nœuds de travail n'est pas optimale.

Ces profils de latence des travailleurs sont trois ensembles de paramètres prédéfinis avec des valeurs soigneusement ajustées qui vous permettent de contrôler la réaction du cluster aux problèmes de latence sans avoir à déterminer les meilleures valeurs manuellement.

Vous pouvez configurer les profils de latence des travailleurs lors de l'installation d'un cluster ou à tout moment lorsque vous constatez une augmentation de la latence dans votre réseau de clusters.

7.8.1. Comprendre les profils de latence des travailleurs

Les profils de latence des travailleurs sont des ensembles multiples de valeurs soigneusement ajustées pour les paramètres **node-status-update-frequency**, **node-monitor-grace-period**, **default-not-ready-toleration-seconds** et **default-unreachable-toleration-seconds**. Ces paramètres vous permettent de contrôler la réaction du cluster aux problèmes de latence sans avoir à déterminer manuellement les meilleures valeurs.

Tous les profils de latence des travailleurs configurent les paramètres suivants :

- **node-status-update-frequency**. Spécifie le temps en secondes pendant lequel un kubelet met à jour son statut auprès de l'opérateur du gestionnaire de contrôleur Kubernetes.
- **node-monitor-grace-period**. Spécifie la durée en secondes pendant laquelle l'opérateur Kubernetes Controller Manager attend une mise à jour d'un kubelet avant de marquer le nœud comme étant malsain et d'ajouter l'erreur **node.kubernetes.io/not-ready** ou **node.kubernetes.io/unreachable** au nœud.
- **default-not-ready-toleration-seconds**. Spécifie la durée en secondes après le marquage d'un nœud insalubre que l'opérateur Kubernetes Controller Manager attend avant d'expulser les pods de ce nœud.
- **default-unreachable-toleration-seconds**. Spécifie la durée en secondes pendant laquelle l'opérateur du gestionnaire de contrôle Kubernetes attend qu'un nœud soit marqué comme inaccessible avant d'expulser les pods de ce nœud.

IMPORTANT

La modification manuelle du paramètre **node-monitor-grace-period** n'est pas possible.

Les opérateurs suivants surveillent les modifications apportées aux profils de latence des travailleurs et réagissent en conséquence :

- L'opérateur de configuration de la machine (MCO) met à jour le paramètre **node-status-update-frequency** sur les nœuds de travail.
- L'opérateur du gestionnaire de contrôleur Kubernetes met à jour le paramètre **node-monitor-grace-period** sur les nœuds du plan de contrôle.

- L'opérateur du serveur API Kubernetes met à jour les paramètres **default-not-ready-toleration-seconds** et **default-unreachable-toleration-seconds** sur les nœuds de la plance de contrôle.

Bien que la configuration par défaut fonctionne dans la plupart des cas, OpenShift Container Platform propose deux autres profils de latence du travailleur pour les situations où le réseau connaît une latence plus élevée que d'habitude. Les trois profils de latence des travailleurs sont décrits dans les sections suivantes :

Profil de latence du travailleur par défaut

Avec le profil **Default**, chaque kubelet signale l'état de son nœud au Kubelet Controller Manager Operator (kube controller) toutes les 10 secondes. L'opérateur du gestionnaire du contrôleur de kubelet vérifie l'état du kubelet toutes les 5 secondes.
L'opérateur du gestionnaire de contrôle Kubernetes attend 40 secondes pour une mise à jour de l'état avant de considérer ce nœud comme malsain. Il marque le nœud avec la taint **node.kubernetes.io/not-ready** ou **node.kubernetes.io/unreachable** et expulse les pods sur ce nœud. Si un pod sur ce nœud a la tolérance **NoExecute**, le pod est expulsé en 300 secondes. Si le pod a le paramètre **tolerationSeconds**, l'expulsion attend la période spécifiée par ce paramètre.

Profile	Composant	Paramètres	Valeur
Défaut	kubelet	**node-status-update-frequency**	10s
	Gestionnaire de contrôleur Kubelet	**node-monitor-grace-period**	40s
	Serveur API Kubernetes	**default-not-ready-toleration-seconds**	300s
	Serveur API Kubernetes	**default-unreachable-toleration-seconds**	300s

Profil de latence du travailleur moyen

Utilisez le profil **MediumUpdateAverageReaction** si la latence du réseau est légèrement plus élevée que d'habitude.
Le profil **MediumUpdateAverageReaction** réduit la fréquence des mises à jour des kubelets à 20 secondes et modifie la période d'attente de ces mises à jour par l'opérateur du contrôleur Kubernetes à 2 minutes. La période d'éviction d'un pod sur ce nœud est réduite à 60 secondes. Si le pod a le paramètre **tolerationSeconds**, l'éviction attend la période spécifiée par ce paramètre.

L'opérateur Kubernetes Controller Manager attend 2 minutes pour considérer qu'un nœud n'est pas sain. Une minute plus tard, le processus d'expulsion commence.

Profile	Composant	Paramètres	Valeur
Moyenne des mises à jour	kubelet	**node-status-update-frequency**	20s

Profile	Composant	Paramètres	Valeur
	Gestionnaire de contrôleur Kubelet	**node-monitor-grace-period**	2m
	Serveur API Kubernetes	**default-not-ready-toleration-seconds**	60s
	Serveur API Kubernetes	**default-unreachable-toleration-seconds**	60s

Profil de latence faible pour les travailleurs

Utilisez le profil **LowUpdateSlowReaction** si la latence du réseau est extrêmement élevée. Le profil **LowUpdateSlowReaction** réduit la fréquence des mises à jour des kubelets à 1 minute et modifie la période d'attente de ces mises à jour par l'opérateur du contrôleur Kubernetes à 5 minutes. La période d'éviction d'un pod sur ce nœud est réduite à 60 secondes. Si le pod a le paramètre **tolerationSeconds**, l'éviction attend la période spécifiée par ce paramètre.

L'opérateur Kubernetes Controller Manager attend 5 minutes pour considérer qu'un nœud n'est pas sain. Dans une minute, le processus d'expulsion commence.

Profile	Composant	Paramètres	Valeur
Faible mise à jourRéaction lente	kubelet	**node-status-update-frequency**	1m
	Gestionnaire de contrôleur Kubelet	**node-monitor-grace-period**	5m
	Serveur API Kubernetes	**default-not-ready-toleration-seconds**	60s
	Serveur API Kubernetes	**default-unreachable-toleration-seconds**	60s

7.8.2. Utilisation des profils de latence des travailleurs

Pour mettre en œuvre un profil de latence du travailleur afin de gérer la latence du réseau, modifiez l'objet **node.config** pour ajouter le nom du profil. Vous pouvez modifier le profil à tout moment lorsque la latence augmente ou diminue.

Vous devez déplacer un profil de latence de travailleur à la fois. Par exemple, vous ne pouvez pas passer directement du profil **Default** au profil **LowUpdateSlowReaction**. Vous devez d'abord passer du profil **default** au profil **MediumUpdateAverageReaction**, puis à **LowUpdateSlowReaction**. De même,

lorsque vous revenez au profil par défaut, vous devez d'abord passer du profil bas au profil moyen, puis au profil par défaut.

NOTE

Vous pouvez également configurer les profils de latence des travailleurs lors de l'installation d'un cluster OpenShift Container Platform.

Procédure

Pour quitter le profil de latence par défaut du travailleur :

1. Passer au profil de latence du travailleur moyen :
   a. Modifiez l'objet **node.config**:

```
$ oc edit nodes.config/cluster
```

   b. Ajouter **spec.workerLatencyProfile: MediumUpdateAverageReaction**:

Exemple d'objet node.config

```
apiVersion: config.openshift.io/v1
kind: Node
metadata:
  annotations:
    include.release.openshift.io/ibm-cloud-managed: "true"
    include.release.openshift.io/self-managed-high-availability: "true"
    include.release.openshift.io/single-node-developer: "true"
    release.openshift.io/create-only: "true"
  creationTimestamp: "2022-07-08T16:02:51Z"
  generation: 1
  name: cluster
  ownerReferences:
  - apiVersion: config.openshift.io/v1
    kind: ClusterVersion
    name: version
    uid: 36282574-bf9f-409e-a6cd-3032939293eb
  resourceVersion: "1865"
  uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
spec:
  workerLatencyProfile: MediumUpdateAverageReaction (1)

 ...
```

(1) Spécifie la politique de latence du travailleur moyen.

La programmation sur chaque nœud de travailleur est désactivée au fur et à mesure de l'application de la modification.

Lorsque tous les nœuds reviennent à l'état **Ready**, vous pouvez utiliser la commande suivante pour vérifier dans le Kubernetes Controller Manager qu'elle a bien été appliquée :

```
$ oc get KubeControllerManager -o yaml | grep -i workerlatency -A 5 -B 5
```

Exemple de sortie

```
...
  - lastTransitionTime: "2022-07-11T19:47:10Z"
    reason: ProfileUpdated
    status: "False"
    type: WorkerLatencyProfileProgressing
  - lastTransitionTime: "2022-07-11T19:47:10Z" 1
    message: all static pod revision(s) have updated latency profile
    reason: ProfileUpdated
    status: "True"
    type: WorkerLatencyProfileComplete
  - lastTransitionTime: "2022-07-11T19:20:11Z"
    reason: AsExpected
    status: "False"
    type: WorkerLatencyProfileDegraded
  - lastTransitionTime: "2022-07-11T19:20:36Z"
    status: "False"
...
```

1 Spécifie que le profil est appliqué et actif.

2. Optionnel : Passez au profil de faible latence du travailleur :

   a. Modifiez l'objet **node.config**:

   ```
   $ oc edit nodes.config/cluster
   ```

   b. Modifiez la valeur de **spec.workerLatencyProfile** en **LowUpdateSlowReaction**:

   Exemple d'objet node.config

   ```
   apiVersion: config.openshift.io/v1
   kind: Node
   metadata:
     annotations:
       include.release.openshift.io/ibm-cloud-managed: "true"
       include.release.openshift.io/self-managed-high-availability: "true"
       include.release.openshift.io/single-node-developer: "true"
       release.openshift.io/create-only: "true"
     creationTimestamp: "2022-07-08T16:02:51Z"
     generation: 1
     name: cluster
     ownerReferences:
     - apiVersion: config.openshift.io/v1
       kind: ClusterVersion
       name: version
       uid: 36282574-bf9f-409e-a6cd-3032939293eb
     resourceVersion: "1865"
     uid: 0c0f7a4c-4307-4187-b591-6155695ac85b
   spec:
     workerLatencyProfile: LowUpdateSlowReaction 1

     ...
   ```

```
-
```

1 Spécifie l'utilisation de la politique de faible latence du travailleur.

La programmation sur chaque nœud de travailleur est désactivée au fur et à mesure de l'application de la modification.

Pour transformer le profil bas en profil moyen ou le profil moyen en profil bas, modifiez l'objet **node.config** et réglez le paramètre **spec.workerLatencyProfile** sur la valeur appropriée.

CHAPITRE 8. NŒUDS DE TÉLÉTRAVAILLEURS À LA PÉRIPHÉRIE DU RÉSEAU

8.1. UTILISATION DE NŒUDS DE TÉLÉTRAVAIL À LA PÉRIPHÉRIE DU RÉSEAU

Vous pouvez configurer les clusters OpenShift Container Platform avec des nœuds situés à la périphérie de votre réseau. Dans cette rubrique, ils sont appelés *remote worker nodes*. Un cluster typique avec des nœuds de travail à distance combine des nœuds maîtres et des nœuds de travail sur site avec des nœuds de travail dans d'autres endroits qui se connectent au cluster. Cette rubrique vise à fournir des conseils sur les meilleures pratiques d'utilisation des nœuds de télétravail et ne contient pas de détails de configuration spécifiques.

Il existe de multiples cas d'utilisation dans différents secteurs, tels que les télécommunications, la vente au détail, la fabrication et le gouvernement, pour utiliser un modèle de déploiement avec des nœuds de travailleurs distants. Par exemple, vous pouvez séparer et isoler vos projets et charges de travail en combinant les nœuds de travailleurs distants dans des zones Kubernetes.

Cependant, l'existence de nœuds de travail distants peut entraîner une latence plus élevée, une perte intermittente de connectivité réseau et d'autres problèmes. Les défis d'un cluster avec des nœuds de travail distants sont notamment les suivants :

- **Network separation**: Le plan de contrôle d'OpenShift Container Platform et les nœuds de télétravail doivent pouvoir communiquer entre eux. En raison de la distance entre le plan de contrôle et les nœuds de télétravail, des problèmes de réseau peuvent empêcher cette communication. Voir Séparation du réseau avec les nœuds de télétravail pour plus d'informations sur la façon dont OpenShift Container Platform réagit à la séparation du réseau et sur les méthodes permettant de réduire l'impact sur votre cluster.

- **Power outage**: Étant donné que le plan de contrôle et les nœuds de télétravail se trouvent dans des endroits distincts, une panne de courant à l'emplacement distant ou à n'importe quel point entre les deux peut avoir un impact négatif sur votre cluster. Voir Power loss on remote worker nodes pour des informations sur la façon dont OpenShift Container Platform réagit à une perte d'alimentation d'un nœud et pour des méthodes permettant de diminuer l'impact sur votre cluster.

- **Latency spikes or temporary reduction in throughput** Comme pour tout réseau, tout changement dans les conditions du réseau entre votre cluster et les nœuds de travail distants peut avoir un impact négatif sur votre cluster. OpenShift Container Platform offre plusieurs *worker latency profiles* qui vous permettent de contrôler la réaction du cluster aux problèmes de latence.

Notez les limitations suivantes lors de la planification d'une grappe avec des nœuds de travail distants :

- OpenShift Container Platform ne prend pas en charge les nœuds de travail à distance qui utilisent un fournisseur de cloud différent de celui utilisé par le cluster sur site.

- Le déplacement des charges de travail d'une zone Kubernetes vers une autre zone Kubernetes peut être problématique en raison de problèmes liés au système et à l'environnement, comme un type de mémoire spécifique qui n'est pas disponible dans une autre zone.

- Les proxys et les pare-feux peuvent présenter des limitations supplémentaires qui dépassent le cadre de ce document. Voir la documentation pertinente de OpenShift Container Platform pour savoir comment traiter ces limitations, comme par exemple Configurer votre pare-feu.

- Vous êtes responsable de la configuration et du maintien de la connectivité réseau de niveau L2/L3 entre le plan de contrôle et les nœuds de bordure du réseau.

8.1.1. Séparation des réseaux avec des nœuds de travail à distance

Tous les nœuds envoient des battements de cœur à l'opérateur du gestionnaire de contrôleur Kubernetes (kube controller) dans le cluster OpenShift Container Platform toutes les 10 secondes. Si le cluster ne reçoit pas de battements de cœur d'un nœud, OpenShift Container Platform répond à l'aide de plusieurs mécanismes par défaut.

OpenShift Container Platform est conçue pour résister aux partitions de réseau et autres perturbations. Vous pouvez atténuer certaines des perturbations les plus courantes, telles que les interruptions dues à des mises à jour logicielles, des divisions de réseau et des problèmes de routage. Les stratégies d'atténuation consistent notamment à s'assurer que les pods sur les nœuds de travail distants demandent la quantité correcte de ressources CPU et mémoire, à configurer une politique de réplication appropriée, à utiliser la redondance entre les zones et à utiliser des budgets de perturbation des pods sur les charges de travail.

Si le contrôleur kube perd le contact avec un nœud après une période configurée, le contrôleur de nœud sur le plan de contrôle met à jour l'état du nœud à **Unhealthy** et marque l'état du nœud **Ready** comme **Unknown**. En réponse, le planificateur arrête de planifier des pods pour ce nœud. Le contrôleur de nœuds sur site ajoute une erreur **node.kubernetes.io/unreachable** avec un effet **NoExecute** au nœud et planifie l'éviction des pods sur le nœud après cinq minutes, par défaut.

Si un contrôleur de charge de travail, tel qu'un objet **Deployment** ou **StatefulSet**, dirige le trafic vers des pods sur le nœud malsain et que d'autres nœuds peuvent atteindre le cluster, OpenShift Container Platform achemine le trafic loin des pods sur le nœud. Les nœuds qui ne peuvent pas atteindre le cluster ne sont pas mis à jour avec le nouveau routage du trafic. Par conséquent, les charges de travail sur ces nœuds peuvent continuer à essayer d'atteindre le nœud malsain.

Vous pouvez atténuer les effets de la perte de connexion en procédant comme suit :

- l'utilisation d'ensembles de démons pour créer des pods qui tolèrent les taches
- utiliser des pods statiques qui redémarrent automatiquement si un nœud tombe en panne
- utiliser les zones Kubernetes pour contrôler l'éviction des pods
- configurer les tolérances des pods pour retarder ou éviter l'éviction des pods
- configurer le kubelet pour contrôler le moment où il marque les nœuds comme étant malsains.

Pour plus d'informations sur l'utilisation de ces objets dans un cluster avec des nœuds de télétravail, voir À propos des stratégies de nœuds de télé travail.

8.1.2. Perte de puissance sur les nœuds des travailleurs distants

Si un nœud de télétravail perd de l'énergie ou redémarre de façon anarchique, OpenShift Container Platform réagit à l'aide de plusieurs mécanismes par défaut.

Si l'opérateur du gestionnaire de contrôleur Kubernetes (contrôleur kube) perd le contact avec un nœud après une période configurée, le plan de contrôle met à jour la santé du nœud à **Unhealthy** et marque l'état du nœud **Ready** comme **Unknown**. En réponse, le planificateur arrête de planifier des pods sur ce nœud. Le contrôleur de nœuds sur site ajoute une tare **node.kubernetes.io/unreachable** avec un effet **NoExecute** au nœud et planifie l'éviction des pods sur le nœud au bout de cinq minutes, par défaut.

Sur le nœud, les pods doivent être redémarrés lorsque le nœud est remis sous tension et reconnecté au plan de contrôle.

NOTE

Si vous souhaitez que les pods redémarrent immédiatement, utilisez des pods statiques.

Après le redémarrage du nœud, le kubelet redémarre également et tente de redémarrer les pods qui ont été planifiés sur le nœud. Si la connexion au plan de contrôle prend plus de temps que les cinq minutes par défaut, le plan de contrôle ne peut pas mettre à jour l'état du nœud et supprimer l'erreur **node.kubernetes.io/unreachable**. Sur le nœud, le kubelet met fin à tous les pods en cours d'exécution. Lorsque ces conditions sont levées, le planificateur peut commencer à planifier des pods sur ce nœud.

Vous pouvez atténuer les effets de la perte de puissance en :

- l'utilisation d'ensembles de démons pour créer des pods qui tolèrent les taches
- utiliser des pods statiques qui redémarrent automatiquement avec un nœud
- configurer les tolérances des pods pour retarder ou éviter l'éviction des pods
- configurer le kubelet pour contrôler le moment où le contrôleur de nœuds marque les nœuds comme étant malsains.

Pour plus d'informations sur l'utilisation de ces objets dans un cluster avec des nœuds de télétravail, voir À propos des stratégies de nœuds de télé travail.

8.1.3. Pics de latence ou réduction temporaire du débit pour les travailleurs à distance

Tous les nœuds envoient des battements de cœur à l'opérateur du contrôleur Kubernetes (kube controller) dans le cluster OpenShift Container Platform toutes les 10 secondes, par défaut. Si le cluster ne reçoit pas de battements de cœur d'un nœud, OpenShift Container Platform répond à l'aide de plusieurs mécanismes par défaut.

Par exemple, si l'opérateur du gestionnaire de contrôleur Kubernetes perd le contact avec un nœud après une période configurée :

1. Le contrôleur de nœuds sur le plan de contrôle met à jour l'état du nœud à **Unhealthy** et marque l'état du nœud **Ready** comme **Unknown**.
2. En réponse, l'ordonnanceur arrête de programmer des pods sur ce nœud.
3. Le contrôleur de nœuds sur site ajoute au nœud une tare **node.kubernetes.io/unreachable** avec un effet **NoExecute** et planifie l'éviction de tous les pods du nœud après cinq minutes, par défaut.

Ce comportement peut poser des problèmes si votre réseau est sujet à des problèmes de latence, en particulier si vous avez des nœuds à la périphérie du réseau. Dans certains cas, l'opérateur du gestionnaire de contrôleur Kubernetes peut ne pas recevoir de mise à jour d'un nœud sain en raison de la latence du réseau. L'opérateur du gestionnaire de contrôleur Kubernetes expulserait alors les pods du nœud, même si celui-ci est sain. Pour éviter ce problème, vous pouvez utiliser *worker latency profiles* pour ajuster la fréquence à laquelle le kubelet et le Kubernetes Controller Manager Operator attendent les mises à jour d'état avant d'agir. Ces ajustements permettent de s'assurer que votre cluster fonctionne correctement dans le cas où la latence du réseau entre le plan de contrôle et les nœuds de travail n'est pas optimale.

Ces profils de latence des travailleurs sont trois ensembles de paramètres prédéfinis avec des valeurs soigneusement ajustées qui vous permettent de contrôler la réaction du cluster aux problèmes de latence sans avoir à déterminer les meilleures valeurs manuellement.

Ressources supplémentaires

- Amélioration de la stabilité des grappes dans les environnements à forte latence à l'aide de profils de latence des travailleurs

8.1.4. Stratégies pour les nœuds de télétravailleurs

Si vous utilisez des nœuds de travail à distance, réfléchissez aux objets à utiliser pour exécuter vos applications.

Il est recommandé d'utiliser des ensembles de démons ou des pods statiques en fonction du comportement souhaité en cas de problèmes de réseau ou de perte d'alimentation. En outre, vous pouvez utiliser les zones et les tolérances Kubernetes pour contrôler ou éviter les évictions de pods si le plan de contrôle ne peut pas atteindre les nœuds de travail distants.

Jeux de démons

Les ensembles de démons constituent la meilleure approche pour gérer les pods sur les nœuds de travail distants pour les raisons suivantes :

- Les ensembles de démons n'ont généralement pas besoin d'un comportement de replanification. Si un nœud se déconnecte du cluster, les pods sur le nœud peuvent continuer à fonctionner. OpenShift Container Platform ne modifie pas l'état des pods daemon set, et laisse les pods dans l'état qu'ils ont signalé pour la dernière fois. Par exemple, si un daemon set pod est dans l'état **Running**, lorsqu'un nœud arrête de communiquer, le pod continue de fonctionner et est supposé être en cours d'exécution par OpenShift Container Platform.

- Par défaut, les pods de l'ensemble démon sont créés avec des tolérances de **NoExecute** pour les taints **node.kubernetes.io/unreachable** et **node.kubernetes.io/not-ready**, sans valeur de **tolerationSeconds**. Ces valeurs par défaut garantissent que les pods de l'ensemble démon ne sont jamais expulsés si le plan de contrôle ne peut pas atteindre un nœud. Par exemple :

  Tolérances ajoutées aux pods du daemon par défaut

```
tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
  - key: node.kubernetes.io/disk-pressure
    operator: Exists
    effect: NoSchedule
  - key: node.kubernetes.io/memory-pressure
    operator: Exists
    effect: NoSchedule
  - key: node.kubernetes.io/pid-pressure
    operator: Exists
    effect: NoSchedule
```

```
- key: node.kubernetes.io/unschedulable
  operator: Exists
  effect: NoSchedule
```

- Les ensembles de démons peuvent utiliser des étiquettes pour s'assurer qu'une charge de travail s'exécute sur un nœud de travail correspondant.
- Vous pouvez utiliser un point d'extrémité de service OpenShift Container Platform pour équilibrer la charge des pods de l'ensemble des démons.

NOTE

Les ensembles de démons ne planifient pas les pods après un redémarrage du nœud si OpenShift Container Platform ne peut pas atteindre le nœud.

Nacelles statiques

Si vous souhaitez que les pods redémarrent en cas de redémarrage d'un nœud, après une panne de courant par exemple, envisagez des pods statiques. Le kubelet sur un nœud redémarre automatiquement les pods statiques lorsque le nœud redémarre.

NOTE

Les pods statiques ne peuvent pas utiliser les secrets et les cartes de configuration.

Zones Kubernetes

Les zones Kubernetes peuvent ralentir le rythme ou, dans certains cas, arrêter complètement les évictions de pods.

Lorsque le plan de contrôle ne peut pas atteindre un nœud, le contrôleur de nœud, par défaut, applique **node.kubernetes.io/unreachable** taints et expulse les pods à un taux de 0,1 nœud par seconde. Cependant, dans un cluster qui utilise des zones Kubernetes, le comportement d'éviction des pods est modifié.

Si une zone est totalement perturbée et que tous les nœuds de la zone ont une condition **Ready** qui est **False** ou **Unknown**, le plan de contrôle n'applique pas l'erreur **node.kubernetes.io/unreachable** aux nœuds de cette zone.

Pour les zones partiellement perturbées, où plus de 55 % des nœuds ont une condition **False** ou **Unknown**, le taux d'éviction des pods est réduit à 0,01 nœud par seconde. Les nœuds des grappes plus petites, comptant moins de 50 nœuds, ne sont pas altérés. Votre cluster doit avoir plus de trois zones pour que ce comportement prenne effet.

Vous affectez un nœud à une zone spécifique en appliquant l'étiquette **topology.kubernetes.io/region** dans la spécification du nœud.

Exemples d'étiquettes de nœuds pour les zones Kubernetes

```
kind: Node
apiVersion: v1
metadata:
  labels:
    topology.kubernetes.io/region=east
```

KubeletConfig objets

Vous pouvez ajuster le temps pendant lequel le kubelet vérifie l'état de chaque nœud.

Pour définir l'intervalle qui affecte le moment où le contrôleur de nœuds sur site marque les nœuds avec la condition **Unhealthy** ou **Unreachable**, créez un objet **KubeletConfig** qui contient les paramètres **node-status-update-frequency** et **node-status-report-frequency**.

Le kubelet de chaque nœud détermine l'état du nœud tel que défini par le paramètre **node-status-update-frequency** et signale cet état au cluster en fonction du paramètre **node-status-report-frequency**. Par défaut, le kubelet détermine l'état du nœud toutes les 10 secondes et le signale toutes les minutes. Cependant, si l'état du nœud change, le kubelet signale le changement au cluster immédiatement. OpenShift Container Platform utilise le paramètre **node-status-report-frequency** uniquement lorsque la porte de fonctionnalité Node Lease est activée, ce qui est l'état par défaut dans les clusters OpenShift Container Platform. Si la porte de la fonctionnalité Node Lease est désactivée, le nœud signale son état en fonction du paramètre **node-status-update-frequency**.

Exemple de configuration d'un kubelet

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: disable-cpu-units
spec:
  machineConfigPoolSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: worker (1)
  kubeletConfig:
    node-status-update-frequency: (2)
      - "10s"
    node-status-report-frequency: (3)
      - "1m"
```

(1) Spécifiez le type de nœud auquel cet objet **KubeletConfig** s'applique en utilisant l'étiquette de l'objet **MachineConfig**.

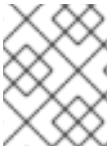
(2) Indique la fréquence à laquelle le kubelet vérifie l'état d'un nœud associé à cet objet **MachineConfig**. La valeur par défaut est **10s**. Si vous modifiez cette valeur par défaut, la valeur **node-status-report-frequency** est modifiée en conséquence.

(3) Indique la fréquence à laquelle le kubelet signale l'état d'un nœud associé à cet objet **MachineConfig**. La valeur par défaut est **1m**.

Le paramètre **node-status-update-frequency** fonctionne avec les paramètres **node-monitor-grace-period** et **pod-eviction-timeout**.

- Le paramètre **node-monitor-grace-period** spécifie la durée pendant laquelle OpenShift Container Platform attend qu'un nœud associé à un objet **MachineConfig** soit marqué **Unhealthy** si le gestionnaire de contrôleur ne reçoit pas le battement de cœur du nœud. Les charges de travail sur le nœud continuent de s'exécuter après ce délai. Si le nœud du travailleur distant rejoint le cluster après l'expiration de **node-monitor-grace-period**, les pods continuent de s'exécuter. De nouveaux pods peuvent être programmés sur ce nœud. L'intervalle **node-monitor-grace-period** est **40s**. La valeur **node-status-update-frequency** doit être inférieure à la valeur **node-monitor-grace-period**.

- Le paramètre **pod-eviction-timeout** spécifie le temps qu'OpenShift Container Platform attend après avoir marqué un nœud associé à un objet **MachineConfig** comme **Unreachable** pour commencer à marquer les pods pour l'éviction. Les pods évincés sont reprogrammés sur d'autres nœuds. Si le nœud de travailleur distant rejoint le cluster après l'expiration de **pod-eviction-timeout**, les pods exécutés sur le nœud de travailleur distant sont arrêtés car le contrôleur de nœud a expulsé les pods sur site. Les pods peuvent alors être reprogrammés sur ce nœud. L'intervalle **pod-eviction-timeout** est **5m0s**.



NOTE

La modification des paramètres **node-monitor-grace-period** et **pod-eviction-timeout** n'est pas possible.

Tolérances

Vous pouvez utiliser des tolérances de pods pour atténuer les effets si le contrôleur de nœuds sur site ajoute une erreur **node.kubernetes.io/unreachable** avec un effet **NoExecute** à un nœud qu'il ne peut pas atteindre.

Une taint avec l'effet **NoExecute** affecte les pods qui s'exécutent sur le nœud de la manière suivante :

- Les pods qui ne tolèrent pas l'altération sont mis en file d'attente pour être expulsés.
- Les pods qui tolèrent l'altération sans spécifier une valeur **tolerationSeconds** dans leur spécification de tolérance restent liés pour toujours.
- Les pods qui tolèrent l'altération avec une valeur **tolerationSeconds** spécifiée restent liés pendant la durée spécifiée. Une fois le temps écoulé, les pods sont mis en file d'attente pour être expulsés.

Vous pouvez retarder ou éviter l'éviction des pods en configurant les tolérances des pods avec l'effet **NoExecute** pour les taints **node.kubernetes.io/unreachable** et **node.kubernetes.io/not-ready**.

Exemple de tolérance dans un spec pod

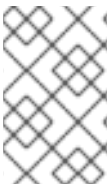
```
...
tolerations:
- key: "node.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute" 1
- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute" 2
  tolerationSeconds: 600
...
```

1. L'effet **NoExecute** sans **tolerationSeconds** permet aux pods de rester pour toujours si le plan de contrôle ne peut pas atteindre le nœud.
2. L'effet **NoExecute** avec **tolerationSeconds**: 600 permet aux pods de rester pendant 10 minutes si le plan de contrôle marque le nœud comme **Unhealthy**.

OpenShift Container Platform utilise la valeur **tolerationSeconds** après l'expiration de la valeur **pod-eviction-timeout**.

Autres types d'objets OpenShift Container Platform

Vous pouvez utiliser des ensembles de répliques, des déploiements et des contrôleurs de réplication. L'ordonnanceur peut replanifier ces pods sur d'autres nœuds après que le nœud a été déconnecté pendant cinq minutes. La replanification sur d'autres nœuds peut être bénéfique pour certaines charges de travail, telles que les API REST, pour lesquelles un administrateur peut garantir qu'un nombre spécifique de pods est en cours d'exécution et accessible.



NOTE

Lorsque vous travaillez avec des nœuds de télétravailleur, la reprogrammation de pods sur différents nœuds peut ne pas être acceptable si les nœuds de télétravailleur sont destinés à être réservés à des fonctions spécifiques.

ne sont pas redémarrés en cas de panne. Les pods restent dans l'état **terminating** jusqu'à ce que le plan de contrôle puisse reconnaître que les pods sont terminés.

Pour éviter de programmer un nœud qui n'a pas accès au même type de stockage persistant, OpenShift Container Platform ne peut pas migrer les pods qui nécessitent des volumes persistants vers d'autres zones dans le cas d'une séparation de réseau.

Ressources supplémentaires

- Pour plus d'informations sur les Daemonesets, voir DaemonSets.
- Pour plus d'informations sur les taches et les tolérances, voir Contrôle du placement des pods à l'aide des taches de nœuds.
- Pour plus d'informations sur la configuration des objets **KubeletConfig**, voir Creating a KubeletConfig CRD.
- Pour plus d'informations sur les ensembles de répliques, voir ReplicaSets.
- Pour plus d'informations sur les déploiements, voir Déploiements.
- Pour plus d'informations sur les contrôleurs de réplication, voir Contrôleurs de réplication.
- Pour plus d'informations sur le gestionnaire de contrôleur, voir Kubernetes Controller Manager Operator.

CHAPITRE 9. NŒUDS DE TRAVAIL POUR LES CLUSTERS OPENSHIFT À NŒUD UNIQUE

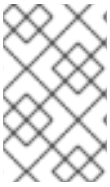
9.1. AJOUTER DES NŒUDS DE TRAVAIL À DES CLUSTERS OPENSHIFT À NŒUD UNIQUE

Les clusters OpenShift à nœud unique réduisent les prérequis de l'hôte pour le déploiement à un seul hôte. Ceci est utile pour les déploiements dans des environnements contraints ou à la périphérie du réseau. Cependant, il est parfois nécessaire d'ajouter une capacité supplémentaire à votre cluster, par exemple, dans les scénarios de télécommunications et de périphérie de réseau. Dans ce cas, vous pouvez ajouter des nœuds de travail au cluster à nœud unique.

Il existe plusieurs façons d'ajouter des nœuds ouvriers à un cluster à nœud unique. Vous pouvez ajouter des nœuds ouvriers à un cluster manuellement, en utilisant Red Hat OpenShift Cluster Manager , ou en utilisant directement l'API REST Assisted Installer.

IMPORTANT

L'ajout de nœuds de travail n'étend pas le plan de contrôle du cluster et ne fournit pas de haute disponibilité à votre cluster. Pour les clusters OpenShift à un seul nœud, la haute disponibilité est gérée en basculant sur un autre site. Il n'est pas recommandé d'ajouter un grand nombre de nœuds de travail à un cluster à nœud unique.



NOTE

Contrairement aux grappes à plusieurs nœuds, tout le trafic entrant est acheminé par défaut vers le seul nœud du plan de contrôle, même après l'ajout de nœuds de travail supplémentaires.

9.1.1. Configuration requise pour l'installation de nœuds de travail OpenShift à un seul nœud

Pour installer un nœud de travail OpenShift à nœud unique, vous devez répondre aux exigences suivantes :

- **Administration host:** Vous devez disposer d'un ordinateur pour préparer l'ISO et suivre l'installation.
- **Production-grade server:** L'installation de nœuds de travail OpenShift à un seul nœud nécessite un serveur disposant de ressources suffisantes pour exécuter les services OpenShift Container Platform et une charge de travail de production.

Tableau 9.1. Minimum resource requirements

Profile	vCPU	Mémoire	Stockage
Minimum	2 cœurs vCPU	8 Go de RAM	100GB

NOTE

One vCPU is equivalent to one physical core when simultaneous multithreading (SMT), or hyperthreading, is not enabled. When enabled, use the following formula to calculate the corresponding ratio:

(threads per core × cores) × sockets = vCPUs

The server must have a Baseboard Management Controller (BMC) when booting with virtual media.

- **Networking:** Le serveur du nœud de travail doit avoir accès à Internet ou à un registre local s'il n'est pas connecté à un réseau routable. Le serveur du nœud de travail doit avoir une réservation DHCP ou une adresse IP statique et être en mesure d'accéder à l'API Kubernetes du cluster OpenShift à nœud unique, à la route d'entrée et aux noms de domaine des nœuds de cluster. Vous devez configurer le DNS pour résoudre l'adresse IP à chacun des noms de domaine pleinement qualifiés (FQDN) suivants pour le cluster OpenShift à nœud unique :

Tableau 9.2. Required DNS records

Utilisation	FQDN	Description
Kubernetes API	**api.<cluster_name>.<base_domain>**	Add a DNS A/AAAA or CNAME record. This record must be resolvable by clients external to the cluster.
Internal API	**api-int.<cluster_name>.<base_domain>**	Add a DNS A/AAAA or CNAME record when creating the ISO manually. This record must be resolvable by nodes within the cluster.
Ingress route	***.apps.<cluster_name>.<base_domain>**	Add a wildcard DNS A/AAAA or CNAME record that targets the node. This record must be resolvable by clients external to the cluster.

Without persistent IP addresses, communications between the **apiserver** and **etcd** might fail.

Ressources supplémentaires

- Ressources minimales requises pour l'installation d'un cluster
- Pratiques recommandées pour la mise à l'échelle du cluster
- Exigences DNS fournies par l'utilisateur
- Creating a bootable ISO image on a USB drive
- Démarrage à partir d'une image ISO servie par HTTP à l'aide de l'API Redfish
- Suppression de nœuds d'une grappe

9.1.2. Ajout de nœuds de travail à l'aide d'Assisted Installer et d'OpenShift Cluster Manager

Vous pouvez ajouter des nœuds de travail à des clusters OpenShift à nœud unique qui ont été créés sur Red Hat OpenShift Cluster Manager à l'aide de l' installateur assisté.

IMPORTANT

L'ajout de nœuds de travail aux clusters OpenShift à nœud unique n'est pris en charge que pour les clusters exécutant OpenShift Container Platform version 4.11 et supérieure.

Conditions préalables

- Avoir accès à un cluster OpenShift à nœud unique installé à l'aide d'Assisted Installer.
- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.
- Assurez-vous que tous les enregistrements DNS requis existent pour le cluster auquel vous ajoutez le nœud de travail.

Procédure

1. Connectez-vous à OpenShift Cluster Manager et cliquez sur le cluster à nœud unique auquel vous souhaitez ajouter un nœud de travail.
2. Cliquez sur **Add hosts**, et téléchargez l'ISO de découverte pour le nouveau nœud de travail, en ajoutant la clé publique SSH et en configurant les paramètres de proxy à l'échelle du cluster, le cas échéant.
3. Démarrez l'hôte cible à l'aide de l'ISO de découverte et attendez que l'hôte soit découvert dans la console. Une fois l'hôte découvert, démarrez l'installation.
4. Au fur et à mesure de l'installation, celle-ci génère des demandes de signature de certificat (CSR) en attente pour le nœud de travail. Lorsque vous y êtes invité, approuvez les CSR en attente pour terminer l'installation.
   Lorsque le nœud de travail est installé avec succès, il est répertorié comme nœud de travail dans la console web du cluster.

IMPORTANT

Les nouveaux nœuds de travail seront cryptés selon la même méthode que le cluster d'origine.

Ressources supplémentaires

- Exigences DNS fournies par l'utilisateur
- Approuver les demandes de signature de certificats pour vos machines

9.1.3. Ajout de nœuds de travail à l'aide de l'API Assisted Installer

Vous pouvez ajouter des nœuds de travail aux clusters OpenShift à nœud unique à l'aide de l'API REST Assisted Installer. Avant d'ajouter des nœuds de travail, vous devez vous connecter à OpenShift Cluster Manager et vous authentifier auprès de l'API.

9.1.3.1. Authentification auprès de l'API REST de l'installateur assisté

Avant de pouvoir utiliser l'API REST d'Assisted Installer, vous devez vous authentifier auprès de l'API à l'aide d'un jeton web JSON (JWT) que vous générez.

Conditions préalables

- Connectez-vous à OpenShift Cluster Manager en tant qu'utilisateur disposant de privilèges de création de cluster.
- Installer **jq**.

Procédure

1. Connectez-vous à OpenShift Cluster Manager et copiez votre jeton API.
2. Définissez la variable **$OFFLINE_TOKEN** à l'aide du jeton API copié en exécutant la commande suivante :

```
$ export OFFLINE_TOKEN=<copied_api_token>
```

3. Définissez la variable **$JWT_TOKEN** en utilisant la variable **$OFFLINE_TOKEN** définie précédemment :

```
$ export JWT_TOKEN=$(
  curl \
  --silent \
  --header "Accept: application/json" \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data-urlencode "grant_type=refresh_token" \
  --data-urlencode "client_id=cloud-services" \
  --data-urlencode "refresh_token=${OFFLINE_TOKEN}" \
  "https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token" \
  | jq --raw-output ".access_token"
)
```

NOTE

Le jeton JWT n'est valable que pendant 15 minutes.

Vérification

- Facultatif : Vérifiez que vous pouvez accéder à l'API en exécutant la commande suivante :

```
$ curl -s https://api.openshift.com/api/assisted-install/v2/component-versions -H
"Authorization: Bearer ${JWT_TOKEN}" | jq
```

Exemple de sortie

```
{
```

```
    "release_tag": "v2.5.1",
    "versions":
    {
        "assisted-installer": "registry.redhat.io/rhai-tech-preview/assisted-installer-rhel8:v1.0.0-175",
        "assisted-installer-controller": "registry.redhat.io/rhai-tech-preview/assisted-installer-reporter-rhel8:v1.0.0-223",
        "assisted-installer-service": "quay.io/app-sre/assisted-service:ac87f93",
        "discovery-agent": "registry.redhat.io/rhai-tech-preview/assisted-installer-agent-rhel8:v1.0.0-156"
    }
}
```

9.1.3.2. Ajout de nœuds de travail à l'aide de l'API REST d'Assisted Installer

Vous pouvez ajouter des nœuds de travail aux clusters à l'aide de l'API REST d'Assisted Installer.

Conditions préalables

- Installez le CLI OpenShift Cluster Manager (**ocm**).
- Connectez-vous à OpenShift Cluster Manager en tant qu'utilisateur disposant de privilèges de création de cluster.
- Installer **jq**.
- Assurez-vous que tous les enregistrements DNS requis existent pour le cluster auquel vous ajoutez le nœud de travail.

Procédure

1. Authentifiez-vous auprès de l'API REST d'Assisted Installer et générez un jeton web JSON (JWT) pour votre session. Le jeton JWT généré n'est valable que pendant 15 minutes.
2. Définissez la variable **$API_URL** en exécutant la commande suivante :

```
export API_URL=<api_url> 1
```

1 Remplacez **<api_url>** par l'URL de l'API d'installation assistée, par exemple, **https://api.openshift.com**

3. Importez le cluster OpenShift à nœud unique en exécutant les commandes suivantes :
   a. Définissez la variable **$OPENSHIFT_CLUSTER_ID**. Connectez-vous au cluster et exécutez la commande suivante :

```
$ export OPENSHIFT_CLUSTER_ID=$(oc get clusterversion -o jsonpath='{.items[].spec.clusterID}')
```

   b. Définissez la variable **$CLUSTER_REQUEST** utilisée pour importer le cluster :

```
$ export CLUSTER_REQUEST=$(jq --null-input --arg openshift_cluster_id "$OPENSHIFT_CLUSTER_ID" '{
  "api_vip_dnsname": "<api_vip>", 1
```

```
  "openshift_cluster_id": $openshift_cluster_id,
  "name": "<openshift_cluster_name>" 2
}')
```

1. Remplacez **<api_vip>** par le nom d'hôte du serveur API du cluster. Il peut s'agir du domaine DNS du serveur API ou de l'adresse IP du nœud unique que le nœud de travail peut atteindre. Par exemple, **api.compute-1.example.com**.

2. Remplacez **<openshift_cluster_name>** par le nom en texte clair de la grappe. Le nom de la grappe doit correspondre à celui qui a été défini lors de l'installation de la grappe le premier jour.

c. Importez le cluster et définissez la variable **$CLUSTER_ID**. Exécutez la commande suivante :

```
$ CLUSTER_ID=$(curl "$API_URL/api/assisted-install/v2/clusters/import" -H
"Authorization: Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-Type:
application/json' \
  -d "$CLUSTER_REQUEST" | tee /dev/stderr | jq -r '.id')
```

4. Générez la ressource **InfraEnv** pour le cluster et définissez la variable **$INFRA_ENV_ID** en exécutant les commandes suivantes :

a. Téléchargez le fichier pull secret depuis Red Hat OpenShift Cluster Manager sur console.redhat.com.

b. Définir la variable **$INFRA_ENV_REQUEST**:

```
export INFRA_ENV_REQUEST=$(jq --null-input \
    --slurpfile pull_secret <path_to_pull_secret_file> \ 1
    --arg ssh_pub_key "$(cat <path_to_ssh_pub_key>)" \ 2
    --arg cluster_id "$CLUSTER_ID" '{
  "name": "<infraenv_name>", 3
  "pull_secret": $pull_secret[0] | tojson,
  "cluster_id": $cluster_id,
  "ssh_authorized_key": $ssh_pub_key,
  "image_type": "<iso_image_type>" 4
}')
```

1. Remplacez **<path_to_pull_secret_file>** par le chemin d'accès au fichier local contenant le secret d'extraction téléchargé depuis Red Hat OpenShift Cluster Manager sur console.redhat.com.

2. Remplacez **<path_to_ssh_pub_key>** par le chemin d'accès à la clé SSH publique requise pour accéder à l'hôte. Si vous ne définissez pas cette valeur, vous ne pourrez pas accéder à l'hôte en mode découverte.

3. Remplacer **<infraenv_name>** par le nom en texte clair de la ressource **InfraEnv**.

4. Remplacez **<iso_image_type>** par le type d'image ISO, soit **full-iso** ou **minimal-iso**.

c. Envoyez l'adresse **$INFRA_ENV_REQUEST** à l'API /v2/infra-envs et définissez la variable **$INFRA_ENV_ID**:

```
$ INFRA_ENV_ID=$(curl "$API_URL/api/assisted-install/v2/infra-envs" -H "Authorization:
Bearer ${JWT_TOKEN}" -H 'accept: application/json' -H 'Content-Type: application/json'
-d "$INFRA_ENV_REQUEST" | tee /dev/stderr | jq -r '.id')
```

5. Obtenez l'URL de l'ISO de découverte pour le nœud de travailleur de cluster en exécutant la commande suivante :

```
$ curl -s "$API_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID" -H "Authorization:
Bearer ${JWT_TOKEN}" | jq -r '.download_url'
```

Exemple de sortie

```
https://api.openshift.com/api/assisted-images/images/41b91e72-c33e-42ee-b80f-
b5c5bbf6431a?
arch=x86_64&image_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NTYwMjYz
NzEsInN1YiI6IjQxYjkxZTcyLWMzM2UtNDJlZS1iODBmLWI1YzViYmY2NDMxYSJ9.1EX_VGaM
NejMhrAvVRBS7PDPIQtbOOc8LtG8OukE1a4&type=minimal-iso&version=4.12
```

6. Télécharger l'ISO :

```
$ curl -L -s '<iso_url>' --output rhcos-live-minimal.iso 1
```

1 Remplacez **<iso_url>** par l'URL de l'ISO de l'étape précédente.

7. Démarrez le nouvel hôte de travail à partir de la version téléchargée de **rhcos-live-minimal.iso**.

8. Obtenez la liste des hôtes du cluster qui sont installés sur *not*. Continuez à exécuter la commande suivante jusqu'à ce que le nouvel hôte apparaisse :

```
$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer
${JWT_TOKEN}" | jq -r '.hosts[] | select(.status != "installed").id'
```

Exemple de sortie

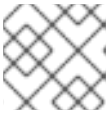
```
2294ba03-c264-4f11-ac08-2f1bb2f8c296
```

9. Définissez la variable **$HOST_ID** pour le nouveau nœud de travail, par exemple :

```
hOST_ID=<host_id> 1
```

1 Remplacez **<host_id>** par l'ID de l'hôte de l'étape précédente.

10. Vérifiez que l'hôte est prêt à être installé en exécutant la commande suivante :



NOTE

Veillez à copier l'intégralité de la commande, y compris l'expression **jq**.

```
$ curl -s $API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID -H "Authorization: Bearer
${JWT_TOKEN}" | jq '
```

```
def host_name($host):
    if (.suggested_hostname // "") == "" then
        if (.inventory // "") == "" then
            "Unknown hostname, please wait"
        else
            .inventory | fromjson | .hostname
        end
    else
        .suggested_hostname
    end;

def is_notable($validation):
    ["failure", "pending", "error"] | any(. == $validation.status);

def notable_validations($validations_info):
    [
        $validations_info // "{}"
        | fromjson
        | to_entries[].value[]
        | select(is_notable(.))
    ];

{
    "Hosts validations": {
        "Hosts": [
            .hosts[]
            | select(.status != "installed")
            | {
                "id": .id,
                "name": host_name(.),
                "status": .status,
                "notable_validations": notable_validations(.validations_info)
            }
        ]
    },
    "Cluster validations info": {
        "notable_validations": notable_validations(.validations_info)
    }
}
' -r
```

Exemple de sortie

```
{
  "Hosts validations": {
    "Hosts": [
      {
        "id": "97ec378c-3568-460c-bc22-df54534ff08f",
        "name": "localhost.localdomain",
        "status": "insufficient",
        "notable_validations": [
          {
            "id": "ntp-synced",
            "status": "failure",
            "message": "Host couldn't synchronize with any NTP server"
          },
```

```
          {
            "id": "api-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          },
          {
            "id": "api-int-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          },
          {
            "id": "apps-domain-name-resolved-correctly",
            "status": "error",
            "message": "Parse error for domain name resolutions result"
          }
        ]
      }
    ]
  },
  "Cluster validations info": {
    "notable_validations": []
  }
}
```


11. Lorsque la commande précédente indique que l'hôte est prêt, démarrez l'installation à l'aide de l'API /v2/infra-envs/{infra_env_id}/hosts/{host_id}/actions/install en exécutant la commande suivante :

```
$ curl -X POST -s "$API_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts/$HOST_ID/actions/install"  -H "Authorization: Bearer ${JWT_TOKEN}"
```

12. Au fur et à mesure de son déroulement, l'installation génère des demandes de signature de certificat (CSR) en attente pour le nœud de travail.

IMPORTANT

Vous devez approuver les CSR pour terminer l'installation.

Continuez à exécuter l'appel API suivant pour surveiller l'installation du cluster :

```
$ curl -s "$API_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "Authorization: Bearer ${JWT_TOKEN}" | jq '{
   "Cluster day-2 hosts":
      [
         .hosts[]
         | select(.status != "installed")
         | {id, requested_hostname, status, status_info, progress, status_updated_at, updated_at, infra_env_id, cluster_id, created_at}
      ]
}'
```

Exemple de sortie

```
{
  "Cluster day-2 hosts": [
    {
      "id": "a1c52dde-3432-4f59-b2ae-0a530c851480",
      "requested_hostname": "control-plane-1",
      "status": "added-to-existing-cluster",
      "status_info": "Host has rebooted and no further updates will be posted. Please check
console for progress and to possibly approve pending CSRs",
      "progress": {
        "current_stage": "Done",
        "installation_percentage": 100,
        "stage_started_at": "2022-07-08T10:56:20.476Z",
        "stage_updated_at": "2022-07-08T10:56:20.476Z"
      },
      "status_updated_at": "2022-07-08T10:56:20.476Z",
      "updated_at": "2022-07-08T10:57:15.306369Z",
      "infra_env_id": "b74ec0c3-d5b5-4717-a866-5b6854791bd3",
      "cluster_id": "8f721322-419d-4eed-aa5b-61b50ea586ae",
      "created_at": "2022-07-06T22:54:57.161614Z"
    }
  ]
}
```


13. En option : Exécutez la commande suivante pour afficher tous les événements du cluster :

```
$ curl -s "$API_URL/api/assisted-install/v2/events?cluster_id=$CLUSTER_ID" -H
"Authorization: Bearer ${JWT_TOKEN}" | jq -c '.[] | {severity, message, event_time, host_id}'
```

Exemple de sortie


```
{"severity":"info","message":"Host compute-0: updated status from insufficient to known (Host
is ready to be installed)","event_time":"2022-07-08T11:21:46.346Z","host_id":"9d7b3b44-
1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from known to installing
(Installation is in progress)","event_time":"2022-07-08T11:28:28.647Z","host_id":"9d7b3b44-
1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing to installing-in-
progress (Starting installation)","event_time":"2022-07-
08T11:28:52.068Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Uploaded logs for host compute-0 cluster 8f721322-419d-4eed-
aa5b-61b50ea586ae","event_time":"2022-07-08T11:29:47.802Z","host_id":"9d7b3b44-1125-
4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host compute-0: updated status from installing-in-progress to
added-to-existing-cluster (Host has rebooted and no further updates will be posted. Please
check console for progress and to possibly approve pending CSRs)","event_time":"2022-07-
08T11:29:48.259Z","host_id":"9d7b3b44-1125-4ad0-9b14-76550087b445"}
{"severity":"info","message":"Host: compute-0, reached installation stage
Rebooting","event_time":"2022-07-08T11:29:48.261Z","host_id":"9d7b3b44-1125-4ad0-9b14-
76550087b445"}
```


14. Connectez-vous au cluster et approuvez les CSR en attente pour terminer l'installation.

Vérification

- Vérifiez que le nouveau nœud de travail a été ajouté avec succès au cluster avec un statut de **Ready**:

```
$ oc get nodes
```

Exemple de sortie

```
NAME                         STATUS   ROLES          AGE   VERSION
control-plane-1.example.com  Ready    master,worker  56m   v1.25.0
compute-1.example.com        Ready    worker         11m   v1.25.0
```

Ressources supplémentaires

- Exigences DNS fournies par l'utilisateur
- Approuver les demandes de signature de certificats pour vos machines

9.1.4. Ajouter manuellement des nœuds de travail à des clusters OpenShift à nœud unique

Vous pouvez ajouter un nœud de travail à un cluster OpenShift à nœud unique manuellement en démarrant le nœud de travail à partir de Red Hat Enterprise Linux CoreOS (RHCOS) ISO et en utilisant le fichier cluster **worker.ign** pour joindre le nouveau nœud de travail au cluster.

Conditions préalables

- Installer un cluster OpenShift à un seul nœud sur du métal nu.
- Installez le CLI OpenShift (**oc**).
- Connectez-vous en tant qu'utilisateur disposant des privilèges **cluster-admin**.
- Assurez-vous que tous les enregistrements DNS requis existent pour le cluster auquel vous ajoutez le nœud de travail.

Procédure

1. Set the OpenShift Container Platform version:

```
oCP_VERSION=<ocp_version> $ OCP_VERSION=<ocp_version> 1
```

1 Replace **<ocp_version>** with the current version, for example, **latest-4.12**

2. Set the host architecture:

```
aRCH=<architecture> $ ARCH=<architecture> 1
```

1 Replace **<architecture>** with the target host architecture, for example, **aarch64** or **x86_64**.

3. Récupérez les données **worker.ign** du cluster à nœud unique en cours d'exécution en exécutant la commande suivante :

```
oc extract -n openshift-machine-api secret/worker-user-data-managed --keys=userData --
to=- > worker.ign
```

4. Hébergez le fichier **worker.ign** sur un serveur web accessible depuis votre réseau.

5. Téléchargez le programme d'installation d'OpenShift Container Platform et mettez-le à disposition en exécutant les commandes suivantes :

```
$ curl -k https://mirror.openshift.com/pub/openshift-
v4/clients/ocp/$OCP_VERSION/openshift-install-linux.tar.gz > openshift-install-linux.tar.gz
```

```
$ tar zxvf openshift-install-linux.tar.gz
```

```
$ chmod +x openshift-install
```

6. Récupérer l'URL de l'ISO RHCOS :

```
$ ISO_URL=$(./openshift-install coreos print-stream-json | grep location | grep $ARCH | grep
iso | cut -d\" -f4)
```

7. Download the RHCOS ISO:

```
$ curl -L $ISO_URL -o rhcos-live.iso
```

8. Utilisez l'ISO RHCOS et le fichier **worker.ign** hébergé pour installer le nœud de travail :

   a. Démarrez l'hôte cible avec l'ISO RHCOS et la méthode d'installation de votre choix.

   b. Lorsque l'hôte cible a démarré à partir de l'ISO RHCOS, ouvrez une console sur l'hôte cible.

   c. Si le DHCP n'est pas activé sur votre réseau local, vous devez créer un fichier d'ignition avec le nouveau nom d'hôte et configurer l'adresse IP statique du nœud de travail avant d'exécuter l'installation de RHCOS. Effectuez les étapes suivantes :

      i. Configurez la connexion réseau de l'hôte travailleur avec une adresse IP statique. Exécutez la commande suivante sur la console de l'hôte cible :

```
$ nmcli con mod <network_interface> ipv4.method manual /
ipv4.addresses <static_ip> ipv4.gateway <network_gateway> ipv4.dns <dns_server>
/
802-3-ethernet.mtu 9000
```

      où :

      <static_ip>
      L'adresse IP statique de l'hôte et le CIDR, par exemple, **10.1.101.50/24**

      <passerelle_de_réseau>
      Il s'agit de la passerelle réseau, par exemple, **10.1.101.1**

      ii. Activez l'interface réseau modifiée :

```
nmcli con up <network_interface>
```

iii. Créez un nouveau fichier d'allumage **new-worker.ign** qui comprend une référence au fichier original **worker.ign** et une instruction supplémentaire que le programme **coreos-installer** utilise pour remplir le fichier **/etc/hostname** sur le nouvel hôte de travail. Par exemple :

```
{
  "ignition":{
    "version":"3.2.0",
    "config":{
      "merge":[
        {
          "source":"<hosted_worker_ign_file>" 1
        }
      ]
    }
  },
  "storage":{
    "files":[
      {
        "path":"/etc/hostname",
        "contents":{
          "source":"data:,<new_fqdn>" 2
        },
        "mode":420,
        "overwrite":true,
        "path":"/etc/hostname"
      }
    ]
  }
}
```

1 **<hosted_worker_ign_file>** est l'URL accessible localement pour le fichier original **worker.ign**. Par exemple, **http://webserver.example.com/worker.ign**

2 **<new_fqdn>** est le nouveau FQDN que vous avez défini pour le nœud de travail. Par exemple, **new-worker.example.com**.

iv. Hébergez le fichier **new-worker.ign** sur un serveur web accessible depuis votre réseau.

v. Exécutez la commande **coreos-installer** suivante, en indiquant les détails de **ignition-url** et du disque dur :

```
$ sudo coreos-installer install --copy-network /
--ignition-url=<new_worker_ign_file> <hard_disk> --insecure-ignition
```

où :

<nouveau_travailleur_fichier_de_signature>

est l'URL accessible localement pour le fichier **new-worker.ign** hébergé, par exemple, **http://webserver.example.com/new-worker.ign**

<disque_dur>

Il s'agit du disque dur sur lequel vous installez RHCOS, par exemple, **/dev/sda**

d. Pour les réseaux dont le protocole DHCP est activé, il n'est pas nécessaire de définir une adresse IP statique. Exécutez la commande suivante **coreos-installer** à partir de la console de l'hôte cible pour installer le système :

```
$ coreos-installer install --ignition-url=<hosted_worker_ign_file> <hard_disk>
```

e. Pour activer manuellement le DHCP, appliquez le CR **NMStateConfig** suivant au cluster OpenShift à nœud unique :

```
apiVersion: agent-install.openshift.io/v1
kind: NMStateConfig
metadata:
  name: nmstateconfig-dhcp
  namespace: example-sno
  labels:
    nmstate_config_cluster_name: <nmstate_config_cluster_label>
spec:
  config:
    interfaces:
      - name: eth0
        type: ethernet
        state: up
        ipv4:
          enabled: true
          dhcp: true
        ipv6:
          enabled: false
  interfaces:
    - name: "eth0"
      macAddress: "AA:BB:CC:DD:EE:11"
```

IMPORTANT

Le CR **NMStateConfig** est requis pour les déploiements réussis des nœuds de travail avec des adresses IP statiques et pour l'ajout d'un nœud de travail avec une adresse IP dynamique si le nœud unique OpenShift a été déployé avec une adresse IP statique. Le réseau cluster DHCP ne définit pas automatiquement ces paramètres réseau pour le nouveau nœud de travailleur.

9. Au fur et à mesure de l'installation, celle-ci génère des demandes de signature de certificat (CSR) en attente pour le nœud de travail. Lorsque vous y êtes invité, approuvez les CSR en attente pour terminer l'installation.

10. Lorsque l'installation est terminée, redémarrez l'hôte. L'hôte rejoint le cluster en tant que nouveau nœud de travail.

Vérification

- Vérifiez que le nouveau nœud de travail a été ajouté avec succès au cluster avec un statut de **Ready**:

```
$ oc get nodes
```

Exemple de sortie

```
NAME                         STATUS   ROLES          AGE   VERSION
control-plane-1.example.com  Ready    master,worker  56m   v1.25.0
compute-1.example.com        Ready    worker         11m   v1.25.0
```

Ressources supplémentaires

- Exigences DNS fournies par l'utilisateur
- Approuver les demandes de signature de certificats pour vos machines

9.1.5. Approving the certificate signing requests for your machines

When you add machines to a cluster, two pending certificate signing requests (CSRs) are generated for each machine that you added. You must confirm that these CSRs are approved or, if necessary, approve them yourself. The client requests must be approved first, followed by the server requests.

Conditions préalables

- You added machines to your cluster.

Procédure

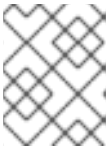
1. Confirm that the cluster recognizes the machines:

```
$ oc get nodes
```

Exemple de sortie

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  63m  v1.25.0
master-1  Ready     master  63m  v1.25.0
master-2  Ready     master  64m  v1.25.0
```

The output lists all of the machines that you created.



NOTE

The preceding output might not include the compute nodes, also known as worker nodes, until some CSRs are approved.

2. Review the pending CSRs and ensure that you see the client requests with the **Pending** or **Approved** status for each machine that you added to the cluster:

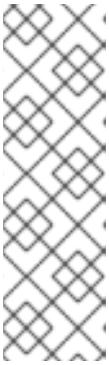
```
$ oc get csr
```

Exemple de sortie

```
NAME        AGE     REQUESTOR                                                   CONDITION
csr-8b2br   15m     system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
csr-8vnps   15m     system:serviceaccount:openshift-machine-config-operator:node-bootstrapper   Pending
...
```

In this example, two machines are joining the cluster. You might see more approved CSRs in the list.

3. If the CSRs were not approved, after all of the pending CSRs for the machines you added are in **Pending** status, approve the CSRs for your cluster machines:



NOTE

Because the CSRs rotate automatically, approve your CSRs within an hour of adding the machines to the cluster. If you do not approve them within an hour, the certificates will rotate, and more than two certificates will be present for each node. You must approve all of these certificates. After the client CSR is approved, the Kubelet creates a secondary CSR for the serving certificate, which requires manual approval. Then, subsequent serving certificate renewal requests are automatically approved by the **machine-approver** if the Kubelet requests a new certificate with identical parameters.

NOTE

For clusters running on platforms that are not machine API enabled, such as bare metal and other user-provisioned infrastructure, you must implement a method of automatically approving the kubelet serving certificate requests (CSRs). If a request is not approved, then the **oc exec**, **oc rsh**, and **oc logs** commands cannot succeed, because a serving certificate is required when the API server connects to the kubelet. Any operation that contacts the Kubelet endpoint requires this certificate approval to be in place. The method must watch for new CSRs, confirm that the CSR was submitted by the **node-bootstrapper** service account in the **system:node** or **system:admin** groups, and confirm the identity of the node.

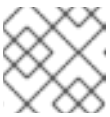
- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** est le nom d'un CSR figurant dans la liste des CSR actuels.

- To approve all pending CSRs, run the following command:

```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}{{end}}{{end}}' | xargs --no-run-if-empty oc adm certificate approve
```



NOTE

Some Operators might not become available until some CSRs are approved.

4. Now that your client requests are approved, you must review the server requests for each machine that you added to the cluster:

```
$ oc get csr
```

Exemple de sortie

```
NAME        AGE    REQUESTOR                                                   CONDITION
csr-bfd72   5m26s  system:node:ip-10-0-50-126.us-east-2.compute.internal
Pending
csr-c57lv   5m26s  system:node:ip-10-0-95-157.us-east-2.compute.internal
Pending
...
```

5. If the remaining CSRs are not approved, and are in the **Pending** status, approve the CSRs for your cluster machines:

- To approve them individually, run the following command for each valid CSR:

```
$ oc adm certificate approve <csr_name> 1
```

1 **<csr_name>** est le nom d'un CSR figurant dans la liste des CSR actuels.

- To approve all pending CSRs, run the following command:

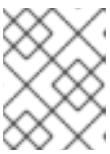
```
$ oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

6. After all client and server CSRs have been approved, the machines have the **Ready** status. Verify this by running the following command:

```
$ oc get nodes
```

Exemple de sortie

```
NAME      STATUS    ROLES   AGE  VERSION
master-0  Ready     master  73m  v1.25.0
master-1  Ready     master  73m  v1.25.0
master-2  Ready     master  74m  v1.25.0
worker-0  Ready     worker  11m  v1.25.0
worker-1  Ready     worker  11m  v1.25.0
```



NOTE

It can take a few minutes after approval of the server CSRs for the machines to transition to the **Ready** status.

Informations complémentaires

- For more information on CSRs, see Certificate Signing Requests.