Images

Images conteneur Kubernetes

Vous créez une image Docker et la poussez dans un registre avant de la référencer depuis un pod Kubernetes.

La propriété `image` d'un conteneur utilise la même syntaxe que la commande `docker`, y compris pour les registres privés et les tags.

Mettre à jour des images

La politique de récupération par défaut est `IfNotPresent`, Kubelet ne récupère alors pas une image si elle est déjà présente sur le nœud. Si vous voulez forcer une récupération à chaque fois, vous pouvez faire une des actions suivantes :

- définissez `imagePullPolicy` du conteneur à `Always`.
- omettez `imagePullPolicy` et utilisez `:latest` comme tag pour l'image à utiliser.
- omettez `imagePullPolicy` et le tag de l'image à utiliser.
- activez l'admission controller AlwaysPullImages.

Notez que vous devez éviter d'utiliser le tag `:latest`, voir Bonnes pratiques pour la configuration pour plus d'informations.

Créer des images multi-architecture à partir de manifestes

La CLI Docker prend maintenant en charge la commande `docker manifest` avec des sous-commandes comme `create`, `annotate` et `push`. Ces commandes peuvent être utilisées pour construire et pousser les manifestes. Vous pouvez utiliser `docker manifest inspect` pour voir le manifeste.

Vous pouvez voir la documentation Docker ici : https://docs.docker.com/edge/engine/reference/commandline/manifest/

Voici comment nous l'utilisons dans notre outil de build: https://cs.k8s.io/?q=docker%20manifest%20(create%7Cpush%7Cannotate)&i=nope&files=&repos=

Ces commandes se basent et sont implémentées purement sur la CLI Docker. Vous devrez soit éditer `$HOME/.docker/config.json` et définir la clé `experimental` à `enabled` ou vous pouvez simplement définir la variable d'environnement `DOCKER_CLI_EXPERIMENTAL` à `enabled` lorsque vous appelez les commandes de la CLI.

> **Note:** Veuillez utiliser les versions *18.06 ou ultérieure*, les versions antérieures ayant des bugs ou ne prenant pas en charge l'option `experimental` pour la ligne de commande. Par exemple https://github.com/docker/cli/issues/1135 cause des problèmes sous `containerd`.

Si vous avez des problèmes en téléchargeant des manifestes viciés, nettoyez les anciens manifestes dans `$HOME/.docker/manifests` pour recommencer de zéro.

Pour Kubernetes, nous avons historiquement utilisé des images avec des suffixes `-$(ARCH)`. Pour une rétrocompatibilité, veuillez générer les anciennes images avec des suffixes. Par exemple, l'image `pause` qui a le manifeste pour toutes les architetures et l'image `pause-amd64` qui est rétrocompatible pour d'anciennes configurations ou des fichiers YAML qui auraient codé en dur les images avec des suffixes.

Utiliser un registre privé

Les registres privés peuvent demander des clés pour pouvoir lire leurs images.

Ces certificats peuvent être fournis de différentes manières :

- En utilisant la Google Container Registry
  - par cluster
  - automatiqueent configuré dans Google Compute Engine ou Google Kubernetes Engine
  - tous les pods peuvent lire le registre privé du projet
- En utilisant Amazon Elastic Container Registry (ECR)
  - utilise les rôles et politiques IAM pour contrôler l'accès aux dépôts ECR
  - rafraîchit automatiquement les certificats de login ECR
- En utilisant Oracle Cloud Infrastructure Registry (OCIR)
  - utilise les rôles et politiques IAM pour contrôler l'accès aux dépôts OCIR
- En utilisant Azure Container Registry (ACR)
- En utilisant IBM Cloud Container Registry
  - utilise les rôles et politiques IAM pour contrôler l'accès à l'IBM Cloud Container Registry
- En configurant les nœuds pour s'authentifier auprès d'un registre privé
  - tous les pods peuvent lire les registres privés configurés
  - nécessite la configuration des nœuds par un administrateur du cluster
- En utilisant des images pré-chargées
  - tous les pods peuvent utiliser toutes les images mises en cache sur un nœud
  - nécessite l'accès root à tous les nœuds pour la mise en place
- En spécifiant ImagePullSecrets dans un Pod
  - seuls les pods fournissant ses propres clés peuvent accéder au registre privé

Chaque option est décrite plus en détails ci-dessous.

Utiliser la Google Container Registry

Kubernetes prend en charge nativement la Google Container Registry (GCR), lorsqu'il s'exécute dans Google Compute Engine (GCE). Si vous exécutez votre cluster dans GCE ou Google Kubernetes Engine, utilisez simplement le nom complet de l'image (par ex. gcr.io/my_project/image:tag).

Tous les pods dans un cluster auront un accès en lecture aux images dans le registre.

Kubelet va s'authentifier auprès de GCR en utilisant le compte de service Google de l'instance. Le compte de service dans l'instance aura un `https://www.googleapis.com/auth/devstorage.read_only` , afin qu'il puisse récupérer depuis le GCR du projet mais qu'il ne puisse pas pousser une image.

Utiliser Amazon Elastic Container Registry

Kubernetes prend en charge nativement Amazon Elastic Container Registry, lorsque les nœuds sont des instances de AWS EC2.

Utilisez simplement le nom complet de l'image (par ex. `ACCOUNT.dkr.ecr.REGION.amazonaws.com/imagename:tag`) dans la définition du Pod.

Tous les utilisateurs du cluster qui peuvent créer des pods auront la possibilité d'exécuter des pods qui utilisent n'importe quelle image du registre ECR.

Kubelet va aller chercher et rafraîchir périodiquement les certificats ECR. Les permissions suivantes sont requises par kubelet :

- `ecr:GetAuthorizationToken`
- `ecr:BatchCheckLayerAvailability`
- `ecr:GetDownloadUrlForLayer`
- `ecr:GetRepositoryPolicy`
- `ecr:DescribeRepositories`
- `ecr:ListImages`
- `ecr:BatchGetImage`

Exigences :

- Vous devez utiliser kubelet version `v1.2.0` ou ultérieure. (exécutez par ex. `/usr/bin/kubelet --version=true`).
- Si vos nœuds sont dans une région différente de votre registre, vous devez utiliser la version `v1.3.0` ou ultérieure.
- ECR doit être disponible dans votre région.

Dépannage :

- Vérifiez toutes les exigences ci-dessus.
- Copiez les certificats de $REGION (par ex. `us-west-2`) sur votre poste de travail. Connectez-vous en SSH sur l'hôte et exécutez Docker manuellement avec ces certificats. Est-ce que ça marche ?
- Vérifiez que kubelet s'exécute avec `--cloud-provider=aws` .
- Augmentez la verbosité des logs de kubelet à au moins 3 et recherchez dans les logs de kubelet (par exemple avec `journalctl -u kubelet`) des lignes similaires à :
- ○ `aws_credentials.go:109] unable to get ECR credentials from cache, checking ECR API`
- ○ `aws_credentials.go:116] Got ECR credentials from ECR API for <AWS account ID for ECR>.dkr.ecr.<AWS region>.amazonaws.com`

Utiliser Azure Container Registry (ACR)

En utilisant Azure Container Registry vous pouvez vous authentifier en utilisant soit un utilisateur admin soit un service principal. Dans les deux cas, l'authentification est faite via l'authentification standard de Docker. Ces instructions assument l'outil en ligne de commande azure-cli.

Vous devez d'abord créer un registre et générer des certificats, la documentation complète pour cela peut être touvée dans la documentation de Azure container registry.

Une fois votre registre de conteneurs créé, vous utiliserez les certificats suivants pour vous connecter :

- `DOCKER_USER` : service principal ou utilisateur admin
- `DOCKER_PASSWORD` : mot de passe du service principal ou utilisateur admin
- `DOCKER_REGISTRY_SERVER` : `${un-nom-de-registre}.azurecr.io`
- `DOCKER_EMAIL` : `${une-adresse-email}`

Une fois que vous avez défini ces variables, vous pouvez configurer un Secret Kubernetes et l'utiliser pour déployer un Pod.

Utiliser IBM Cloud Container Registry

IBM Cloud Container Registry fournit un registre d'images multi-tenant privé que vous pouvez utiliser pour stocker et partager de manière sécurisée vos images. Par défaut, les images de votre registre privé sont scannées par le Vulnerability Advisor intégré pour détecter des failles de sécurité et des vulnérabilités potentielles. Les utilisateurs de votre compte IBM Cloud peuvent accéder à vos images, ou vous pouvez des rôles et politiques IAM pour fournir l'accès aux namespaces de l'IBM Cloud Container Registry.

Pour installer le plugin du CLI de IBM Cloud Container Registry et créer un namespace pour vos images, voir Débuter avec IBM Cloud Container Registry.

Si vous utilisez le même compte et la même région, vous pouvez déployer des images stockées dans IBM Cloud Container Registry vers la namespace `default` de votre cluster IBM Cloud Kubernetes Service sans configuration supplémentaire, voir Construire des conteneurs à partir d'images. Pour les autres options de configuration, voir Comprendre comment autoriser votre cluster à télécharger des images depuis un registre.

Configurer les nœuds pour s'authentifier auprès d'un registre privé

> **Note:** Si vous travaillez dans Google Kubernetes Engine, vous trouverez un `.dockercfg` sur chaque nœud avec les certificats pour Google Container Registry. Vous ne pourrez pas utiliser cette méthode.

> **Note:** Si vous travaillez dans AWS EC2 et utilisez EC2 Container Registry (ECR), kubelet sur chaque nœud va gérer et mettre à jour les certificats du login ECR. Vous ne pourrez pas utiliser cette méthode.

> **Note:** Cette méthode est utilisable si vous avez le contrôle sur la configuration des nœuds. Elle ne marchera pas correctement sur GCE, et sur tout autre fournisseur cloud qui fait du remplacement de nœud automatique.

> **Note:** Kubernetes prend pour l'instant en charge uniquement les sections `auths` et `HttpHeaders` de la config docker. Cela veut dire que les aides aux certificats (`credHelpers` ou `credsStore`) ne sont pas pris en charge.

Docker stocke les clés pour les regisres privés dans le fichier `$HOME/.dockercfg` ou `$HOME/.docker/config.json` . Si vous placez le même fichier dans un des chemins de recherche ci-dessous, kubelet l'utilise comme fournisseur de clés lorsque les images sont récupérées.

- `{--root-dir:-/var/lib/kubelet}/config.json`
- `{cwd of kubelet}/config.json`
- `${HOME}/.docker/config.json`
- `/.docker/config.json`
- `{--root-dir:-/var/lib/kubelet}/.dockercfg`
- `{cwd of kubelet}/.dockercfg`
- `${HOME}/.dockercfg`
- `/.dockercfg`

> **Note:** Vous pouvez avoir à définir `HOME=/root` explicitement dans votre fichier d'environnement pour kubelet.

Voici les étapes recommandées pour configurer vos nœuds pour qu'ils utilisent un registre privé. Dans cet exemple, exécutez-les sur votre poste de travail :

1. Exécutez `docker login [server]` pour chaque jeu de certificats que vous désirez utiliser. Ceci met à jour `$HOME/.docker/config.json` .
2. Examinez `$HOME/.docker/config.json` dans un éditeur pour vous assurer qu'il contient uniquement les certificats que vous désirez utiliser.
3. Récupérez la liste de vos nœuds, par exemple :
   - si vous voulez connaître les noms : `nodes=$(kubectl get nodes -o jsonpath='{range.items[*].metadata}{.name} {end}')`
   - si vous voulez connaître les IPs : `nodes=$(kubectl get nodes -o jsonpath='{range .items[*].status.addresses[?(@.type=="ExternalIP")]}{.address} {end}')`
4. Copiez votre fichier `.docker/config.json` local dans un des chemins de recherche ci-dessus.
   - par exemple : `for n in $nodes; do scp ~/.docker/config.json root@$n:/var/lib/kubelet/config.json; done`

Vérifiez en créant un pod utilisant une image privée, par ex. :

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: test-image-privee-1
spec:
  containers:
    - name: utilise-image-privee
      image: $NOM_IMAGE_PRIVEE
      imagePullPolicy: Always
      command: [ "echo", "SUCCESS" ]
EOF
```

```
pod/test-image-privee-1 created
```

Si tout fonctionne, alors, après quelques instants, vous pouvez exécuter :

```
kubectl logs test-image-privee-1
```

et voir que la commande affiche :

```
SUCCESS
```

Si vous suspectez que la commande a échouée, vous pouvez exécuter :

```
kubectl describe pods/test-image-privee-1 | grep 'Failed'
```

En cas d'échec, l'affichage sera similaire à :

```
  Fri, 26 Jun 2015 15:36:13 -0700    Fri, 26 Jun 2015 15:39:13 -0700
   19    {kubelet node-i2hq}    spec.containers{uses-private-image}
  failed        Failed to pull image "user/privaterepo:v1": Error: im
age user/privaterepo:v1 not found
```

Vous devez vous assurer que tous les nœuds du cluster ont le même fichier `.docker/config.json` . Dans le cas contraire, les pods vont s'exécuter sur certains nœuds et échouer sur d'autres. Par exemple, si vous utilisez l'autoscaling des nœuds, alors chaque modèle d'instance doit inclure le fichier `.docker/config.json` ou monter un disque le contenant.

Tous les pods auront un accès en lecture aux images d'un registre privé dès que les clés du registre privé sont ajoutées au fichier `.docker/config.json` .

Images pré-chargées

> **Note:** Si vous travaillez dans Google Kubernetes Engine, vous trouverez un `.dockercfg` sur chaque nœud avec les certificats pour Google Container Registry. Vous ne pourrez pas utiliser cette méthode.

> **Note:** Cette méthode est utilisable si vous avez le contrôle sur la configuration des nœuds. Elle ne marchera pas correctement sur GCE, et sur tout autre fournisseur cloud qui fait du remplacement de nœud automatique.

Par défaut, kubelet essaiera de récupérer chaque image depuis le registre spécifié. Cependant, si la propriété `imagePullPolicy` du conteneur est `IfNotPresent` ou `Never` , alors une image locale est utilisée (respectivement de préférence ou exclusivement).

Si vous désirez vous reposer sur des images pré-chargées pour éviter l'authentification à un registre, vous devez vous assurer que tous les nœuds du cluster ont les mêmes images pré-chargées.

Ceci peut être utilisé pour pré-charger certaines images pour gagner du temps, ou comme une alternative à l'authentification à un registre privé.

Tous les pods auront un accès en lecture aux images pré-chargées.

Spécifier ImagePullSecrets dans un Pod

> **Note:** Cette méthode est actuellement la méthode recommandée pour Google Kubernetes Engine, GCE, et tout autre fournisseur de cloud où la création de nœuds est automatisée.

Kubernetes permet de spécifier des clés de registre dans un pod.

Créer un Secret avec une config Docker

Exécutez la commande suivante, en substituant les valeurs en majuscule :

```
kubectl create secret docker-registry <name> --docker-server=SERVEUR_REGISTRE_DOCKER --
docker-username=UTILISATEUR_DOCKER --docker-password=MOT_DE_PASSE_DOCKER --docker-email=
EMAIL_DOCKER
secret/myregistrykey created.
```

Si vous avez déjà un fichier de clés Docker, alors, plutôt que d'utiliser la commande ci-dessus, vous pouvez importer le fichier de clés comme un Secret Kubernetes. [Créer un Secret basé sur des clés Docker existantes]() explique comment s'y prendre. Ceci est particulièrement utile si vous utilisez plusieurs registres privés, `kubectl create secret docker-registry` créant un Secret ne fonctionnant qu'avec un seul registre privé.

> **Note:** Les pods peuvent référencer des pull secrets dans leur propre namespace uniquement, ces étapes doivent donc être faites pour chaque namespace.

Se référer à un imagePullSecrets dans un Pod

Vous pouvez maintenant créer des pods qui référencent ce secret en ajoutant une section `imagePullSecrets` dans la définition du pod.

```
cat <<EOF > pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: foo
  namespace: awesomeapps
spec:
  containers:
    - name: foo
      image: janedoe/awesomeapp:v1
  imagePullSecrets:
    - name: myregistrykey
EOF

cat <<EOF >> ./kustomization.yaml
resources:
- pod.yaml
EOF
```

Ceci doit être fait pour chaque pod utilisant un registre privé.

Cependant, la définition de ce champ peut être automatisé en définissant `imagePullSecrets` dans une ressource serviceAccount. Voyez Ajouter un ImagePullSecrets à un Service Account pour des instructions détaillées.

Vous pouvez utiliser cette méthode en conjonction avec un `.docker/config.json` par nœud. Les certificats seront alors regroupés. Cette approche fonctionnera dans Google Kubernetes Engine.

Cas d'utilisation

Il y a plusieurs solutions pour configurer des registres privés. Voici quelques cas d'utilisation classiques et des propositions de solutions.

1. Cluster exécutant uniquement des images non propriétaires (par ex. open-source). Inutile de protéger les images.
   - Utilisez des images publiques dans le Hub Docker.
     - Pas de configuration requise.
     - Dans GCE/Google Kubernetes Engine, un miroir local est automatiquement utilisé pour améliorer la vitesse et la disponibilité.
2. Cluster exécutant quelques images propriétaires qui doivent être protégées de l'extérieur de l'entreprise, mais visibles pour tous les utilisteurs du cluster.
   - Utilisez un registre Docker hébergé privé.
     - Il peut être hébergé sur le Hub Docker, ou ailleurs.
     - Configurez manuellement .docker/config.json sur caque nœud comme décrit ci-dessus.
   - Ou, utilisez un registre privé interne derrière votre pare-feu avec un accès ouvert en lecture.
     - Aucune configuration Kubernetes n'est nécessaire.
   - Ou, dans GCE/Google Kubernetes Engine, utilisez le Google Container Registry du projet.
     - Cela fonctionnera mieux pour l'autoscaling du cluster que la configuration manuelle des nœuds.
   - Ou, dans un cluster où le changement de la configuration des nœuds est difficile, utilisez `imagePullSecrets` .
3. Cluster avec des images propriétaires, dont quelques-unes nécessitent un contrôle d'accès plus strict.
   - Assurez-vous que l'admission controller AlwaysPullImages est actif. Autrement, tous les Pods ont potentiellement accès à toutes les images.
   - Déplacez les données sensibles dans une ressource "Secret", plutôt que de les intégrer dans une image.
4. Un cluster multi-tenant où chaque *tenant* doit avoir son propre registre privé.

- Assurez-vous que l'admission controller AlwaysPullImages est actif. Autrement, tous les Pods ont potentiellement accès à toutes les images.
- Utilisez un registre privé nécessitant l'autorisation.
- Générez des certificats de registre pour chaque *tenant*, placez-les dans des secrets, et placez ces secrets dans les namespaces de chaque *tenant*. pod - Le *tenant* ajoute ce secret dans les imagePullSecrets de chaque pod.

Si vous devez accéder à plusieurs registres, vous pouvez créer un secret pour chaque registre. Kubelet va fusionner tous les `imagePullSecrets` dans un unique `.docker/config.json` virtuel.

Feedback

Cette page est elle utile ?

Oui Non

Dernière modification May 30, 2020 at 3:36 PM PST: add fr pages (7daf3c55e9)