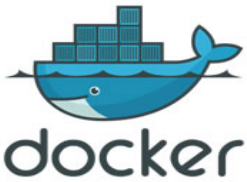




Mise en place d'un registre privé sur Kubernetes





Mise en place d'un registre privé sur kubernetes

Docker Registry est une application qui vous aide à stocker et à distribuer des images de conteneur. Le registre de conteneurs le plus populaire est **DockerHub**, qui est le registre public standard pour Docker et Kubernetes. Mais vous pourriez faire face à une situation où internet est Inaccessible temporairement. Dans ce cas, la configuration d'un registre Docker privé vous fournit plusieurs options de stockage et d'authentification qui peuvent être personnalisées selon vos besoins. Dans ce didacticiel, nous allons examiner le déploiement d'un registre Docker privé compatible TLS en tant que Pod dans un environnement Kubernetes. Cela nous aidera à pousser nos images personnalisées vers le registre, qui peut ensuite être extrait par l'un des nœuds de travail et exécuté en tant que conteneurs dans les Pods. Mon cluster K3s dérivé de k8s ici se compose de 3 machines virtuelles Photon avec 1 nœud maître et 2 nœuds de travail.

Vérifions les versions :

```
k3s -v
```

k3s version **v1.24.3+k3s1** (990ba0e8)

go version go1.18.1

```
uname -a
```

Linux k8s-server **4.19.132-5**.ph3-esx #1-photon SMP Wed Aug 12 21:02:13 UTC 2020 x86_64 GNU/Linux

```
cat /etc/photon-release
```

VMware Photon **OS 3.0**

PHOTON_BUILD_NUMBER=11dd065

```
docker -v
```

Docker **version 19.03.10**, build 9424aea

Créer le répertoire registry et le sous répertoire certs

```
cd /root
mkdir -p registry && cd "$_"
mkdir certs
```

```
cd ..
```

Création de fichiers pour l'authentification

Commençons par créer des certificats auto-signés et l'authentification des utilisateurs pour renforcer la sécurité de notre registre Docker privé. Les certificats TLS sont créés à l'aide **d'openssl** où nous devons spécifier le nom avec lequel nous voulons accéder à notre registre dans le champ Nom commun **« /CN= »**. Ici, je souhaite accéder à mon registre en utilisant le nom **docker-registry**.

Créez avec open ssl le certificat auto-signés :

```
openssl req -x509 -newkey rsa:4096 -days 22365 -nodes -sha256 -keyout registry/certs/tls.key -out registry/certs/tls.crt -subj "/CN=docker-registry" -extensions san -config <(echo '[req]'; echo 'distinguished_name=req';
        echo '[san]'; echo 'subjectAltName=DNS:docker-registry')
```

La commande req réquisitionne la création d'un certificat **au format PKCS#10.**

Le certificat auto-signé avec sha256 tls.crt et un fichier de clé avec RSA 2048 bits tls.key.

La validité du certificat est 22365 jours

Un SAN ou Subject Alternative Name est une extension de la norme **X509**, qui permet d'ajouter des informations complémentaires à un certificat. Un SAN peut contenir plusieurs informations comme des adresses mails, des adresses IP ou des noms de domaine.

Pour ajouter l'extension SANv3 subjectAltName IP et DNS on a:

-subj "/CN=docker-registry" -extensions san -config <(echo '[req]'; echo 'distinguished_name=req';

        echo '[san]'; echo 'subjectAltName=DNS:docker-registry')

Créons de nouveau répertoire :

```
mkdir -p ./registry/certs ./registry/auth
```

Vérifions la communication avec le modem routeur pour l'accès à Internet

Ping @ipdumodemrouteur

et lançons le script suivant ;

bash ./gensecretauthmanifest.sh

```
!/bin/bash
echo "Entrez le login d'authentification d'accès au registry privé Docker"
read LOGIN
echo "Entrez le mot de passe d'authentification d'accès au registry privé Docker"
read PASSWORD
mkdir -p ./registry/certs ./registry/auth
echo "Generating tls..."
SECRETMANIFEST="secrets-auth.yaml"
echo "Generating tls Manifest"
cat << EOF > "$SECRETMANIFEST"
apiVersion: v1
kind: Secret
metadata:
  name: certs-secret-docker-registry
type: kubernetes.io/tls
data:
  tls.crt: $(cat registry/certs/tls.crt | base64 --wrap=0)
  tls.key: $(cat registry/certs/tls.key | base64 --wrap=0)
EOF

echo "Generation auth file"
AUTHFILE="registry/auth/htpasswd"
docker run --rm --entrypoint htpasswd registry:2.6.2 -Bbn "$LOGIN" "$PASSWORD" > "$AUTHFILE"

# Check
echo "CHECK..."
if [ ! -f "$AUTHFILE" ];then
        echo "Le login/password n'a pas été créé par le container htpasswd"
        exit 1
fi
echo " [*] $AUTHFILE content : "
cat "$AUTHFILE"
echo "Displaying architecture created"
ls registry/*

echo "Generating auth Manifest"
echo "---" >> "$SECRETMANIFEST"
cat << EOF >> "$SECRETMANIFEST"
apiVersion: v1
```

```
kind: Secret
metadata:
  name: auth-secret-docker-registry
type: Opaque
data:
  username: $(echo "$LOGIN" | base64 --wrap=0)
  password: $(echo "$PASSWORD" | base64 --wrap=0)
EOF

echo "---" >> "$SECRETMANIFEST"
cat << EOF >> "$SECRETMANIFEST"
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret-docker-registry
type: Opaque
data:
  htpasswd: $(cat registry/auth/htpasswd | base64 --wrap=0)
EOF

echo "Execute now : "
echo ""
echo "    kubectl: apply -f $SECRETMANIFEST"
echo ""
```

Le script permet de saisir un login et un mot de passe au format base 64 afin de générer un ficher d'authentification htpasswd dans le répertoire registry/auth.

Il crée un répertoire registry, contenant le certificat qui devra être disposés sur tous les workers une fois la tache terminée le conteneur est effacée

```
docker run --rm --entrypoint htpasswd registry:2.6.2 -Bbn "$LOGIN" "$PASSWORD"
> "$AUTHFILE"
```

Le script précédent permet de générer un fichier de configuration yaml secrets-auth.yaml qui contient tous les secrets pour kubernetes

```
apiVersion: v1
kind: Secret
metadata:
  name: certs-secret-docker-registry
type: kubernetes.io/tls
data:
  tls.crt: ------------------------------
```

```
  tls.key: ------------------------
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret-docker-registry
type: Opaque
data:
  username: cm9vdAo=
  password: dHByZXNlYXUxMQo=
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret-docker-registry
type: Opaque
data:
  htpasswd: cm----------------------------------------------------------------------
```

Vérifions l'arborescence du répertoire registry :

```
ls -R registry
```

registry:

auth certs

registry/auth:

htpasswd

registry/certs:

tls.crt tls.key

Par la suite on Ajoute les secrets dans le cluster K3s

```
kubectl apply -f secrets-auth.yaml
```

Création d'un volume persistant et d'une revendication pour le stockage du référentiel

Les images qui sont poussées vers notre registre doivent être placées dans un emplacement de stockage cohérent. Par conséquent, nous utiliserons un **volume persistant** de 3 Go hébergé à un emplacement temporaire dans le nœud où notre Pod de registre s'exécutera. L'espace utilise une **revendication de volume persistant** qui sera liée au volume nouvellement créé sous la forme d'un mappage un-à-un.

Créez un fichier volumes.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: docker-repo-pv
spec:
  capacity:
    storage: 3Gi
  accessModes:
  - ReadWriteOnce
  hostPath:
    path: /tmp/repository
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: docker-repo-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Création de l'espace de registre

Par la suite on crée un Pod qui va nous permettre grâce à son service de permettre l'accès à d'autres pods à notre registre privé

Dans le fichier yaml **volume.yaml précédent**, l'image que nous utilisons pour notre registre est appelée **registre** qui est téléchargé à partir de DockerHub. Les images poussées vers ce registre seront enregistrées dans le répertoire **/var/lib/registry** en interne, c'est pourquoi nous montons notre volume persistant à l'aide du **docker-repo-pvc** Claim pour conserver les images de manière permanente. Les variables d'environnement requises par le conteneur de registre sont prises en charge par les secrets que nous montons en tant que volumes. Le service est nommé **docker-registry, avec lequel** nous voulons accéder à notre registre privé docker. Notez qu'il s'agit du nom exact qui a été donné dans le champ Nom commun « /CN= » lors de la génération des certificats TLS. Le conteneur de registre par défaut est exposé au port **5000** et nous lions notre service à ce port en conséquence.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: docker-registry-pod
  labels:
    app: registry
spec:
  containers:
  - name: registry
    image: registry:2.6.2
    volumeMounts:
    - name: repo-vol
      mountPath: "/var/lib/registry"
    - name: certs-vol
      mountPath: "/certs"
      readOnly: true
    - name: auth-vol
      mountPath: "/auth"
      readOnly: true
    env:
    - name: REGISTRY_AUTH
      value: "htpasswd"
```

```
    - name: REGISTRY_AUTH_HTPASSWD_REALM
      value: "Registry Realm"
    - name: REGISTRY_AUTH_HTPASSWD_PATH
      value: "/auth/htpasswd"
    - name: REGISTRY_HTTP_TLS_CERTIFICATE
      value: "/certs/tls.crt"
    - name: REGISTRY_HTTP_TLS_KEY
      value: "/certs/tls.key"
    - name: REGISTRY_STORAGE_DELETE_ENABLED
      value: "true"
  volumes:
  - name: repo-vol
    persistentVolumeClaim:
      claimName: docker-repo-pvc
  - name: certs-vol
    secret:
      secretName: certs-secret-docker-registry
  - name: auth-vol
    secret:
      secretName: auth-secret-docker-registry
---
apiVersion: v1
kind: Service
metadata:
  name: docker-registry
spec:
  selector:
    app: registry
  ports:
  - port: 5000
    targetPort: 5000
```

Créez les volumes, l'espace Registre et le service à l'aide de la commande suivante :

```
kubectl apply -f volumes.yaml
```

on peut aussi créer un déploiement avec une réplication pour deux pods

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: docker-repo-pv
spec:
  capacity:
```

```
    storage: 3Gi
  accessModes:
  - ReadWriteOnce
  hostPath:
    path: /tmp/repository
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: docker-repo-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: "docker-registry-pod"
spec:
  selector:
    matchLabels:
      app: registry
  replicas: 2
  template:
    metadata:
      labels:
        app: registry
    spec:
      containers:
      - name: registry
        image: registry:2.6.2
        volumeMounts:
        - name: repo-vol
          mountPath: "/var/lib/registry"
        - name: certs-vol
          mountPath: "/certs"
          readOnly: true
        - name: auth-vol
          mountPath: "/auth"
          readOnly: true
```

```
        env:
        - name: REGISTRY_AUTH
          value: "htpasswd"
        - name: REGISTRY_AUTH_HTPASSWD_REALM
          value: "Registry Realm"
        - name: REGISTRY_AUTH_HTPASSWD_PATH
          value: "/auth/htpasswd"
        - name: REGISTRY_HTTP_TLS_CERTIFICATE
          value: "/certs/tls.crt"
        - name: REGISTRY_HTTP_TLS_KEY
          value: "/certs/tls.key"
        - name: REGISTRY_STORAGE_DELETE_ENABLED
          value: "true"
      volumes:
      - name: repo-vol
        persistentVolumeClaim:
          claimName: "docker-repo-pvc"
      - name: certs-vol
        secret:
          secretName: certs-secret-docker-registry
      - name: auth-vol
        secret:
          secretName: auth-secret-docker-registry
---
apiVersion: v1
kind: Service
metadata:
  name: docker-registry
spec:
  selector:
    app: registry
  ports:
  - port: 5000
    targetPort: 5000
```

Attention relancez les nœuds si difficulté de création des pods

```
kubectl apply -f volumes_deploy.yaml
```

Vérifiez si les pods sont bien créés

```
kubectl get pods -o wide
```

Notamment

docker-registry-pod 1/1 Running 0 30m

Pour un déploiement on observe les deux pods :

```
docker-registry-pod-74894df87-gdbbp 1/1 Running 0 5h13m 10.42.1.47 k8s-node1 <none> <none>
docker-registry-pod-74894df87-jtt8r 1/1 Running 0 5h13m 10.42.1.46 k8s-node1 <none> <none>
```

K8s-node1 a créé le répertoire /tmp/repository en effet si le pod se lance d'un nœud celui-ci mettra en place les volumes persistants PVC et PV sur ce nœud. Vérifions

```
kubectl get pvc
```

kubectl describe pv pvc-c02cb6d3-4ece-4d06-b7d6-83ef6fcc9247

```
Name:            pvc-c02cb6d3-4ece-4d06-b7d6-83ef6fcc9247
Labels:          <none>
Annotations:     pv.kubernetes.io/provisioned-by: rancher.io/local-path
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    local-path
Status:          Bound
Claim:           default/docker-repo-pvc
Reclaim Policy:  Delete
Access Modes:    RWO
VolumeMode:      Filesystem
Capacity:        3Gi
Node Affinity:
  Required Terms:
    Term 0:      kubernetes.io/hostname in [k8s-node1]
```

Le volume permanent est bien sur **[k8s-node1]** cela posera un problème si vous perdez le noeud1 pas de possibilité d'atteindre le registre privé et ses images lors de la création d'un pod dont l'image se situe dans le registre privée

Ce pod permettra l'accès aux registres aux autres pods

Pour voir si les pods sont bien générés et en running

```
kubectl describe pod
```

Créons deux variables pour la suite :

```
export REGISTRY_NAME="docker-registry"
```

Vérifions l'adresse IP du service docker-registry

```
kubectl get svc
```

docker-registry ClusterIP **10.43.38.212** <none> 5000/TCP 25h

```
export REGISTRY_IP=" 10.43.38.212"
```

Maintenant, nous allons ajouter l'entrée « **10.43.38.212 docker-registry** » au fichier **/etc/hosts** de tous les nœuds de notre cluster afin que cette adresse IP soit résolue au nom **docker-registry**. L'étape ci-dessus peut également être effectuée à l'aide d'une seule commande du nœud maître (entrez le mot de passe, si vous y êtes invité):

```
for x in $(kubectl get nodes -o jsonpath='{
$.items[*].status.addresses[?(@.type=="InternalIP")].address }'); do ssh root@$x
"echo '$REGISTRY_IP $REGISTRY_NAME' >> /etc/hosts"; done
```

Ensuite, nous devons copier le **fichier certificat tls.crt** que nous avons créé précédemment en tant que **« ca.crt »** dans un répertoire **personnalisé /etc/docker/certs.d/docker-registry:5000** dans tous les nœuds de notre cluster pour nous assurer que notre certificat auto-signé est approuvé par Docker. Notez que le répertoire créé dans **/etc/docker/certs.d** doit porter le nom du format**<registry_name>:<registry_port>**.

Cette étape peut être effectuée manuellement ou à l'aide d'une seule commande du nœud maître comme suit :

```
for x in $(kubectl get nodes -o jsonpath='{
$.items[*].status.addresses[?(@.type=="InternalIP")].address }'); do ssh root@$x
"rm -rf /etc/docker/certs.d/$REGISTRY_NAME:5000;mkdir -p
/etc/docker/certs.d/$REGISTRY_NAME:5000"; done
```

puis

```
for x in $(kubectl get nodes -o jsonpath='{
$.items[*].status.addresses[?(@.type=="InternalIP")].address }'); do scp
registry/certs/tls.crt
root@$x:/etc/docker/certs.d/$REGISTRY_NAME:5000/ca.crt; done
```

L'étape ci-dessus oblige Docker à vérifier notre certificat auto-signé même s'il n'est pas signé par une autorité connue.

essayons de nous connecter au registre à partir du nœud maître, en utilisant les mêmes informations d'identification que nous avons créées précédemment:

```
docker login docker-registry:5000 -u root -p tpreseau11
```

```
mypasswd
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in
/root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
Login Succeeded
```

Cette action va créer la propriété "auth" dans le fichier ~/.docker/config.json et donc permettre à docker de se connecter automatiquement à la registry privée.

Utilisons ce secret pour créer le secret "regcred" dans kubernetes :

```
kubectl create secret generic regcred --from-
file=.dockerconfigjson=/root/.docker/config.json --
type=kubernetes.io/dockerconfigjson
```

Créer votre manifest secrets.yaml , et ajouter ce composant

```
---
apiVersion: v1
kind: Secret
metadata:
  name: auth-secret-docker-registry
  namespace: default
type: kubernetes.io/dockerconfigjson
data:
  .dockerconfigjson: ewoJImF1dGhzIjo
```

où `ewoJImF1dGhzIjo` sera la valeur contenu dans le secret regcred et obtenu avec la commande :

```
kubectl get secret regcred -o jsonpath='{.data}' | sed -E 's/\{|\}|"/ /g' | xargs
```

```
kubectl apply -f secrets.yaml
```

Au niveau du "Deployment", on ajoutera la propriété "imagePullSecrets" : avec *regcred comme name (voir par la suite)*

On peut aussi copier le certificat dans le répertoire de tous les workers :/etc/ssl/certs/ de openssl

for x in $(kubectl get nodes -o jsonpath='{ $.items[*].status.addresses[?(@.type=="InternalIP")].address }'); do scp registry/certs/tls.crt root@$x:/etc/ssl/certs/tls.pem; done

Création d'un déploiement de apache dans notre registre privée :

Téléchargez l'image et stockez la dans le registre privée

```
docker pull httpd
docker tag httpd:latest docker-registry:5000/httpd:v1
docker push docker-registry:5000/httpd:v1
```

On peut relancer la prise en compte des certificats :

```
tdnf install -y openssl-c_rehash
rehash_ca_certificates.sh
```

Notre manifest httpd.yaml pour apache est le suivant :

```
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-devops
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  hostPath:
    path: /mnt/security
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-devops
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: manual
  resources:
```

```
    requests:
      storage: 2Gi
---

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: httpd
  name: apache-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: httpd
  template:
    metadata:
      labels:
        app: httpd
    spec:
      containers:
        -
          image: "docker-registry:5000/httpd:v1"
          name: container-devops
          ports:
            -
              containerPort: 80
          volumeMounts:
            -
              mountPath: /usr/local/apache2/htdocs/
              name: storage-devops
      imagePullSecrets:
        -
          name: regcred
      volumes:
        -
          name: storage-devops
          persistentVolumeClaim:
            claimName: pvc-devops
---
apiVersion: v1
kind: Service
metadata:
```

```
  name: web-devops
  labels:
    app: httpd
spec:
  selector:
    app: httpd
  type: NodePort
  ports:
    - protocol: TCP
      nodePort: 30008
      port: 80
      targetPort: 80
      name: http
```

Création d'un volume persistent pour apache de 2GO

Création d'un déploiement de pods de deux replicas sur les workers ou servers

Récupération de l'image dans notre registre privée docker-registry:5000/httpd:v1.

Le point de montage du pod se sera dans /usr/local/apache2/htdocs/
N'oublions pas notre secret à déployer

```
imagePullSecrets:
      -
        name: regcred
```

notre service nous mets à disposition notre service web sur le port 30008

```
ports:
   - protocol: TCP
     nodePort: 30008
```

Pour copier les fichier de l'OS Hosts dans la VM sous VMware Workstation pro

Créer un dossier partagé sous Vmware

Sous la VM vérifiez si le partage est effectif

vmware-hgfsclient

mkdir /mnt/hgfs/share1

mettre en place la partage sur le dossier partagé ici Sycobs5

vmhgfs-fuse .host:/Sycobs5 /mnt/hgfs/share1

cd /mnt/hgfs/share1

Pour copier des fichiers de la VM vers le pod

```
kubectl cp /root/*.html apache-deployment-5b5f64795b-q5jrp:/usr/local/apache2/htdocs
```

avec apache-deployment-5b5f64795b-q5jrp le nom de pod

Pour recupérer le shell du pod

```
kubectl exec -it apache-deployment-5b5f64795b-q5jrp -- /bin/bash
```

attention aux droits des fichiers de apache dans :/usr/local/apache2/htdocs

```
chmod 777 *
```

pour sortir du shell du pod

```
exit
```

Extraire une image d'un registre docker privé dans Kubernetes Pod

Pour extraire une image de Docker Hub, vous devez vous authentifier auprès d'un registre.

Utilisez l'outil docker pour vous connecter au Docker Hub comme ci-dessous.

Un nom d'utilisateur et un mot de passe sont nécessaires pour se connecter.

```
Docker login -u xxxx -p xxxxx
```

Le processus de connexion crée ou met à jour le fichier config.json qui contient un jeton d'autorisation.

```
cat /root/.docker/config.json
```

```
{
    "auths": {
        "docker-registry:5000": {
            "auth": "cm9vdDp0cHJlc2VhdTEx"
        }
    },
    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.10 (linux)"
    }
}
```

Créer un secret basé sur les informations d'identification existantes

Un cluster Kubernetes utilise le type Secret of kubernetes.io/dockerconfigjson

pour s'authentifier auprès d'un registre de conteneur afin d'extraire une image privée.

Si vous avez déjà exécuté la connexion docker, vous pouvez copier ces informations d'identification dans Kubernetes :

```
kubectl create secret generic regcred --from-
file=.dockerconfigjson=/root/.docker/config.json --
type=kubernetes.io/dockerconfigjson
```

Vous pouvez inspecter le secret comme ci-dessous.

```
kubectl get secret regcred --output=yaml
```

```
apiVersion: v1
data:
  .dockerconfigjson: ewoJImF1dGxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
kind: Secret
metadata:
  creationTimestamp: "2022-09-05T05:12:01Z"
  name: regcred
  namespace: default
  resourceVersion: "2191"
  uid: 85983d1c-e56e-42a1-a160-efb2aab7a021
type: kubernetes.io/dockerconfigjson
```

La valeur du champ .dockerconfigjson est une représentation en base64 de vos informations d'identification Docker.

Pour comprendre ce qui se trouve dans le champ .dockerconfigjson, convertissez les données secrètes dans un format lisible :

```
kubectl get secret regcred --output="jsonpath={.data.\.dockerconfigjson}" | base64 -
-decode
```

```
{
      "auths": {
            "docker-registry:5000": {
                  "auth": "cm9vdDp0cHJlc2VhdTEx"
            }
```

```
},
"HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.10 (linux)"
}
```

Pour Créer un pod qui utilise le secret pour extraire l'image

```
vi my-private-reg-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: private-reg
spec:
  containers:
  - name: private-reg-container
    image: <your-private-image>
  imagePullSecrets:
  - name: regcred
```

```
kubectl apply -f my-private-reg-pod.yaml
kubectl get pod private-reg
```

Notez que le champ imagePullSecrets spécifie que Kubernetes doit obtenir les informations d'identification d'un secret nommé regcred afin d'extraire une image de conteneur de Docker Hub.

<your-private-image> est le nom de ton image dans le registre privé