eBook Gratuit

APPRENEZ kubernetes

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#kubernetes

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: kubernetes

It is an unofficial and free kubernetes ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official kubernetes.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec kubernetes

Remarques

Kubernetes est une plate-forme open source permettant d'automatiser le déploiement, la mise à l'échelle et les opérations des conteneurs d'applications sur des clusters d'hôtes, fournissant une infrastructure centrée sur les conteneurs.

Avec Kubernetes, vous pouvez répondre rapidement et efficacement aux demandes des clients:

- Déployez vos applications rapidement et de manière prévisible.
- Mettez à l'échelle vos applications à la volée.
- Déployer de manière transparente de nouvelles fonctionnalités.
- Optimisez l'utilisation de votre matériel en utilisant uniquement les ressources dont vous avez besoin.

Pourquoi ai-je besoin de Kubernetes et que peut-il faire?

Kubernetes peut planifier et exécuter des conteneurs d'applications sur des grappes de machines physiques ou virtuelles. Cependant, Kubernetes permet également aux développeurs de "couper le cordon" aux machines physiques et virtuelles, en passant d'une infrastructure **centrée sur l'hôte** à une infrastructure **centrée sur** les **conteneurs** , qui offre tous les avantages et bénéfices inhérents aux conteneurs. Kubernetes fournit l'infrastructure nécessaire à la création d'un environnement de développement véritablement centré sur les conteneurs.

Kubernetes répond à un certain nombre de besoins communs des applications en cours de production, telles que:

- la co-localisation des processus d'assistance, la facilitation des applications composites et la préservation du modèle à une application par conteneur,
- montage de systèmes de stockage,
- distribuer des secrets,
- vérification de la santé des applications,
- répliquer des instances d'application,
- mise à l'échelle automatique horizontale,
- dénomination et découverte,
- l'équilibrage de charge,
- mises à jour roulantes,
- suivi des ressources,
- consigner l'accès et l'ingestion,
- prise en charge de l'introspection et du débogage, et
- identité et autorisation.

Cela offre la simplicité de la plateforme en tant que service (PaaS) avec la flexibilité de l'infrastructure en tant que service (IaaS) et facilite la portabilité entre les fournisseurs d'infrastructure.

Versions

Version	Date de sortie
1,7	2017-06-28
1.6	2017-02-22
1,5	2016-12-13
1.4	2016-09-26
1.3	2016-07-06
1.2	2016-03-17
1.1	2015-09-09
1.0	2015-07-18

Examples

Installation de Minikube

Minikube crée un cluster local de machines virtuelles pour exécuter Kubernetes. C'est la méthode la plus simple pour mettre la main sur Kubernetes sur votre machine locale.

La documentation de Minikube est disponible sur http://kubernetes.io/docs/getting-started-guides/minikube/

Exigences

- Sur les hyperviseurs macOS, xhyve , VirtualBox ou VMware Fusion
- Sur les hyperviseurs Linux, VirtualBox ou KVM
- Sur les hyperviseurs Windows VirtualBox ou Hyper-V
- Virtualisation VT-x / AMD-v activée

Pour vérifier si la prise en charge de la virtualisation est activée, exécutez la commande appropriée ci-dessous. La commande affichera quelque chose si la virtualisation est activée.

```
# On Linux
cat /proc/cpuinfo | grep 'vmx\|svm'
# On OSX
sysctl -a | grep machdep.cpu.features | grep VMX
```

Installation

Minikube est un binaire unique. Ainsi, l'installation est aussi simple que de télécharger le binaire et de le placer sur votre chemin.

```
# Specify the version of minikube to download.
# Latest version can be retrieved from
# https://github.com/kubernetes/minikube/releases
VERSION=v0.16.0

# If on Linux
OS=linux
# If on OSX
# OS=darwin

# URL to download minikube binary from
URL=https://storage.googleapis.com/minikube/releases/$VERSION/minikube-$OS-amd64

# Download binary and place in path.
curl -Lo minikube $URL
chmod +x minikube
sudo mv minikube /usr/local/bin/
```

Usage

Pour démarrer un nouveau cluster:

```
minikube start
```

Cela créera un nouveau cluster de machines virtuelles locales avec Kubernetes déjà installé et configuré.

Vous pouvez accéder au tableau de bord Kubernetes avec:

```
minikube dashboard
```

Minikube crée un contexte associé à `kubectl` qui peut être utilisé avec:

```
kubectl config use-context minikube
```

Une fois prêts, les Kubernetes locaux peuvent être utilisés:

```
kubectl run hello-minikube --image=gcr.io/google_containers/echoserver:1.4 --port=8080
kubectl expose deployment hello-minikube --type=NodePort
curl $(minikube service hello-minikube --url)
```

Pour arrêter le cluster local:

```
minikube stop
```

Pour supprimer le cluster local, notez que la nouvelle adresse IP sera attribuée après la création:

```
minikube delete
```

Installer sur Google Cloud

Kubernetes a été initialement développé par Google pour alimenter leur moteur conteneur . En tant que tel, les clusters Kubernetes sont un citoyen de première classe chez Google.

La création d'un cluster Kubernetes dans le moteur de conteneur nécessite la commande `gcloud` partir du SDK Google Cloud . Pour installer cette commande localement, utilisez l'une des options suivantes:

- utilisez l'installateur interactif (le moyen le plus simple pour les nouveaux arrivants):

```
curl https://sdk.cloud.google.com | bash
exec -l $SHELL
gcloud init
```

- Téléchargez le kit de développement depuis https://cloud.google.com/sdk/ et exécutez le fichier d'installation approprié.

  Par exemple, pour installer sous Linux (x86_64):

```
curl -Lo gcloud-sdk.tar.gz https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-142.0.0-linux-x86_64.tar.gz
tar xvf ./gcloud-sdk.tar.gz
./google-cloud-sdk/install.sh
gcloud init
```

Une fois que `gcloud` est installé, créez un cluster Kubernetes avec:

```
# Give our cluster a name
CLUSTER_NAME=example-cluster

# Number of machines in the cluster.
NUM_NODES=3

gcloud container clusters create $CLUSTER_NAME --num_nodes=$NUM_VMS
```

Configurez kubectl

Un cluster Kubernetes est contrôlé à l'aide de la commande `kubectl` . La méthode de configuration de `kubectl` dépend de l'emplacement d'installation de Kubernetes.

Google Cloud (moteur de conteneur)

Pour installer kubectl à l'aide du SDK Google Cloud:

```
gcloud components install kubectl
```

Pour configurer kubectl pour contrôler un cluster Kubernetes existant dans Container Engine:

```
gcloud container clusters get-credentials $CLUSTER_NAME
```

Minikube

Lorsque vous utilisez minikube, le binaire kubectl doit être téléchargé manuellement et placé dans le chemin.

```
# Version of Kubernetes.
K8S_VERSION=$(curl -sS https://storage.googleapis.com/kubernetes-release/release/stable.txt)
# Operating System. Can be one of {linux, darwin}
GOOS=linux
# Architecture. Can be one of {386, amd64, arm64, ppc641e}
GOARCH=amd64

# Download and place in path.
curl -Lo kubectl http://storage.googleapis.com/kubernetes-
release/release/${K8S_VERSION}/bin/${GOOS}/${GOARCH}/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/
```

Le binaire minikube configure automatiquement kubectl au démarrage d'un cluster.

```
minikube start
# kubectl is now ready to use!
```

Kubectl en ligne de commande

Une fois que vous avez un cluster en cours d'exécution, vous pouvez le gérer avec la commande `kubectl` . La plupart des commandes que vous pouvez obtenir avec la commande `kubectl --help` , mais je vous montre les commandes les plus courantes, pour gérer et obtenir des informations sur votre cluster, vos nœuds, vos modules, vos services et vos étiquettes.

Pour obtenir des informations sur le cluster, vous pouvez utiliser la commande suivante

```
kubectl cluster-info
```

Il vous montrera l'adresse courante et le port.

Pour obtenir de brèves informations sur les nœuds, les modules, les services, etc. ou les ressources qui ont une place sur le cluster, vous pouvez utiliser la commande suivante

```
kubectl get {nodes, pods, services, ...}
```

La sortie est principalement une ligne par ressource.

Pour obtenir une description détaillée des ressources, vous pouvez utiliser l'indicateur `describe` pour le `kubectl`

```
kubectl describe {nodes, pods, ...}
```

Les applications déployées ne sont visibles qu'à l'intérieur du cluster. Si vous souhaitez obtenir la sortie de l'extérieur du cluster, vous devez créer une route entre le cluster et le cluster Kubernetes.

```
kubectl proxy
```

Il va ouvrir une API, où nous pouvons tout obtenir du cluster. Si vous souhaitez obtenir le nom des modules pour obtenir des informations, vous devez utiliser la commande suivante:

```
kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}'
```

Il listera les modules pour une utilisation ultérieure.

```
curl http://localhost:8001/api/v1/proxy/namespaces/default/pods/{pod_name}/
```

Deux autres commandes communes sont les journaux de lecture et l'exécution d'une commande depuis / dans l'application conteneurisée.

```
kubectl logs {pod_name}
kubectl exec {pod_name} {command}
```

La configuration de l'achèvement de l'onglet pour votre shell peut être effectuée avec:

```
source <(kubectl completion zsh)   # if you're using zsh
source <(kubectl completion bash)  # if you're using bash
```

ou plus par programme:

```
source <(kubectl completion "${0/-/}")
```

Bonjour le monde

Une fois que votre cluster Kubernetes est en cours d'exécution et que `kubectl` est configuré, vous pouvez exécuter votre première application en quelques étapes. Cela peut être fait en utilisant les commandes impératives qui n'ont pas besoin de fichiers de configuration.

Pour exécuter une application, vous devez fournir un nom de déploiement (`bootcamp`), l'emplacement de l'image du conteneur (`docker.io/jocatalin/kubernetes-bootcamp:v1`) et le port (`8080`)

```
$ kubectl run bootcamp --image=docker.io/jocatalin/kubernetes-bootcamp:v1 --port=8080
```

Confirmer qu'il a travaillé avec:

```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bootcamp               1         1         1            1         6s
```

Pour exposer votre application et la rendre accessible depuis l'extérieur:

```
$ kubectl expose deployment/bootcamp --type="LoadBalancer" --port 8080
```

Confirmer qu'il a travaillé avec:

```
$ kubectl get services
NAME                  CLUSTER-IP    EXTERNAL-IP          PORT(S)     AGE
kubernetes            10.0.0.1      <none>               443/TCP     3m
bootcamp              10.3.245.61   104.155.111.170      8080:32452/TCP   2m
```

Pour accéder aux services, utilisez l'adresse IP externe et le port d'application, par exemple:

```
$ export EXTERNAL_IP=$(kubectl get service bootcamp --
output=jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ export PORT=$(kubectl get services --output=jsonpath='{.items[0].spec.ports[0].port}')
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-2fhnk | v=1
```

La même chose pourrait être faite manuellement avec les données fournies dans:

```
$ kubectl describe service bootcamp
Name:                    bootcamp
Namespace:               default
Labels:                  run=bootcamp
Selector:                run=bootcamp
Type:                    LoadBalancer
IP:                      10.3.245.61
LoadBalancer Ingress:    104.155.111.170
Port:                    <unset> 8080/TCP
NodePort:                <unset> 32452/TCP
Endpoints:               10.0.0.3:8080
... events and details left out ....

$ export NODE=104.155.111.170
$ export PORT=8080
```

Une fois que cela a fonctionné, vous pouvez augmenter votre application avec:

```
$ kubectl scale deployments/bootcamp --replicas=4
```

Et vérifiez le résultat avec:

```
$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
bootcamp               4         4         4            4          30s

$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-2fhnk | v=1
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-390780338-gmtv5 | v=1
```

Attention à la modification de l'identifiant du pod.

Pour lancer une nouvelle version de l'application, exécutez:

```
kubectl set image deployments/bootcamp bootcamp=jocatalin/kubernetes-bootcamp:v2
```

Et confirmez avec:

```
$ curl "$EXTERNAL_IP:$PORT"
Hello Kubernetes bootcamp! | Running on: bootcamp-284539476-gafwev3 | v=2
```

Le nettoyage est enfin fait avec:

```
$ kubectl delete deployment bootcamp
$ kubectl delete service bootcamp
```

Lire Démarrer avec kubernetes en ligne: https://riptutorial.com/fr/kubernetes/topic/4099/demarrer-avec-kubernetes

Chapitre 2: Appel de l'API Kubernetes

Examples

Utilisation de Kubernetes Go Client - Hors Cluster

```
package main

import (
  "fmt"

  "k8s.io/client-go/1.5/kubernetes"
  "k8s.io/client-go/1.5/pkg/api/v1"
  "k8s.io/client-go/1.5/tools/clientcmd"
)

func main()  {
    config, err := clientcmd.BuildConfigFromFlags("", <kube-config-path>)
    if err != nil {
      return nil, err
    }

    c, err := kubernetes.NewForConfig(config)
    if err != nil {
      return nil, err
    }

    // Get Pod by name
    pod, err := c.Pods(v1.NamespaceDefault).Get("my-pod")
    if err != nil {
        fmt.Println(err)
        return
    }

    // Print its creation time
    fmt.Println(pod.GetCreationTimestamp())
}
```

Liste des réplicas par déploiement donné avec le client kubernetes go

```
package main

import (
    "k8s.io/kubernetes/pkg/api"
    unver "k8s.io/kubernetes/pkg/api/unversioned"
    "k8s.io/kubernetes/pkg/apis/extensions"
    "k8s.io/kubernetes/pkg/client/restclient"
    client "k8s.io/kubernetes/pkg/client/unversioned"
    "log"
    "os"
    "strings"
)

var logger *log.Logger
```

```
const (
    SERVER             string = "http://172.21.1.11:8080"
    RevisionAnnotation        = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func getReplicaSetsByDeployment(c *client.Client, deployment *extensions.Deployment)
([]extensions.ReplicaSet, error) {

    namespace := deployment.Namespace
    selector, err := unver.LabelSelectorAsSelector(deployment.Spec.Selector)
    if err != nil {
        return nil, err
    }
    options := api.ListOptions{LabelSelector: selector}
    rsList, err := c.Extensions().ReplicaSets(namespace).List(options)

    return rsList.Items, nil
}

func getDeploymentByReplicaSet(namespace string, c *client.Client, rs *extensions.ReplicaSet)
([]extensions.Deployment, error) {

    selector, err := unver.LabelSelectorAsSelector(rs.Spec.Selector)
    if err != nil {
        return nil, err
    }
    options := api.ListOptions{LabelSelector: selector}
    dps, err := c.Extensions().Deployments(namespace).List(options)
    if err != nil {
        return nil, err
    }

    return dps.Items, nil
}

func getDeploymentByReplicaSetName(namespace string, c *client.Client, rs
*extensions.ReplicaSet) (*extensions.Deployment, error) {

    name := rs.Name
    index := strings.LastIndex(name, "-")
    deploymentName := name[:index]

    dp, err := c.Extensions().Deployments(namespace).Get(deploymentName)
    if err != nil {
        return nil, err
    }

    return dp, nil
}

func main() {

    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
```

```
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    list, err := c.Extensions().Deployments(api.NamespaceDefault).List(api.ListOptions{})
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }

    logger.Printf("Deployment -------> ReplicaSet: ")

    for _, deployment := range list.Items {
        rses, err := getReplicaSetsByDeployment(c, &deployment)
        if err != nil {
            logger.Fatalf("GetReplicaSetsByDeployment Error: err=%s\n", err)
        }

        for _, rs := range rses {
            logger.Printf("ReplicaSet assioated with Deployment: rs-name=%s, revision=%s, dp-
name=%s\n",
                rs.Name, rs.Annotations[RevisionAnnotation], deployment.Name)
        }
    }

    logger.Printf("\n\nReplicaSet -------> Deployment: ")
    rsList, err := c.Extensions().ReplicaSets(api.NamespaceDefault).List(api.ListOptions{})
    if err != nil {
        log.Fatalf("Could not list ReplicaSet")
    }

    for _, rs := range rsList.Items {
        dp, err := getDeploymentByReplicaSetName(api.NamespaceDefault, c, &rs)
        if err != nil {
            logger.Fatalf("GetDeploymentByReplicaSet Error: err=%s\n", err)
        }

        logger.Printf("Deployment assioated with ReplicaSet: rs-name=%s, revision=%s, dp-
name=%s\n",
            rs.Name, rs.Annotations[RevisionAnnotation], dp.Name)
    }
}
```

Restauration avec la révision de réplicas à l'aide de kubernetes go client

```
package main

import (
    //"k8s.io/kubernetes/pkg/api"
    "k8s.io/kubernetes/pkg/client/restclient"
    "log"
    "os"
    // unver "k8s.io/kubernetes/pkg/api/unversioned"
    "k8s.io/kubernetes/pkg/apis/extensions"
    client "k8s.io/kubernetes/pkg/client/unversioned"
)

var logger *log.Logger

var annotations = map[string]string{
```

```
    "Image":                      "nginx:1.7.9",
    "UserId":                     "2",
    "kubernetes.io/change-cause": "version mismatch",
}

const (
    DEPLOYMENT         string = "nginx-test"
    REVERSION          int64  = 4
    SERVER             string = "http://172.21.1.11:8080"
    RevisionAnnotation        = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func rollBack(c *client.Client, dp *extensions.Deployment, revision int64) error {

    dr := new(extensions.DeploymentRollback)
    dr.Name = dp.Name
    dr.UpdatedAnnotations = annotations
    dr.RollbackTo = extensions.RollbackConfig{Revision: revision}

    // Rollback
    err := c.Extensions().Deployments("ops").Rollback(dr)
    if err != nil {
        logger.Printf("Deployment Rollback Error: err=%s\n", err)
        return err
    }

    return nil
}

func main() {
    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    dp, err := c.Extensions().Deployments("ops").Get(DEPLOYMENT)
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }

    rollBack(c, dp, REVERSION)

}
```

Mise à jour progressive avec des répliasets utilisant le client kubernetes go

```
package main

import (
    //    "k8s.io/kubernetes/pkg/api"
    // unver "k8s.io/kubernetes/pkg/api/unversioned"
```

```
    "k8s.io/kubernetes/pkg/apis/extensions"
    "k8s.io/kubernetes/pkg/client/restclient"
    client "k8s.io/kubernetes/pkg/client/unversioned"
    "k8s.io/kubernetes/pkg/util/intstr"
    "log"
    "os"
)

var logger *log.Logger

const (
    //DEPLOYMENT           string = "nginx-deployment"
    DEPLOYMENT           string = "nginx-test"
    RevisionHistoryLimit int32  = 5
    SERVER               string = "http://172.21.1.11:8080"
    RevisionAnnotation   string = "deployment.kubernetes.io/revision"
)

func init() {
    logger = log.New(os.Stdout, "", 0)
}

func rollingUpdate(c *client.Client, dp *extensions.Deployment) error {

    // New a DeploymentStrategy
    ds := new(extensions.DeploymentStrategy)
    ds.Type = extensions.RollingUpdateDeploymentStrategyType
    ds.RollingUpdate = new(extensions.RollingUpdateDeployment)
    ds.RollingUpdate.MaxUnavailable = intstr.FromInt(int(dp.Spec.Replicas))

    dp.Spec.Strategy = *ds

    // Image
    //dp.Spec.Template.Spec.Containers[0].Image = "nginx:1.9.7"
    dp.Spec.Template.Spec.Containers[0].Image = "nginx:1.9"

    // Update
    //_, err := c.Extensions().Deployments(api.NamespaceDefault).Update(dp)
    _, err := c.Extensions().Deployments("ops").Update(dp)
    if err != nil {
        logger.Printf("Update Deployment Error: err=%s\n", err)
        return err
    }
    return nil
}

func main() {
    config := &restclient.Config{
        Host: SERVER,
    }

    c, err := client.New(config)
    if err != nil {
        logger.Fatalf("Could not connect to k8s api: err=%s\n", err)
    }

    //dp, err := c.Extensions().Deployments(api.NamespaceDefault).Get(DEPLOYMENT)
    dp, err := c.Extensions().Deployments("ops").Get(DEPLOYMENT)
    if err != nil {
        logger.Fatalf("Could not list deployments: err=%s\n", err)
    }
```

```
    rollingUpdate(c, dp)

}
```

Utilisation de Kubernetes Go Client - À l'intérieur du cluster

```
package main

import (
  "fmt"

  "k8s.io/client-go/1.5/kubernetes"
  "k8s.io/client-go/1.5/pkg/api/v1"
  "k8s.io/client-go/1.5/rest"
)

func main()  {
    config, err = rest.InClusterConfig()
    if err != nil {
      return nil, err
    }

    c, err := kubernetes.NewForConfig(config)
    if err != nil {
      return nil, err
    }

    // Get Pod by name
    pod, err := c.Pods(v1.NamespaceDefault).Get("my-pod")
    if err != nil {
        fmt.Println(err)
        return
    }

    // Print its creation time
    fmt.Println(pod.GetCreationTimestamp())
}
```

Lire Appel de l'API Kubernetes en ligne: https://riptutorial.com/fr/kubernetes/topic/5926/appel-de-l-api-kubernetes

Chapitre 3: Kubernetes en production

Introduction

Présenter comment utiliser les kubernetes dans un environnement de production

Examples

Déployer un cluster zookeeper en production à l'aide de kubernetes et de ceph

Dockerize zookeeper-3.4.6

Créez un fichier Docker:

```
#########################################################
# Image: img.reg.3g:15000/zookeeper:3.4.6
#########################################################

FROM img.reg.3g:15000/jdk:1.7.0_67

MAINTAINER lth9739@gmail.com

USER root

ENV ZOOKEEPER_VERSION 3.4.6

ADD Dockerfile /

ADD zookeeper/ /opt/

COPY zoo.cfg /opt/zookeeper/conf/zoo.cfg

RUN mkdir -p /opt/zookeeper/{data,log}

WORKDIR /opt/zookeeper

VOLUME ["/opt/zookeeper/conf", "/opt/zookeeper/data", "/opt/zookeeper/log"]

COPY config-and-run.sh /opt/zookeeper/bin/

EXPOSE 2181 2888 3888

CMD ["/opt/zookeeper/bin/config-and-run.sh"]
```

Voir plus de détails

Déployer le contrôleur de réplique zookeeper dans le cluster kubernetes

Vous pouvez utiliser cette commande pour déployer le contrôleur de réplique de zookeeper:

```
kubectl create -f zookeeper-rc-1.json
```

```
{
  "apiVersion": "v1",
  "kind": "ReplicationController",
  "metadata": {
    "labels": {
      "component": "zookeeper"
    },
    "name": "zookeeper-1"
  },
  "spec": {
    "replicas": 1,
    "selector": {
      "server-id": "1",
      "role": "zookeeper-1"
    },
    "template": {
      "metadata": {
        "labels": {
          "server-id": "1",
          "role": "zookeeper-1"
        },
        "name": "zookeeper-1"
      },
      "spec": {
        "containers": [
          {
            "env": [
              {
                "value": "1",
                "name": "SERVER_ID"
              },
              {
                "value": "5",
                "name": "MAX_SERVERS"
              }
            ],
            "image": "img.reg.3g:15000/fabric8/zookeeper:latest",
            "name": "zookeeper-1",
            "ports": [
              {
                "containerPort": 2181,
                "name": "client",
                "protocol": "TCP"
              },
              {
                "containerPort": 2888,
                "name": "followers",
                "protocol": "TCP"
              },
              {
                "containerPort": 3888,
                "name": "election",
                "protocol": "TCP"
              }
            ],
            "volumeMounts": [
              {
                "mountPath": "/opt/zookeeper/data",
                "name": "zookeeper-1"
```

```
              }
            ]
          }
        ],
        "restartPolicy": "Always",
        "volumes": [
          {
            "name": "zookeeper-1",
            "rbd": {
              "monitors": [
                "10.151.32.27:6789",
                "10.151.32.29:6789",
                "10.151.32.32:6789"
              ],
              "pool": "rbd",
              "image": "log-zookeeper-1",
              "user": "admin",
              "secretRef": {
                "name": "ceph-secret-default"
              },
              "fsType": "ext4",
              "readOnly": false
            }
          }
        ]
      }
    }
  }
}
```

Déployer le service zookeeper dans le cluster kubernetes

Vous pouvez utiliser cette commande pour déployer le service de zookeeper:

```
kubectl create -f zookeeper-svc-1.json
```

```
{
  "kind": "Service",
  "apiVersion": "v1",
  "metadata": {
    "name": "zookeeper-1",
    "labels": {
      "name": "zookeeper-1"
    }
  },
  "spec": {
    "ports": [
      {
        "name": "client",
        "port": 2181,
        "targetPort": 2181
      },
      {
        "name": "followers",
        "port": 2888,
        "targetPort": 2888
      },
      {
```

```
        "name": "election",
        "port": 3888,
        "targetPort": 3888
      }
    ],
    "selector": {
      "server-id": "1"
    }
  }
}
```

Cluster Zookeeper

Si vous voulez obtenir un cluster zookeeper avec 5 nœuds, vous pouvez écrire les fichiers zookeeper-rc-2/3/4 / 5.json et zookeeper-svc-2/3/4 / 5.jan comme décrit ci-dessus et utiliser la commande kubectl pour déployez-les dans le cluster Kubernetes.

Lire Kubernetes en production en ligne: https://riptutorial.com/fr/kubernetes/topic/9153/kubernetes-en-production

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec kubernetes	cledoux, Community, Dimitris, idvoretskyi, jayantS, jkantihub, mohan08p, Ogre Psalm33, pagid, PumpkinSeed, webdizz
2	Appel de l'API Kubernetes	litanhua, Rush
3	Kubernetes en production	litanhua