1. Table des matières

Chapitre 1 : Installation

Pré-requis kubectl
Installation minikube
Installation kubeadm

2. Installation kubectl

Nous allons installer kubectl sur un poste de travail (Desktop Windows/Mac/Linux)

2.1 Linux

2.1.1 Avec curl

Télécharger le fichier binaire statique

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/
linux/amd64/kubectl && chmod +x ./kubectl
```

Déplacer le binaire

```
sudo install kubectl /usr/local/bin
```

2.1.2 Avec homebrew

```
brew install kubectl
```

2.2 Windows

2.2.1 Avec Windows Installer

Télécharger le fichier kubectl installer

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.20.0/
bin/windows/amd64/kubectl.exe
```

Puis ajouter le binaire dans le PATH

2.2.2 Avec Chocolatey

```
choco install kubernetes-cli
```

2.3 MAC OS

2.3.1 Avec curl

Télécharger le fichier kubectl installer

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/
darwin/amd64/kubectl && chmod +x ./kubectl
```

Déplacer le binaire

```
sudo mv ./kubectl /usr/local/bin
```

2.3.2 Avec homebrew

```
brew install kubectl
```

2.4 Vérifier l'installation

Afficher la version

```
kubectl version
```

Afficher le help

```
minikube --help
```

3. Installation minikube

Nous allons installer minikube sur un poste de travail (Desktop Windows/Mac/Linux)

3.1 Linux

3.1.1 Avec curl

Télécharger le fichier binaire statique

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/
minikube-linux-amd64 && chmod +x minikube
```

Déplacer le binaire

```
sudo install minikube /usr/local/bin
```

3.1.2 Avec homebrew

```
brew install minikube
```

3.2 Windows

3.2.1 Avec Windows Installer

Télécharger le fichier minikube installer

```
https://github.com/kubernetes/minikube/releases/tag/v1.17.1/minikube-windows-
amd64.exe
```

3.2.2 Avec Chocolatey

```
choco install minikube
```

3.3 MAC OS

3.3.1 Avec curl

Télécharger le fichier binaire statique

```
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/
minikube-darwin-amd64 && chmod +x minikube
```

Déplacer le binaire

```
sudo mv minikube /usr/local/bin
```

3.3.2 Avec homebrew

```
brew install minikube
```

3.4 Vérifier l'installation

Afficher la version

```
minikube version
```

Afficher le help

```
minikube --help
```

3.5 Installer le cluster kubernetes sur minikube

Installer et démarrer l'instance minikube avec le driver par défaut

```
minikube start
```

Installer et démarrer l'instance minikube avec le driver hyper-v

```
minikube start --driver=hyperv
```

4. Installation avec kubeadm

4.1 Installation de Docker

4.1.1 Pré-requis

Mettre à jour les fichiers /etc/hosts des 3 nodes

```
vi /etc/hosts
```

```
192.168.56.101 node1
192.168.56.102 node2
192.168.56.103 node3
```

4.1.2 Installation de Docker

Installer des packages pré-requis pour Docker

```
sudo yum install -y yum-utils net-tools
```

Ajouter le fichier repository sur les noeuds

```
sudo yum-config-manager  --add-repo   https://download.docker.com/linux/
centos/docker-ce.repo
```

Installer les packages docker version 19.03.9 sur les 3 noeuds

```
sudo yum install docker-ce-19.03.9 docker-ce-cli-19.03.9 containerd.io -y
```

Démarrer et activer le daemon

```
sudo systemctl start docker
sudo systemctl enable docker
```

Ajouter le user "centos" au group "docker"

```
sudo usermod -aG docker centos
```

Passer en tant que user "centos" et vérifier la version de docker

```
su - centos
docker version
```

4.2 Installation de Kubernetes

4.2.1 Pré-requis

Sur le Master: Ouvrir les ports réseau requis

```
sudo firewall-cmd --permanent --add-port={6443,2379,2380,10250,10251,10252}/
tcp
sudo firewall-cmd --reload
```

Sur les Workers: Ouvrir les ports réseau requis

```
sudo firewall-cmd --permanent --add-port={10250,30000-32767}/tcp
sudo firewall-cmd --reload
```

Sur tous les noeuds: Ouvrir les ports réseau pour Calico

```
sudo firewall-cmd --permanent --add-port={179,5473,443}/tcp
sudo firewall-cmd --reload
```

Sur tous les noeuds: Autoriser le trafic en mode bridge (exécuter en tant que root)

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

Sur tous les noeuds: ajouter le repository (exécuter en tant que root)

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://
packages.cloud.google.com/yum/doc/rpm-package-key.gpg
exclude=kubelet kubeadm kubectl
EOF

# Set SELinux in permissive mode (effectively disabling it)
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

```
sudo yum install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
sudo systemctl enable --now kubelet
```

4.2.2 Installation du Master

Sur le Master: Disable l swap

```
swapoff -a
sed -e '/swap/s/^/#/g' -i /etc/fstab
```

Sur le Master: initialiser le cluster Kubernetes

```
kubeadm init --apiserver-advertise-address=192.168.56.101 --pod-network-
cidr=192.168.0.0/16
```

Transferer le fichier du config sur le poste client de l'utilisateur

```
scp /etc/kubernetes/admin.conf centos@poste:/home/centos/.kube/config
```

4.2.3 Installation de Calico

Sur le Client en tant que user, appliquer la spécification de Calico

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

Une fois que le Master est READY, exécuter sur les Workers la commande join affichée par la commande kubeadm init

```
kubeadm join
```

5. Fichier de config

5.1 Ajout de fichier de config dans le HOME

Transférer le fichier de configuration

```
scp root@node1:/etc/kubernetes/admin.conf .kube/config
```

Visualier la configuration

```
kubectl config view
```

6. Variable d'environnement

6.1 KUBECONFIG

Créer la variable d'environnement avec les 2 fichiers de config des 2 clusters

```
export KUBECONFIG=/home/centos/.kube/config:/home/centos/.kube/admin.conf
```

Visualier la configuration

```
kubectl config view
```

7. Fusion de plusieurs fichiers de config

Créer la variable d'environnement avec les 2 fichiers de config des 2 clusters

```
export KUBECONFIG=/home/centos/.kube/config:/home/centos/.kube/admin.conf
```

Créer un fichier avec le mode raw de config view

```
kubectl config view --raw > new-config
mv new-config .kube/config
unset KUBECONFIG
```

8. Lab1

8.1 Premier Pod

Créer un pod nommé web basé sur une image nginx

8.2 Deuxième Pod

1. Créer un pod nommé debug basé sur une image alpine
2. Se connecter sur le pod debug et faire un curl sur l'IP du pod web

8.3 Troisième Pod

1. Créer un pod nommé all basé sur 2 conteneurs (nginx et alpine)
2. Se connecter sur le pod all dans le conteneur alpine et faire un curl sur l'IP localhost

9. Lab1

9.1 Premier Pod

9.1.1 Créer un pod nommé web basé sur une image nginx

```
cat web.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
spec:
  containers:
  - name: www
    image: nginx:1.17-alpine
```

```
kubectl apply -f web.yml
```

9.2 Deuxième Pod

9.2.1 Créer un pod nommé debug basé sur une image alpine

```
cat debug.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
  - name: debug
    image: alpine:3.12
    command: ["sleep","3600"]
```

```
kubectl apply -f debug.yml
```

9.2.2 Se connecter sur le pod debug et faire un curl sur l'IP du pod web

```
kubectl exec -it debug -- sh
apk add curl
curl <IP WEB>
```

9.3 Troisième Pod

9.3.1 Créer un pod nommé all basé sur 2 conteneurs (nginx et alpine)

```
cat all.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: all
spec:
  containers:
  - name: www
    image: nginx:1.17-alpine
  - name: debug
    image: alpine:3.12
    command: ["sleep","3600"]
```

```
kubectl apply -f all.yml
```

9.3.2 Se connecter sur le pod all dans le conteneur alpine et faire un curl sur l'IP localhost

```
kubectl exec -it all -c debug -- sh
apk add curl
curl 127.0.0.1
```

10. Lab2 SCHEDULER

10.1 Premier Pod

1. Créer un pod nommé web basé sur une image nginx
2. Ajouter un label app=web
3. Ajouter un nodeSelector sur un label disktype=ssd
4. Appliquer le fichier de spécification et remarquer que le status du pod est Pending

10.2 Deuxième Pod

1. Créer un pod nommé debug basé sur une image alpine
2. Ajouter une Affinity de type pod sur un Label app=web
3. Appliquer le fichier de spécification et remarquer que le status du pod est Pending

10.3 Worker

1. Créer sur un worker un Label disktype=ssd
2. Remarquer que le status des pods nginx et alpine est Running

11. Lab2 SCHEDULER

11.1 Premier Pod

11.1.1 Créer un pod nommé web basé sur une image nginx

```
vi web.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app: web
spec:
  containers:
  - name: www
    image: nginx:1.17-alpine
  nodeSelector:
    disktype: ssd
```

11.1.2 Appliquer les spécifications

```
kubectl apply -f web.yml
```

11.1.3 Visualiser le résultat

```
kubectl get pods -o wide --show-labels
```

11.2 Deuxième Pod

11.2.1 Créer un pod nommé debug basé sur une image alpine

```
vi debug.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
  - name: debug
```

```
    image: alpine
    command: ["sleep","3600"]
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchExpressions:
          - key: app
            operator: In
            values:
            - web
        topologyKey: kubernetes.io/hostname
```

11.2.2 Appliquer les spécifications

```
kubectl apply -f debug.yml
```

11.2.3 Visualiser le résultat

```
kubectl get pods -o wide --show-labels
```

11.3 Worker

11.3.1 Créer sur un worker un Label disktype=ssd

```
vi host13.yml
```

```
apiVersion: v1
kind: Node
metadata:
  labels:
    disktype: ssd
  name: host13
```

11.3.2 Appliquer la spécifiviion

```
kubectl apply -f host13.yml
```

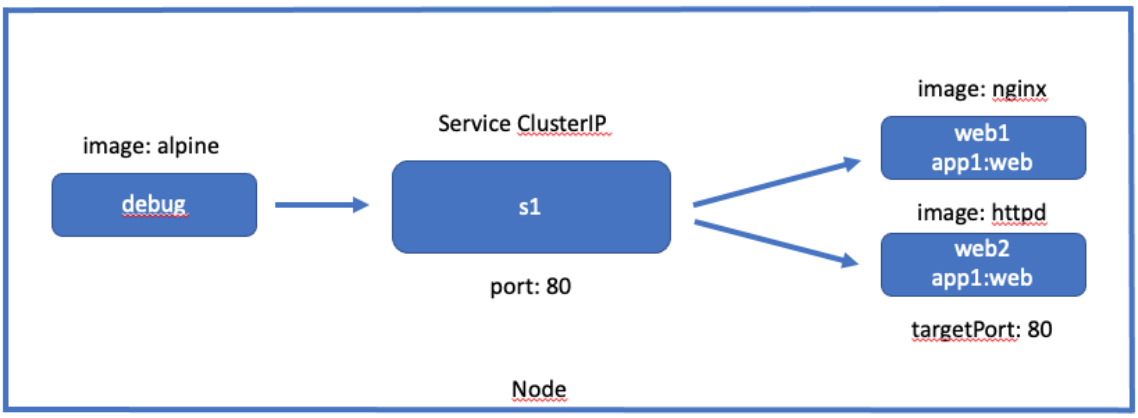
11.3.3 Remarquer que le status des pods nginx et alpine est Running

```
kubectl get pods -o wide --show-labels
```

12. Lab3 SERVICE

12.1 Service ClusterIP

1. Créer un service nommé s1 de type ClusterIP rassemblant des pods ayant comme label app1=web
2. Créer un pod web1 basé sur l'image nginx avec comme label app1=web
3. Créer un pod web2 basé sur l'image httpd avec comme label app1=web
4. Créer un pod debug basé sur l'image alpine
5. Se connecter sur le pod debug et exécuter plusieurs fois la commande "curl s1"



12.2 Service NodePort

1. Créer un service nommé s2 de type NodePort rassemblant des pods ayant comme label app2=web
2. Créer un pod web3 basé sur l'image nginx avec comme label app2=web
3. Créer un pod web4 basé sur l'image httpd avec comme label app2=web
4. Depuis le poste de travail, exécuter plusieurs fois la commande "curl node1:30000"

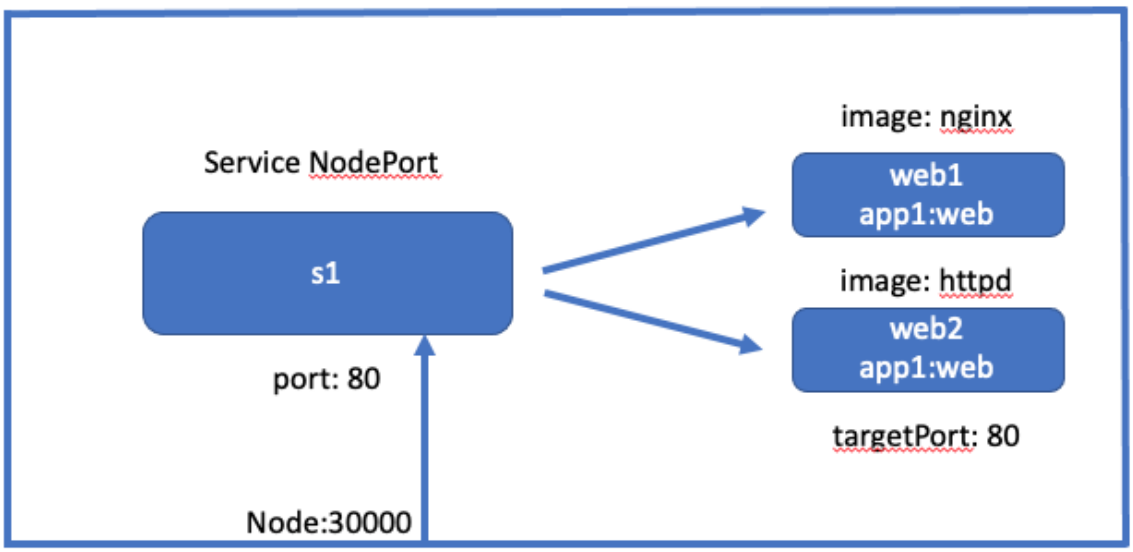
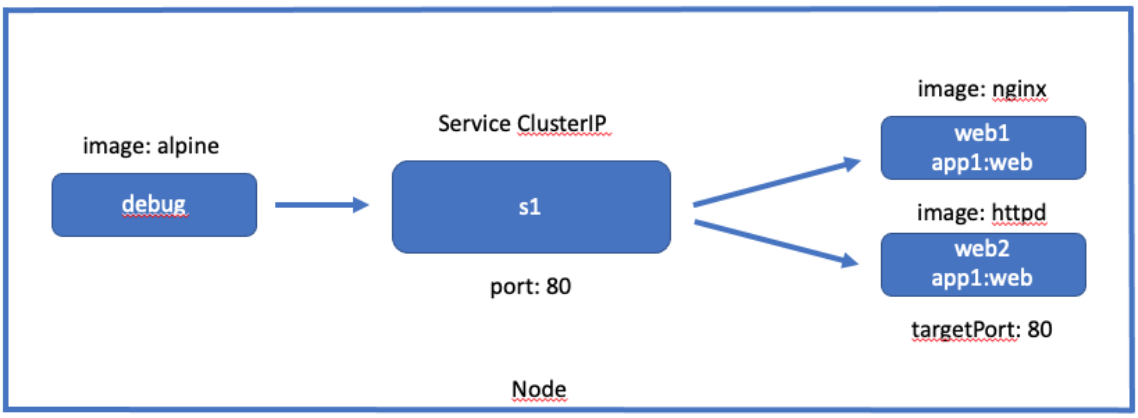


image: nginx
Service NodePort
web1
app1:web
s1
image: httpd
web2
app1:web
port: 80
targetPort: 80
Node:30000

13. Lab3 SERVICE

13.1 Service ClusterIP



13.1.1 Créer un service nommé s1 de type ClusterIP rassemblant des pods ayant comme label app1=web

```
vi s1.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: s1
spec:
  selector:
    app1: web
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
```

13.1.2 Créer un pod web1 basé sur l'image nginx avec comme label app1=web

```
vi web1.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web1
  labels:
    app1: web
```

```
spec:
  containers:
  - name: nginx
    image: nginx:1.17-alpine
```

13.1.3 Créer un pod web2 basé sur l'image httpd avec comme label app1=web

```
vi web2.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web2
  labels:
    app1: web
spec:
  containers:
  - name: nginx
    image: httpd:2.4-alpine
```

13.1.4 Créer un pod debug basé sur l'image alpine

```
vi debug.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
  - name: debug
    image: alpine:3.12
    command: ["sleep","3600"]
```

13.1.5 Se connecter sur le pod debug et exécuter plusieurs fois la commande "curl s1"

Appliquer les spécifications

```
kubectl apply -f .
```

```
pod/debug created
service/s1 created
pod/web1 created
pod/web2 created
```

Visualiser les pods et le service

```
kubectl get svc,pods
```

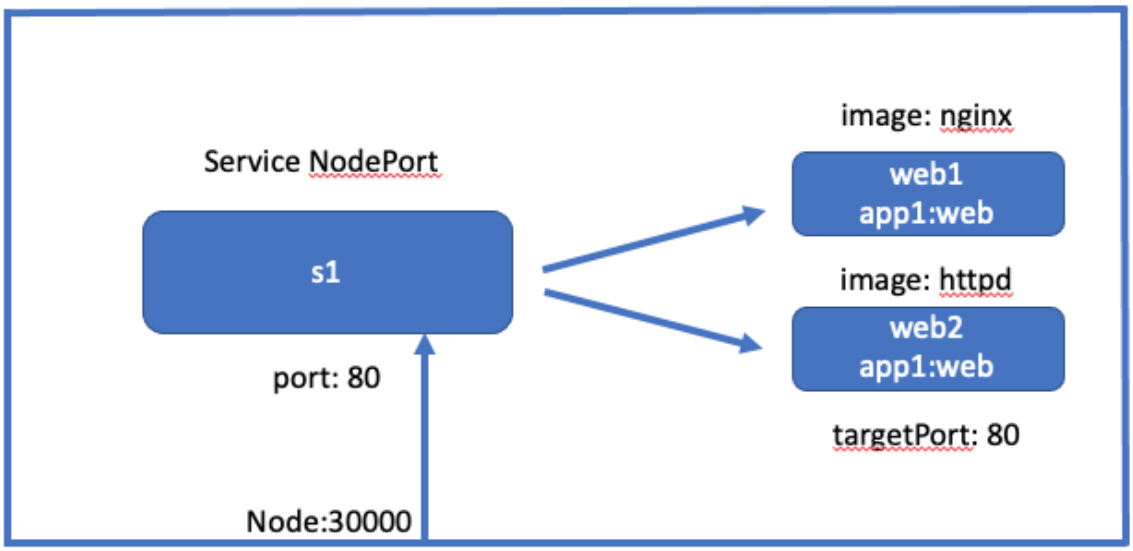
```
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
service/s1           ClusterIP   10.96.245.226   <none>        80/TCP    63s

NAME         READY   STATUS    RESTARTS   AGE
pod/debug    1/1     Running   0          63s
pod/web1     1/1     Running   0          63s
pod/web2     1/1     Running   0          63s
```

Se connecter sur le pod debug et exécuter plusieurs fois la commande "curl s1"

```
kubectl exec -it debug -- sh
```

```
apk add curl
while true
do
curl s1
done
```

13.2 Service NodePort



13.2.1 Créer un service nommé s2 de type NodePort rassemblant des pods ayant comme label app2=web

```
vi s2.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: s2
spec:
  selector:
    app2: web
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30000
```

13.2.2 Créer un pod web3 basé sur l'image nginx avec comme label app2=web

```
vi web3.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web3
  labels:
    app2: web
spec:
  containers:
  - name: nginx
    image: nginx:1.17-alpine
```

13.2.3 Créer un pod web4 basé sur l'image httpd avec comme label app2=web

```
vi web4.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web4
  labels:
    app2: web
spec:
  containers:
  - name: nginx
    image: httpd:2.4-alpine
```

13.2.4 Depuis le poste de travail, exécuter plusieurs fois la commande "curl node1:30000"

Appliquer les spécifications

```
kubectl apply -f .
```

```
service/s2 created
pod/web3 created
pod/web4 created
```

Visualiser les pods et le service

```
kubectl get svc,pods
```

```
NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE
service/s2           NodePort    10.96.34.252   <none>        80:30000/TCP
57s

NAME       READY   STATUS    RESTARTS   AGE
pod/web3   1/1     Running   0          57s
pod/web4   1/1     Running   0          57s
```

Depuis le poste de travail, exécuter plusieurs fois la commande "curl node1:30000"

```
while true
do
curl node1:3000
done
```

14. Lab4 Deployment & DaemonSet

14.1 Deployment

1. Créer un deploiement nommé web (avec 4 replicas) basé sur l'image httpd
2. Créer un service nommé web de type NodePort rassemblant les pods du deploiment web sur le nodePort 30001

14.2 DaemonSet

1. Créer un DaemonSet nommé debug basé sur l'image alpine

15. Lab4 Deployment & DaemonSet

15.1 Deployment

15.1.1 Créer un deploiement nommé web (avec 4 replicas) basé sur l'image httpd

```
vi deployment.yml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 4
  selector:
    matchLabels:
      app3: web
  template:
    metadata:
      labels:
        app3: web
    spec:
      containers:
      - name: apache
        image: httpd:2.4-alpine
```

15.1.2 Créer un service nommé web de type NodePort rassemblant les pods du deploiment web sur le nodePort 30001

```
vi service.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app3: web
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
```

15.1.3 Appliquer les spécifications

```
kubectl apply -f .
```

15.1.4 Visualiser le résultat

```
kubectl get deploy,rs,pods,svc -o wide
```

```
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS
IMAGES             SELECTOR
deployment.apps/web   4/4     4            4           5m20s   apache
httpd:2.4-alpine   app3=web

NAME                               DESIRED   CURRENT   READY   AGE
CONTAINERS   IMAGES              SELECTOR
replicaset.apps/web-66bcf7cf77   4         4         4       5m20s
apache       httpd:2.4-alpine   app3=web,pod-template-hash=66bcf7cf77

NAME                         READY   STATUS        RESTARTS   AGE
IP              NODE     NOMINATED NODE   READINESS GATES
pod/app3-ds-7ldcc            1/1     Terminating   0          3m25s
192.168.0.145   host13   <none>           <none>
pod/app3-ds-vkt88            1/1     Terminating   0          3m25s
192.168.0.72    host12   <none>           <none>
pod/web-66bcf7cf77-2q644     1/1     Running       0          5m20s
192.168.0.143   host13   <none>           <none>
pod/web-66bcf7cf77-bnchn     1/1     Running       0          5m20s
192.168.0.144   host13   <none>           <none>
pod/web-66bcf7cf77-kmmfk     1/1     Running       0          5m20s
192.168.0.71    host12   <none>           <none>
pod/web-66bcf7cf77-pwnnq     1/1     Running       0          5m20s
192.168.0.70    host12   <none>           <none>

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)
AGE     SELECTOR
service/web          NodePort    10.97.155.18   <none>        80:30001/TCP
5m20s   app3=web
```

15.2 DaemonSet

15.2.1 Créer un DaemonSet nommé debug basé sur l'image alpine

```
vi daemonset.yml
```

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: app3-ds
spec:
```

```
  selector:
    matchLabels:
      app3: daemon
  template:
    metadata:
      labels:
        app3: daemon
    spec:
      containers:
      - name: daemon
        image: alpine:3.12
        command: ["sleep","3600"]
```

15.2.2 Appliquer la spécification

```
kubectl apply -f daemonset.yml
```

15.2.3 Visualiser le résultat

```
kubectl get ds,pods -o wide
```

```
NAME                     DESIRED   CURRENT   READY   UP-TO-DATE
AVAILABLE   NODE SELECTOR   AGE   CONTAINERS   IMAGES        SELECTOR
daemonset.apps/app3-ds   2         2         2       2
2           <none>          47s   daemon       alpine:3.12   app3=daemon

NAME                READY   STATUS    RESTARTS   AGE   IP
NODE     NOMINATED NODE   READINESS GATES
pod/app3-ds-l6gcv   1/1     Running   0          47s   192.168.0.146
host13   <none>           <none>
pod/app3-ds-tr6z2   1/1     Running   0          47s   192.168.0.73
host12   <none>           <none>
```

16. Lab5 Namespace

16.1 Namespace

1. Créer un namespace nommé **production**
2. Créer un namespace nommé **development**

16.2 Context

1. Créer un contexte **prod** qui fait référence au namespace **production**
2. Créer un contexte **dev** qui fait référence au namespace **development**

16.3 Utilisation

1. Lancer un pod **web** dans le contexte **dev**
2. Lancer le même pod **web** dans le contexte **prod**

17. Lab5 Namespace

17.1 Namespace

17.1.1 Créer un namespace nommé **production**

```
vi prod.yml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: production
```

17.1.2 Créer un namespace nommé **development**

```
vi dev.yml
```

```
apiVersion: v1
kind: Namespace
metadata:
  name: development
```

17.1.3 Afficher le résultat

```
kubectl get namespaces
```

```
NAME              STATUS   AGE
default           Active   3d12h
development       Active   15h
kube-node-lease   Active   3d12h
kube-public       Active   3d12h
kube-system       Active   3d12h
production        Active   15h
```

17.2 Context

17.2.1 Créer un contexte **prod** qui fait référence au namespace **production**

```
kubectl config set-context prod --cluster=kubernetes --user=kubernetes-admin
--namespace=production
```

17.2.2 Créer un contexte **dev** qui fait référence au namespace **development**

```
kubectl config set-context dev --cluster=kubernetes --user=kubernetes-admin
--namespace=development
```

17.2.3 Afficher le résultat

```
kubectl config get-contexts
```

```
CURRENT   NAME                          CLUSTER      AUTHINFO
NAMESPACE
          dev                           kubernetes   kubernetes-admin
development
*         kubernetes-admin@kubernetes   kubernetes   kubernetes-admin
          prod                          kubernetes   kubernetes-admin
production
```

17.3 Utilisation

17.3.1 Lancer un pod **web** dans le contexte **dev**

```
vi web.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app: web
spec:
  containers:
  - name: www
    image: nginx:1.17-alpine
```

```
kubectl config use-context dev
kubectl apply -f web.yml
```

17.3.2 Lancer le même pod **web** dans le contexte **prod**

```
kubectl config use-context prod
kubectl apply -f web.yml
```

17.3.3 Afficher le résultat

```
kubectl get pods --all-namespaces
```

```
NAMESPACE     NAME                                       READY   STATUS
RESTARTS   AGE
development   web                                        1/1     Running
0          8s
kube-system   calico-kube-controllers-bcc6f659f-r6rkz    1/1     Running
3          3d1h
kube-system   calico-node-992r2                          1/1     Running
1          3d1h
kube-system   calico-node-dxwqg                          1/1     Running
1          3d1h
kube-system   calico-node-n5d6v                          1/1     Running
5          3d1h
kube-system   coredns-74ff55c5b-dztqx                    1/1     Running
3          3d13h
kube-system   coredns-74ff55c5b-kptvf                    1/1     Running
4          3d13h
kube-system   etcd-host11                                1/1     Running
5          3d13h
kube-system   kube-apiserver-host11                      1/1     Running
4          3d13h
kube-system   kube-controller-manager-host11             1/1     Running
5          3d13h
kube-system   kube-proxy-6tzzg                           1/1     Running
3          3d13h
kube-system   kube-proxy-bcxjr                           1/1     Running
1          3d1h
kube-system   kube-proxy-tfml9                           1/1     Running
1          3d1h
kube-system   kube-scheduler-host11                      1/1     Running
5          3d13h
production    web                                        1/1     Running
0          14s
```

18. Lab6 Volumes

18.1 Volume

1. Créer un volume nommé **web** de type **HostPath**
2. Créer un pod basé sur l'image alpine
3. Monter le volume web dans le conteneur sur le chemin /web
4. Se connecter sur le pod et initialiser le fichier /web/index.html
5. Supprimer le pod web

18.2 Serveur Web

1. Créer un pod nommé web basé sur l'image httpd
2. Monter le volume web dans le conteneur sur le chemin htdocs de Apache

18.3 Service NodePort

1. Créer un service nommé web de type NodePort qui rassemble le pod web

18.4 Utilisation

1. Depuis le poste de travail, exécuter la commande "curl node1:30000"

19. Lab6 Volumes

19.1 Volume

1. Créer un pod basé sur l'image alpine
2. Créer un volume nommé **web** de type **HostPath**
3. Monter le volume web dans le conteneur sur le chemin /web

```
vi init-vol.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: init
spec:
  containers:
  - name: debug
    image: alpine:3.12
    command: ["sleep","3600"]
    volumeMounts:
    - name: data
      mountPath: /web
  volumes:
  - name: data
    hostPath:
      path: /tmp
```

```
kubectl apply -f init-vol.yml
```

19.1.1 Se connecter sur le pod et initialiser le fichier /web/index.html

```
kubectl exec -it init -- sh
# vi /web/index.html
Hello from Apache
# exit
```

19.1.2 Supprimer le pod web

```
kubectl delete po/init
```

19.2 Serveur Web

19.2.1 Créer un pod nommé web basé sur l'image httpd

```
vi web.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app3: web
spec:
  containers:
  - name: web
    image: nginx:1.17-alpine
    volumeMounts:
    - name: data
      mountPath: /usr/share/nginx/html
  volumes:
  - name: data
    hostPath:
      path: /tmp
```

19.3 Service NodePort

19.3.1 Créer un service nommé web de type NodePort qui rassemble le pod web

```
vi web.yml
```

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app3: web
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
```

19.4 Utilisation

19.4.1 Depuis le poste de travail, exécuter la commande "curl node1:30000"

20. Lab7 ConfigMap

20.1 ConfigMap de type Volume

1. Copier le fichier /etc/nginx/nginx.conf à partir d'un pod web existant
2. Modifier le fichier et créer un ConfigMap nommé nginx-conf
3. Créer un pod web avec le ConfigMap nginx-conf
4. Vérifier le fichier de conf /etc/nginx/nginx.conf du pod web

20.2 ConfigMap de type variable d'environnement

1. Créer un ConfigMap nommé mysql-pass qui contient une clé password et une valeur root
2. Créer un pod mysql et initialiser le ROOT_PASSWORD de mysql
3. Vérifier que le pod mysql est Running

21. Lab7 ConfigMap

21.1 ConfigMap de type Volume

21.1.1 Copier le fichier /etc/nginx/nginx.conf à partir d'un pod web existant

```
kubectl cp web:/etc/nginx/nginx.conf nginx.conf
```

21.1.2 Modifier le fichier et créer un ConfigMap nommé nginx-conf

```
kubectl create configmap nginx-conf --from-file=nginx.conf
```

21.1.3 Créer un pod web avec le ConfigMap nginx-conf

```
vi web.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app: web
spec:
  containers:
  - name: www
    image: nginx:1.17-alpine
    volumeMounts:
    - name: config
      mountPath: "/etc/nginx/nginx.conf"
      subPath: "nginx.conf"
  volumes:
  - name: config
    configMap:
      name: nginx-conf
```

21.1.4 Vérifier le fichier de conf /etc/nginx/nginx.conf du pod web

```
kubectl exec -it web -- sh
cat /etc/nginx/nginx.conf
```

21.2 ConfigMap de type variable d'environnement

21.2.1 Créer un ConfigMap nommé mysql-pass qui contient une clé password et une valeur root

```
vi cm-mysql.yml
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-pass
data:
  password: root
```

```
kubectl apply -f cm-mysql.yml
```

21.2.2 Créer un pod mysql et initialiser le ROOT_PASSWORD de mysql

```
vi mysql.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mysql
spec:
  containers:
  - image: mysql:5.6
    name: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      valueFrom:
        configMapKeyRef:
          name: mysql-pass
          key: password
```

```
kubectl apply -f mysql.yml
```

21.2.3 Vérifier que le pod mysql est Running

```
kubectl get pods
```