

Rendu Kubernetes



WINTZ ENZO

MSI-DevOps-A

Prérequis	3
Installation de kubectl	3
Installation de minikube.....	3
Démarrage avec minikube.....	3
Producteur.....	4
Création d'un daemonSet producteur basé sur une image alpine	4
Création d'un volume nommé web.....	4
Liaison du volume web sur le conteneur	5
Ecriture dans le fichier /web/index.html du nom du hostname + date toutes les 60 secondes	6
Consommateur.....	6
Création d'un déploiement web (avec 2 réplicas) basé sur httpd	6
Liaison du volume web sur le chemin htdocs du serveur Apache	7
Service NodePort.....	8
Création d'un service web de type NodePort rassemblant les pods du déploiement web.....	8
Utilisation	10
Exécution de la commande « curl node1 :30000 » toutes les minutes sur le poste de travail.....	10

Prérequis

Afin de pouvoir faire la démonstration suivante, il sera nécessaire d'installer chocolatey (car cela se fera sous Windows), pour ce faire, voir le site <https://chocolatey.org/install>.

De même, il sera nécessaire d'installer VirtualBox, on peut l'installer via le lien suivant : <https://www.virtualbox.org/>.

De plus, à des fins de compatibilité, nous nous baserons sur Minikube.

Installation de kubectl

Nous ouvrons une fenêtre PowerShell en administrateur et nous exécutons la commande suivante :

choco install kubernetes-cli

On renseigne Y pour que l'installation puisse poursuivre.

On vérifie l'installation avec **kubectl version**.

Installation de minikube

Pour l'installation, on fera **choco install minikube** :

```
PS C:\WINDOWS\system32> choco install minikube
Chocolatey v0.10.15
Installing the following packages:
minikube
By installing you accept licenses for the packages.
Progress: Downloading Minikube 1.18.1... 100%

Minikube v1.18.1 [Approved]
minikube package files install completed. Performing other installation steps.
ShimGen has successfully created a shim for minikube.exe
The install of minikube was successful.
Software install location not explicitly set, could be in package or
default install location if installer.

Chocolatey installed 1/1 packages.
See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).
```

On vérifie l'installation de minikube avec la commande **minikube version** :

```
PS C:\WINDOWS\system32> minikube version
minikube version: v1.18.1
commit: 09ee84d530de4a92f00f1c5dbc34cead092b95bc
```

Démarrage avec minikube

On prépare la machine virtuelle sous minikube avec la commande **minikube start** (il faudra cocher Oui deux fois pour que la machine virtuelle puisse se créer) :

```
PS C:\WINDOWS\system32> minikube start
* minikube v1.18.1 sur Microsoft Windows 10 Education 10.0.19042 Build 19042
* Choix automatique du pilote virtualbox
* Downloading VM boot image ...
  > minikube-v1.18.0.iso.sha256: 65 B / 65 B [-----] 100.00% ? p/s 0s
  > minikube-v1.18.0.iso: 212.99 MiB / 212.99 MiB [ ] 100.00% 30.77 MiB p/s 7s
* Démarrage du noeud de plan de contrôle minikube dans le cluster minikube
* Downloading Kubernetes v1.20.2 preload ...
  > preloaded-images-k8s-v9-v1...: 491.22 MiB / 491.22 MiB 100.00% 29.69 Mi
* Création de VM virtualbox (CPUs=2, Mémoire=2200MB, Disque=20000MB)...
* Préparation de Kubernetes v1.20.2 sur Docker 20.10.3...
  - Generating certificates and keys ...
  - Booting up control plane ...
  - Configuring RBAC rules ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v4
* Enabled addons: storage-provisioner, default-storageclass

! C:\Program Files\Docker\Docker\resources\bin\kubectl.exe est la version 1.18.8, qui peut comporter des incompatibilités avec Kubernetes 1.20.2.
  - Want kubectl v1.20.2? Try 'minikube kubectl -- get pods -A'
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

On peut vérifier que minikube vient d'apparaître dans VirtualBox :



Producteur

Création d'un daemonSet producteur basé sur une image alpine

Pour créer notre daemonSet, nous créerons le fichier **1.DaemonSet.yml** avec à l'intérieur de ce fichier :

```
'''
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: producteur-ds
spec:
  selector:
    matchLabels:
      producteur-ds: daemon
  template:
    metadata:
      labels:
        producteur-ds: daemon
    spec:
      containers:
        - name: daemon-set-prod-alp
          image: alpine:3.12
          command: ["sleep", "3600"]
'''
```

Afin de valider les changements, il faudra se positionner dans l'invite powershell à l'endroit où se situe le fichier YAML et exécuter la commande **kubectl apply -f 1.DaemonSet.yml** :

```
PS C:\Users\enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f 1.DaemonSet.yml
daemonset.apps/producteur-ds created
```

On peut vérifier que le daemonset est créé avec la commande **kubectl get ds,pods -o wide** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get ds,pods -o wide
NAME                                DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE SELECTOR  AGE  CONTAINERS  IMAGES  SELECTOR
daemonset.apps/producteur-ds       1         1         1         1             1          <none>         8s   daemon-set-prod-alp  alpine:3.12  producteur-ds=daemon

NAME                                READY  STATUS   RESTARTS  AGE  IP          NODE    NOMINATED NODE  READINESS GATES
pod/producteur-ds-dmvl5             1/1    Running   0          8s   172.17.0.3  minikube  <none>          <none>
```

Création d'un volume nommé web

Pour créer notre volume, nous allons créer un fichier qui se nommera **2.VolumeWeb.yml**, à l'intérieur, on y intégrera le code suivant :

```
'''
apiVersion: v1
kind: PersistentVolume
metadata:
  name: web-vol-pers
spec:
  storageClassName: manual
'''
```

```
capacity:
  storage: 1Gi
accessModes:
  - ReadWriteOnce
hostPath:
  path: /data/web
'''
```

Une fois le fichier enregistré, on appliquera ces paramètres avec la commande **kubectl apply -f .\2.VolumeWeb.yml** :

```
PS C:\Users\enzow\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f .\2.VolumeWeb.yml
persistentvolume/web-vol-pers created
```

Nous pouvons voir que notre volume a bien été créé avec la commande **kubectl get pv** :

```
PS C:\Users\enzow\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM          STORAGECLASS  REASON   AGE
web-vol-pers  1Gi       RWO           Retain          Available                manual                2m48s
```

Liaison du volume web sur le conteneur

Nous allons reprendre notre fichier **1.DaemonSet.yml** et rajouter le code (**en vert**) à la suite afin de reconfigurer notre container dans notre daemonset :

```
'''
apiVersion: apps/v1
[...]
containers:
  - name: daemon-set-prod-alp
    image: alpine:3.12
    command: ["sleep", "3600"]
    volumeMounts:
      - name: web-vol-pers
        mountPath: /web
    volumes:
      - name: web-vol-pers
        hostPath:
          path: /data/web
'''
```

```
PS C:\Users\enzow\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f .\1.DaemonSet.yml
daemonset.apps/producteur-ds configured
```

Pour vérifier que notre volume a bien été placé sur le container précédemment créé, nous allons nous connecter sur le pod en question grâce aux commandes **kubectl get pods** et **kubectl exec -it producteur-ds-vg6xd -- sh** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get pods
NAME                READY  STATUS   RESTARTS  AGE
producteur-ds-vg6xd  1/1    Running  0         19m
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl exec -it producteur-ds-vg6xd -- sh
/#
```

Nous sommes à présent sur notre container, pour voir si le dossier de montage « web » apparaît, on fait un **ls** :

```
/ # ls
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var  web
/ #
```

Nous voyons à présent que notre volume est bien monté sur notre container.

Ecriture dans le fichier `/web/index.html` du nom du hostname + date toutes les 60 secondes

On va reprendre notre fichier **1.DaemonSet.yml** et rajouter une ligne d'argument (**en rouge**) exécutant notre commande **kubectl apply -f .\1.DaemonSet.yml** (le nom du pod va changer, il faudra prendre en compte cette modification) :

```
'''
```

apiVersion: apps/v1

[...]

containers:

- name: daemon-set-prod-alp

image: alpine:3.12

command: ["/bin/sh"]

args: ["-c", "while true; do echo `hostname` - `date +%d%m` >> /web/index.html;sleep 60;done"]

[...]

```
'''
```

Pour vérifier l'incrémentation de notre fichier `index.html`, nous allons nous connecter sur notre container avec la commande **kubectl exec -it producteur-ds-b74m4 -- sh** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl exec -it producteur-ds-b74m4 -- sh
/ #
```

On se rend dans le dossier `web` avec la commande **cd web** :

```
/ # cd web/
/web #
```

Pour lire le fichier `index.html`, nous ferons un **cat index.html** et nous aurons le résultat suivant :

```
/web # cat index.html
producteur-ds-b74m4-2003
producteur-ds-b74m4-2003
producteur-ds-b74m4-2003
```

Nous avons bien le nom de notre hostname avec la date qui s'incrémente toutes les 60 secondes.

Consommateur

Création d'un déploiement web (avec 2 réplicas) basé sur httpd

Afin de créer notre déploiement, nous allons créer un fichier qui se nommera **3.Deploy.yml** et qui contiendra les informations suivantes :

```
'''
```

apiVersion: apps/v1

kind: Deployment

metadata:

name: web

spec:

replicas: 2

```

selector:
  matchLabels:
    deploy: apache
template:
  metadata:
    labels:
      deploy: apache
  spec:
    containers:
      - name: apache
        image: httpd : 2.4-alpine
'''

```

Une fois le fichier enregistré, nous allons faire la commande **kubectl apply -f .\3.Deploy.yml** :

```

PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f .\3.Deploy.yml
deployment.apps/web created

```

Pour voir notre déploiement, nous ferons la commande suivante : **kubectl get deploy** :

```

PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get deploy
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
web     2/2     2            2           10s

```

Nous pouvons également voir les pods avec leur réplica avec la commande **kubectl get pods** :

```

PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
producteur-ds-b74m4                1/1     Running   1          14h
web-5b8c7b9dcd-jmbxt               1/1     Running   0          17s
web-5b8c7b9dcd-s7b8v               1/1     Running   0          17s

```

Nos réplicas sont bien présents et sont en statuts « **RUNNING** ».

Liaison du volume web sur le chemin `htdocs` du serveur Apache

Pour ce faire, nous allons modifier **3.Deploy.yml** en ajoutant les lignes suivantes (**en bleu**) :

```

'''
apiVersion: apps/v1
[...]
containers:
  - name: apache
    image: httpd:2.4-alpine
    volumeMounts:
      - name: web-vol-pers
        mountPath: /usr/local/apache2/htdocs/web
    volumes:
      - name: web-vol-pers
        hostPath:
          path: /data/web
'''

```

Une fois le fichier YAML enregistré, nous allons faire la commande **kubectl apply -f .\3.Deploy.yml** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f .\3.Deploy.yml
deployment.apps/web configured
```

Notre déploiement se configure (à noter que les pods ont changés de nom afin de prendre en compte les nouveaux paramètres).

On peut vérifier que nos pods sont en statut « **RUNNING** » avec la commande **kubectl get pods** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
producteur-ds-b74m4                1/1     Running   1           17h
web-d5b766d45-fwn6s                1/1     Running   0           30s
web-d5b766d45-lf99l                1/1     Running   0           32s
```

Afin de vérifier si le volume a bien été monté sur le chemin **/usr/local/apache2/htdocs/web**, il faudra exécuter la commande **kubectl exec -it web-d5b766d45-fwn6s -- sh**

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl exec -it web-d5b766d45-fwn6s -- sh
/usr/local/apache2 #
```

Nous sommes à présent sur notre pod, nous allons faire un **ls** pour voir les dossiers et nous constatons la présence du **htdocs** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl exec -it web-d5b766d45-fwn6s -- sh
/usr/local/apache2 # ls
bin      build    cgi-bin  conf     error    htdocs   icons    include  logs     modules
/usr/local/apache2 #
```

Nous nous rendons dans ce dossier avec la commande **cd htdocs** et faisons un **ls** et nous voyons le dossier **web** qui s'est ajouté (le volume web s'est bien positionné sur notre pod) :

```
/usr/local/apache2 # cd htdocs/
/usr/local/apache2/htdocs # ls
index.html  web
```

Cette vérification pourra se faire avec le second pod **web-d5b766d45-lf99l** en reproduisant les mêmes commandes citées précédemment.

Service NodePort

Création d'un service web de type NodePort rassemblant les pods du déploiement web

Nous allons créer un fichier qui se nommera **4.NodePort.yml** avec le contenu suivant à l'intérieur :

'''

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    deploy: apache
  type: NodePort
  ports:
    - port: 80
```


targetPort: 80

nodePort: 30000

'''

Une fois enregistré, nous allons faire la commande **kubectl apply -f .\4.NodePort.yml** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl apply -f .\4.NodePort.yml
service/web-service created
```

Nous pouvons voir que notre **web-services** est créé avec la commande **kubectl get services** :

```
PS C:\Users\Enzo\OneDrive\Documents\K8S\TP_RENDU_FINAL> kubectl get services
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP          20h
web-service    NodePort      10.97.11.87   <none>         80:30000/TCP     5s
```

Pour vérifier que nos pods ont bien été ajoutés à notre service de type **NodePort**, nous allons tout d'abord faire un **kubectl describe pod <NomDuPod>** sur chaque pods, soit les commandes :

- **kubectl describe pod web-d5b766d45-lf99l**
- **kubectl describe pod web-d5b766d45-fwn6s**

Pour la commande **kubectl describe pod web-d5b766d45-lf99l**, dans le champ « **IP** », nous avons l'adresse suivante : **172.17.0.4**

```
Priority: 0
Node: minikube/192.168.99.102
Start Time: Sun, 21 Mar 2021 14:19:36 +0100
Labels: deploy=apache
        pod-template-hash=d5b766d45
Annotations: <none>
Status: Running
IP: 172.17.0.4
IPs:
  IP: 172.17.0.4
Controlled By: ReplicaSet/web-d5b766d45
Containers:
  apache:
    Container ID: docker://21af1bac6ca05a60b8a2644838b52cd21eed23af80cc17bdd6530292d7d77135
    Image: httpd:2.4-alpine
    Image ID: docker-pullable://httpd@sha256:8c16a28de3e8a715c613bc84868f8ccb984eca1027800f18bfc7a0fab377f475
    Port: <none>
    Host Port: <none>
    State: Running
      Started: Sun, 21 Mar 2021 14:19:38 +0100
    Ready: True
    Restart Count: 0
    Environment: <none>
    Mounts:
      /usr/local/apache2/htdocs/web from web-vol-pers (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-8zjj7 (ro)
Conditions:
```

Pour la commande **kubectl describe pod web-d5b766d45-fwn6s**, nous avons dans le champ « **IP** » l'adresse suivante : **172.17.0.5**

```

Node: minikube/192.168.99.102
Start Time: Sun, 21 Mar 2021 14:19:38 +0100
Labels:
  deploy=apache
  pod-template-hash=d5b766d45
Annotations: <none>
Status: Running
IP: 172.17.0.5
IPs:
  IP: 172.17.0.5
Controlled By: ReplicaSet/web-d5b766d45
Containers:
  apache:
    Container ID: docker://28ddc2d88179c0dee33dc059648f5a4a73196a76d29a1856ac1be47a485e3936
    Image: httpd:2.4-alpine
    Image ID: docker-pullable://httpd@sha256:8c16a28de3e8a715c613bc84868f8ccb984eca1027800f18bfc7a0fab377f475
    Port: <none>
    Host Port: <none>
    State: Running
      Started: Sun, 21 Mar 2021 14:19:40 +0100
    Ready: True
    Restart Count: 0
    Environment: <none>
    Mounts:
      /usr/local/apache2/htdocs/web from web-vol-pers (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-8zjj7 (ro)

```

Pour vérifier que nos pods sont bien intégrés au NodePort créé précédemment, nous allons exécuter la commande **kubectl describe service web-service** et nous verrons dans le champ « **Endpoints** » les deux adresses des pods (**172.17.0.4:80,172.17.0.5:80**) qui sont mappés sur le port http 80 :

```

IP: 10.97.11.87
IPs: 10.97.11.87
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 30000/TCP
Endpoints: 172.17.0.4:80,172.17.0.5:80
Session Affinity: None
External Traffic Policy: Cluster

```

Nous avons bien notre service de type NodePort qui regroupe nos deux pods de déploiement web.

Utilisation

Exécution de la commande « **curl node1 :30000** » toutes les minutes sur le poste de travail

Nous allons tout d'abord devoir forwarder notre service NodePort, pour obtenir nos services, on exécutera la commande **kubectl get svc** :

```

PS C:\Windows\system32> kubectl get svc
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP     10.96.0.1     <none>         443/TCP          21h
web-service    NodePort      10.97.11.87   <none>         80:30000/TCP     59m

```

Pour que notre NodePort soit accessible depuis notre poste de travail, nous allons faire la commande **kubectl port-forward service/web-service 30000:80** afin que le service soit accessible depuis l'adresse IP **127.0.0.1:30000** :

```

PS C:\Windows\system32> kubectl port-forward service/web-service 30000:80
Forwarding from 127.0.0.1:30000 -> 80
Forwarding from [::1]:30000 -> 80

```

Notre service est à l'écoute et attend qu'on l'interroge.

Afin d'avoir la commande **curl 127.0.0.1:30000** qui s'exécute toutes les minutes, nous allons faire un script powershell en .ps1 qui se nommera **CurlRequest.ps1** qui contiendra le code suivant :

```

'''
$InterServ = curl 127.0.0.1:30000
#Boucle infinie qui se répète toute les 60 secondes grâce au sleep
while ($InterServ)
{
    curl 127.0.0.1:30000
    sleep 60
}
'''

```

Le but du script étant qu'il vérifie si le **\$InterServ** est **True** et tant que c'est le cas, il exécute le code. Une fois exécuté, nous avons comme résultat ce qu'il y a sur la capture suivante :

```

StatusCode      : 200
StatusDescription : OK
Content          : <html><body><h1>It works!</h1></body></html>

RawContent      : HTTP/1.1 200 OK
                  Accept-Ranges: bytes
                  Content-Length: 45
                  Content-Type: text/html
                  Date: Sun, 21 Mar 2021 14:59:03 GMT
                  ETag: "2d-432a5e4a73a80"
                  Last-Modified: Mon, 11 Jun 2007 18:53:14 GMT
                  Server:...
Forms            : {}
Headers          : {[Accept-Ranges, bytes], [Content-Length, 45], [Content-Type, text/html], [Date, Sun, 21 Mar 2021 14:59:03 GMT]...}
Images           : {}
InputFields      : {}
Links            : {}
ParsedHtml       : mshtml.HTMLDocumentClass
RawContentLength : 45

```

Et nous pourrions constater que sur la fenêtre où nous avons exécuté la commande **kubectl port-forward service/web-service 30000:80**, à chaque nouvelle tentative d'interaction avec notre service NodePort, celui-ci a bien reçu le curl qui lui était destiné :

```

PS C:\Windows\system32> kubectl port-forward service/web-service 30000:80
Forwarding from 127.0.0.1:30000 -> 80
Forwarding from [::1]:30000 -> 80
Handling connection for 30000
Handling connection for 30000
Handling connection for 30000
Handling connection for 30000
Handling connection for 30000
Handling connection for 30000

```