

Chapitre 5 : Déploiement d'applications avec Kubernetes

Docker et Kubernetes

CCI Strasbourg

Chapitre 5 : Déploiement d'applications avec Kubernetes

Objectifs

En suivant ce chapitre, nous allons aborder Kubernetes avec les modules suivants :

- 5.1 Définition de pods et services avec des fichiers YAML
- 5.2 Gestion des configurations avec ConfigMaps et Secrets
- 5.3 Déploiements et mise à jour des applications
- 5.4 Utilisation de Helm pour gérer des packages

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1

Définition de pods et services avec des fichiers YAML

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML

Sommaire :

- Les fichiers YAML et Kubernetes
- Définition des pods avec YAML
- Définition des services avec YAML
- Manipulation Kubernetes avec YAML (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML

Les fichiers YAML et Kubernetes



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Les fichiers YAML avec Kubernetes

Les fichiers YAML

Les fichiers YAML sont des fichiers structurés avec des tabulations (uniquement avec des espaces), ils permettent de définir des configurations en précisant notamment :

- des entiers,
- des chaînes de caractères,
- des listes.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Les fichiers YAML avec Kubernetes

Les fichiers YAML dans Kubernetes

Lors du chapitre précédent, nous avons interagit avec Kubernetes en utilisant directement des commandes kubectl.

Pour rappel, Kubernetes fonctionne en déclarant ses besoins, afin de pouvoir réutiliser facilement des opérations et de définir une application réutilisable, il est possible d'utiliser des fichiers YAML pour définir une application.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Les fichiers YAML avec Kubernetes

Les fichiers YAML dans Kubernetes

La commande **kubectl apply -f fichier.yaml** va permettre de d'appliquer une configuration à Kubernetes.

- La commande **kubectl delete -f fichier.yaml** va permettre de supprimer une configuration à Kubernetes.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML

Définition des pods avec YAML



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des pods avec YAML

Exemple simple

Ce fichier basique donne un exemple pour la déclaration d'un pod avec :

- **apiVersion** : indique la version de l'API Kubernetes à utiliser
- **kind** : Spécifie le type d'objet à traiter, ici « Pod »
- **metadata** : définit des meta-données sur le Pod
 - **name** : Nom du pod
 - **labels** : étiquettes que l'on attribue au Pod pour pouvoir le sélectionner plus tard.
- **spec** : donne les spécifications du pod
 - **containers** : liste les conteneurs à l'intérieur du pod et leurs configurations
 - **name** : nom du conteneur
 - **image** : image à utiliser
 - **ports** :
 - **containerPort** : port à exposer du conteneur

```
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
  labels:
    app: example
spec:
  containers:
    - name: mon-conteneur
      image: mon-image:version
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des pods avec YAML

Exemple simple

Ce fichier basique donne un exemple pour la déclaration d'un pod

```
apiVersion: v1
kind: Pod
metadata:
  name: mon-pod
  labels:
    app: example
spec:
  containers:
  - name: mon-conteneur
    image: mon-image:version
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
  labels:
    app: app-nginx
spec:
  containers:
  - name: ctn-nginx
    image: nginx
    ports:
    - containerPort: 80
```

`kubectl apply -f nginx_basique_pod.yml`

`kubectl describe pod pod-nginx`

```
Name: pod-nginx
Namespace: cci
Priority: 0
Service Account: default
Node: docker-desktop/192.168.65.3
Start Time: Fri, 15 Dec 2023 15:46:45 +0100
Labels: app=app-nginx
Annotations: <none>
Status: Running
IP: 10.1.0.36
IPs:
  IP: 10.1.0.36
Containers:
  ctn-nginx:
    Container ID: docker://c5de327971b780aeaf4c5a2a3b21cd7232db2306
    Image: nginx
    Image ID: docker-pullable://nginx@sha256:10d1f5b58f74683ad3
    Port: 80/TCP
    Host Port: 0/TCP
    State: Running
      Started: Fri, 15 Dec 2023 15:46:47 +0100
    Ready: True
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des pods avec YAML


Exemple simple

Pour tester le Pod, nous pouvons utiliser un pod « utilitaire » qui sera dans le cluster et qui va pouvoir atteindre l'adresse du pod crée précédemment. Puis à l'intérieur, réaliser une commande **curl**.

kubectrl run mycurlpod --image=curlimages/curl -i --tty -- sh

La commande **attach** va permettre de revenir sur le pod quand la session sera arrêtée :

kubectrl attach mycurlpod -c mycurlpod -i -t



```
~ $ curl http://10.1.0.36
<!DOCTYPE html>
<html>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des pods avec YAML

Autres paramètres optionnels

env : spécifie des variables d'environnements

securityContext : permet de préciser l'utilisateur qui va être utilisé

volumes : gestion des volumes

```
spec:
  containers:
  - name: mon-conteneur
    env:
    - name: MA_VARIABLE_ENV
      value: "valeur"
    securityContext:
      runAsUser: 1000
  volumes:
  - name: mon-volume
    emptyDir: {}
  containers:
  - name: mon-conteneur
    volumeMounts:
    - name: mon-volume
      mountPath: "/chemin/dans/le/conteneur"
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML

Définition des services avec YAML



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple

Ce fichier basique donne un exemple pour la déclaration d'un service avec :

- **apiVersion** : indique la version de l'API Kubernetes à utiliser
- **kind** : Spécifie le type d'objet à traiter, ici « Service »
- **metadata** : définit des meta-données sur le service
 - **name** : Nom du service
- **spec** : donne les spécifications du service
 - **selector**: sélectionne les pods en fonction du label qui a été défini sur les pods
 - **app**: le nom à filtrer dans les labels
 - **ports** :
 - **protocol**: protocole à utiliser (TCP, UDP, etc.)
 - **port** : Port sur lequel le service écoute
 - **targetPort** : Port sur lequel les pods répondent
 - **type** : valeurs possibles « ClusterIP », « NodePort », « LoadBalancer » ou « ExternalName »

```
apiVersion: v1
kind: Service
metadata:
  name: mon-service
spec:
  selector:
    app: mon-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple

Ce fichier basique donne un exemple pour la déclaration d'un service en reprenant le pod précédent.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx
  labels:
    app: app-nginx
spec:
  containers:
  - name: ctn-nginx
    image: nginx
    ports:
    - containerPort: 80
```

```
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
spec:
  selector:
    app: app-nginx
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

```
kubectl apply -f nginx_basique_service.yml
```

```
kubectl describe service service-nginx
```

```
Name: service-nginx
Namespace: cci
Labels: <none>
Annotations: <none>
Selector: app=app-nginx
Type: ClusterIP
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.98.219.240
IPs: 10.98.219.240
Port: <unset> 80/TCP
TargetPort: 80/TCP
Endpoints: 10.1.0.36:80
Session Affinity: None
Events: <none>
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple

Test du service :

curl sur le service

```
~ $ curl http://10.98.219.240
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
```

curl sur le pod

```
~ $ curl http://10.1.0.36
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple

Test du service après recréation du pod :

curl sur le service

```
~ $ curl http://10.98.219.240
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
```

curl sur le pod

```
~ $ curl http://10.1.0.38
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
```

- On observe que l'IP du service n'a pas changé, mais celui du Pod est bien différent !


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Autres paramètres optionnels

Nous pouvons ajouter des annotations pour décrire un peu mieux le service :

```
apiVersion: v1
kind: Service
metadata:
  name: service-nginx
  annotations:
    description: "Service pour mon application nginx"
spec:
  selector:
    app: app-nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```



```
Name: service-nginx
Namespace: cci
Labels: <none>
Annotations: description: Service pour mon application nginx
Selector: app=app-nginx
Type: ClusterIP
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.101.228.199
IPs: 10.101.228.199
Port: <unset> 80/TCP
TargetPort: 80/TCP
Endpoints: 10.1.0.38:80
Session Affinity: None
Events: <none>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple avec un type NodePort

```
apiVersion: v1
kind: Service
metadata:
  name: service-nginx-nodeport
spec:
  # On indique le type Nodeport
  type: NodePort
  selector:
    app: app-nginx
  ports:
    # 3 types de ports
    # nodePort - un port static qui sera assigné à chaque noeud du cluster
    # port - port exposé en interne du cluster
    # targetPort - port du conteneur
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

```
Name: service-nginx-nodeport
Namespace: cci
Labels: <none>
Annotations: <none>
Selector: app=app-nginx
Type: NodePort
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.108.137.156
IPs: 10.108.137.156
LoadBalancer Ingress: localhost
Port: <unset> 8080/TCP
TargetPort: 80/TCP
NodePort: <unset> 30163/TCP
Endpoints: 10.1.0.38:80
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Exemple simple avec un type NodePort

```
Name: service-nginx-nodeport
Namespace: cci
Labels: <none>
Annotations: <none>
Selector: app=app-nginx
Type: NodePort
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.108.137.156
IPs: 10.108.137.156
LoadBalancer Ingress: localhost
Port: <unset> 8080/TCP
TargetPort: 80/TCP
NodePort: <unset> 30163/TCP
Endpoints: 10.1.0.38:80
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

127.0.0.1:30163

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

```
~ $ curl http://10.108.137.156:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML

Manipulation (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition des services avec YAML

Manipulation Kubernetes avec YAML (TP)

- Suivre le TP fournit par le formateur.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.1 – Définition de pods et services avec des fichiers YAML



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2

Gestion des configurations avec ConfigMaps et Secrets

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets

Sommaire :

- Présentation et utilisation de ConfigMap
- Présentation et utilisation de Secrets
- Manipulation ConfigMaps et Secrets (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets

Présentation et utilisation de ConfigMaps



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Présentation de ConfigMap

ConfigMap est un élément de Kubernetes qui permet de stocker des configurations sous la forme de paires clé-valeur. Ces données seront stockées dans la base **etcd** de K8s.

ConfigMap offre une séparation entre les configurations et les applications, cela permet de :

- partager des configurations entre plusieurs pods simplement
- centraliser et faciliter la gestion dans une base

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Présentation de ConfigMap : stockage de valeurs

```
kubectl create configmap myvalues  
--from-literal=key1=value1 --from-literal=key2=value2
```



Cluster Kubernetes



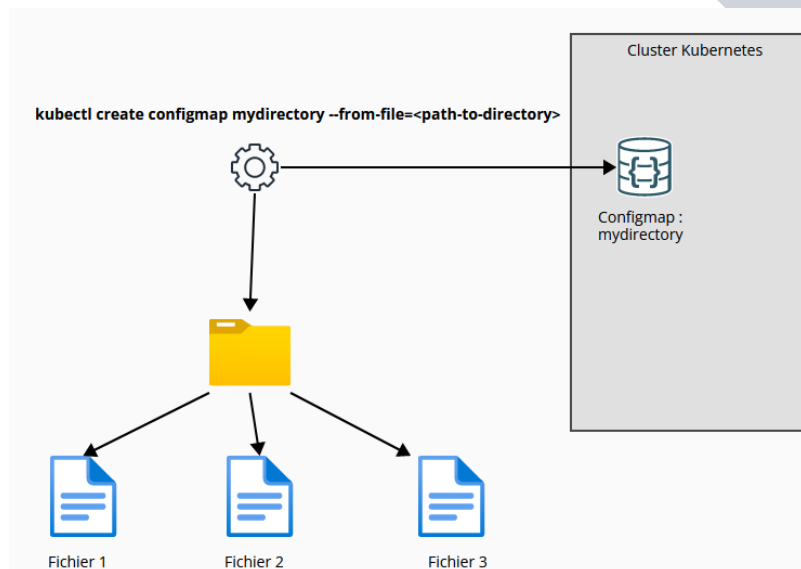
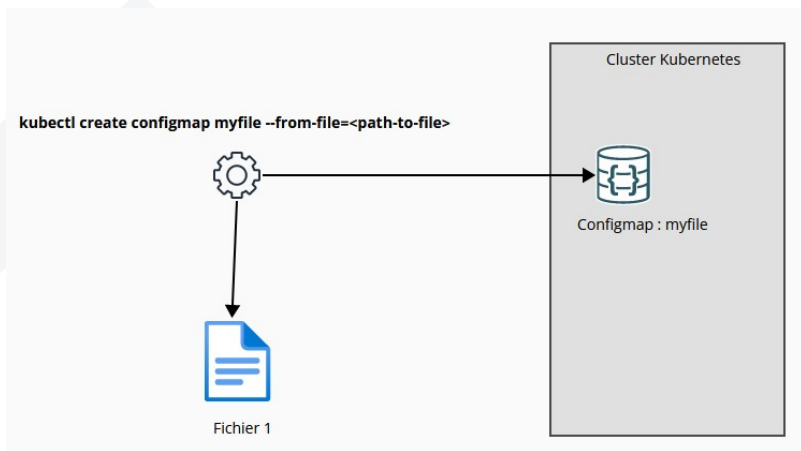
Configmap :
myvalues

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Présentation de ConfigMap : stockage de fichiers

Ces 2 illustrations montrent qu'il est possible de **stocker des fichiers dans le cluster via configMap** en utilisant la commande **kubectl create configmap**.

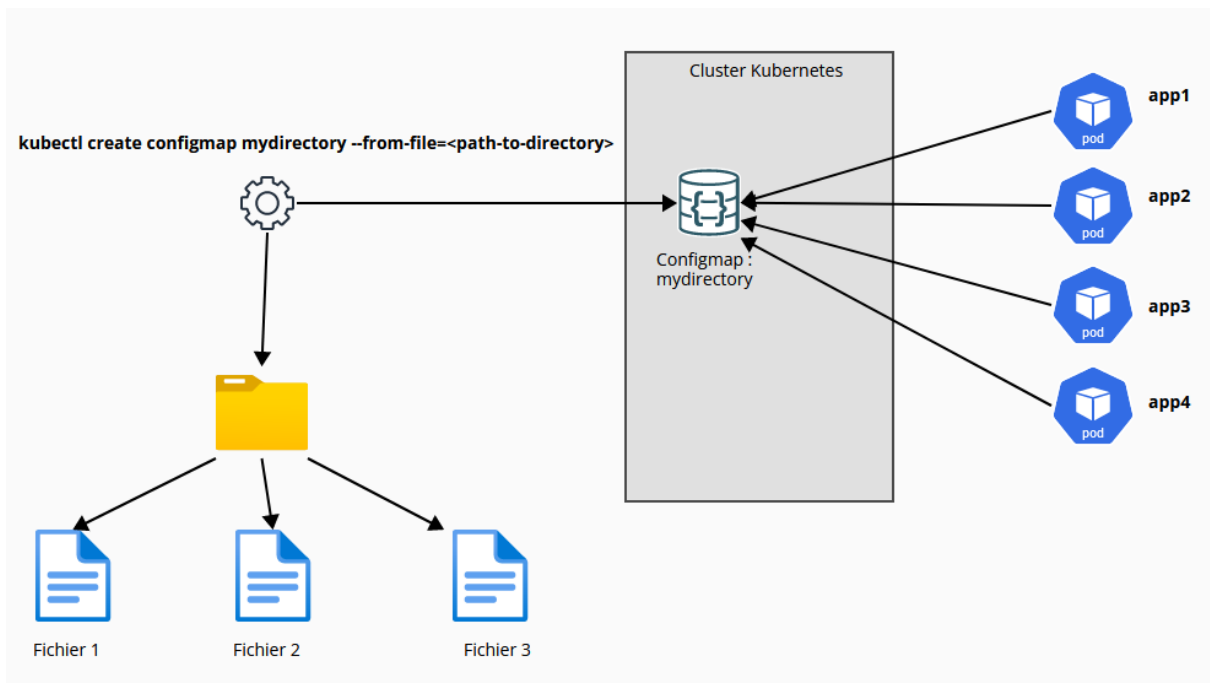


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Intérêts de ConfigMap

ConfigMap va apporter une factorisation et une réutilisation possible des données pour des applications (pods) différents.



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

ConfigMap avec kubectl

Il est possible de gérer les ConfigMaps directement avec l'outil kubectl (ou avec des fichiers YAML que l'on abordera plus tard).

- **kubectl create configmap <configmap-name> --from-file=<path-to-file-or-directory>** : créer un configMap à partir d'un fichier ou d'un répertoire (tous les fichiers du répertoire seront intégrés dans le configMap)
- **kubectl create configmap <configmap-name> --from-env-file=<file_env>** : créer un configmap depuis un fichier env
- **kubectl create configmap <configmap-name> --from-literal=<key1>=<value1> --from-literal=<key2>=<value2>** : créer un configMap depuis les valeurs passées de la ligne de commande (utile pour définir des valeurs de configuration, comme l'adresse d'un serveur par exemple)
- **kubectl get configmap <configmap-name>** : affiche les informations d'un configMap
- **kubectl describe configmap <configmap-name>** : affiche le contenu d'un configMap
- **kubectl delete configmap <configmap-name>** : supprime un config map
- **kubectl get configmaps <configmap-name> -o yaml** : exporte un configMap en fichier YAML

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps


ConfigMap avec kubectl – exemple de création

1/ Création d'un configMap pour des données de configuration

```
kubectl create configmap mailserveur --from-literal=server=cci-mail-server.local
```

2/ Afficher les configMaps disponibles :


```
kubectl get configmaps
```



NAME	DATA	AGE
kube-root-ca.crt	1	4d21h
mailserveur	1	20s

3/ Afficher le contenu du configMap

```
kubectl describe configmap mailserveur
```



```
Name:      mailserveur
Namespace: cci
Labels:    <none>
Annotations: <none>

Data
====
server:
----
cci-mail-server.local

BinaryData
=====

Events: <none>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps


ConfigMap avec kubectl – exemple de création avec fichier

1/ Création d'un configMap pour un fichier index.html

```
kubectl create configmap htmlfile --from-file=index.html
```

2/ Afficher le contenu du configMap

```
kubectl describe configmap htmlfile
```



```
Data
====
index.html:
----
<html>\r
<body>\r
Cocou depuis mon serveur\r
</body>\r
</html>

BinaryData
=====

Events:  <none>
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps


ConfigMap création avec des fichiers YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prenom
data:
  # Définissez vos données de configuration ici
  prenom: "Vincent"
```

```
kubectl apply -f configmap_prenom.yml
```



```
kubectl describe configmap prenom
```



```
Data
====
prenom:
----
Vincent

BinaryData
=====

Events: <none>
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

ConfigMap - édition

Il y a plusieurs moyens d'éditer un configMap existant :

- En utilisant la commande **kubectl edit configmap <votre_configmap>** pour l'éditer à la main
- La commande **kubectl create configmap <configmap name> --from-file <file/directory> -o yaml --dry-run=client | kubectl apply -f -** pour mettre à jour un fichier ou répertoire

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de ConfigMap

Il y a différents types d'utilisation des ConfigMaps, on peut citer principalement :

- **Récupérer des valeurs** depuis un configMap pour les utiliser dans des manifestes YAML (comme des variables d'environnement par exemple)
- Monter des volumes avec configMap pour **déployer des fichiers de configurations** ou du contenu dans les pods

Afin que les modifications prennent effets sur les pods, il faut les redémarrer.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de ConfigMap : exemple

Exemple pour intégrer des fichiers dans les pods via les configMaps

En reprenant le configmap crée précédemment qui contient un fichier index.html, nous pouvons l'inclure dans la déclaration d'un pod nginx via un manifeste YAML :

- **volumeMounts** : indique le nom du volume qui va être utilisé (via **name**) et le chemin où il va être monté (**mountPath**)
- **volumes** : définit le volume
 - **name** : le nom du volume définit
 - **configMap** :
 - **name** : le nom du config map qui est utilisé

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: config-volume
      mountPath: /usr/share/nginx/html
  volumes:
  - name: config-volume
    configMap:
      name: htmlfile
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de ConfigMap : exemple

Exemple pour intégrer des fichiers dans les pods via les configMaps

Nous aurions pu définir l'exemple précédent dans un seul et unique fichier YAML :

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  index.html: |
    <html>
      <body>
        <h1>Coucou depuis mon serveur</h1>
      </body>
    </html>
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: config-volume
          mountPath: /usr/share/nginx/html
  volumes:
    - name: config-volume
      configMap:
        name: nginx-config
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de ConfigMap : exemple

Exemple pour définir des valeurs dans les pods via les configMaps

L'exemple suivant montre l'utilisation d'une variable d'environnement qui est définie dans un configMap.

- **command** : lance le shell pour executer des instructions pour afficher les variables d'environnements
- **args** : les arguments à passer au shell (dans ce cas utiliser la commande env et faire un grep dessus)
- **valueFrom** : va rechercher la valeur depuis une source
 - **configMapKeyRef** :
 - **name** : nom du configMap à utiliser
 - **key** : clé à utiliser pour rechercher la valeur

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-prenom
spec:
  containers:
  - name: ctn-prenom
    image: debian
    command: [ "/bin/sh", "-c" ]
    args:
    - env | grep PRENOM
    env:
    - name: ENV_PRENOM
      valueFrom:
        configMapKeyRef:
          name: prenom
          key: prenom
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de ConfigMap : exemple

Exemple pour définir des valeurs dans les pods via les configMaps

Il est possible d'importer toutes les valeurs d'un configMap en tant que variables d'environnement via l'instruction **envFrom**.

- Remarque, en intégrant les valeurs de cette manière, les clés des configMaps seront utilisées comme les noms des variables d'environnements

```
apiVersion: v1
kind: Pod
metadata:
  name: dapi-test-pod
spec:
  containers:
    - name: test-container
      image: registry.k8s.io/busybox
      command: [ "/bin/sh", "-c", "env" ]
      envFrom:
        - configMapRef:
            name: prenom
  restartPolicy: Never
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets

Présentation et utilisation de Secrets



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Présentation de Secrets

Les Secrets ont un fonctionnement très similaires aux configMaps, la différence est la possibilité de rendre opaque les valeurs pour les protéger.

Les Secrets vont donc être utilisés pour des données plus sensibles comme définir des mots de passes par exemple.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Type de Secrets

En terme d'utilisation, les secrets disposent de plusieurs types (liste non exhaustive) :

- **Opaque** : le type par défaut qui est utilisé pour le stockage de données utilisateurs (mot de passe, tokens d'API, etc.). Ces données sont encodées en base64 (attention, elles ne sont **pas** chiffrées).
- **kubernetes.io/ssh-auth** : permet de stocker des informations en utilisant une paire de clé SSH
- **kubernetes.io/tls** : spécialement conçu pour stocker des certificats et clés privées associées.
- **kubernetes.io/service-account-token** : permet d'enregistrer un token d'authentification pour un ServiceAccount (utilisateur d'un Pod).

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Création de Secrets Opaque

Nous allons aborder dans ce chapitre le cas le plus courant pour comprendre les Secrets avec le type **Opaque**.

- **kubectl create secret generic <secretname> --from-literal=<key>=<value>**
 - **generic** : définit un secret de type **Opaque**

```
kubectl create secret generic mes-acces \  
--from-literal=username=MonUtilisateur \  
--from-literal=password=TOTO
```

kubectl describe secret mes-acces

```
Type: Opaque  
Data  
====  
password: 4 bytes  
username: 14 bytes
```

- On observe que les valeurs ne sont pas affichées

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Création de Secrets Opaque avec YAML

Nous pouvons reprendre la création de l'exemple précédent avec un fichier YAML :

```
apiVersion: v1
kind: Secret
metadata:
  name: mes-acces
type: Opaque
data:
  username: TW9uVXRpbG1zYXRldXI=
  password: VE9UTw==
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Présentation et utilisation de ConfigMaps

Utilisation de Secrets Opaque dans les Pods : exemple

Tout comme les configMaps, les secrets peuvent être utilisés pour des valeurs et fichiers. Dans ce cas, nous illustrons le passage de valeurs pour des variables d'environnements :

- **secretKeyRef** : précise que nous allons utiliser un Secret en indiquant le nom du Secret et la clé que nous allons utiliser

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-env-pod
spec:
  containers:
  - name: ctn-acces
    image: debian
    command: [ "/bin/sh", "-c" ]
    args:
    - env | grep SECRET_
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: mes-acces
          key: username
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
          name: mes-acces
          key: password
  restartPolicy: Never
```

kubectl logs secret-env-pod

SECRET_PASSWORD=TOTO
SECRET_USERNAME=MonUtilisateur

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets

Manipulation ConfigMaps et Secrets (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets

Manipulation ConfigMap et Secrets (TP)

- Suivre le TP fournit par le formateur.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.2 – Gestion des configurations avec ConfigMaps et Secrets



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3

Déploiements et mise à jour des applications

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Sommaire :

- Rappel sur les déploiements
- Création d'un déploiement
- Mise à jour d'une application avec Deployment
- Gérer un déploiement (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Rappel sur les déploiements



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Rappel sur les déploiements

Les déploiements

Un déploiement est un **type d'objet** dans Kubernetes qui permet de gérer des applications dans un cluster.

Le déploiement assure :

- La déclaration et la **mise à jour** des applications
- De pouvoir effectuer des **montées de versions sans rupture de service**
- **La mise à l'échelle** (permettant la stabilité de l'application et la gestion de la charge) via les **replicaSets**
- Le retour arrière en cas de problème

Sur un cluster productif, il est usuel d'utiliser des déploiements pour profiter de ces avantages plutôt que déclarer des « pods manuellement ».

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Création d'un déploiement



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Rappel sur les déploiements

Création d'un déploiement

Un déploiement est un type d'objet dans Kubernetes qui permet de gérer des applications dans un cluster.

- **apiVersion** : **apps/v1**, précise à Kubernetes que l'on souhaite utiliser ce type d'API (cf documentation)
- **kind** : précise le type « Deployment »
- **spec** : spécifications du déploiement
 - **replicas** : le nombre de replicas souhaité
 - **selector** : comme les services, précise la correspondance pour intégrer les pods dans le déploiement qui dispose de ce label
- **template** : indique les spécifications des pods à intégrer

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-nginx
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-nginx
  template:
    metadata:
      labels:
        app: mon-nginx
    spec:
      containers:
        - name: mon-ctn-nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Rappel sur les déploiements

Création d'un déploiement

Lors de l'exécution de ce déploiement, nous allons observer la création de plusieurs pods :

NAME	READY	STATUS	RESTARTS	AGE
dapi-test-pod	0/1	Completed	0	40h
mes-nginx-6c847bbc7b-chzp4	1/1	Running	0	8s
mes-nginx-6c847bbc7b-f64jk	1/1	Running	0	8s
mes-nginx-6c847bbc7b-qqfwq	1/1	Running	0	8s
mes-nginx-6c847bbc7b-s8fjp	1/1	Running	0	8s
mes-nginx-6c847bbc7b-skhsj	1/1	Running	0	8s
mes-nginx-6c847bbc7b-vzk9t	1/1	Running	0	8s
mes-nginx-6c847bbc7b-xt8wc	1/1	Running	0	8s

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Mise à jour d'une application avec Deployment



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Les mises à jours

Il existe plusieurs types de mise à jour avec les déploiements :

rolling update

- La méthode de mise à jour la plus fine possible, Kubernetes va effectuer mise à jour incrémentale des pods :
 - Création d'un pod v2 → lancement du pod v2 → suppression d'un pod v1
 - Ce cycle sera répété jusqu'à ce qu'il n'y ai plus de v1
 - Ceci est un exemple de stratégie de rolling update « standard » mais il est possible de faire différemment.
- Il s'agit du comportement par défaut de Kubernetes

recreate

- Kubernetes supprime toutes les versions d'un pod et effectue une recréation avec une v2

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple : rolling update par défaut

- Pour simuler une mise à jour d'applications, nous allons reprendre l'exemple d'un chapitre précédent de Docker et effectuer un déploiement initial avec la version 1.0, puis nous allons le rendre accessible à travers un service de type **NodePort**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  template:
    metadata:
      labels:
        app: mon-php
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:1.0
          ports:
            - containerPort: 80
```



```
apiVersion: v1
kind: Service
metadata:
  name: service-mon-php
spec:
  # On indique le type Nodeport
  type: NodePort
  selector:
    app: mon-php
  ports:
    # 3 types de ports
    # nodePort - un port statique
    # port - port exposé en interne
    # targetPort - port du container
    - nodePort: 30999
      port: 8080
      targetPort: 80
```



Votre prenom est CCI

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple : rolling update par défaut

- Maintenant, nous passons depuis la version 1.0 vers la version 2.0 (c'est d'ailleurs l'unique modification du fichier)
- **Pour rappel**, nous n'avons pas besoin de modifier le service car il fournit un moyen stable pour accéder à l'application.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  template:
    metadata:
      labels:
        app: mon-php
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:2.0
          ports:
            - containerPort: 80
```



Votre prenom est NOUVELLE_VERSION

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple : rolling update par défaut

- Avec la commande **kubectl get all**, nous pouvons observer que notre déploiement de manière standard à terminer les pods de manière progressive pour ensuite les recréer :

```
pod/mes-sites-php-7fc6f77d97-dwk74    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-frbpd    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-hc8x5    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-pssn8    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-q62g9    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-rhwk7    1/1    Terminating    0    9m3s
pod/mes-sites-php-7fc6f77d97-v875n    1/1    Terminating    0    9m3s
pod/mes-sites-php-dc4df8c86-7slqr     1/1    Running         0    10s
pod/mes-sites-php-dc4df8c86-9kxh9     1/1    Running         0    10s
pod/mes-sites-php-dc4df8c86-cnm9j     1/1    Running         0    9s
pod/mes-sites-php-dc4df8c86-g2vg2     1/1    Running         0    9s
pod/mes-sites-php-dc4df8c86-p6tz6     1/1    Running         0    11s
pod/mes-sites-php-dc4df8c86-ss5sr     1/1    Running         0    11s
pod/mes-sites-php-dc4df8c86-ww2qg     1/1    Running         0    11s
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple avec une stratégie recreate

- En précisant une **stratégie de type Recreate**, Kubernetes va arrêter tous les pods et recréer des pods à la bonne version (attention dans ce cas au downtime)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mon-php
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:1.0
          ports:
            - containerPort: 80
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple avec une stratégie rolling update personnalisée

- Avec un rolling update personnalisée, il est possible de contrôler de manière plus fine le déploiement au lieu de laisser Kubernetes gérer cette partie de manière automatique.
- strategy :**
 - type** : RollingUpdate, on définit la stratégie de manière explicite
 - rollingUpdate** : on précise le comportement du rollingUpdate
 - maxSurge** : le nombre supplémentaires de pods que K8s peut créer
 - maxUnavailable** : le nombre de pods down max autorisé
champs facultatifs :
 - minReadySeconds** : spécifie le nombre minimum de secondes où K8s attend qu'un Pod nouvellement créé est prêt sans plantage de ses conteneurs, il sera ainsi déclaré comme disponible et K8s pourra procéder à la suite du déploiement
 - progressDeadlineSeconds** : permet d'indiquer à partir de quand on considère que le déploiement est au point mort (si des conteneurs n'arrivent pas à se créer par exemple)
 - revisionHistoryLimit** : définit le nombre de replicaSets à stocker en historique pour pouvoir effectuer une restauration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      # Nombre de pods supplémentaires possibles par rapports aux repl
      # dans ce cas on pourrait donc aller jusqu'à 8 pods
      maxSurge: 1
      # Nombre de pods down possibles
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:1.0
          ports:
            - containerPort: 80
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Gérer le déploiement et son versionning

Kubernetes fournit une commande rollout qui permet d'interagir avec le déploiement.

Voici les cas d'utilisations de cette commande :

kubectl rollout status deploy <nomDéploiement> : permet de consulter l'état du déploiement

kubectl rollout pause deploy <nomDéploiement> : permet de mettre en pause le déploiement

kubectl rollout resume deploy <nomDéploiement> : permet de reprendre le déploiement

kubectl rollout history deploy <nomDéploiement> : affiche l'historique du déploiement

kubectl rollout history deploy <nomDéploiement> --revision=1 : affiche l'historique d'une révision spécifique

kubectl rollout undo deploy <nomDéploiement> : effectue un retour arrière sur la version précédente

kubectl rollout undo deploy <nomDéploiement> --to-revision=1 : effectue un retour arrière vers la révision 1

Nous pouvons faire un parallèle avec un système de gestion de versions comme GIT qui permet de versionner des éléments et de revenir vers un état stable.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Gérer le déploiement et son versionning

Les annotations vont permettre de spécifier des commentaires sur un déploiement :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version initiale"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdlilens/site-php-image:1.0
          ports:
            - containerPort: 80
```

kubectl describe deploy mes-sites-php

```
CreationTimestamp:    Mon, 18 Dec 2023 13:25:45 +0100
Labels:               <none>
Annotations:          deployment.kubernetes.io/revision: 1
Selector:             app=mon-php
Replicas:             7 desired | 7 updated | 7 total | 7 available | 0 unavailable
StrategyType:         RollingUpdate
MinReadySeconds:      0
RollingUpdateStrategy: 0 max unavailable, 1 max surge
Pod Template:
  Labels:  app=mon-php
  Annotations:  kubernetes.io/change-cause: version initiale
  Containers:
    mon-ctn-php:
      Image:  vdlilens/site-php-image:1.0
      Port:  80/TCP
      Host Port:  0/TCP
      Environment:  <none>
      Mounts:       <none>
      Volumes:      <none>
  Conditions:
    Type           Status    Reason
    ----           -
    Available      True     MinimumReplicasAvailable
    Progressing    True     NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   mes-sites-php-5fdc7b55c6 (7/7 replicas created)
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Gérer le déploiement et son versionning

Les annotations vont permettre de spécifier des commentaires sur un déploiement :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version initiale"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdlilens/site-php-image:1.0
          ports:
            - containerPort: 80
```

kubectl describe deploy mes-sites-php

```
CreationTimestamp:    Mon, 18 Dec 2023 13:25:45 +0100
Labels:               <none>
Annotations:          deployment.kubernetes.io/revision: 1
Selector:              app=mon-php
Replicas:              7 desired | 7 updated | 7 total | 7 available | 0 unavailable
StrategyType:          RollingUpdate
MinReadySeconds:       0
RollingUpdateStrategy: 0 max unavailable, 1 max surge
Pod Template:
  Labels:  app=mon-php
  Annotations:  kubernetes.io/change-cause: version initiale
  Containers:
    mon-ctn-php:
      Image:          vdlilens/site-php-image:1.0
      Port:           80/TCP
      Host Port:      0/TCP
      Environment:    <none>
      Mounts:         <none>
      Volumes:        <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Available      True   MinimumReplicasAvailable
    Progressing    True   NewReplicaSetAvailable
OldReplicaSets:    <none>
NewReplicaSet:     mes-sites-php-5fdc7b55c6 (7/7 replicas created)
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple sur un cas pratique

Sur ce cas pratique, nous allons reprendre l'exemple précédent et le passer vers la version 2 :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version initiale"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:1.0
          ports:
            - containerPort: 80
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version 2"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:2.0
          ports:
            - containerPort: 80
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple sur un cas pratique

Sur ce cas pratique, nous allons reprendre l'exemple précédent et le passer vers la version 2 :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version initiale"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:1.0
          ports:
            - containerPort: 80
```



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mes-sites-php
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mon-php
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: mon-php
      annotations:
        kubernetes.io/change-cause: "version 2"
    spec:
      containers:
        - name: mon-ctn-php
          image: vdillens/site-php-image:2.0
          ports:
            - containerPort: 80
```


Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Mise à jour d'une application avec Déploiement

Exemple sur un cas pratique

Suite à la migration de v1 vers v2, une commande history va nous afficher ce résultat :

```
kubectl rollout history deploy mes-sites-php
```




REVISION	CHANGE-CAUSE
1	version initiale
2	version 2


On observe bien que nous sommes sur la version 2

Si on procède à un retour arrière via la commande undo :

```
kubectl rollout undo deploy mes-sites-php
```



```
kubectl rollout history deploy mes-sites-php
```



REVISION	CHANGE-CAUSE
2	version 2
3	version initiale

On observe que l'historique a changé et que la révision 3 est repassée sur la version initiale. Un describe va confirmer ce point :

```
CreationTimestamp: Mon, 18 Dec 2023 13:25:45 +0100
Labels:           <none>
Annotations:      deployment.kubernetes.io/revision: 3
Selector:         app=mon-php
Replicas:         7 desired | 7 updated | 7 total | 7 available | 0 unavailable
StrategyType:     RollingUpdate
MinReadySeconds:  0
RollingUpdateStrategy: 0 max unavailable, 1 max surge
Pod Template:
  Labels:  app=mon-php
  Annotations:  kubernetes.io/change-cause: version initiale
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Gérer un déploiement (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications

Gérer un déploiement (TP)

- Suivre le TP fournit par le formateur.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.3 – Déploiements et mise à jour des applications



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4

Utilisation de Helm pour gérer les packages

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

Sommaire :

- Présentation de Helm
- Installation de Helm
- Utilisation de Helm
- Manipulation avec Helm (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

Présentation de Helm



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm - présentation

Helm (traduction : gouvernail pour pouvoir piloter Kubernetes) est un gestionnaire de packages pour Kubernetes afin de simplifier la distribution et la gestion des applications dans un cluster K8s.

Dans le principe, on pourrait donc comparer Helm à un gestionnaire comme Apt pour Linux ou encore l'Appstore / Playstore pour les smartphones Apple / Androids.

Helm va définir tous ces éléments dans des configurations que l'on appelle package Helm ou encore « **charts** ». Un chart Helm va contenir les données nécessaires pour déployer une application (pods, services ,etc...).

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm - présentation

Comme nous l'avons vu précédemment, il est possible dans K8s de déclarer des ressources (pods, services, etc.) en ligne de commandes ou dans des fichiers manifestes YAML. Ces fichiers YAML permettent de sauvegarder une configuration et de pouvoir la réutiliser mais ce n'est pas idéal pour la distribution.

Comment rendre paramétrable facilement l'application?

Comment livrer l'application de manière cohérente et intégrée?

Comment gérer le versionning de l'application avec ces fichiers?

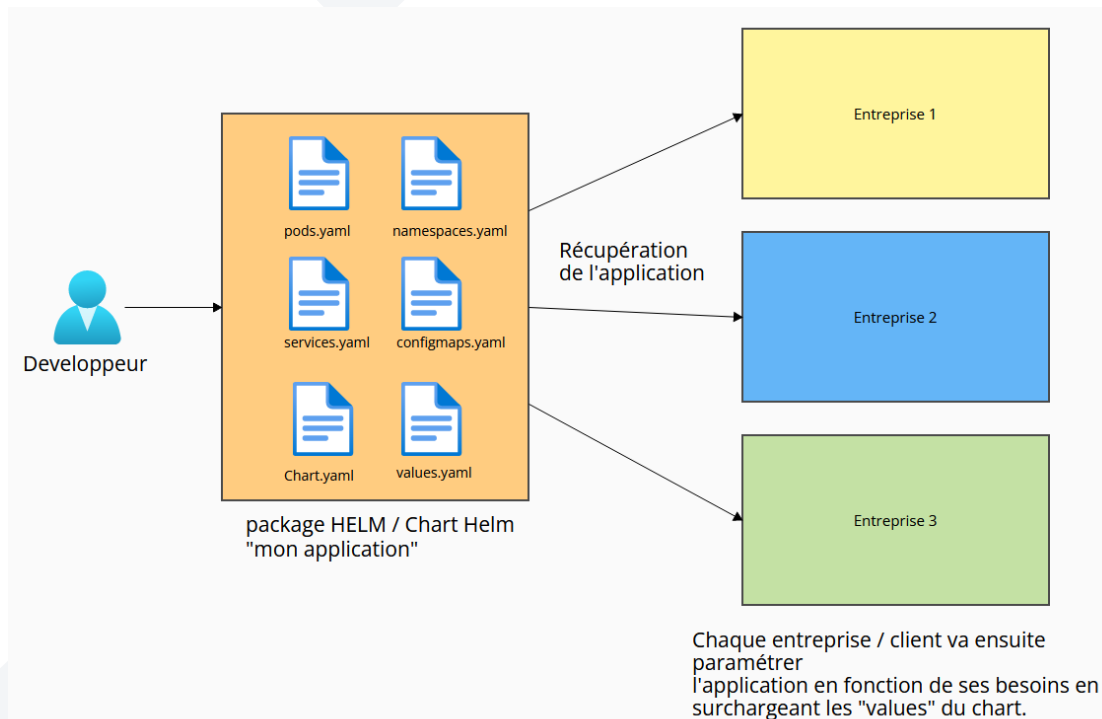
Helm va répondre à chacun de ces points.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm - présentation

L'illustration ci-dessous résume le fonctionnement de Helm.



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – les dépôts

Les charts sont stockés dans des dépôts et leur gestion se fait à travers un client « helm » disponible sur IOS, Linux et Windows. Les charts sont livrés sous forme d'archives (extension .tgz).

Le dépôt public officiel est **artifact HUB** (<https://artifacthub.io>). Ce dépôt va contenir des meta-données sur le package Helm, des informations pour faciliter l'utilisation du package. Il va également contenir un lien vers un dépôt GIT où les fichiers versionnés seront sauvegardés.

A noter, il est possible d'ajouter des dépôts supplémentaires (public ou privée) pour diffuser des charts.

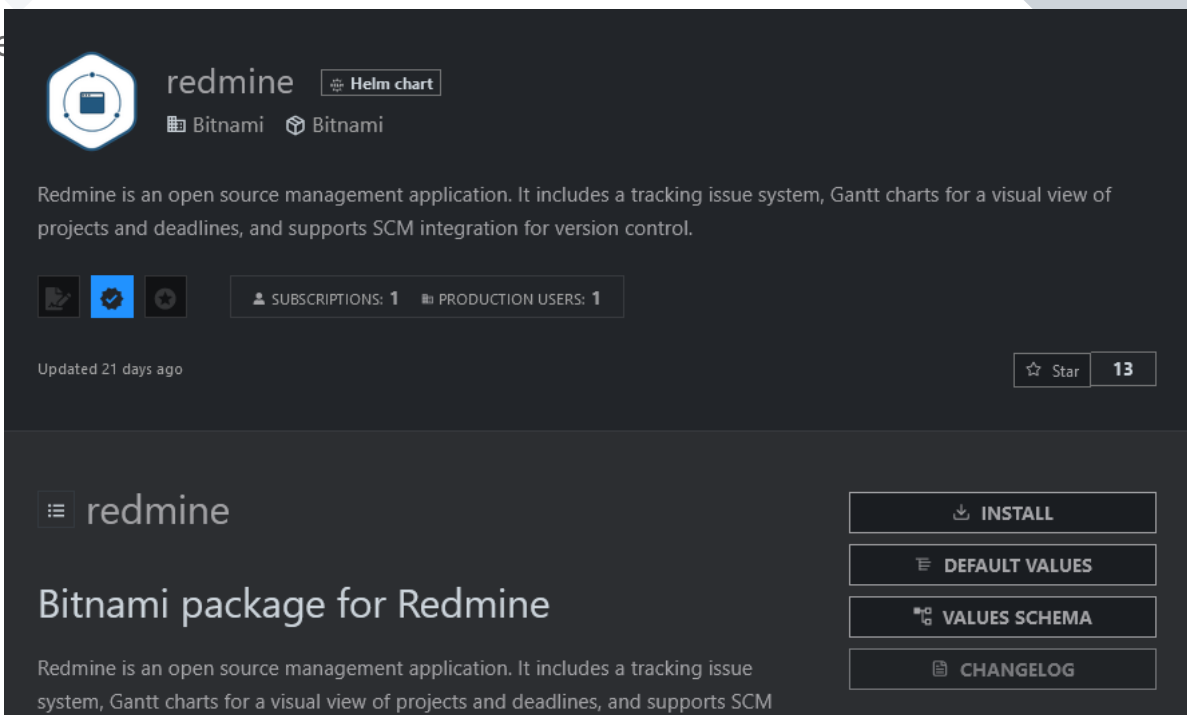
Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – les dépôts : artifactHub

L'image ci-dessous montre la page de présentation de l'application « redmine ».

- **Install** : procédure pour installer le package
- **Default values** : les valeurs par défaut qui sont définies dans le package et qui permettent de retrouver les clés à surcharger pour déployer le package
- **Values schema** : une documentation sur les valeurs qui sont définies



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – les dépôts privés

Un dépôt privé techniquement de manière simplifié est un serveur HTTP qui héberge un fichier **index.yaml** et des charts (sous forme de fichier avec l'extension .tgz).

Le fichier **index.yaml** va présenter un référentiel de tous les charts du dépôt. Il va également définir les meta-données des charts.

- Exemple partiel du contenu **index.yaml** :

```
apiVersion: v1
entries:
  site-php:
    - apiVersion: v2
      appVersion: "2.0"
      created: "2023-12-19T11:41:31.2811987+01:00"
      description: Mon site php
      digest: 9e2f5acdca4628cd7a4dbc3813859e1411aef803a9eb8f49d259ce6d52d821c4
      home: www.vdillenschneider.fr
      keywords:
        - php
        - prenom
        - application
      maintainers:
        - email: dillenschneider.v@gmail.com
          name: Vincent DILLENSCHNEIDER
      name: site-php
      urls:
        - https://vdillens.github.io/site-php-helm/site-php-1.0.tgz
      version: "1.0"
generated: "2023-12-19T11:41:31.2737374+01:00"
```

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – structure d'une chart

La structure d'une chart se compose de la manière suivante :

- **Chart.yaml** : définit la description de votre chart en précisant les meta-données (version, description, mainteneur, etc.)
- **charts** : si votre chart dépend d'autres charts, elles sont configurées dans ce dossier (en indiquant les valeurs par défaut, les versions à utiliser, etc...). Ce dossier est donc optionnel
- **crds** : (Custom Resource Definitions)
- **templates** : ce sont des Go templates permettant d'incorporer du dynamisme via des variables dans les fichiers de configuration YAML
- **values.schema.json** : définit les règles pour valider les variables et sert également de documentation
- **values.yaml** : valeurs par défaut

▼ SITE-PHP-CHART

▼ charts

▼ crds

▼ templates

≡ Notes.txt

! site_php_deploiement.yaml

! site_php_service_nodeport.yaml

! Chart.yaml

🔑 LICENSE

📘 README.MD

{ } values.schema.json

! values.yaml

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – quelques remarques

- **Attention aux versions** ! Un chart dispose d'une version et il peut embarquer une application qui dispose d'un numéro de version différent :

NAME	CHART VERSION	APP VERSION	DESCRIPTION
vdillens/site-php	1.0	2.0	Mon site php

- Utiliser la notation .yaml plutôt que .yml pour les extensions de fichiers (helm est un peu capricieux sur ce point).
- On parle aussi de la notion de **Release** dans Helm, il s'agit d'une instance de chart qui s'exécute sur un cluster K8s. Par exemple, nous pourrions avoir plusieurs instances d'une base de données MariaDB sur un cluster, chacune de ces instances est une release du chart MariaDB avec à chaque fois un nom de release différent.

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Présentation de Helm

Helm – les avantages

Pour terminer cette présentation de Helm, voici ces avantages :

- **Simplifier les déploiements** : avec une commande Helm il est possible d'installer et de configurer une nouvelle application dans son cluster K8s
- **Des packages qui sont vérifiés** : tout comme pour Docker avec des images officielles, il existe des versions officielles des packages Helm qui garantissent le bon fonctionnement de l'application et sa sécurité.
- **Les packages sont personnalisables** : en utilisant la surcharge des valeurs, il est possible d'adapter l'application à son environnement

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

Installation de Helm



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Installation de Helm

Helm – Installation

Les charts sont stockés dans des dépôts et leur gestion se fait à travers un client « helm » disponible sur IOS, Linux et Windows.

Pour son installation, en fonction de votre environnement il existe plusieurs moyens d'installation, on pourra citer par exemple depuis :

- Votre repository si vous utilisez Linux
- Télécharger le binaire pour Windows
- Ou utiliser brew pour macOS

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

Utilisation de Helm



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm

Helm – commandes de base

helm version : vérifier la version du client

Gestion des dépôts

helm repo list : liste les dépôts configurés sur le poste local

helm repo add <nomDepot> <UrlDepot> : ajoute un dépôt

helm repo remove <nomDepot> : supprime un dépôt sur le poste

helm repo update : mets à jour la base local avec les données des dépôts distants (similaire à apt-get update)

helm repo search <nom> : recherche le nom d'une application dans vos dépôts configurés

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm

Helm – commandes de base

Gestion des charts

helm install <nomRelease> <nomPackage> : installe un chart avec un nom de release

helm uninstall <nomRelease> : Supprime la release

helm list : Liste les releases qui sont déployées

helm upgrade : Mets à jour une release existante (pour modifier une configuration par exemple)

helm rollback <nomRelease> <version> : revenir à une version ultérieure de la release

helm history <nomRelease> : affiche l'historique de la release (maximum 256 révisions par défaut)

Création de chart :

helm create <nomChart> : Créer un chart initial avec un squelette par défaut

helm lint <cheminChart> : Vérifie le contenu du chart et indique les éventuels erreurs de syntaxe.

helm package <nomChart> : Créer le package .tgz

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm


Helm – exemple avec l'application Redmine

Installation du chart : <https://artifacthub.io/packages/helm/bitnami/redmine>

helm repo add bitnami <https://charts.bitnami.com/bitnami>

helm install my-redmine bitnami/redmine --version 25.0.7 --set redminePassword=toto --set service.ports.http=8090

kubectl get all



NAME	READY	STATUS	RESTARTS	AGE
pod/my-redmine-5596c544dc-mvxw4	1/1	Running	0	90s
pod/my-redmine-mariadb-0	1/1	Running	0	90s
pod/my-redmine-postgresql-0	1/1	Running	0	90s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	38m
service/my-redmine	LoadBalancer	10.103.72.31	localhost	8090:32738/TCP	90s
service/my-redmine-mariadb	ClusterIP	10.102.212.42	<none>	3306/TCP	90s
service/my-redmine-postgresql	ClusterIP	10.97.213.73	<none>	5432/TCP	90s
service/my-redmine-postgresql-hl	ClusterIP	None	<none>	5432/TCP	90s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/my-redmine	1/1	1	1	90s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/my-redmine-5596c544dc	1	1	1	90s

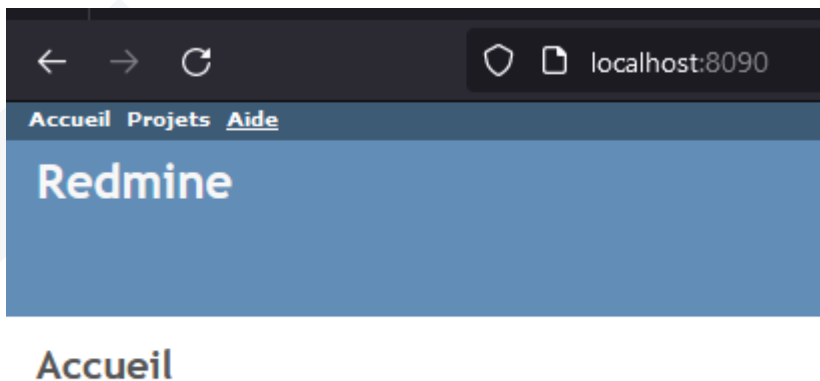
NAME	READY	AGE
statefulset.apps/my-redmine-mariadb	1/1	90s
statefulset.apps/my-redmine-postgresql	1/1	90s

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm

Helm – exemple avec l'application Redmine

L'installation de notre package a fonctionné et nous pouvons maintenant utiliser l'application sur le port 8090 :



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm

Helm – exemple avec l'application Redmine

Maintenant, sur cette application, nous souhaitons modifier le le port de l'application :

helm upgrade my-redmine bitnami/redmine --set service.ports.http=80

→ cela ne fonctionne pas !

En effet, le système indique qu'il a besoin des mots de passes pour modifier les données qui sont hébergées dans les bases de données pour mettre à jour la release

```
Error: UPGRADE FAILED: execution error at (redmine/templates/NOTES.txt:77:4):
PASSWORDS ERROR: You must provide your current passwords when upgrading the release.
Note that even after reinstallation, old credentials may be needed as they may be kept in persistent volume claims.
Further information can be obtained at https://docs.bitnami.com/general/how-to/troubleshoot-helm-chart-issues/#credential-errors-while-upgrading-chart-releases

'mariadb.auth.rootPassword' must not be empty, please add '--set mariadb.auth.rootPassword=$MARIADB_ROOT_PASSWORD' to the command. To get the current value:
    export MARIADB_ROOT_PASSWORD=$(kubectl get secret --namespace "default" my-redmine-mariadb -o jsonpath="{.data.mariadb-root-password}" | base64 -d)
'mariadb.auth.password' must not be empty, please add '--set mariadb.auth.password=$MARIADB_PASSWORD' to the command. To get the current value:
    export MARIADB_PASSWORD=$(kubectl get secret --namespace "default" my-redmine-mariadb -o jsonpath="{.data.mariadb-password}" | base64 -d)
```

Après avoir récupéré les 2 mots de passes mentionnés, cette commande va fonctionner :

helm upgrade my-redmine bitnami/redmine --set service.ports.http=80 --set mariadb.auth.rootPassword=<mdp1> --set mariadb.auth.password=<mdp2>

Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

Manipulation avec Helm (TP)



Chapitre 5 : Déploiement d'applications avec Kubernetes

Module 5.4 – Utilisation de Helm pour gérer les packages

