Correction TD3 : Manipulation kubectl

Commandes Kubernetes essentielles

IUT Grand Ouest Normandie – BUT Informatique S5

Année 2024/2025

Table des matières

1 Introduction

Ce document présente la correction complète du TD3 sur la manipulation de kubectl, l'outil en ligne de commande pour interagir avec Kubernetes. Ce TD couvre les commandes essentielles utilisées quotidiennement en entreprise.

2 Concepts de base

2.1 Question 1 : Créer un namespace et un pod nginx

Énoncé : Créer un namespace appelé 'monnamespace' et un pod avec une image nginx appelé nginx sur ce namespace.

```
# Creer le namespace
kubectl create namespace monnamespace

# Creer le pod nginx dans ce namespace
kubectl run nginx --image=nginx --namespace=monnamespace

# Verifier
kubectl get pods -n monnamespace
```

2.2 Question 2 : Créer le pod en YAML

Énoncé : Créer le pod qui vient d'être décrit en utilisant YAML (`-o yaml`).

```
# Generer le YAML sans creer le pod
kubectl run nginx --image=nginx \
  --namespace=monnamespace \
  --dry-run=client -o yaml > nginx-pod.yaml

# Contenu du fichier nginx-pod.yaml genere :
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: monnamespace
spec:
  containers:
  - image: nginx
    name: nginx
```

```
# Creer le pod a partir du fichier YAML
kubectl apply -f nginx-pod.yaml
```

2.3 Question 3 : Pod busybox avec commande env

Énoncé : Créez un pod busybox qui exécute la commande "env" et voyez la sortie.

```
# Methode 1 : Executer et voir la sortie immediatement
kubectl run busybox --image=busybox --restart=Never \
  --rm -it -- env

# Methode 2 : Creer le pod et voir les logs
kubectl run busybox --image=busybox --restart=Never \
```

```
  -- env

# Voir la sortie dans les logs
kubectl logs busybox

# Nettoyer
kubectl delete pod busybox
```

2.4 Question 4 : YAML d'un namespace sans le créer

Énoncé : Obtenir le YAML pour un nouveau namespace appelé 'monns' sans le créer.

```
# Generer le YAML sans creer
kubectl create namespace monns --dry-run=client -o yaml

# Resultat :
```

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

```
# Sauvegarder dans un fichier
kubectl create namespace monns --dry-run=client -o yaml > monns.yaml
```

2.5 Question 5 : YAML d'un ResourceQuota

Énoncé : Créer le YAML pour un ResourceQuota appelé 'monrq' avec 1 CPU, 1G mémoire et 2 pods.

```
# Generer le YAML de base
kubectl create quota monrq \
  --hard=cpu=1,memory=1G,pods=2 \
  --dry-run=client -o yaml

# Resultat :
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: monrq
spec:
  hard:
    cpu: "1"
    memory: 1G
    pods: "2"
```

```
# Sauvegarder
kubectl create quota monrq \
  --hard=cpu=1,memory=1G,pods=2 \
  --dry-run=client -o yaml > monrq.yaml

# Appliquer dans un namespace specifique
kubectl apply -f monrq.yaml -n monnamespace
```

2.6 Question 6 : Afficher pods sur tous les namespaces

Énoncé : Afficher les pods sur tous les namespaces.

```
# Option 1 : Avec --all-namespaces
kubectl get pods --all-namespaces

# Option 2 : Avec -A (raccourci)
kubectl get pods -A

# Avec plus de details
kubectl get pods -A -o wide

# Trier par namespace
kubectl get pods -A --sort-by=.metadata.namespace
```

2.7 Question 7 : Pod nginx exposant le port 80

Énoncé : Créer un pod nginx et exposer le trafic sur le port 80.

```
# Creer le pod avec le port expose
kubectl run nginx --image=nginx --port=80

# Verifier
kubectl get pod nginx

# Voir les details du port
kubectl describe pod nginx | grep Port
```

Note : Cette commande expose le port dans la définition du pod, mais ne crée pas de Service. Pour exposer réellement le pod, il faut créer un Service.

2.8 Question 8 : Changer l'image du pod

Énoncé : Changer l'image du pod en nginx :1.7.1 et observer le redémarrage.

```
# Methode 1 : Avec kubectl set image
kubectl set image pod/nginx nginx=nginx:1.7.1

# Observer le pod en temps reel
kubectl get pod nginx --watch

# Methode 2 : Editer directement
kubectl edit pod nginx
# Changer "image: nginx" en "image: nginx:1.7.1"
# Sauvegarder et quitter

# Verifier l'image utilisee
kubectl describe pod nginx | grep Image

# Voir les events
kubectl get events --field-selector involvedObject.name=nginx
```

2.9 Question 9 : Tester le pod avec wget

Énoncé : Obtenir l'IP du pod nginx et utiliser busybox pour wget son '/'.

```
# Obtenir l'IP du pod nginx
NGINX_IP=$(kubectl get pod nginx -o jsonpath='{.status.podIP}')
echo $NGINX_IP

# Ou directement :
kubectl get pod nginx -o wide

# Utiliser un pod busybox temporaire pour tester
kubectl run busybox --image=busybox --rm -it --restart=Never \
  -- wget -O- $NGINX_IP:80

# Alternative sans variable
kubectl run busybox --image=busybox --rm -it --restart=Never \
  -- wget -O- $(kubectl get pod nginx -o jsonpath='{.status.podIP}'):80
```

2.10 Question 10 : Informations détaillées du pod

Énoncé : Obtenir des informations détaillées sur le pod nginx.

```
# Describe pour voir tous les details
kubectl describe pod nginx

# Voir les logs
kubectl logs nginx

# Voir les events lies au pod
kubectl get events --field-selector involvedObject.name=nginx

# Status detaille en YAML
kubectl get pod nginx -o yaml

# Status detaille en JSON
kubectl get pod nginx -o json

# Voir seulement les problemes potentiels
kubectl describe pod nginx | grep -A 5 "Events:"
```

3 Pods Multi-Containers

3.1 Question 1 : Pod avec deux conteneurs

Énoncé : Créez un Pod avec deux conteneurs busybox, se connecter au deuxième et exécuter 'ls'.

```
# Creer le fichier YAML
cat > multi-container-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: multi-container
spec:
  containers:
  - name: container1
    image: busybox
    command: ["sh", "-c", "echo hello ; sleep 3600"]
  - name: container2
```

```
    image: busybox
    command: ["sh", "-c", "echo hello ; sleep 3600"]
EOF

# Appliquer
kubectl apply -f multi-container-pod.yaml

# Verifier
kubectl get pod multi-container

# Se connecter au deuxieme conteneur
kubectl exec -it multi-container -c container2 -- sh

# Dans le shell du conteneur :
ls
exit

# Ou directement :
kubectl exec multi-container -c container2 -- ls
```

4 Conception de pods

4.1 Labels et annotations

4.1.1 Question 1 : Créer 3 pods avec label app=v1

```
# Creer les 3 pods
kubectl run nginx1 --image=nginx --labels="app=v1"
kubectl run nginx2 --image=nginx --labels="app=v1"
kubectl run nginx3 --image=nginx --labels="app=v1"

# Verifier
kubectl get pods --show-labels
```

4.1.2 Question 2 : Afficher tous les labels

```
# Voir tous les labels
kubectl get pods --show-labels

# Voir dans un format plus lisible
kubectl get pods -o custom-columns=NAME:.metadata.name,LABELS:.metadata.
    labels

# Alternative avec wide
kubectl get pods -o wide --show-labels
```

4.1.3 Question 3 : Modifier le label de nginx2

```
# Methode 1 : Avec kubectl label (overwrite)
kubectl label pod nginx2 app=v2 --overwrite

# Verifier
kubectl get pod nginx2 --show-labels
```

4.1.4 Question 4 : Afficher la colonne APP

```
# Afficher le label app dans une colonne
kubectl get pods -L app

# Ou avec custom-columns
kubectl get pods -o custom-columns=NAME:.metadata.name,APP:.metadata.
    labels.app
```

4.1.5 Question 5 : Afficher uniquement app=v2

```
# Filtrer par label
kubectl get pods -l app=v2

# Avec details
kubectl get pods -l app=v2 --show-labels

# Avec output wide
kubectl get pods -l app=v2 -o wide
```

4.1.6 Question 6 : Ajouter label tier=web

```
# Ajouter tier=web aux pods app=v2
kubectl label pods -l app=v2 tier=web

# Ajouter tier=web aux pods app=v1
kubectl label pods -l app=v1 tier=web

# Ou en une commande pour app=v2 OU app=v1
kubectl label pods -l 'app in (v1,v2)' tier=web

# Verifier
kubectl get pods --show-labels
```

4.1.7 Question 7 : Ajouter annotation owner

```
# Ajouter l'annotation aux pods app=v2
kubectl annotate pods -l app=v2 owner=marketing

# Verifier les annotations
kubectl describe pods -l app=v2 | grep Annotations

# Voir toutes les annotations
kubectl get pods -l app=v2 -o yaml | grep -A 5 annotations
```

4.2 Placement du Pod

4.2.1 Question 1 : Pod sur nœud avec label GPU

Énoncé : Créer un pod déployé sur un nœud avec label accelerator=nvidia-tesla-p100.

```
# D'abord, labelliser un noeud (si necessaire)
kubectl label nodes <node-name> accelerator=nvidia-tesla-p100

```

```
# Creer le fichier YAML
cat > gpu-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  nodeSelector:
    accelerator: nvidia-tesla-p100
  containers:
  - name: cuda-app
    image: nvidia/cuda:11.0-base
EOF

# Appliquer
kubectl apply -f gpu-pod.yaml

# Verifier le placement
kubectl get pod gpu-pod -o wide
```

4.2.2 Question 2 : Pod sur control-plane

Énoncé : Créer un pod placé sur le nœud control-plane avec nodeSelector et tolerations.

```
# Voir les taints du control-plane
kubectl describe node <control-plane-name> | grep Taint

# Creer le fichier YAML
cat > control-plane-pod.yaml <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: control-plane-pod
spec:
  nodeSelector:
    node-role.kubernetes.io/control-plane: ""
  tolerations:
  - key: node-role.kubernetes.io/control-plane
    operator: Exists
    effect: NoSchedule
  containers:
  - name: nginx
    image: nginx
EOF

# Appliquer
kubectl apply -f control-plane-pod.yaml

# Verifier le placement
kubectl get pod control-plane-pod -o wide
```

5 Déploiement

5.1 Question 1 : Créer un déploiement

Énoncé : Créer un déploiement nginx :1.18.0 avec 2 replicas.

```
# Creer le deploiement
kubectl create deployment nginx \
  --image=nginx:1.18.0 \
  --replicas=2 \
  --port=80

# Verifier
kubectl get deployment nginx
kubectl get pods -l app=nginx
```

5.2 Question 2 : Afficher le YAML du déploiement

```
# Afficher le YAML
kubectl get deployment nginx -o yaml

# Sauvegarder dans un fichier
kubectl get deployment nginx -o yaml > nginx-deployment.yaml

# Voir seulement la spec
kubectl get deployment nginx -o yaml | grep -A 30 "spec:"
```

5.3 Question 3 : YAML des ReplicaSets

```
# Lister les ReplicaSets
kubectl get replicaset -l app=nginx

# Afficher le YAML de tous les ReplicaSets
kubectl get replicaset -l app=nginx -o yaml

# Afficher le YAML d'un ReplicaSet specifique
RS_NAME=$(kubectl get rs -l app=nginx -o jsonpath='{.items[0].metadata.
    name}')
kubectl get replicaset $RS_NAME -o yaml
```

5.4 Question 4 : État d'avancement du déploiement

```
# Verifier l'etat
kubectl rollout status deployment/nginx

# Voir l'historique
kubectl rollout history deployment/nginx

# Details du deployment
kubectl describe deployment nginx

# Voir les events
kubectl get events --field-selector involvedObject.name=nginx
```

5.5 Question 5 : Mettre à jour l'image

Énoncé : Mettre à jour l'image nginx vers nginx :1.19.8.

```
# Mettre a jour l'image
kubectl set image deployment/nginx nginx=nginx:1.19.8

# Suivre la mise a jour en temps reel
kubectl rollout status deployment/nginx --watch

# Verifier la nouvelle image
kubectl describe deployment nginx | grep Image

# Voir les pods en cours de mise a jour
kubectl get pods -l app=nginx --watch
```

5.6 Question 6 : Vérifier l'historique

```
# Voir l'historique complet
kubectl rollout history deployment/nginx

# Voir les details d'une revision specifique
kubectl rollout history deployment/nginx --revision=2

# Verifier que les replicas sont correctes
kubectl get deployment nginx
kubectl get replicaset -l app=nginx
kubectl get pods -l app=nginx

# Details des replicas
kubectl describe deployment nginx | grep -i replica
```

5.7 Question 7 : Rollback du déploiement

Énoncé : Annuler le dernier déploiement et vérifier l'ancienne image.

```
# Rollback vers la revision precedente
kubectl rollout undo deployment/nginx

# Suivre le rollback
kubectl rollout status deployment/nginx

# Verifier l'image (doit etre nginx:1.18.0)
kubectl describe deployment nginx | grep Image

# Verifier les pods
kubectl get pods -l app=nginx

# Verifier l'image dans un pod
POD_NAME=$(kubectl get pods -l app=nginx -o jsonpath='{.items[0].
    metadata.name}')
kubectl describe pod $POD_NAME | grep Image

# Rollback vers une revision specifique
kubectl rollout undo deployment/nginx --to-revision=1
```

6 Commandes utiles supplémentaires

6.1 Débogage

```
# Voir les logs d'un pod
kubectl logs <pod-name>

# Logs d'un conteneur specifique
kubectl logs <pod-name> -c <container-name>

# Suivre les logs en temps reel
kubectl logs -f <pod-name>

# Logs des pods precedents (apres crash)
kubectl logs <pod-name> --previous

# Exec dans un pod
kubectl exec -it <pod-name> -- bash

# Port-forward
kubectl port-forward <pod-name> 8080:80

# Copier des fichiers
kubectl cp <pod-name>:/path/to/file ./local-file
kubectl cp ./local-file <pod-name>:/path/to/file
```

6.2 Gestion des ressources

```
# Voir la consommation de ressources
kubectl top nodes
kubectl top pods

# Scaler un deployment
kubectl scale deployment nginx --replicas=5

# Autoscaling
kubectl autoscale deployment nginx --min=2 --max=10 --cpu-percent=80

# Supprimer des ressources
kubectl delete pod <pod-name>
kubectl delete deployment <deployment-name>
kubectl delete -f fichier.yaml

# Tout supprimer dans un namespace
kubectl delete all --all -n <namespace>
```

6.3 Contextes et namespaces

```
# Voir les contextes
kubectl config get-contexts

# Changer de contexte
kubectl config use-context <context-name>

# Definir un namespace par defaut
```

```
kubectl config set-context --current --namespace=<namespace>

# Voir la configuration actuelle
kubectl config view
```

7 Cheat Sheet kubectl

7.1 Ressources courantes

```
# Pods
kubectl get pods
kubectl get pods -o wide
kubectl get pods --all-namespaces
kubectl describe pod <pod-name>

# Deployments
kubectl get deployments
kubectl describe deployment <deployment-name>
kubectl edit deployment <deployment-name>

# Services
kubectl get services
kubectl describe service <service-name>
kubectl expose deployment <name> --port=80 --type=NodePort

# Namespaces
kubectl get namespaces
kubectl create namespace <name>
kubectl delete namespace <name>

# ConfigMaps et Secrets
kubectl get configmaps
kubectl get secrets
kubectl create configmap <name> --from-literal=key=value
kubectl create secret generic <name> --from-literal=password=secret
```

7.2 Formatage de sortie

```
# YAML
kubectl get pod <name> -o yaml

# JSON
kubectl get pod <name> -o json

# JSONPath
kubectl get pods -o jsonpath='{.items[*].metadata.name}'

# Custom columns
kubectl get pods -o custom-columns=NAME:.metadata.name,STATUS:.status.
    phase

# Wide (plus de colonnes)
kubectl get pods -o wide

# Labels
```

```
kubectl get pods --show-labels
kubectl get pods -L app,tier
```

8 Conclusion

Ce TD vous a permis de maîtriser :

— Les commandes kubectl de base (create, get, describe, delete)
— La gestion des pods, deployments et services
— L'utilisation des labels et annotations
— Le placement des pods avec nodeSelector et tolerations
— Les déploiements et rollouts
— Le débogage avec logs, exec et describe

La maîtrise de kubectl est essentielle pour tout développeur ou DevOps travaillant avec Kubernetes. Ces commandes sont utilisées quotidiennement en entreprise pour déployer, gérer et déboguer des applications.

Ressources complémentaires :

— `https://kubernetes.io/docs/reference/kubectl/cheatsheet/`
— `https://kubernetes.io/docs/reference/kubectl/overview/`