TP – Utilisation de Kubectl

Kubectl est LA commande en ligne utilisée pour interagir avec les clusters Kubernetes. Il vous permet de gérer et de contrôler les ressources présentes dans un cluster Kubernetes.

Create (créer) : Cette action est utilisée pour créer une nouvelle ressource dans le cluster Kubernetes. Vous spécifiez le fichier de configuration YAML ou JSON contenant les détails de la ressource que vous souhaitez créer, et kubectl l'ajoute au cluster.

```
ludo@kubernetes:~$ kubectl create deployment nginx --image=nginx:latest
```

Get (obtenir) : Cette action vous permet d'obtenir des informations sur les ressources présentes dans le cluster. Par exemple, vous pouvez utiliser "kubectl get nodes" pour obtenir la liste des nœuds (nodes) du cluster, ou "kubectl get pods" pour obtenir la liste des pods en cours d'exécution.

```
ludo@kubernetes:~$ kubectl get deployments

ludo@kubernetes:~$ kubectl get deployments -o wide
```

Describe (décrire) : Cette action vous fournit des informations détaillées sur une ressource spécifique. Par exemple, "kubectl describe pod <nom-du-pod>" affiche des informations détaillées sur un pod spécifique, telles que son état, ses événements, ses conteneurs, etc.

```
ludo@kubernetes:~$ kubectl describe deployment nginx-deployment
```

Delete (supprimer) : Cette action vous permet de supprimer une ressource du cluster. Vous pouvez utiliser "kubectl delete" suivi du type de ressource et de son nom pour supprimer la ressource spécifiée. Par exemple, "kubectl delete pod <nom-du-pod>" supprime un pod spécifique.

```
ludo@kubernetes:~$ kubectl delete deployment nginx-deployment
```

Apply (appliquer) : Cette action est utilisée pour créer ou mettre à jour des ressources à partir d'un fichier de configuration. Vous spécifiez le fichier de configuration YAML ou JSON et kubectl applique les modifications au cluster en conséquence. Cela permet de créer des ressources ou de mettre à jour des ressources existantes de manière déclarative.

Quelques ressources spécifiques :

Node (nœud) : Un nœud est une machine physique ou virtuelle qui fait partie du cluster Kubernetes. Il exécute les pods et les services.

```
ludo@kubernetes:~$ kubectl get nodes
```

```
ludo@kubernetes:~$ kubectl get node -o wide
```

Namespace (espace de noms) : Un espace de noms est une façon de diviser un cluster Kubernetes en plusieurs parties logiques. Il permet d'isoler les ressources et les objets au sein d'un cluster.

```
ludo@kubernetes:~$ kubectl create namespace formation

ludo@kubernetes:~$ kubectl get namespaces

ludo@kubernetes:~$ kubectl get ns
```

Pod : Un pod est l'unité de base dans Kubernetes. Il représente un ou plusieurs conteneurs qui partagent un environnement d'exécution commun. Les pods sont déployés sur les nœuds du cluster.

```
ludo@kubernetes:~$ kubectl run  nginx --image=nginx:latest

ludo@kubernetes:~$ kubectl get pods  nginx

ludo@kubernetes:~$ kubectl pods nginx -o wide
ludo@kubernetes:~$ kubectl describe pod nginx
ludo@kubernetes:~$ kubectl delete pod nginx
```

Deployment (déploiement) : Un déploiement est une ressource Kubernetes qui définit comment les pods doivent être créés et mis à l'échelle. Il facilite le déploiement et la gestion des applications dans le cluster.

```
ludo@kubernetes:~$ kubectl create deployment nginx --image=nginx:latest

ludo@kubernetes:~$ kubectl get deployments

ludo@kubernetes:~$ kubectl get deployments -o wide

ludo@kubernetes:~$  kubectl describe deployment nginx-deployment
ludo@kubernetes:~$  kubectl delete deployment nginx-deployment
```

Créer des alias
Vous pouvez créer des alias pour simplifier l'utilisation de la commande kubectl. Voici quelques exemples d'alias couramment utilisés

```
ludo@kubernetes:~$  alias kga='kubectl get all' >> .bashrc

ludo@kubernetes:~$  alias kapply='kubectl apply -f' >> .bashrc

ludo@kubernetes:~$  alias kdel='kubectl delete ' >> .bashrc

ludo@kubernetes:~$  source .bashrc
```

TP – Les Pods

1.Manipuler notre premier pod

Création en mode impératif

```
ludo@kubernetes:~$ kubectl run nginx-ludo –image=nginx:latest

ludo@kubernetes:~$ kubectl get all
```

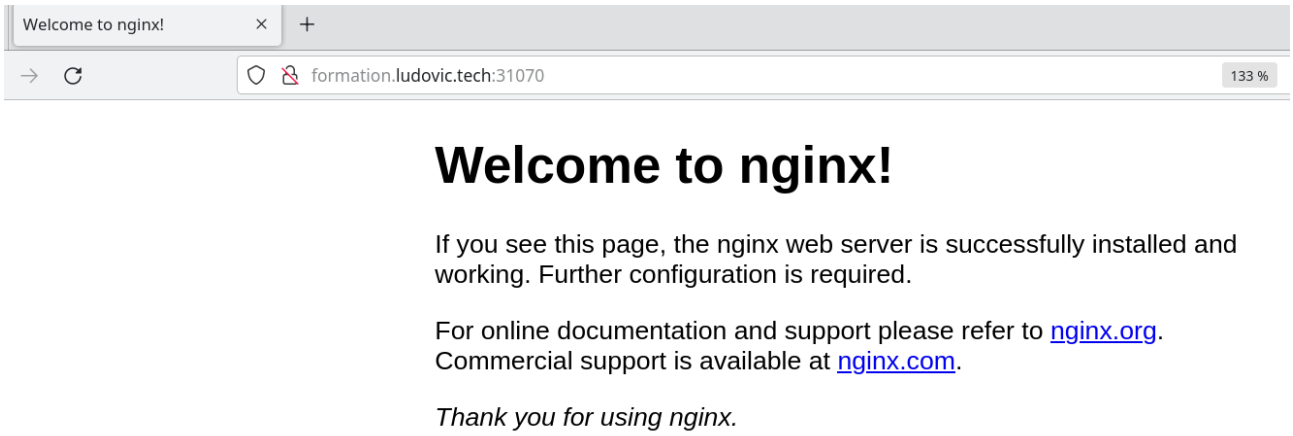
Exposer le pod sur extérieur avec un service NodePort

```
ludo@kubernetes:~$ kubectl expose pod nginx-ludo --type NodePort --port 80

ludo@kubernetes:~$ kubectl get service
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
nginx    NodePort    10.99.252.217   <none>                    80:31070/TCP
```

afficher les endpoints du service

```
ludo@kubernetes:~$ kubectl get pod nginx-ludo -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP               NODE
nginx   1/1     Running   0          68s   192.168.196.92   worker04

ludo@kubernetes:~$ kubectl get endpoints nginx
NAME    ENDPOINTS           AGE
nginx   192.168.196.92:80   5m6s
```

On « requête» le pod



On rentre dans le pod et on modifie le ficher html

```
ludo@kubernetes:~$ kubectl exec -it nginx-ludo -- bash
root@nginx-ludo:/# echo "<h1>Bienvenue sur le serveur de Ludo</h1>"  \
   /usr/share/nginx/html/index.html
```

On supprime le pod et le service

```
ludo@kubernetes:~$ kubectl delete pod nginx
pod "nginx" deleted

ludo@kubernetes:~$ kubectl delete service nginx
service "nginx" deleted
```

2.Gérer les pods avec des manifestes

Se rendre dans le répertoires $HOME/formation/pod , ouvrir le fichier nginx-pod.yaml et on change le nom du pod.

```
ludo@kubernetes:/pod$ vim nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-ludo
  labels:
    app: web
spec:
  containers:
  - image: nginx
    name: nginx
```

On applique le manifeste à l'api server

```
ludo@kubernetes:/pod$ kubectl appply -f nginx-pod.yaml
```

Exposer un pod avec un manifest

```
ludo@kubernetes:/pod$ vim service-nginx.yaml
apiVersion: v1
kind: Service
metadata:
  name: nodeport
spec:
  type: NodePort
  selector:
    app: web
  ports:
    - port: 80
      targetPort: 80
```

On applique

```
ludo@kubernetes:/pod$ kubectl apply -f service-nginx.yaml
```

TP - La gestion des sondes Kubernetes

La LivenessProbe

Ouvrir le fichier kuard-pod-liveness.yaml

```
ludo@kubernetes:/pod$ vim kuard-pod-liveness.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kuard-ludo
  labels:
    app: liveness-ludo
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    livenessProbe:
      httpGet:
        path: /healthy
        port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 1
      periodSeconds: 10
      failureThreshold: 2
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

On change le nom du pod , les labels et on applique

```
ludo@kubernetes:/pod$ kubectl apply -f kuard-pod-liveness.yaml
pod/kuard-ludo created
```

On expose l'application

```
ludo@kubernetes:/pod$ vim service-kuard.yaml
apiVersion: v1
kind: Service
metadata:
  name: kuard-ludo
spec:
  type: NodePort
  selector:
    app: demo-ludo
  ports:
    - port: 8080
      targetPort: 8080
```

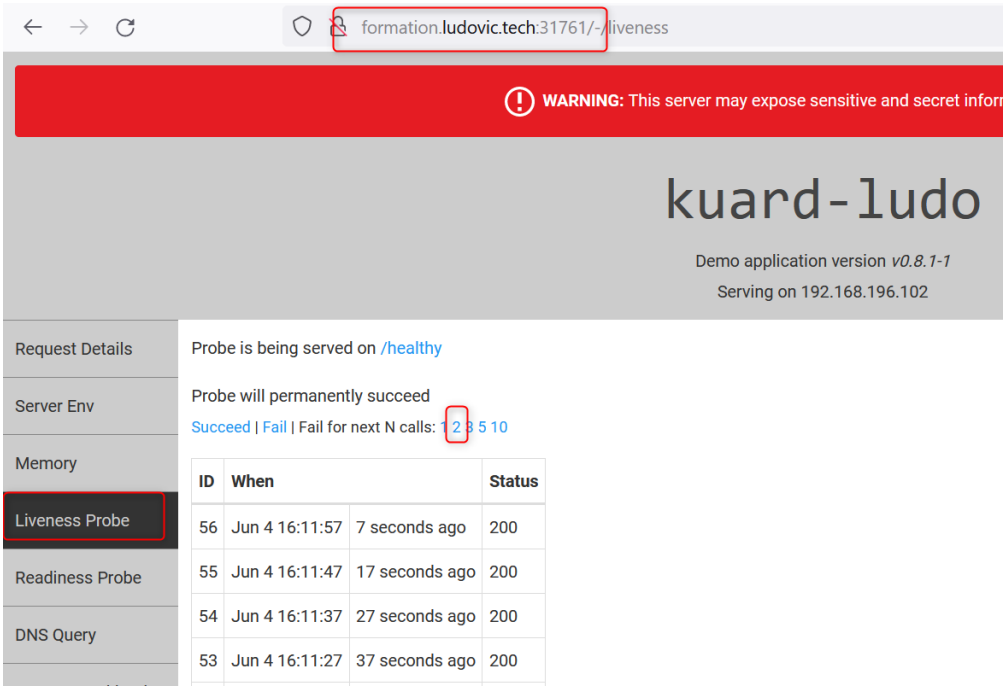
On applique le manifeste

```
ludo@kubernetes:/pod$ kubectl apply -f service-kuard.yaml
service/kuard-ludo created
```

On liste les ressources

```
ludo@kubernetes:/pod$ kubectl get all
NAME             READY   STATUS    RESTARTS   AGE
pod/kuard-ludo   1/1     Running   0          2m10s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/kuard-ludo   NodePort    10.111.216.187   <none>        8080:30101/TCP
10s
```

On teste la sonde




```
ludo@kubernetes:/pod$ kubectl get pod kuard-ludo

Every 2.0s: kubectl get pod
kubernetes.ludovic.tech: Sun Jun  4 18:14:13 2023

NAME         READY   STATUS    RESTARTS   AGE
kuard-ludo   1/1     Running   0          11m

NAME         READY   STATUS    RESTARTS      AGE
kuard-ludo   1/1     Running   1 (9s ago)    12m
```

On supprime tout

```
ludo@kubernetes:/pod$ kubectl delete all --selector app=demo-ludo
```

Readiness

Ouvrir le fichier kuard-pod-readiness.yaml

```
ludo@kubernetes:/pod$ vim kuard-pod- readiness.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kuard
  labels:
    app: demo
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
   readinessProbe:
      httpGet:
        path: /ready
        port: 8080
      initialDelaySeconds: 30
      timeoutSeconds: 1
      periodSeconds: 10
      failureThreshold: 1
```

On change le nom du pod , les labels et on applique

```
ludo@kubernetes:/pod$ kubectl apply -f kuard-pod- readiness.yaml
pod/kuard-ludo created

ludo@kubernetes:/pod$  kubectl get all --show-labels
NAME        READY   STATUS    RESTARTS   AGE   LABELS
pod/kuard   0/1     Running   0          7s    app=demo-ludo


ludo@kubernetes:/pod$  kubectl get all
NAME        READY   STATUS    RESTARTS   AGE
pod/kuard   1/1     Running   0          47s
```

On expose l'application

```
ludo@kubernetes:/pod$ vim service-kuard.yaml
apiVersion: v1
kind: Service
metadata:
```

```
  name: kuard-ludo
spec:
  type: NodePort
  selector:
    app: demo-ludo
  ports:
    - port: 8080
      targetPort: 8080
```

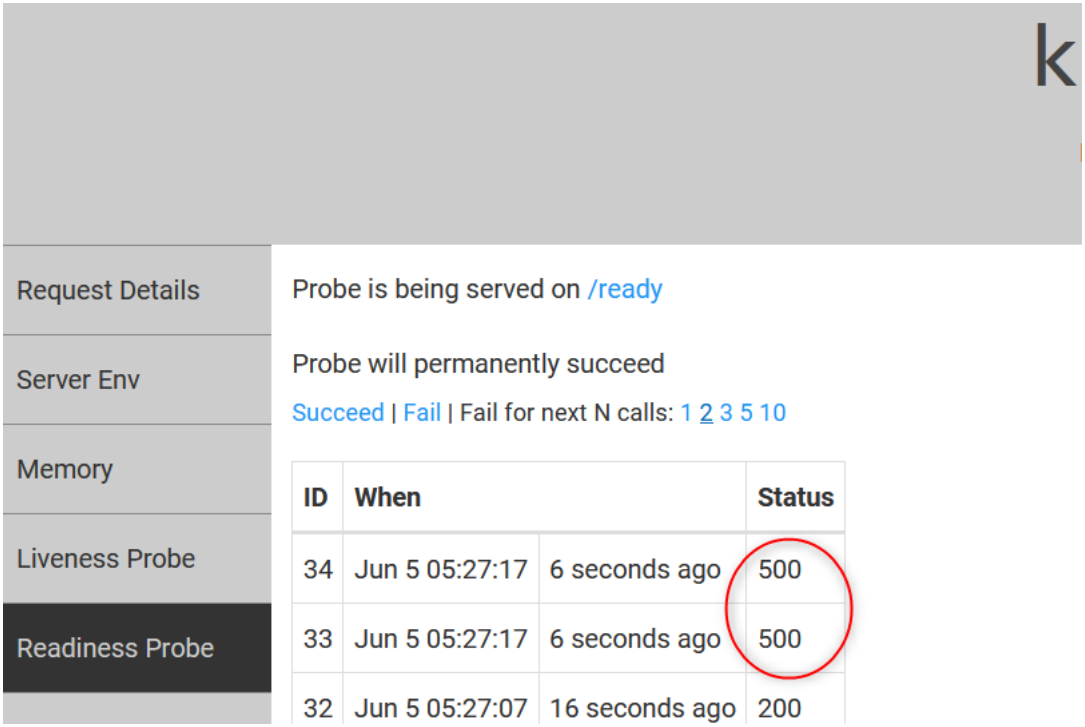
On applique le manifeste

```
ludo@kubernetes:/pod$ kubectl apply -f service-kuard.yaml
service/nodeport created
```

On liste les ressources

```
ludo@kubernetes:/pod$ kubectl get all
NAME             READY   STATUS    RESTARTS   AGE
pod/kuard-ludo   1/1     Running   0          2m10s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/kuard-ludo   NodePort    10.111.216.187   <none>        8080:30101/TCP
10s
```

On teste la sonde



On «watch» les ressources

```
ludo@kubernetes:/pod$ watch kubectl get pod,endpoint
Every 2.0s: kubectl get pod,endpoints

NAME             READY   STATUS    RESTARTS   AGE
pod/kuard-ludo   1/1     Running   0          3m10s

NAME                   ENDPOINTS             AGE
endpoints/kuard-ludo   192.168.196.117:8080   3m16



NAME             READY   STATUS    RESTARTS   AGE
pod/kuard-ludo   0/1     Running   0          4m57s

NAME                   ENDPOINTS       AGE
endpoints/kuard-ludo                   5m3s
```

La gestion des ressources.

TP -Haute disponibilité
Création d'un réplicaset
Mise a l'échelle des pods

TP – Déployer des applications
Création d'un déployement
Gestion des mise a jours – RollingUpdate et Recreate
Mise a l' échelle des applications

TP – Le stockage persistent
Création d' un PVC
Utiliser des Classe de stockage
Les volumes dans les pods
Déployement d'applications avec du stockage

TP – Réseaux
Les services ClusterIP
La découverte de service DNS
Les ingress
les ingress et la réécriture d'url

TP – Helm
Déploiements d'applications avec Helm
Personnalisé un chart
Créer son chart
Déployer son chart