

TP 2: Prise en main de kubernetes

Prérequis

Avant de commencer ce tp, il est nécessaire d'avoir installer **Podman Desktop** ou **Docker Destop**

Installation de kubernetes et des outils

Minikube

Suivez la procédure d'installation disponible à cette [adresse](#)

Kubectl

L'outil de management kubernetes kubectl disponible à cette [adresses](#)

Pensez à faire un alias :

```
alias k="kubectl"
```

Vérifiez si kubectl est installé en exécutant la commande suivante :

```
kubectl version
```

Memento des commandes [kubectl](#)

Liste des ressources sous format court **kubectl api-resources**:

Short name	Full name
csr	certificatesigningrequests
cs	componentstatuses
cm	configmaps
ds	daemonsets
deploy	deployments
ep	endpoints
ev	events
hpa	horizontalpodautoscalers
ing	ingresses
limits	limitranges

Short name	Full name
ns	namespaces
no	nodes
pvc	persistentvolumeclaims
pv	persistentvolumes
po	Pods
pdb	poddisruptionbudgets
psp	podsecuritypolicies
rs	replicasets
rc	replicationcontrollers
quota	resourcequotas
sa	serviceaccounts
svc	services

kubecolor (optionnel)

Vous permet d'avoir kubectl en couleur disponible à cette [adresse](#)

Pensez à faire un alias:

```
alias kubectl="kubecolor"  
alias k="kubecolor"
```

Lens

GUI pour kubernetes disponible à cette [adresse](#)

Premier Pas avec minikube

Créer un cluster kubernetes

1. Ouvrez une fenêtre de terminal ou une invite de commande.
2. Exécutez la commande suivante pour démarrer Minikube et créer un cluster Kubernetes local :

```
minikube start
```

Warning Si vous utilisez podman : `minikube start --driver=podman --container-runtime=cri-o`

Minikube téléchargera et configurera automatiquement les composants nécessaires pour le cluster Kubernetes.

3. Vérification de l'état du cluster : Vous pouvez vérifier l'état du cluster Kubernetes avec la commande suivante :

```
minikube status
```

Elle affichera des informations sur l'état du cluster, comme l'adresse IP et l'état de Minikube lui-même.

4. Interagir avec le cluster Kubernetes :

Pour interagir avec le cluster Kubernetes, vous utiliserez l'outil en ligne de commande kubectl.

```
kubectl cluster-info
```

Exemple :

```
> kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:42413
CoreDNS is running at https://127.0.0.1:42413/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```




Minikube addons


Minikube fournit des addons kubernetes qui permet d'ajouter des fonctionnalités rapidement à notre cluster kubernetes grâce à la commande `minikube addons enable <ADDON_NAME>`

Ci-dessous la commande permettant de lister l'ensemble des addons disponibles.

```
> minikube addons list
```

ADDON NAME	PROFILE	STATUS	MAINTAINER
ambassador (Ambassador)	minikube	disabled	3rd party
auto-pause	minikube	disabled	Google
cloud-spanner	minikube	disabled	Google
csi-hostpath-driver	minikube	disabled	Kubernetes

dashboard	minikube	enabled 	Kubernetes
default-storageclass	minikube	enabled 	Kubernetes
efk (Elastic)	minikube	disabled	3rd party
freshpod	minikube	disabled	Google
gcp-auth	minikube	disabled	Google
gvisor	minikube	disabled	Google
headlamp (kinvolk.io)	minikube	disabled	3rd party
helm-tiller	minikube	disabled	3rd party (Helm)
inaccel (InAccel [info@inaccel.com])	minikube	disabled	3rd party
ingress	minikube	disabled	Kubernetes
ingress-dns	minikube	disabled	Google
istio (Istio)	minikube	disabled	3rd party
istio-provisioner (Istio)	minikube	disabled	3rd party
kong HQ)	minikube	disabled	3rd party (Kong)
kubevirt (KubeVirt)	minikube	disabled	3rd party
logviewer (unknown)	minikube	disabled	3rd party
metallb (MetalLB)	minikube	disabled	3rd party
metrics-server	minikube	enabled 	Kubernetes
nvidia-driver-installer	minikube	disabled	Google
nvidia-gpu-device-plugin (Nvidia)	minikube	disabled	3rd party
olm (Operator Framework)	minikube	disabled	3rd party
pod-security-policy (unknown)	minikube	disabled	3rd party
portainer (Portainer.io)	minikube	disabled	3rd party
registry	minikube	disabled	Google
registry-aliases (unknown)	minikube	disabled	3rd party
registry-creds Enterprises)	minikube	disabled	3rd party (UPMC)

storage-provisioner	minikube	enabled 	Google
storage-provisioner-gluster (Gluster)	minikube	disabled	3rd party
volumesnapshots	minikube	disabled	Kubernetes
-----	-----	-----	-----

Activation du dashboard kubernetes

Nous allons installer l'addon des dashboard afin de pouvoir visualiser dans son navigateur son cluster kubernetes

`minikube addons enable dashboard`

Si l'addon metrics-server n'est pas activé, il faudra l'activer avec la commande : `minikube addons enable metrics-server`

Accéder au dashboard kubernetes

La commande `minikube dashboard` va vous ouvrir un navigateur avec le dashboard kubernetes

Questions

Que voyez-vous ?

Combien y-a-t-il de nodes kubernetes ?

Combien y-a-t-il de namespaces ?

Objets Kubernetes

Explorons notre cluster, il est composé de plein d'objets divers, organisés entre eux de façon dynamique pour décrire des applications, tâches de calcul, services et droits d'accès.

La commande `get` est générique et peut être utilisée pour récupérer la liste de tous les types de ressources.

Nodes

Lister les nodes de notre cluster avec la commande `k get nodes`

Combien y-a-t-il de node dans notre cluster ?

Est-ce normal ?

Namespaces

Lister les namespaces

Le namespace permet d'isoler des groupes de ressources au sein d'un seul cluster.

Lister l'ensemble des namespaces de votre cluster `k get ns` ou `k get namespaces`

Combien y-a-t-il de namespaces dans votre cluster ?

Configuration d'un namespace

La commande `k describe` permet d'afficher l'état détaillé d'une ou plusieurs ressources, en incluant par défaut les ressources non initialisées. `k describe TYPE/NOM`

Regardons en détails la configuration d'un namespace `k describe ns/default`

Quel est le status du namespace ?

Y-a-t-il des quotas ?

Y-a-t-il un LimitRange ?

Y-a-t-il des labels ?

Namespace en yaml

Vous ne connaissez pas la définition sous format YAML de l'objet namespace. La commande `k get ns default -o yaml` va vous permettre d'exporter la configuration sous format YAML

```
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2023-05-22T08:32:17Z"
  labels:
    kubernetes.io/metadata.name: default
  name: default
  resourceVersion: "193"
  uid: 8e739e79-46e6-498b-93a6-db690c2b7787
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Créer son propre namespace

Par défaut dans un cluster kubernetes vous trouverez par défaut le namespace **default** mais vous pouvez créer autant de namespace que vous le désirez.

Créer votre propre namespace avec la commande `k create ns tp2`

Résultat: `namespace/tp2 created`

Combien y-a-t-il de namespaces dans votre cluster ?

Supprimer son namespace

La commande **k delete** permet de supprimer une ressources par noms de fichiers, stdin, noms, ou par sélecteur d'étiquettes.

Supprimons notre namespace:

```
> k delete ns tp2
namespace "tp2" deleted
```

Pods

Un pod est la plus petite unité de déploiement dans Kubernetes.

Nous allons déployer un pod nginx dans un namespace **tp2**

Créer le namespace **tp2**

Création

Afin de pouvoir créer un pods, nous devons avoir une définition en YAML décrivant ce pod.

Ci-dessous la définition d'un pod nginx(OS: debian) :

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

Sauvegardez ce fichier dans un fichier nommé: **pod.yaml**

Vous pouvez enregistrer de manière permanente un namespace pour toutes les commandes kubectl, **kubectl config set-context --current --namespace=<namespace>** sans avoir besoin d'ajouter **-n <namespace>** à chaque commande **kubectl**

Enregistrez le namespace avec la commande **kubectl config set-context --current --namespace=tp2**

Afin de pouvoir créer cet objet dans notre cluster kubernetes, nous allons utiliser la commande **k apply -f fichier**

Exemple:

```
> k apply -f pod.yaml
pod/nginx created
```

Lister

Une fois que votre pod, nous allons utiliser la commande `k get pod`. Elle va nous permettre d'avoir l'ensemble des pods dans le namespace

Résultat:

```
> k get pods
NAME      READY   STATUS    RESTARTS   AGE
nginx     1/1     Running   0           8m11s
```

Décrire

Grâce à la commande `k describe`, nous allons pouvoir avoir encore plus d'information sur notre pod.

Quel est l'état de votre pod ?

Quel est son containerID ?

Quel est son IP ?

Debug

Il est possible selon comment votre image a été fabriquée de se connecter à l'intérieur d'un conteneur présent dans `k exec pod/<pod_name> -it -- bash`

Résultat:

```
> k exec pod/nginx -it -- bash
root@nginx:/#
```

Il se peut aussi que nous n'ayez pas tous les packages nécessaire afin de débbuger.

Pensez à installer les packages nécessaire (procps)

Combien y-a-t-il de process nginx ?

Quel est l'interet d'avoir un conteneur minimaliste ?

Warning Il y a des images sans shell c'est ce qu'on appelle ***distroless container images***. Il est compliqué de pouvoir s'y connecter mais vous pouvez suivre cette [page](#) si besoin

Supression

Votre pod continuera de fonctionner pour l'arrêter nous devons le supprimer.

La commande `k delete` va nous aider à supprimer des ressources.

Nous avons 2 façons disponibles pour supprimer notre pod:

- soit à partir du fichier yaml: `k delete -f pod.yaml`
- soit directement: `k delete pod nginx`

Deploiment

Le **deployment** permet de déployer et de gérer des applications dans un cluster Kubernetes de manière automatisée.

Créer un deploiment

Ci-dessous l'exemple d'un deploiement nginx au format YAML à enregistrer dans un fichier sous le nom **deployment.yml**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
      resources:
        limits:
          memory: 512Mi
          cpu: "1"
        requests:
          memory: 256Mi
          cpu: "0.2"
```

Appliquer ce deploiement dans votre cluster kubernetes. Quel est la commande a exécuté ? Quel est le résultat de votre commande ? Combien y-a-t-il de pod lié à votre deployment ?

Etat de deploiement et historique

La commande `kubectl rollout status deployment/<deployment_name>` va nous donner l'état de notre deployment La commande `k rollout history deployment/<deployment_name>` va nous donner l'historique de nos deployments

Quel est l'état de notre deployment ?

Replicaset

Le deployment gère l'objet **replicaset**, la commande `k get rs` permet de lister les replicaset. Combien y-a-t-il de réplicaset ?

Avoir les détails de son déploiement

Grâce à la commande `k describe deployment/<deployment_name>`, nous allons pouvoir avoir plus de détails sur notre déploiement.

Y-a-t-il des request en cpu et ram ? Quel est la stratégie d'update (StrategyType) ?

Mise à l'échelle de son déploiement

Vous pouvez mettre à l'échelle un déploiement à l'aide de la commande suivante `k scale deployment/nginx-deployment --replicas=10`

Mais nous pouvons aussi modifier le fichier **deployment.yaml** afin de changer la spec replicas pour la passer à 10.

Combien de pods sont démarrés ? Pourquoi l'ensemble des pods ne sont démarrés ? (Vous pouvez utiliser le dashboard pour avoir plus d'informations si besoin)

Supprimer son déploiement

Afin de libérer des ressources de notre cluster, supprimer le deployment.

Quel est la commande ?

Service

Nous avons vu comment deployer un nginx mais comment faire pour y accéder.

C'est là que le service va nous permettre d'exposer une application en tant que service réseau stable

Créer un service interne

Ci-dessous le YAML d'un service à enregistrer dans un fichier sous le nom **service.yaml** puis l'appliquer avec la commande `k apply -f svc.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
```

```
selector:
  app: nginx
ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Quel est le résultat de la commande ?

Lister les service

La commande `k get svc` va lister les services de votre namespace. Minikube permet aussi de lister les services du cluster `minikube service list`

Quel est le résultat de la commande avec kubectl ? Quel est l'IP de votre service kubernetes ? Quel est le type de votre service ?

Accéder à nginx depuis son ordinateur

Nous avons deux possibilité afin de pouvoir accéder à notre nginx.

- Kubectl

Nous allons utilisé le port-forward: `kubectl port-forward svc/nginx 8080:80` Puis dans un autre terminal `curl http://127.0.0.1:8080`

Quel est le résultat de la commande ?

- Minikube La commande `minikube service <service>` ici `minikube service nginx`, une fenêtre s'ouvrira dans votre navigateur

Créer un ssrvice Loadbalancer

Les services de type LoadBalancer peuvent être exposés via la `minikube tunnel` commande. Il doit être exécuté dans une fenêtre de terminal distincte pour continuer à fonctionner.

Ctrl-C dans le terminal peut être utilisé pour mettre fin au processus, moment auquel les routes réseau seront nettoyées.

Lancer la commande: `minikube tunnel`

Créer un service de type Loadbalancer en utilisant le yaml ci-dessous :

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: LoadBalancer
  selector:
    app: nginx
```

```
ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
```

Puis faire `k get svc` et récupérer l'adresse IP de la colonne EXTERNAL-IP

Exemple:

```
> k get svc
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
nginx         LoadBalancer  10.105.7.108    127.0.0.1        8080:31816/TCP   11m
```

Puis faire un la commande `curl http://127.0.0.1:8080`.

Quel est le résultat ?

Utilisation d'un ingress

Minikube support un ingress, pour l'activer: `minikube addons enable ingress`

Le namespace ingress-nginx est créé avec les objets ci-dessous :

```
> k get all -n ingress-nginx
```

NAME	READY	STATUS	
RESTARTS	AGE		
pod/ingress-nginx-admission-create-v96c8	0/1	Completed	0
2m59s			
pod/ingress-nginx-admission-patch-tj5lp	0/1	Completed	1
2m59s			
pod/ingress-nginx-controller-6cc5ccb977-nhcrb	1/1	Running	0
2m59s			

NAME	TYPE	CLUSTER-IP
EXTERNAL-IP	PORT(S)	AGE
service/ingress-nginx-controller	NodePort	10.107.142.172
<none>	80:30072/TCP,443:30428/TCP	2m59s
service/ingress-nginx-controller-admission	ClusterIP	10.110.46.189
<none>	443/TCP	2m59s

NAME	READY	UP-TO-DATE	AVAILABLE
AGE			
deployment.apps/ingress-nginx-controller	1/1	1	1
2m59s			

NAME	DESIRED	CURRENT
READY		
AGE		
replicaset.apps/ingress-nginx-controller-6cc5ccb977	1	1
1		
2m59s		

NAME	COMPLETIONS	DURATION	AGE
job.batch/ingress-nginx-admission-create	1/1	8s	2m59s
job.batch/ingress-nginx-admission-patch	1/1	9s	2m59s

Récupère l'adresse ip de votre ingress dans le champs ADDRESS :

```
> k get ingress
NAME      CLASS   HOSTS          ADDRESS          PORTS   AGE
nginx     nginx   nginx.info     192.168.49.2    80      6m16s
```

Créer l'objet ingress comme dans le YAML ci-dessous :

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: nginx.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: nginx
                port:
                  number: 8080
```

Ajouter dans votre /etc/host : <adresse_ip> nginx.info

Gestion des ressources

Se placer dans le namespace tp2 ou le créer s'il a été supprimé.

Quota

Appliquer le YAML ci-dessous, il permet de limiter la consommation de RAM à 2Go et la cpu à 2:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: my-resource-quota
spec:
  hard:
```

```
limits.memory: "2Gi"    # Limite maximale de la mémoire
limits.cpu: "2"          # Limite maximale de l'utilisation du CPU
```

Appliquer le YAML ci-dessous permettant de deployer nginx

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
          resources:
            limits:
              memory: 512Mi
              cpu: "0.75"
            requests:
              memory: 256Mi
              cpu: "0.2"
```

Utilisation du quota

Vu que nous avons mis en place une ressource quota sur le namespace, nous pouvons lancer la commande qui permet d'avoir les détails du namespace afin d'avoir la consommation du quota : Exemple :

```
> k describe ns/tp2
Name:          tp2
Labels:        kubernetes.io/metadata.name=tp2
Annotations:   <none>
Status:        Active
```

Resource Quotas

```
Name:          my-resource-quota
Resource       Used    Hard
-----
limits.cpu     1750m  2
limits.memory  1536Mi 2Gi
```

Passer le nombre de replicas de 2 à 3.

Combien voyez-vous de pods ?

Qu'est qui se passe ?

Pour vous aider, les commandes :

- `k get events` permet de lister les événements kubernetes
- `k describe rs/<replicaset_name>`: permet d'avoir les details sur le scale des pods

Une fois terminée, supprimez le deployment et le quota

LimitRange

Appliquer le YAML ci-dessous permettant de définir un limitRange quand la spec n'est pas présente dans les pods

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
    - default: # this section defines default limits
      cpu: 500m
      defaultRequest: # this section defines default requests
        cpu: 500m
      max: # max and min define the limit range
        cpu: "1"
      min:
        cpu: 100m
      type: Container
```

En récupérant les informations détaillées du namespace, nous avons :

```
> k describe ns/tp2
Name:          tp2
Labels:        kubernetes.io/metadata.name=tp2
Annotations:   <none>
Status:        Active

No resource quota.

Resource Limits
Type          Resource  Min  Max  Default Request  Default Limit  Max
Limit/Request Ratio
-----
```

Container	cpu	100m	1	500m	500m	—

Notre limitrange est bien en place.

Utilisation du limitRange

Appliquer le YAML ci-dessous qui permet de créer un pod sans limit et request

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

Avec la commande `k describe pod/nginx` que remarquez-vous au niveau des limits en CPU ? Une fois terminée, supprimez le limitRange et le pod

HPA

Un HorizontalPodAutoscaler (HPA en abrégé) met automatiquement à jour une ressource de charge de travail (telle qu'unDéploiement ou StatefulSet), dans le but d'adapter automatiquement la charge de travail à la demande.

Créer les objets dans le YAML ci-dessous :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: registry.k8s.io/hpa-example
```



```

    ports:
    - containerPort: 80
  resources:
    limits:
      cpu: 500m
    requests:
      cpu: 200m
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache

```

Maintenant que le serveur est en cours d'exécution, créez l'autoscaler à l'aide de kubectl. Il existe kubectl autoscale une sous-commande, qui fait partie de kubectl, qui vous aide à le faire.

Créer l'autoscale : `kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10`

Vérifier son état: `kubectl get hpa`

Augmenter la charge

Dans un notre terminal, lancer la commande ci-dessous:

```

# Run this in a separate terminal
# so that the load generation continues and you can carry on with the rest
of the steps
kubectl run -i --tty load-generator --rm --image=busybox:1.28 --
restart=Never -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-
apache; done"

```

Exécuter la commande qui permet de suivre l'hpas: `kubectl get hpa php-apache --watch`

En une minute environ, vous devriez voir la charge CPU plus élevée et le nombre de réplicas qui a augmenté

Par exemple:

```
> kubectl get hpa php-apache --watch
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
REPLICAS	AGE			

php-apache 98s	Deployment/php-apache	0%/50%	1	10	1
php-apache 2m	Deployment/php-apache	84%/50%	1	10	1
php-apache 2m15s	Deployment/php-apache	84%/50%	1	10	2

Le déploiement montre également qu'il a été augmenté

```
> kubectl get deployment php-apache
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
php-apache	4/4	4	4	5m58s

Arrêter de générer de la charge

Dans l'autre terminal, arrêter la commande avec un Ctrl+C Puis suivre le hpa, vous allez constater que le nombre de replicas va petit à petit réduire

```
# type Ctrl+C to end the watch when you're ready
kubectl get hpa php-apache --watch
```

Exemple:

```
> kubectl get hpa php-apache --watch
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS
php-apache 9m6s	Deployment/php-apache	42%/50%	1	10
php-apache 10m	Deployment/php-apache	11%/50%	1	10
php-apache 11m	Deployment/php-apache	0%/50%	1	10
php-apache 14m	Deployment/php-apache	0%/50%	1	10
php-apache 15m	Deployment/php-apache	0%/50%	1	10
php-apache 15m	Deployment/php-apache	0%/50%	1	10
php-apache 16m	Deployment/php-apache	0%/50%	1	10

Cronjob

Appliquer le YAML ci-dessous qui permet de scheduler un job toutes les minutes en affichant l'heure et un message

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: hello
              image: busybox:1.28
              imagePullPolicy: IfNotPresent
              command:
                - /bin/sh
                - -c
                - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

La commande `k get cj` permet de lister les cronjobs

Les jobs issue du cronjob

La commande `k get jobs` permet de lister les jobs.

Combien avez-vous de jobs ?

Voir la log du cronjob

La commande `k logs TYPE/name` permet de voir les logs de la ressource cependant avec les cronjobs cela n'est pas implémenté encore.

Exemple:

```
> kubectl logs cronjobs.batch/hello
error: cannot get the logs from *v1.CronJob: selector for *v1.CronJob not implemented
```

Pour cela, il faut récupérer le pod généré par le job et regarder la log :

Exemple Job:

```
> k get jobs.batch
NAME                                COMPLETIONS  DURATION  AGE
```

hello-28079285	1/1	3s	2m54s
hello-28079286	1/1	4s	114s
hello-28079287	1/1	4s	54s

Exemple Pod:

```
> k get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hello-28079288-79jq6	0/1	Completed	0	2m25s
hello-28079289-kc7mj	0/1	Completed	0	85s
hello-28079290-tp79p	0/1	Completed	0	25s

Exemple log:

```
> k logs pods/hello-28079289-kc7mj
Mon May 22 12:09:00 UTC 2023
Hello from the Kubernetes cluster
```

Pour aller plus loin <https://medium.com/@pranay.shah/how-to-get-logs-from-cron-job-in-kubernetes-last-completed-job-7957327c7e76>

Le Stockage

Une StorageClass est une ressource qui permet de définir des propriétés communes pour les volumes persistants utilisés par les pods.

La commande `k get sc` permet de lister les classes de stockage présentes dans votre cluster.

La commande `k describe sc/standard` permet d'avoir les détails de class de storage

Minikube prend en charge les PersistentVolumes de type hostPath prêts à l'emploi. Ces PersistentVolumes sont mappés à un répertoire à l'intérieur de l'instance minikube en cours d'exécution

Utilisation des persistant Volumes

Appliquer le YAML ci-dessous permettant de créer un volume d'1Go

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: my-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

La commande `k get pvc` permet de lister les pvc.

La commande `k get pv` permet de lister les pv.

```
kind: Pod
apiVersion: v1
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: my-pv-claim
      persistentVolumeClaim:
        claimName: my-pv-claim
  containers:
    - name: nginx
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: my-pv-claim
```

Rentrez dans le pod :

`k exec pod/task-pv-pod -it -- bash`

Puis placez dans le repertoire **`/usr/share/nginx/html`**

Créez un fichier test.html comme ci-dessous

```
<html>
<body>
Hello World
</body>
</html>
```

Puis effectuez un `curl http://127.0.0.1/test.html`

Quel est le retour du curl ?

Sortez du pod et faire un `k delete pod task-pv-pod`

Recréer le pod à nouveau puis rentrez à nouveau dans le pod.

Lister les fichiers dans le repertoire **`/usr/share/nginx/html`**

Quel est le résultat ?

Quel est le résultat de la commande `curl http://127.0.0.1/test.html` ?

Une fois les commandes terminées, supprimer le pod et le pvc.

Les composants du control Plane

Se placer dans le namespace **kube-system** et lister l'ensemble des pods

Quel(s) composant(s) du control plane retrouvez-vous ? Comment sont-ils déployés ?

Securité

Gestion des serviceaccounts, roles et rolebinding

Se placer dans le namespace **tp2** puis créer un serviceaccount avec la commande suivante **kubectl create serviceaccount my-service-account**

Créer le role en appliquant le YAML ci-dessous :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: my-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["get", "list", "create", "update"]
- apiGroups: [""]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "list"]
```

Créer le role binding en appliquant le YAM ci-dessous :

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-role-binding
subjects:
- kind: ServiceAccount
  name: my-service-account
  namespace: default
roleRef:
  kind: Role
  name: my-role
  apiGroup: rbac.authorization.k8s.io
```

Lancer la commande suivante qui permet d'utiliser le serviceaccount et de verifier les droits sur un objet : `k auth can-i get pods --as=system:serviceaccount:tp2:my-service-account -n tp2`

Quel est le résultat de la commande ? Est-ce normal ? Pourquoi ?

Lancer la commande suivante : `k auth can-i get pods --as=system:serviceaccount:tp2:my-service-account -n default` Quel est le résultat de la commande ? Est-ce normal ? Pourquoi ?

Lancer la commande suivante : `k auth can-i get svc --as=system:serviceaccount:tp2:my-service-account -n tp2` Quel est le résultat de la commande ? Est-ce normal ? Pourquoi ?

Lancer la commande suivante : `k auth can-i get secret --as=system:serviceaccount:tp2:my-service-account -n default` Quel est le résultat de la commande ? Est-ce normal ? Pourquoi ?