TP KUBERNETES

M2 TIW

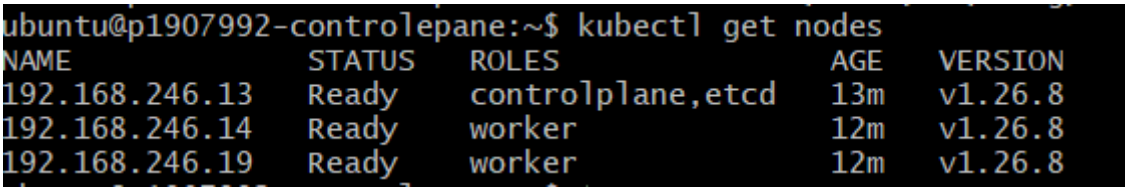
BAHADI Imane p1907992

Table des matières

A. Utilisation du cluster

1. Installation et configuration de kubectl

- `kubectl get nodes`
    - Quel est l'état des nœuds ?



Il y a trois nœuds dans le cluster :

- **192.168.246.13**: C'est le nœud Control Plane et il est également membre de l'ensemble etcd. Il est dans l'état Ready, ce qui signifie qu'il est opérationnel et capable de gérer le cluster.

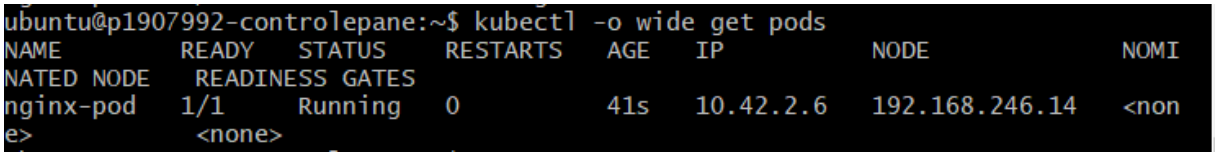
- **192.168.246.14** & **192.168.246.19**: Ce sont les nœuds Workers. Ils sont aussi dans l'état Ready, indiquant qu'ils sont prêts à accepter et exécuter des conteneurs et des services.

Le fait que tous les nœuds soient dans l'état Ready signifie que le cluster est configuré correctement et que tous les nœuds sont fonctionnels et communiquent avec succès avec le Control Plane.

2. Création d'un pod

- Sur quel nœud le Pod a-t-il été lancé ?

Le pod nginx-pod a été lancé sur le nœud Worker 1 avec l'adresse IP 192.168.246.14.



- Vérifiez l'accessibilité du Pod en interrogeant le port 8080 de tous les nœuds. Que pouvez-vous conclure ?

Pour tester l'accessibilité d'un service exposé sur le port 8080, nous pouvons utiliser **curl**. Par exemple: **curl http://192.168.246.14:8080**

Pour le Worker Node 2 et Control Plane, ça ne marche pas ; Le Pod Nginx n'est pas réparti entre tous les nœuds du cluster.

3. Création d'un deployment

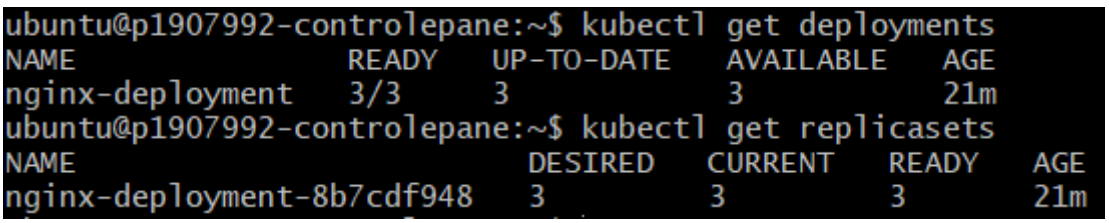
Créez cet objet de type Deployment dans le cluster

- Quels rôles jouent les labels et les sélecteurs ?
    - Labels:
        - Les labels sont des paires clé-valeur qu'on attache à des objets comme les Pods. Ils sont utilisés pour sélectionner et effectuer des opérations sur des groupes d'objets. Dans le fichier **nginx_deployment.yml**, *app: web* est un label attaché aux Pods créés par le déploiement. Ce label peut être utilisé pour identifier tous les pPods appartenant à l'application web.
    - Sélecteurs:
        - Les sélecteurs sont utilisés pour sélectionner un ensemble d'objets basé sur leurs labels. Ils sont souvent utilisés pour indiquer à un service quel groupe de pods doit être ciblé pour le trafic réseau, ou à un déploiement. Dans le fichier de déploiement, le sélecteur *matchLabels* est utilisé pour indiquer au déploiement de gérer tous les pods ayant le label *app: web*.
- Sur quelle image seront basés les conteneurs créés ?
    - Dans le déploiement créé, chaque pod contiendra un conteneur basé sur l'image nginx. Cela signifie que l'image utilisée sera la dernière version de nginx disponible sur le registre par défaut de Docker, Docker Hub.

Vous pouvez suivre le processus de déploiement et visualiser l'état des déploiements avec les commandes suivantes

- Combien de réplicas ont été créés par le déploiement ?

D'après la sortie des commandes, le déploiement nginx-deployment a créé 3 réplicas.



Mettez à l'échelle votre déploiement pour avoir 6 replicas

- Que voyez-vous dans la liste des événements du déploiement ?

Les événements indiquent l'activité du contrôleur de déploiement qui a d'abord mis à l'échelle le ReplicaSet à 3 réplicas il y a 24 minutes (ce qui correspond à la création initiale du déploiement), puis il l'a mis à l'échelle à 6 réplicas il y a 20 secondes (ce qui correspond à notre commande scale).

```
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  24m   deployment-controller  Scaled up replica set
nginx-deployment-8b7cdf948 to 3
  Normal  ScalingReplicaSet  20s   deployment-controller  Scaled up replica set
nginx-deployment-8b7cdf948 to 6 from 3
```

Vérifiez que votre déploiement a lancé un bon nombre des réplicas

```
ubuntu@p1907992-controlepane:~$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   6/6     6            6           28m
```

Visualisez les pods lancées par le déploiement

```
ubuntu@p1907992-controlepane:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-8b7cdf948-4bstk    1/1     Running   0          29m
nginx-deployment-8b7cdf948-7zlnr    1/1     Running   0          29m
nginx-deployment-8b7cdf948-922cn    1/1     Running   0          4m54s
nginx-deployment-8b7cdf948-hmght    1/1     Running   0          4m54s
nginx-deployment-8b7cdf948-qnc2c    1/1     Running   0          4m54s
nginx-deployment-8b7cdf948-r7qxm    1/1     Running   0          29m
nginx-pod                           1/1     Running   0          26h
```

**

- Comment sont distribués les pods entre les nœuds Workers? (utilisez l'option `-o wide`)

4 des 6 pods créés par le déploiement sont sur le nœud **192.168.246.14** qui est le *Worker Node 1* créé précédemment et les 2 autres sur le nœud **192.168.246.19** qui est le *Worker Node 2*.

```
ubuntu@p1907992-controlepane:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP          NODE              NOMINATED NODE   READINESS GATES
nginx-deployment-8b7cdf948-4bstk    1/1     Running   0          38m   10.42.2.8   192.168.246.14    <none>           <none>
nginx-deployment-8b7cdf948-7zlnr    1/1     Running   0          38m   10.42.2.7   192.168.246.14    <none>           <none>
nginx-deployment-8b7cdf948-922cn    1/1     Running   0          14m   10.42.2.9   192.168.246.14    <none>           <none>
nginx-deployment-8b7cdf948-hmght    1/1     Running   0          14m   10.42.1.6   192.168.246.19    <none>           <none>
nginx-deployment-8b7cdf948-qnc2c    1/1     Running   0          14m   10.42.1.7   192.168.246.19    <none>           <none>
nginx-deployment-8b7cdf948-r7qxm    1/1     Running   0          38m   10.42.1.5   192.168.246.19    <none>           <none>
nginx-pod                           1/1     Running   0          26h   10.42.2.6   192.168.246.14    <none>           <none>
```

4. Création d'un service

- Quel est l'intérêt de la section `selector` dans la description du service?

Le *selector* dans la définition du Service Kubernetes est essentiel pour définir quelles sont les pods qui seront accessibles via ce Service. Quand nous créons un Service, Kubernetes utilise le selector pour trouver tous les pods qui correspondent aux critères définis, et il va diriger le trafic vers ces pods. Dans le fichier **nginx_service.yml**, le selector est configuré pour sélectionner les pods qui ont le label *app: web*.

- Que permet de faire un **Service** de type `NodePort`?

Un service de type NodePort expose le service sur l'IP de chaque nœud sur un port statique (le NodePort). Un service ClusterIP, vers lequel le service NodePort est automatiquement créé. Il est possible de contacter le service NodePort, depuis l'extérieur du cluster, en demandant <NodeIP>: <NodePort>.

Détectez quel port est exposé sur les nœuds pour atteindre le service

Le service est accessible à l'intérieur du cluster via le port 80 et à l'extérieur du cluster via le port 31199 sur chaque nœud du cluster.

```
ubuntu@p1907992-controlepane:~$ kubectl get services
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.43.0.1      <none>        443/TCP        27h
nginx-service   NodePort    10.43.10.187   <none>        80:31199/TCP   58s
```

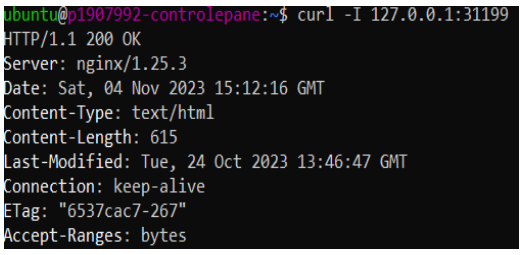
- Quelles adresses sont affichées dans la liste des ENDPOINTS?

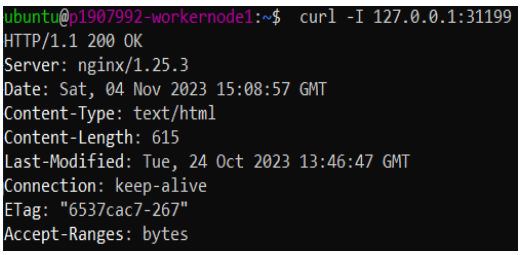
```
ubuntu@p1907992-controlepane:~$ kubectl get endpoints
NAME            ENDPOINTS                                              AGE
kubernetes      192.168.246.13:6443                                    27h
nginx-service   10.42.1.5:80,10.42.1.6:80,10.42.1.7:80 + 3 more...     8m6s
```

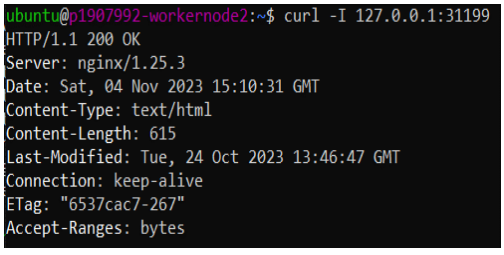
Voici les 3 autres adresses qui ne sont pas affichées à cause de la limitation de la largeur de la console : 10.42.2.7, 10.42.2.8, 10.42.2.9.

Les adresses IP listées dans la section des endpoints du service nginx-service sont les mêmes que celles des pods créées avec le déploiement nginx-deployment. En effet, le Service nginx-service est configuré pour sélectionner les pods avec le label app=web.

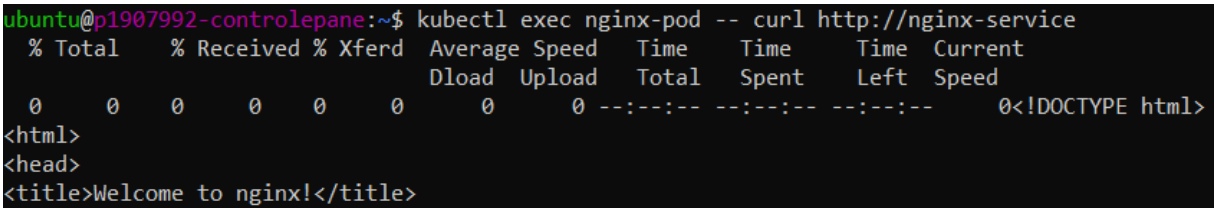
Vérifiez que le déploiement est bien accessible depuis n'importe quel nœud du cluster







- Quelle commande avez-vous utilisée pour effectuer la requête HTTP avec `curl` à partir du **Pod** `nginx-pod`? Que pouvez-vous conclure?

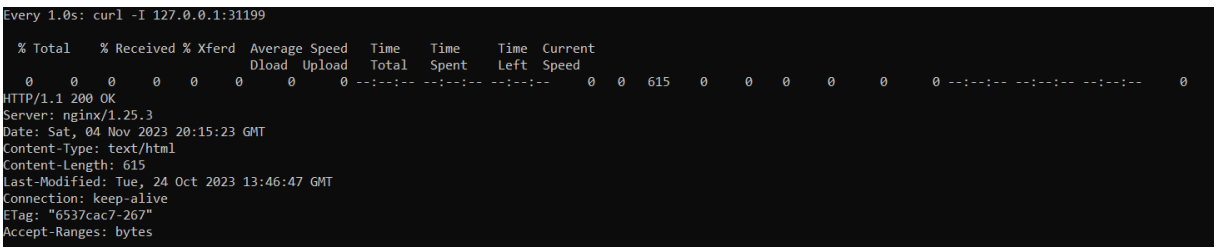


La commande retourne bien le code HTML de la page par défaut de nginx. Le Service nginx-service est correctement configuré pour diriger le trafic vers les Pods de l'ensemble nginx-deployment.

5. Rollbacks

Suivez le processus de Rollback

Nous pouvons en conclure que le processus de Rollback du déploiement nginx-deployment s'est déroulé avec succès et sans interruption de service. La commande curl renvoie des réponses HTTP 200 OK en continue, ce qui indique que le service est resté opérationnel pendant le processus de Rollback. La réponse de curl montre aussi un changement dans la version du serveur Nginx de 1.16.0 à 1.25.3. Cela suggère que les Pods sont revenus à une version précédente, ce qui est cohérent avec un Rollback.



Récupérez l'historique du déploiement

- Affichez les détails de la révision 2 du **Deployment**. Quelle commande utiliserez-vous ?

```
ubuntu@p1907992-controlepane:~$ kubectl rollout history deployment nginx-deployment --revision=2
deployment.apps/nginx-deployment with revision #2
Pod Template:
  Labels:       app=web
        pod-template-hash=859474fc96
  Containers:
   nginx:
    Image:      nginx:1.16.0
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

Nous observons qu'une nouvelle révision (4) est créée. En effet, Dans Kubernetes, chaque fois qu'un rollback est effectué, une nouvelle révision est créée pour refléter cet état. C'est pourquoi après le rollback vers la révision 2, la révision 2 n'est plus visible dans l'historique, mais plutôt la révision 4, qui représente maintenant l'état du déploiement qui était auparavant celui de la révision 2. Cela permet de garder une trace claire de toutes les modifications et rollbacks effectués dans le système.

6. Volumes

- Que signifie l'accès mode "ReadWriteOnce"?

Le volume peut être monté en lecture-écriture par un seul nœud. Le mode d'accès ReadWriteOnce peut toujours permettre à plusieurs pods d'accéder au volume lorsque les pods s'exécutent sur le même nœud.

- Quel est le statut du PV après création ?

Il est dans l'état **Available.** Cela signifie que le Persistent Volume a été créé et est prêt à être lié à un Persistent Volume Claim (PVC). Un PV dans l'état **Available** est un volume qui n'est pas encore lié à un PVC.

```
ubuntu@p1907992-controlepane:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
task-pv-volume   200Mi      RWO            Retain           Available           manual                  3m25s
```

- Quelle est la stratégie de rétention de volume persistant créée et que signifie-t-elle ?

La stratégie de rétention **Retain** signifie que lorsque le PersistentVolumeClaim lié à ce PV est supprimé, le PV ne sera pas automatiquement supprimé par Kubernetes. Au lieu de cela, il restera dans le cluster jusqu'à ce qu'il soit manuellement supprimé. Cette politique est utile pour conserver les données même après la suppression du PVC, permettant à un administrateur de les récupérer ou de les gérer manuellement.

Visualisez l'état des Persistant Volumes et Persistant Volume Claims

- Quel est le statut du PV et du PVC après la création de la claim ?

Le statut Bound du PV indique qu'il est désormais lié à un PVC spécifique. Le champ CLAIM montre que le PV est maintenant lié au PVC task-pv-claim dans le namespace default.
Le statut Bound du PVC indique que la demande de volume a été satisfaite et qu'elle est liée à un volume spécifique, dans ce cas task-pv-volume.

```
ubuntu@p1907992-controlepane:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                  STORAGECLASS   REASON   AGE
task-pv-volume   200Mi      RWO            Retain           Bound    default/task-pv-claim   manual                  15m
ubuntu@p1907992-controlepane:~$ kubectl get pvc
NAME           STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim   Bound    task-pv-volume   200Mi      RWO            manual         28s
```

Est-ce que deux claims peuvent utiliser le même volume persistant ?

Création d'un deuxième PVC :

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: task-pv-claim-2
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

Non, deux claims ne peuvent pas utiliser le même volume persistant. L'état Pending signifie que le PVC est en attente d'un PV disponible correspondant à ses critères de demande, mais puisque le PV est déjà lié à un autre PVC, il ne peut pas être utilisé par task-pv-claim-2.

```
ubuntu@p1907992-controlepane:~$ kubectl get pvc
NAME             STATUS    VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
task-pv-claim    Bound     task-pv-volume   200Mi      RWO            manual         13m
task-pv-claim-2  Pending                                               manual         4s
```

Trouvez un moyen de vérifier que le volume persistent fonctionne correctement

- Comment l'avez-vous vérifié ?

Dans le fichier **pvc_pod.yml**, il est indiqué le volume persistant (fourni par le PVC task-pv-claim) doit être monté dans le Pod mongodb-pvc au chemin /data/db. Ainsi, si la commande **ls /data/db** liste des fichiers, cela suggère que MongoDB a réussi à accéder au volume.

Voici une partie du résultat de la commande :

```
ubuntu@p1907992-controlepane:~$ kubectl exec mongodb-pvc -- ls /data/db
WiredTiger
WiredTiger.lock
WiredTiger.turtle
WiredTiger.wt
WiredTigerHS.wt
_mdb_catalog.wt
collection-0--1687831510590482823.wt
collection-2--1687831510590482823.wt
collection-4--1687831510590482823.wt
```

7. Variables d'environnement

Visualisez les logs du pod

- Que voyez-vous dans les logs du Pod?

La commande dans le fichier env_var_pod.yml était :

```
command: ['sh', '-c', 'echo "Username: $USERNAME" && sleep 99999']
env:
- name: USERNAME
  value: administrator
```

Cela signifie que le shell dans le conteneur exécute la commande echo "Username: $USERNAME" qui imprime la valeur de la variable d'environnement USERNAME (dans ce cas, administrator). Après cela, le conteneur exécute sleep 99999 pour rester actif, ce qui vous permet d'accéder aux logs et de voir la sortie. Cela confirme que l'environnement du conteneur a été bien configuré.

```
ubuntu@p1907992-controlepane:~$ kubectl logs env-var-pod
Username: administrator
```

8. Secrets

Modifiez le secret

- Quel est le contenu du fichier `secret.yml` après les modifications?

D'abord, on encode « Lyon 1 » en base64 :

```
ubuntu@p1907992-controlepane:~$ echo -n 'Lyon1' | base64
THlvbjE=
```

Le fichier secret.yml devient :

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: 42-secret
data:
  username: THlvbjE=
  password: THlvbjE=
```

- Modifiez la description du **Pod** afin que le répertoire `/secret` soit monté en tant que volume du secret et que la variable d'environement `SECRET_USERNAME` contienne la valeur `username` du secret.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-secret
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'ls -al /secret && echo "Username: $SECRET_USERNAME" && sleep 99999']
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: 42-secret
          key: username
    volumeMounts:
    - name: secret-volume
      mountPath: /secret
  volumes:
  - name: secret-volume
    secret:
      secretName: 42-secret
```

Visualisez les logs du pod

```
ubuntu@p1907992-controlepane:~$ kubectl logs pod-with-secret
total 4
drwxrwxrwt    3 root     root          120 Nov  5 15:46 .
drwxr-xr-x    1 root     root         4096 Nov  5 15:46 ..
drwxr-xr-x    2 root     root           80 Nov  5 15:46 ..2023_11_05_15_46_36.1750130255
lrwxrwxrwx    1 root     root           32 Nov  5 15:46 ..data -> ..2023_11_05_15_46_36.1750130255
lrwxrwxrwx    1 root     root           15 Nov  5 15:46 password -> ..data/password
lrwxrwxrwx    1 root     root           15 Nov  5 15:46 username -> ..data/username
Username: Lyon1
```

- Que voyez-vous dans les logs du Pod?

- Le résultat de la commande ls -al /secret montre le contenu du répertoire /secret :
- La commande echo "Username: $SECRET_USERNAME" affiche Username: Lyon1, ce qui indique que la variable d'environnement SECRET_USERNAME a été correctement définie à partir du secret.

- Que contiennent les fichiers du répertoire `/secret`?

Il y a plusieurs entrées :

- . et .. sont des références standards au répertoire courant et au répertoire parent, respectivement.
- ..2023_11_05_15_46_36.1750130255 est un répertoire créé par Kubernetes pour stocker les données du secret.
- ..data est un lien symbolique qui pointe vers le répertoire actuel contenant les données du secret.
- des liens symboliques pour password et username. Les fichiers password et username contiennent les valeurs des secrets.

```
ubuntu@p1907992-controlepane:~$ kubectl exec pod-with-secret -- cat /secret/username
Lyon1ubuntu@p1907992-controlepane:~$
ubuntu@p1907992-controlepane:~$ kubectl exec pod-with-secret -- cat /secret/password
Lyon1ubuntu@p1907992-controlepane:~$
```

9. Init containers

Surveillez le déploiement du Pod

- Sur quel nœud **Worker** le **Pod** a-t-il été lancé ?

Il a été lancé sur le nœud Worker avec l'adresse IP **192.168.246.19** qui est le Worker 2.

```
ubuntu@p1907992-controlepane:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP           NODE              NOMINATED NODE   READINESS GATES
env-var-pod                         1/1     Running   0          4h49m   10.42.1.16   192.168.246.19    <none>           <none>
mongodb-pvc                         1/1     Running   0          6h24m   10.42.2.14   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-749qt   1/1     Running   0          23h     10.42.1.14   192.168.246.19    <none>           <none>
nginx-deployment-859474fc96-lvhlm   1/1     Running   0          23h     10.42.2.13   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-q679c   1/1     Running   0          23h     10.42.1.15   192.168.246.19    <none>           <none>
nginx-pod                           1/1     Running   0          2d8h    10.42.2.6    192.168.246.14    <none>           <none>
nginx-pod-with-init                 1/1     Running   0          93s     10.42.1.17   192.168.246.19    <none>           <none>
pod-with-secret                     1/1     Running   0          4h27m   10.42.2.15   192.168.246.14    <none>           <none>
```

Testez le Pod deploye precedement avec la commande `curl` depuis le nœud Worker

- Que pouvez-vous constater?

Dans le Worker 2, le fait que "Kubernetes" soit renvoyé suggère que le initContainer a correctement modifié le fichier index.html avant le démarrage du conteneur nginx.

```
ubuntu@p1907992-workernode2:~$ curl 127.0.0.1:8081
Kubernetes
```

Dans le Worker 1, on a Connection refused. Cela est dû au fait que le Pod **`nginx-pod-with-init`** n'est pas en cours d'exécution sur ce nœud, donc aucune application n'écoute sur le port 8081 de l'adresse locale (127.0.0.1) de ce nœud.

```
ubuntu@p1907992-workernode1:~$ curl 127.0.0.1:8081
curl: (7) Failed to connect to 127.0.0.1 port 8081 after 0 ms: Connection refused
```

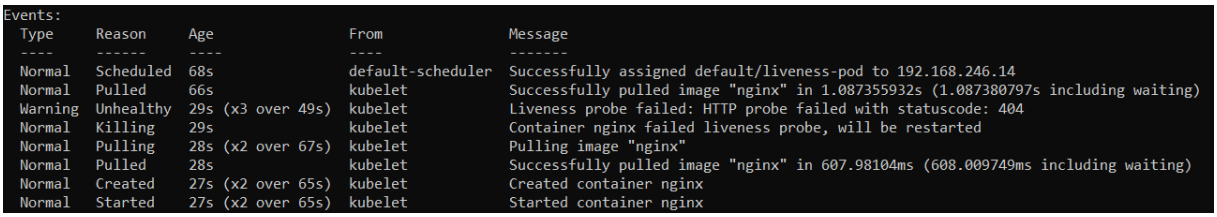
10. Sondes de Liveness et Readiness

Liveness probe

- Que fait Kubernetes en cas d'échec de la Liveness probe?

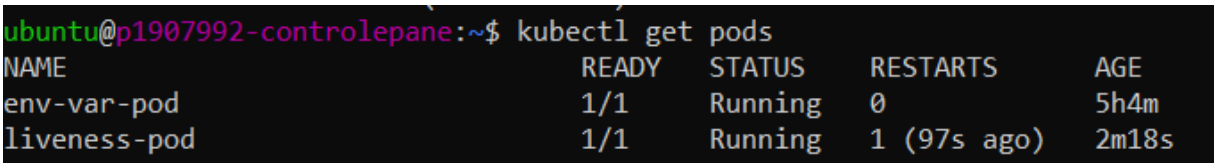
En cas d'échec de la Liveness probe, Kubernetes tente de résoudre le problème en redémarrant le conteneur concerné.

Les événements montrent que la Liveness probe a échoué « Liveness probe failed: HTTP probe failed with statuscode: 404 » et que le conteneur nginx va être redémarré dans le Pod « Container nginx failed liveness probe, will be restarted ».



```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  68s                default-scheduler  Successfully assigned default/liveness-pod to 192.168.246.14
  Normal   Pulled     66s                kubelet            Successfully pulled image "nginx" in 1.087355932s (1.087380797s including waiting)
  Warning  Unhealthy  29s (x3 over 49s)  kubelet            Liveness probe failed: HTTP probe failed with statuscode: 404
  Normal   Killing    29s                kubelet            Container nginx failed liveness probe, will be restarted
  Normal   Pulling    28s (x2 over 67s)  kubelet            Pulling image "nginx"
  Normal   Pulled     28s                kubelet            Successfully pulled image "nginx" in 607.98104ms (608.009749ms including waiting)
  Normal   Created    27s (x2 over 65s)  kubelet            Created container nginx
  Normal   Started    27s (x2 over 65s)  kubelet            Started container nginx
```

Le compteur RESTARTS a augmenté (1 (97s ago)), ce qui indique que le conteneur dans ce Pod a été redémarré par Kubernetes.



```
ubuntu@p1907992-controlepane:~$ kubectl get pods
NAME                                READY   STATUS    RESTARTS      AGE
env-var-pod                         1/1     Running   0             5h4m
liveness-pod                        1/1     Running   1 (97s ago)   2m18s
```

Readiness probe

Etudiez le comportement des Pods avec une sonde Readiness

Surveillez les pods déployés

- Que remarquez-vous?

Pod nginx-good :

- Status: Running et Ready: 1/1.
- Cela signifie que le Pod est en cours d'exécution et que sa Readiness Probe indique qu'il est prêt à accepter le trafic. Dans ce cas, le serveur nginx a démarré rapidement et répond aux requêtes HTTP sur le chemin /.

Pod nginx-slow :

- Status: Running mais Ready: 0/1.
- Ce Pod est en cours d'exécution, mais sa Readiness Probe indique qu'il n'est pas encore prêt. Cela est dû à la commande sleep 300 dans sa spécification, qui retarde le démarrage du serveur nginx pendant 300 secondes.

```
ubuntu@p1907992-controlepane:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS      AGE     IP           NODE              NOMINATED NODE   READINESS GATES
env-var-pod                         1/1     Running   0             6h20m   10.42.1.16   192.168.246.19    <none>           <none>
liveness-pod                        1/1     Running   1 (77m ago)   78m     10.42.2.16   192.168.246.14    <none>           <none>
mongodb-pvc                         1/1     Running   0             7h55m   10.42.2.14   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-749qt   1/1     Running   0             25h     10.42.1.14   192.168.246.19    <none>           <none>
nginx-deployment-859474fc96-lvhlm   1/1     Running   0             25h     10.42.2.13   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-q679c   1/1     Running   0             25h     10.42.1.15   192.168.246.19    <none>           <none>
nginx-good                          1/1     Running   0             2m33s   10.42.1.18   192.168.246.19    <none>           <none>
nginx-pod                           1/1     Running   0             2d10h   10.42.2.6    192.168.246.14    <none>           <none>
nginx-pod-with-init                 1/1     Running   0             92m     10.42.1.17   192.168.246.19    <none>           <none>
nginx-slow                          0/1     Running   0             2m33s   10.42.2.17   192.168.246.14    <none>           <none>
pod-with-secret                     1/1     Running   0             5h58m   10.42.2.15   192.168.246.14    <none>           <none>
```

Surveillez la liste des endpoints du service

- Que remarquez-vous?

On remarque que le service « nginx-readiness » contient les adresses IP 10.42.1.18:80 et 10.42.2.17:80 qui correspondent respectivement aux Pods nginx-good et nginx-slow. Malgré le fait que le Pod nginx-slow n'était pas prêt au moment de la dernière vérification (READY était 0/1), il apparaît toujours dans la liste des endpoints. Cela indique que le service nginx-readiness inclut tous les Pods correspondants, peu importe leur état de disponibilité (readiness).

```
ubuntu@p1907992-controlepane:~$ kubectl get endpoints
NAME              ENDPOINTS                                  AGE
kubernetes        192.168.246.13:6443                        2d10h
nginx-readiness   10.42.1.18:80,10.42.2.17:80                8m36s
nginx-service     10.42.1.14:80,10.42.1.15:80,10.42.2.13:80  31h
```

Le service répond-il aux requêtes?

- Comment pouvez-vous expliquer un tel comportement?

Le service répond bien aux requêtes.

Bien que le Pod `nginx-slow` ait été configuré pour démarrer lentement (avec un délai initial de 300 secondes avant de lancer Nginx), le test avec `curl` montre que le service n'a pas dirigé de trafic vers ce Pod tant qu'il n'était pas prêt.

Le Pod `nginx-good`, qui avait une Readiness probe configurée pour répondre immédiatement, est probablement celui qui a répondu à votre requête.

```
ubuntu@p1907992-controlepane:~$ curl 127.0.0.1:31683
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

Attendez 5 minutes et réétudiez le comportement des Pods et la liste des endpoints du service

- Que remarquez-vous? Comment pouvez-vous expliquer un tel comportement?

Le Pod nginx-slow est désormais dans l'état Running et READY 1/1.
Il était configuré pour démarrer lentement (avec une commande sleep 300), ce qui signifie qu'il ne serait pas prêt à recevoir du trafic immédiatement. Après la fin du délai de 300 secondes (5 minutes), le serveur nginx dans ce Pod démarre, permettant à la Readiness probe de réussir.

```
ubuntu@p1907992-controlepane:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS      AGE     IP           NODE              NOMINATED NODE   READINESS GATES
env-var-pod                         1/1     Running   0             6h37m   10.42.1.16   192.168.246.19    <none>           <none>
liveness-pod                        1/1     Running   1 (94m ago)   95m     10.42.2.16   192.168.246.14    <none>           <none>
mongodb-pvc                         1/1     Running   0             8h      10.42.2.14   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-749qt   1/1     Running   0             25h     10.42.1.14   192.168.246.19    <none>           <none>
nginx-deployment-859474fc96-lvhlm   1/1     Running   0             25h     10.42.2.13   192.168.246.14    <none>           <none>
nginx-deployment-859474fc96-q679c   1/1     Running   0             25h     10.42.1.15   192.168.246.19    <none>           <none>
nginx-good                          1/1     Running   0             19m     10.42.1.18   192.168.246.19    <none>           <none>
nginx-pod                           1/1     Running   0             2d10h   10.42.2.6    192.168.246.14    <none>           <none>
nginx-pod-with-init                 1/1     Running   0             109m    10.42.1.17   192.168.246.19    <none>           <none>
nginx-slow                          1/1     Running   0             19m     10.42.2.17   192.168.246.14    <none>           <none>
pod-with-secret                     1/1     Running   0             6h15m   10.42.2.15   192.168.246.14    <none>           <none>
```

11. Création d'un Ingress

```
ubuntu@p1907992-controlepane:~$ ping bahadi.tiw-csv.os.univ-lyon1.fr.
PING bahadi.tiw-csv.os.univ-lyon1.fr (192.168.246.14) 56(84) bytes of data.
64 bytes from 192.168.246.14 (192.168.246.14): icmp_seq=1 ttl=64 time=0.817 ms
64 bytes from 192.168.246.14 (192.168.246.14): icmp_seq=2 ttl=64 time=0.556 ms
64 bytes from 192.168.246.14 (192.168.246.14): icmp_seq=3 ttl=64 time=0.744 ms
64 bytes from 192.168.246.14 (192.168.246.14): icmp_seq=4 ttl=64 time=0.809 ms
```

Visualisez la liste des Ingress

- Quelles adresses se trouvent dans le colonne `ADDRESS` ? Si vous n'avez rien dans cette colonne, attendez un peu et réexécutez la commande.

Les adresses des 2 Workers.

```
ubuntu@p1907992-controlepane:~$ kubectl get ingress
NAME            CLASS   HOSTS   ADDRESS                          PORTS   AGE
nginx-ingress   nginx   *       192.168.246.14,192.168.246.19    80      58s
```

Essayez d'accéder au Service en utilisant le nom DNS précédemment créé à parir de votre navigateur ou en executant la commande `curl`

- Que pouvez-vous constater ?

La commande curl retourne la page par défaut de Nginx. Cela indique que l'Ingress dirige correctement le trafic vers le service Nginx.

```
ubuntu@p1907992-controlepane:~$ curl bahadi.tiw-csv.os.univ-lyon1.fr.
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

B. Un déploiement plus complexe

12. Service Redis

PersistantVolume

```yaml
kind: PersistentVolume
apiVersion: v1
metadata:
  name: redis-pv
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 500Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/redis"
```

PersistantVolumeClaim

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: redis-pvc
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

On voit que le PVC redis-pvc est dans l'état Pending car le PV redis-pv est lié au 2ème PVC task-pv-claim2 crée précédemment pour le test dans la partie Volume. Il faut donc supprimer le PVC task-pv-claim2 avec kubectl delete pvc task-pv-claim-2.

```
ubuntu@p1907992-controlepane:~$ kubectl get pvc
NAME             STATUS    VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
redis-pvc        Pending                                               manual         112m
task-pv-claim    Bound     task-pv-volume   200Mi      RWO            manual         6d2h
task-pv-claim-2  Bound     redis-pv         500Mi      RWO            manual         6d1h
ubuntu@p1907992-controlepane:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
redis-pv         500Mi      RWO            Retain           Bound    default/task-pv-claim-2 manual                 112m
task-pv-volume   200Mi      RWO            Retain           Bound    default/task-pv-claim   manual                 6d2h
```

L'état du pv redis-pv devient Released. Il faut maintenant supprimer la référence au PVC pvc-task-claim2 avec kubectl patch pv redis-pv -p '{"spec":{"claimRef": null}}'.

```
ubuntu@p1907992-controlepane:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM                   STORAGECLASS   REASON   AGE
redis-pv         500Mi      RWO            Retain           Released   default/task-pv-claim-2 manual                 124m
```

Maintenant, les 2 états sont Bound :

```
ubuntu@p1907992-controlepane:~$ kubectl get pvc
NAME           STATUS   VOLUME           CAPACITY   ACCESS MODES   STORAGECLASS   AGE
redis-pvc      Bound    redis-pv         500Mi      RWO            manual         127m
task-pv-claim  Bound    task-pv-volume   200Mi      RWO            manual         6d2h
ubuntu@p1907992-controlepane:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS   REASON   AGE
redis-pv         500Mi      RWO            Retain           Bound    default/redis-pvc       manual                  129m
task-pv-volume   200Mi      RWO            Retain           Bound    default/task-pv-claim   manual                  6d2h
```

Secret

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: redis-secret
data:
  username: cmVkaXNwYXNzd29yZA==
  password: cmVkaXNwYXNzd29yZA==
```

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deployment
spec:
  selector:
    matchLabels:
      app: redis
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
    spec:
      volumes:
      - name: redis-config
        emptyDir: {}
      - name: redis-data
        persistentVolumeClaim:
            claimName: redis-pvc
      initContainers:
      - name: redis-config-init
        image: busybox
        command: ["sh", "-c", "echo requirepass $PASSWORD > /etc/redis/redis.conf"]
        env:
        - name: PASSWORD
          valueFrom:
            secretKeyRef:
                name: redis-secret
                key: password
        volumeMounts:
        - name: redis-config
          mountPath: /etc/redis
      containers:
      - name: redis
        image: redis:7.0.2
        command: ["redis-server", "/etc/redis/redis.conf"]
        volumeMounts:
          - name: redis-config
            mountPath: /etc/redis
          - name: redis-data
            mountPath: /data
        livenessProbe:
          exec:
            command: ["redis-cli", "ping"]
          initialDelaySeconds: 5
          periodSeconds: 5
```

Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: redis-service
spec:
  type: ClusterIP
  selector:
    app: redis
  ports:
    - protocol: TCP
      port: 6379
      targetPort: 6379
```

Vérification du bon fonctionnement du service Redis

```
ubuntu@p1907992-controlepane:~$ kubectl exec -it busybox-74c6795-g92gw -- sh
/ #
/ # telnet redis-service 6379
Connected to redis-service
GET counter
-NOAUTH Authentication required.
AUTH redispassword
+OK
MGET *
*1
$-1
QUIT
+OK
Connection closed by foreign host
```

13. Service Counter

Deployment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: counter-deployment
spec:
  selector:
    matchLabels:
      app: counter
  replicas: 3
  template:
    metadata:
      labels:
        app: counter
    spec:
      volumes:
      - name: counter-app
        emptyDir: {}
      - name: redis-secret
        secret:
          secretName: redis-secret
      initContainers:
      - name: counter-app-init
        image: alpine
        command: ['wget', 'https://forge.univ-lyon1.fr/vladimir.ostapenco/counter-application/-/raw/main/index.php', '-O', '/var/www/html/index.php']
        volumeMounts:
        - name: counter-app
          mountPath: /var/www/html
      containers:
      - name: counter-app
        image: vladost/php:7.2-apache-redis
        env:
          - name: REDIS_HOST
            value: redis-service
        volumeMounts:
          - name: counter-app
            mountPath: /var/www/html
          - name: redis-secret
            mountPath: /credentials
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          periodSeconds: 5
```

Service

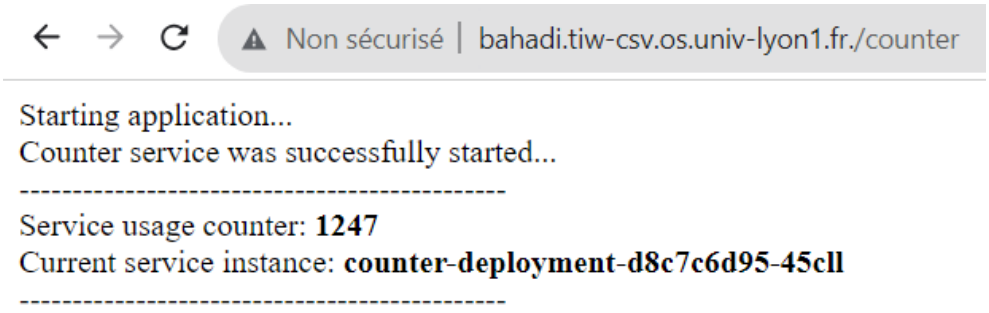
```yaml
apiVersion: v1
kind: Service
metadata:
  name: counter-service
spec:
  selector:
    app: counter
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Ingress

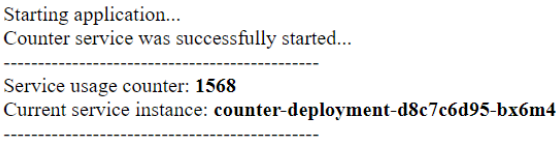
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: counter-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - http:
        paths:
          - path: /counter
            pathType: Prefix
            backend:
              service:
                name: counter-service
                port:
                  number: 80
```

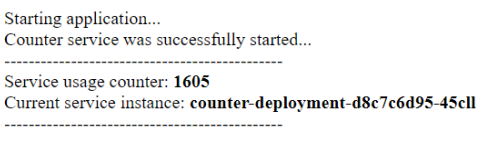
Vérification de l'application

Vérifier le bon fonctionnement de l'application



Surveillez la valeur du compteur, attendez une minute et mettez à jour la page.





Le compteur augmente grâce à la Liveness Probe. On lui dit de requeter chaque 5 secondes à /80.

Cela est dû au nombre de réplicas : 6, le temps augmente donc de 6*12.

```
livenessProbe:
  httpGet:
    path: /
    port: 80
  initialDelaySeconds: 5
  periodSeconds: 5
```

Le changement de l'instance de Pod montre que la charge est répartie entre plusieurs réplicas.

Mise à l'échelle du déploiement

- Quelle commande avez-vous utilisé ?

On utilise la commande **kubectl scale deployment counter-deployment --replicas=6.**

```
ubuntu@p1907992-controlepane:~$ kubectl scale deployment counter-deployment --replicas=6
deployment.apps/counter-deployment scaled
ubuntu@p1907992-controlepane:~$ kubectl get deployments
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
busybox              1/1     1            1           77m
counter-deployment   3/6     6            3           50m
nginx-deployment     3/3     3            3           7d3h
redis-deployment     1/1     1            1           146m
```

C. BONUS: Déploiement du cluster Kubernetes avec kubeadm

kubeadm est un outil qui fournit des commandes pour initialiser, configurer et gérer les composants du cluster Kubernetes. Il simplifie grandement le processus de mise en place et de maintenance d'un cluster Kubernetes.

Le kubelet est un agent qui s'exécute sur chaque nœud du cluster. Il s'assure que les conteneurs sont en cours d'exécution dans un Pod.

kubectl est une ligne de commande pour interagir avec l'API server de Kubernetes. Il permet aux utilisateurs de déployer des applications, d'inspecter et de gérer les ressources du cluster, et de visualiser les logs.

- Quelle commande avez-vous utilisée pour initialiser le nœud Control Plane ?

```
ubuntu@p1907992-controlepane:~$ kubeadm init --cri-socket unix:///var/run/cri-dockerd.sock --pod-network-cidr=10.244.0.0/16
```

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.246.13:6443 --token 6hciyk.vhdjwc4ekgxl90rk \
        --discovery-token-ca-cert-hash sha256:acea0b1817f5d5c414498a555f6697ff9e5e605a694605fe0a55883865982a46
```

- Quelle commande avez-vous utilisée sur chaque nœud Worker pour rejoindre le cluster ?

sudo kubeadm join 192.168.246.13:6443 --token 0877p7.he70ioh9ux4gddb4 --discovery-token-ca-cert-hash sha256:bfd4400423c8641332f8c41dcab17130d7c54949b701c4af8361be10ccbf408e --cri-socket unix:///var/run/cri-dockerd.sock

```
This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

- Déployez le CNI (Container Network Interface) `flannel`

```
ubuntu@p1907992-controlepane:~$ kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml

namespace/kube-flannel created
serviceaccount/flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
```

- Verifiez l'etat du cluser avec la commande `kubectl get nodes`

```
ubuntu@p1907992-controlepane:~$ kubectl get nodes
NAME                     STATUS   ROLES           AGE     VERSION
p1907992-controlepane    Ready    control-plane   20m     v1.28.4
p1907992-workernode1     Ready    <none>          5m8s    v1.28.4
p1907992-workernode2     Ready    <none>          5m24s   v1.28.4
```

- Déployez les objets `nginx-deployment` et `nginx-service` vus précédemment et vérifiez s'ils fonctionnent correctement.

```
ubuntu@p1907992-controlepane:~$ kubectl apply -f nginx_deployment.yml
deployment.apps/nginx-deployment created
ubuntu@p1907992-controlepane:~$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           28s
```

```
ubuntu@p1907992-controlepane:~$ kubectl apply -f nginx_service.yml
service/nginx-service created
ubuntu@p1907992-controlepane:~$ kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        24m
nginx-service   NodePort    10.107.157.228   <none>        80:32455/TCP   6s
```