Fase 4 - Activitat 10.1: Orquestració de contenidors amb Kubernetes amb microk8s

0- Identificació del grup i activitat:

Curs: ASIX2
Projecte: PJ9 DevOps i Cloud Computing
Fase: 4
Activitat: 10.1
Grup/Individual: Individual
Membres/Alumne:

1- Introducció i objectius de l'activitat 10.1

a) Conceptes bàsics sobre Kubernetes.
b) Creació dels nodes del cluster amb el programari de contenidors i Kubernetes
c) Creació del cluster de Kubernetes
d) Desplegament d'una aplicació sobre el cluster
e) Gestió del cluster i l'aplicació.

2- Conceptes de Kubernetes

a) Què és Kubernetes?

Sobre **Kubernets** podem dir que:

- És una eina open-source d'orquestració de contenidors.
- Amb aquesta eina podem agrupar multiples hosts (ordinadors físics i virtuals) formant un cluster. El cluster semblarà un únic sistema sobre el qual serà fàcil desplegar multiples contenidors.
- El contenidors es distribuiran al llarg dels host sque formen part del cluster.
- El contenidors estaran intercomunicats dins del cluster tot i que formin part d'un host diferent.

b) Quin és el propòsit d'utilitzar Kubernetes?

El propòsit de **Kubernetes** és que, dins d'un entorn de producció a on una aplicació hagi d'estar disponible als usuaris finals, es pugui garantir:

- Poder desplegar i gestionar fàcilment l'aplicació posant en marxa desenes, centenars o milers de contenidors distribuits entre multiples hosts i intercomunicats.
- L'escalabilitat.
- Tolerancia a les fallades: L'aplicació continua funcionant sense interrupció encara que una part del sistema (per exemple, un o més contenidors) falli.
- Alta Disponiblitat: Habilitat de l'apliació de respondre sempre a qualsevol petició reduint al màxim el temps de no disponiblitat.
- Optima utilització de recursos dels hosts del cluster (CPU, RAM, xarxa, espai de disc,....)
- Seamless Updates i Rolling updates → Actualitzacions sense interrupció i temps d'espera zero.
- Rollback → Facilitat de tornar a una versió anterior si falla una actualització sense temps de no disponiblitat.
- Poder implementar polítiques de seguretat d'accés a les aplicacions dins dels contenidors
- Balanceig de carrega

c) Que és microk8s?

Sobre **microk8s** podem dir que:

- És una implementació de kubernetes. N'existeixen altres com per exemple **minikube** o **k3s**.
- Aquesta implementació és una opció interessant perquè proporciona documentació, eines d'instal·lació i eines de gestió de clusters que fan de **microk8s** una opció molt interessant tan per aprendre com per utilitzar **Kubernets** en entorns de producció.
- Està preparada per poder ser desplegat en sistemes i entorns en producció i té certificació CNCF i per tant, compleix totes les especificacions d'una eina d'orquestració de contenidors **Kubernetes**.

d) Quins són els components bàsics de Kubernetes?

Dins de **Kubernetes** poden diferenciar els següents elements:

- El **cluster**: agrupació multiples ordinadors físics i virtuals que treballen coordinadament formant un únic sistema sobre el qual es depleguen i gestionen els contenidors.
- Els **nodes**: cadascún dels ordinador físics o virtuals del cluster sobre els quals es despleguen els contenidors.
- Els **pods**:
  - És la unitat mínima d'instal·lació d'una aplicació. Una aplicació necessita com a mínim un Pod per funcionar. Normalment, quan posem en marxa una aplicació ho fem sobre multiples Pods.
  - Dins d'un pod tenim els contenidors de l'aplicació, la seva xarxa, els volums compartits i el **.yml** de com han de desplagar-se els contenidors (ports, nom de contenidors, imatges, etc...).
  - Dins d'un **node** poden tenir en marxa 1 o més **pods**.
  - Són efimers. Això vol dir que si per qualsevol motiu un Pod deixa de funcionar, automàticament es posa en marxa un altre per substituir-lo.
- El gestor del cluster anomenat **control plane**. Aquest component pot estar situat a un node o distribuït entre diversos nodes de manera que si falla un encara pugui donar servei amb la resta de nodes. El **control plane** gestiona els **nodes** i **pods** del cluster.

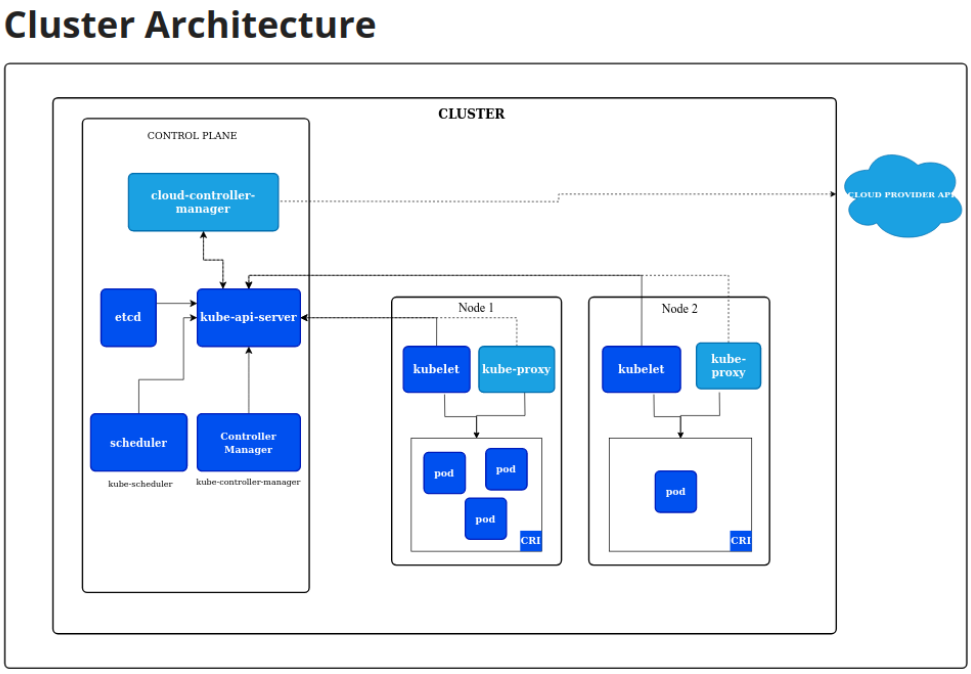
En **kubernetes** el element més petit que podem gestionar és un **pod**. Per exemple:

- Si volem escalar una aplicació haurem de crear nous pods o esborrar-ne.
- Si cal redistribuir la carrega, caldra veure de quin **node** s'eliminen uns **pods** i a quin altre **node** es posen en marxa altres **pods**.

En **kubernetes**:

- Cada **node** controla té les eines necessàries per controlar els seus **pods** i estar en contacte via **control plane** amb la resta del cluster.
- El **control plane** té les eines necessàries per controlar el cluster, els nodes i les comunicacions amb l'exterior del cluster. Generalment es troba dins d'un node però pot distribuir-se entre diferentes nodes del cluster.
- un **worker** és qualsevol node sobre el qual s'executa una aplicació i que no forma part del **control plane.**

Un diagrama bàsic d'un **cluster** és aquest:



e) Què és un Deployment (desplegament)?

- Per mitjà d'un **Deployment** posem en marxa els **Pods** d'una aplicació sobre un cluster de **Kubernetes**. En altres paraules, instal·lem i posem en marxa l'aplicacío utilitzant **Kubernetes**.
- Els **Deployments** no serveixen únicament per posar en marxa els Pods d'una aplicació. També ens permeten indicar les característiques dels **Pods** a posar en marxa. Això vol dir entre d'altres coses, indicar:
  - Imatges de contenidors utilitzades per crear els contenidors dels Pods.
  - El nom dels contenidors dels Pods.
  - Els volums i xarxes dins dels Pods.
  - Ports exposats per fer l'apliació disponible a l'usuari
  - Recursos de memòria RAM i CPU necessaris.
- Els **Deployments** també s'utilitzen per gestionar els Pods. Això vol dir:
  - Proporcionar escalabilitat.
  - Permetre fer actualitzacions en temps real amb temps d'espera mínim o zero.
  - Permetre tornar a version posteriors (rollback) en temps real amb temps d'espera mínim o zero.
  - Proporcionar alta disponibilitat.
  - Monitoritzar l'estat dels Pods
  - Desplegar nous Pods i substituir-ne els antics si el seu estat no és el desitjat.
- Els **Deployments** permeten automatizar la gestió (escalabilitat, actualitzacions, rollbacks, monitorització, substitució) dels Pods i aixó s'eviten molts errors humans, s'allibera temps dels administradors i aquestes tasques es fan més ràpidament.

f) Què és un Service (Servei)?

- Per mitjà d'un **Service** exposarem tots els Pods de l'apliació com si fosin un únic servei de xarxa i també podrem fer balanceig de carrega.
- Amb un **service** fem disponible i accessible l'aplicació als com si fos una única aplicació sobre un únic servidor.

g) Què és un Reverse-Prxoy Ingress?

- Un reverse-proxy de Kubernetes que serà necessari per poder accedir a un **Service** des de fora del cluster.
- Ingress redirecciona les peticions HTTP (o HTTPS) des de fora del cluster al **Service** que exposa l'aplicació a l'exterior del cluster.

h) Què és un manifest?

- És un fitxer en format **YAML** (també por ser JSON) amb l'inventari (o sigui la llista) de tots els Pods, Services i Deployments que es volem desplegar-se sobre un cluster i les seves configuracions.
- Amb un fitxer de manifest i una ordre de Kubernetes simple poden crear i gestionar fàcilment els Pods, Deployments, Services i un Ingress necessaris per fer accessibles aplicacions als usuaris.
- Com sempre, aplicar el manifest és fàcil, allò que és complicat és la creació un fitxer de manifest.
- El fitxers de manifest tenen normalment l'extensió .yml o .yaml.

3- Desplegament d'una aplicació amb Kubernetes

3.1- Creació dels nodes necessaris per establir un cluster de Kubernetes

a) Crea una carpeta de nom **a10**. A continuació, dins de la carpeta **a10** crea una carpeta de nom **pj9f4a10**. Dins de **pj9f4a10** crea un fitxer **Vagrantfile** amb aquest contingut:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# -*- mode: ruby -*-
# vi: set ft=ruby :

#################################
#### Definició de variables ####
#################################

IMATGE_BOX_NODES = "???????"
PROVIDER = "???????"
NUM_NODES = ???????
NOM_BASE_NODES="???????"
NOM_DOMINI_NODES="???????"
MEMORIA_RAM_NODES = ???????
NUM_CPUS_NODES = ???????
TARGETA_XARXA="???????"

################################################
#### Definició de les màquines. Provision ####
################################################

Vagrant.configure("2") do |config|

 ##############################
 #### Creació dels Nodes ####
 ##############################
 (1..NUM_NODES).each do |i|
    config.vm.define NOM_BASE_NODES+"#{i}" do |node|
       node.vm.box = IMATGE_BOX_NODES
       node.vm.hostname = NOM_BASE_NODES+"#{i}"+"."+NOM_DOMINI_NODES
       node.vm.network "public_network", bridge: TARGETA_XARXA
       node.vm.provider PROVIDER do |prov|
             prov.name = NOM_BASE_NODES+"#{i}"
             prov.memory = MEMORIA_RAM_NODES
             prov.cpus = NUM_CPUS_NODES
             prov.customize ['modifyvm', :id, '--clipboard', 'bidirectional', '--groups', '/KURBERNETES', '--mac-address1', "0800278DC04"+"#{i}"]
          end
    end
 end

 ##############################################################################################################
 #### Programari a instal·lar i instruccions a executar a les màquines virtuals durant la seva creació ####
 ##############################################################################################################
 config.vm.provision "shell", inline: <<-SHELL
   #### Instal·lació d'algunes eines de sistema
   sudo apt-get update -y
   sudo apt-get install -y net-tools whois aptitude git zip unzip curl
   #### Instal·lació de Docker
   #### Instal·lació de microk8s
   SHELL
end
```

b) Ara canviarem els paràmetres de configuració:

- Utilitzarem el box **debian/bookworm64**.
- Treballarem amb **virtualbox** com a eina de virtualització (provider).
- El nom base del sistema i l'identificador dins de VirtualBox dels nodes serà **node**.
- El nom de domini dels nodes serà **fjeclot.net**
- La RAM dels nodes serà **2048MB**
- Les CPUs assignades als nodes seran **2**
- Crearem un **cluster** amb **3** nodes
- Seleccionarem la targeta de xarxa WiFi a partir del seu identificador dins de VirtualBox. Executa l'ordre:
  - Windows (Powershell) → **VBoxManage.exe list bridgedifs | Select-String "Name"** → comprova el nom (Name) de les teves targetes de xarxa des del punt de vista de VirtualBox.
  - Linux → **VBoxManage list bridgedifs | grep ^Name** → comprova el nom (Name) de les teves targetes de xarxa des del punt de vista de VirtualBox.

c) Afegeix a la secció **provision** el programari **Docker**. Després de la línia *# Instal·lació de Docker* dins de **Vagrantfile** escriu:

```
sudo apt-get update -y
sudo apt-get install -y net-tools whois aptitude git zip unzip curl
sudo apt-get -y install apt-transport-https ca-certificates curl gnupg
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker.gpg] \
https://download.docker.com/linux/debian bookworm stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get -y update
sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
sudo gpasswd -a vagrant docker
```

d) Afegeix a la secció **provision** el programari **microk8s**. Després de la línia *# Instal·lació de microk8s* dins de **Vagrantfile** escriu:

```
sudo apt-get install -y snapd
sudo snap install snapd
echo 'export PATH=/snap/bin:$PATH' >> /home/vagrant/.bashrc
source /home/vagrant/.bashrc
sudo snap install microk8s --classic
sudo gpasswd -a vagrant microk8s
sudo  chown -f -R vagrant /home/vagrant/.kube
echo "alias kubectl='microk8s kubectl'" >> /home/vagrant/.bashrc
source /home/vagrant/.bashrc
exit
```

e) Executa:

- **vagrant box update**
- **vagrant up**

i posa en marxa els **3** nodes del cluster amb el programari **docker** i **micro8ks**.

f) Executa **vagrant status** i comprova s'han creat les màquines **node1**, **node2** i **node3**.

g) Comprova que les màquines virtual són vísibles a **VirtualBox** dins d'un grup de nom **KUBERNETES**.

h) Accedeix a les màquina virtual **node1**. Executa:

vagrant ssh node1

i) Comprova dins de **node1** que:

- L'usuari **vagrant** es membre del grups **docker** i **micro8ks**. Executa: **id vagrant**
- El programari **micro8ks** i **docker** està disponible. Executa:
  - **microk8s version** (hauria de sortir `MicroK8s v1.32.9 revision 8511` o posterior)
  - **doker -v** (hauria de sortir `Docker version 29.2.0, build 0b9d198` o posterior)
- Els noms de sistema del **node1**. Executa: **hostname --fqdn**. Els nom han de ser **node1.fjeclot.net**.
- Comprova l'adreça IP i MAC de la interfície **eth1** executant: **ip addr show eht1**
- Comprova l'adreça MAC de la interfície **eth0** executant: **ip addr show eht0**

j) Surt de **node1** i accedeix a **node2**. Fes les mateixes comprovacions que has fet a **node1**. Assegura't que tot funciona correctament i que les adreces **MAC** d'**eth0** i **eth1** són diferents a les del **node1**. Assegura't també que l'adreça IP d'**eth1** és diferent.

k) Surt de **node2** i accedeix a **node3**. Fes les mateixes comprovacions que has fet a **node1**. Assegura't que tot funciona correctament i que les adreces **MAC** d'**eth0** i **eth1** són diferents a les de **node1** i **node2**. Assegura't també que l'adreça IP d'**eth1** és diferent.

3.2- Creació d'un cluster de Kubernetes a partir dels 3 nodes creats a l'apartat anterior

a) Dins de **node1** executarem la següent ordre per descarrgar i habilitar l'add-on **dns** necessari per poder treballar amb els noms del nodes dins del cluster:

microk8s enable dns

Aquesta ordre pot trigar una estona en executar-se.

b) Després executarem:

microk8s add-node

i el resultat serà una cosa similar a això:

```
vagrant@node1:~$ microk8s add-node
From the node you wish to join to this cluster, run the following:
microk8s join 10.0.2.15:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6

Use the '--worker' flag to join a node as a worker not running the control plane, eg:
microk8s join 10.0.2.15:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6 --worker

If the node you are adding is not reachable through the default interface you can use one of the following:
microk8s join 10.0.2.15:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6
microk8s join 192.168.1.47:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6
microk8s join 172.17.0.1:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6
microk8s join fd00::a00:27ff:fe8d:c041:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6
vagrant@node1:~$
```

Ordre a executar en **node2** i **node3** per unir-lo al cluster. L'adreça IP és la interfície eth1. Aquest adreça IP, molt probablament serà diferent en el teu cas.

Ara **node1** és el **control plane** d'un cluster i amb les ordres indicades podem unir altres nodes al cluster.

c) Afegeix **node2** al **cluster** executant dins de **node2** l'ordre indicada a l'apartat **a)** , i el resultat serà similar a això:

```
vagrant@node2:~$ microk8s join 192.168.1.47:25000/87211305ceff5fc2cf42601cc093ce09/e77cfeb02eb6
Contacting cluster at 192.168.1.47
Waiting for this node to finish joining the cluster. .. .. .. ..
Successfully joined the cluster.
vagrant@node2:~$
```

d) Fes el mateix procés per afegir **node3** al cluster. És a dir:
- Genera un nou token dins de **node1** amb l'ordre **micro8ks node-add**
- Executa l'ordre mostrada en el **node1** dins del **node3**.

e) Dins de **node1** executa:

microk8s kubectl get nodes

i el resultat hauria de ser:

```
vagrant@node1:~$ microk8s kubectl get nodes
NAME    STATUS   ROLES    AGE     VERSION
node1   Ready    <none>   75m     v1.32.9
node2   Ready    <none>   6m36s   v1.32.9
node3   Ready    <none>   2m12s   v1.32.9
vagrant@node1:~$
```

Si a la columna **STATUS** surt **Ready** a tots els nodes és que ja tenim un cluster amb **3** nodes.

3.3- Creació d'un Deployment dins d'un cluster de Kubernetes

3.3.1- Creació d'una imatge de Docker per fer el Deployment

a) Dins de **node1** crea una carpeta de nom **projectes**. Dins de la carpeta **projectes** crea una carpeta de nom **ppv**. Dins de la carpeta **ppv** crea una carpeta de nom **codi**.

b) Dins de la carpeta **ppv** descarrega un fitxer **Dockerfile** des de l'adreça URL:

https://raw.githubusercontent.com/asix2pj9/ppv/refs/heads/main/Dockerfile

c) Dins de la carpeta **codi** descarrega els arxius **index.html** i **ppv.php** que es troben a les adreces URL:

https://raw.githubusercontent.com/asix2pj9/ppv/refs/heads/main/codi/index.html

https://raw.githubusercontent.com/asix2pj9/ppv/refs/heads/main/codi/ppv.php

d) Canvia a la línia 34 del fitxer index.html l'autor i escriu en comptes de dacomo el teu nom de compte d'usuari de **Github**.

e) Crea una imatge de contenidor de nom **ippv** i versió **1.0**. Dins del directori **ppv** executa:

docker build -t ippv:1.0 .

f) Comprova que s'ha creat la imatge executant:

docker images

3.3.2- Puja la imatge a Docker Hub

a) Accedeix a **Docker Hub** amb el compte que vas crear a l'activiat **7.2**. Crea un repository per les **imatges** de l'aplicació **ppv** amb les següents caracterísiques:
- Repository name: **ippv**
- Short description: **App ppv (pay per view)**
- Visibility: **Public**

b) Dins del directori **ppv** executa la següent ordre per crear un **tag** de la imatge local **ippv**:**1.0** necessari per poder pujar-la al teu compte de **Docker Hub**:

docker tag ippv:1.0 <nom_usuari_dockerhub>/ippv:1.0

a on has de canviar **<nom_usuari_dockerhub>** pel teu nom d'usuari de **DockerHub** real.

c) Comprova que s'ha creat el tag. Executa:

docker images

d) Fes un **login** al teu compte de Docker Hub:

docker login -u <nom_usuari_dockerhub>

a on has de canviar **<nom_usuari_dockerhub>** pel teu nom d'usuari de **DockerHub** real.

e) Si et pots validar dins de **Docker Hub** sortirà el missatge: **`Login Suceeded`**

f) Ara puja la imatge al dipòsit de **Docker Hub** executant:

docker push <tag_de_la imatge_local_ippv:1.0>

a on **<tag_de_la imatge_local_ippv:1.0>** ha de ser el tag que vas crear a l'apartat **b)**.

g) Refresqueu el vostre dipòsit de **Docker Hub** i comproveu que s'han pujat la imatge **ippv:1.0** dins del dipòsit **ippv** del teu compte.

h) Si s'has pogut pujar la imatge, ara ja pots fer un **logout** del teu compte de Docker Hub:

docker logout

i també surt del teu compte de **Docker Hub** des del navegador.

3.3.3- Creant un manifest i fent un Deployment a partir del manifest

a) Dins de la carpeta **ppv** crea un fitxer de nom **ppv-deployment.yaml**. Dins d'aquest fitxer escriu el següent codi de desplegament de l'aplicació:

```yaml
apiVersion: apps/v1
kind: Deployment # Especificica el recurs a desplegar: Deployment, Service , Pod, etc.
metadata:
     name: depl-ppv # Nom identificador del Deployment i nom base dels Pods en el sistema
spec: #spec del Deployment
     replicas: 3 # Quantitat de Pods a crear quan es fa el desplegament
     selector:
          matchLabels:
               app: pods-ppv # Etiqueta per selecciona els Pods que seran gestionats per aquest Deployment .
     template: # Configuració dels Pods
          metadata:
               labels:
                    app: pods-ppv # Etiquetes assignades als Pods que formaran part del Deployment.
          spec: #spec dels Pods
               containers:
               - name: cppv # Nom base dels contenidors dins del Pod
                 image: <nom_usuari_dockerhub>/ippv:1.0 # Imatge per crear contenidors
                 ports:
                 - containerPort: 80 # Port pel qual s'exposa l'aplicació.
```

NOTES:

- *Canvia <nom_usuari_dockerhub> pel nom del compte de **Dockerhub** utilitzat a l'apartat **3.3.2**.*
- *Això és un fitxer YAML i les sagnies són importants. Si copies i enganxes d'un PDF hauràs de comprovar els espais.*
- *Has d'assegurar-te que els comentaris queden al mateix lloc a on es veuen si copies i enganxes d'un PDF.*

b) Aquest manifest:

- Crearà un **Desployment** de nom **depl-ppv** dins del **namespace** identificat com **ns-ppv**.
- Crearà **3 Pods** de nom base **depl-ppv-**
- Cadascun dels **Pods** tindrà un contenidor de nom **cPpv** creat a partir de la imatge **ippv.1.0**.
- El port **80** de cada contenidor serà exposat a l'exterior per poder fer accessible l'aplicació.

c) Ara farem el desplegament. Executarem dins de la carpeta **ppv** l'ordre:

```
microk8s kubectl apply -f ppv-deployment.yaml
```

i el sistema hauria de mostrar el missatge: `deployment.apps/depl-ppv created`

d) Comprova l'estat del **Deployment**. Executa:

```
microk8s kubectl get deployment
```

i comprova que el resultat és:

```
vagrant@node1:~/projectes/ppv$ microk8s kubectl get deployment
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
depl-ppv   3/3     3            3           61s
```

NOTA: Pot trigar des d'un uns pocs segons a uns minuts en mostrar el resultat perquè tots els Pods i els seus components necessiten un temps per ser creats i també perquè pot ser necessari descarregar imatges des de Docker Hub.

e) Finalment, comprova que l'estat dels **3 Pods** que s'han creat. Executa:

```
microk8s kubectl get pods
```

i comprova que el resultat és semblant a aquest:

```
vagrant@node1:~/projectes/ppv$ microk8s kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
depl-ppv-697f8bdb5-8287p    1/1     Running   0          4m21s
depl-ppv-697f8bdb5-f7f6b    1/1     Running   0          4m21s
depl-ppv-697f8bdb5-wb82b    1/1     Running   0          4m21s
```

f) Si els resultats són correctes (tots el **Pods** estan **Running**) llavors, ja has fet un **Deployment**, o sigui has instal·lat i posat en marxa una aplicació utilitzant **Kubernetes**. Però encara l'aplicació no està disponible per poder ser utilitzada pels usuaris. Per fer disponible l'aplicació als usuaris, primer de tot cal crear un **Service** associat al **Deployment**.

3.4 Creació d'un Service a partir del Deployment de l'aplicació

a) La creació d'un **Service** de Kubernetes és un pas necessari fer disponible una aplicació desplegada amb Kubernetes i que permet exposar tots els Pods de l'apliació com si fosin un únic servei de xarxa.

b) Dins de la la carpeta **ppv** crea un fitxer per crear un servei associat al desplegament fet a l'apartat anterior de nom **ppv-service.yaml**. Dins d'aquest fitxer escriu el següent codi:

```
apiVersion: v1
kind: Service  #Especificica el recurs a desplegar: Deployment, Service , Pod, etc.
metadata:
     name: serv-ppv # Nom del service que s'ha de crear
spec:
     type: NodePort
     selector:
          app: pods-ppv # Etiquetes identificadores dels Pods dins de ppv-deployment.yaml
     ports:
     - port: 4000 # Port del Service  que es redirigirà cap el port dels Pods.
       targetPort: 80 # Port pel qual els Pods exposen l'aplicació.
       nodePort: 30000 # Port creat dins de la màquina que es redirigirà cap el port del Service.
```

NOTES:

- *Això és un fitxer YAML i les sagnies són importants. Si copies i enganxes d'un PDF hauràs de comprovar els espais.*
- *Has d'assegurar-te que els comentaris queden al mateix lloc a on es veuen si copies i enganxes d'un PDF.*

Amb aquesta configuració crearem un **Service** amb aquestes característiques:

- Nom del servei: **serv-ppv**
- Pods utilitzats pel servei → Els pods **pods-ppv** definits dins de **ppv-deployment.yaml**.
- Redirecció de ports:

Port del node		Port del Service		Port dels Pods
30000/tcp	→	4000/tcp	→	80/tcp

Una connexió des de l'exterior al port **30000/tcp** a un node del cluster serà redirigit al port **4000/tcp** del Service **ppv-serv** que el redigirirà al port **80/tcp** del Pod seleccionat a cada moment utilitzant el balanceig de carrega que ens proporciona el **Service**.

c) Ara crearem el **Service**. Executa dins de la carpeta **ppv** l'ordre:

```
microk8s kubectl create -f ppv-service.yaml
```

i el sistema hauria de mostrar el missatge: **`service/serv-ppv created`**

d) Comprova que l'estat del **Service**. Executa:

```
microk8s kubectl get services
```

i comprova que el resultat és semblant a aquest:

```
vagrant@node1:~/projectes/ppv$ microk8s kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.152.183.1     <none>        443/TCP          3h12m
serv-ppv     NodePort    10.152.183.131   <none>        4000:30000/TCP   36s
```

e) Si els resultats són correctes, llavors ja has creat un **Setvice**, pel **Deployment** creat a l'apartat anterior. Però encara l'aplicació no està disponible per poder ser utilitzada pels usuaris des de fora del cluster de **Kubernetes**. Per fer disponible l'aplicació als usuaris fora del cluster, cal posar en marxa un servei de **Reverse-Proxy** dins dels nodes del cluster.

3.5- Utilització d'un reverse-proxy Ingress per accedir a l'aplicació des de fora del cluster

a) Per accedir a l'aplicació des de fora del cluster ens cal un servidor **reverse-proxy** que **redireccioni** les peticions HTTP (o HTTPS) des de fora del cluster al servei **serv-ppv** dins del cluster. **Kubernetes** en proporciona un servei de **reverse-proxy** anomenat Ingress que haurem d'habilitar i configurar per fer les redireccions.

b) Habilita **Ingress**. Executa dins de **ppv** l'ordre:

```
microk8s enable ingress
```

i si tot funciona bé, el sistema mostrarà un missatge a on la **darrera línia** diu: **`Ingress is enabled`**

c) Configura **Ingress**. Dins de la carpeta **ppv** crea un fitxer de nom **ppv-ingress.yaml**. Dins d'aquest fitxer escriu el següent codi de configuració:

```
apiVersion: networking.k8s.io/v1
kind: Ingress #Especificica el recurs a desplegar: Deployment, Service , Pod, etc.
metadata:
     name: proxy-ppv # Nom del service del recurs tipus ingress a crear
     annotations:
          spec.ingressClassName: "nginx"
spec:
     rules:
     - host: ppv.fjeclot.net #Nom i domini amb el qual es pot accedir des de l'exterior
       http:
            paths:
            - path: /
              pathType: Prefix
              backend:
                   service:
                        name: serv-ppv # Nom del service del qual ha de fer de  reverse-proxy
                        port:
                             number: 4000 # El mateix número que - port dins ppv-service.yaml
```

NOTES:

- *Això és un fitxer YAML i les sagnies són importants. Si copies i enganxes d'un PDF hauràs de comprovar els espais.*
- *Has d'assegurar-te que els comentaris queden al mateix lloc a on es veuen si copies i enganxes d'un PDF.*

Amb aquesta configuració, crearem un **reverse-proxy** que permet que qualsevol accés a **http://ppv.fjeclot.net:30000** és redireccioni al port **4000/tcp** del Service **serv-ppv** que autòmaticament es redirigirà al port **80/tcp** del qualsevol dels **Pods** de l'aplicació.

d) A continuació exexcuta dins de **ppv** l'ordre:

```
microk8s kubectl  create  -f  ppv-ingress.yaml
```

i el sistema hauria de mostrar el missatge: **`ingress.networking.k8s.io/proxy-ppv created`**

e) Comprova que l'estat del **reverse-proxy**. Executa:

```
microk8s kubectl get ingress
```

i el resultat hauria de ser:

```
NAME        CLASS    HOSTS              ADDRESS     PORTS   AGE
proxy-ppv   public   ppv.fjeclot.net    127.0.0.1   80      5m56s
```

f) Troba l'adreça IP de la màquina virtual **node1**. Executa: **`ip -4 -br add show eth1`**

3.6- Accedint a l'aplicació

a) Dins de la teva màquina host:

- Si treballes amb Windows:
  - Obre amb **notepad** el fitxer **C:\Windows\System32\drivers\etc\hosts**
  - Afegeix al final del fitxer l'entrada la relació entre IP i nom de màquina.
  - Exemple:
    - **192.168.1.47 ppv.fjeclot.net**
      S'ha de canviar **192.168.1.47** amb l'adreça obtinguda a l'apartat **f)**.
- Si treballes amb Linux:
  - Obre amb **nano** el fitxer **/etc/hosts**
  - Afegeix al final del fitxer l'entrada la relació entre IP i nom de màquina.
  - Exemple:
    - **192.168.1.47 ppv.fjeclot.net**
      S'ha de canviar **192.168.1.47** amb l'adreça obtinguda a l'apartat **f)**.

b) Comprova que pots fer **ping** de la màquina host a la màquina virtual **node1**.

c) Des del navegador de la teva màquina host accedeix a **http://ppv.fjeclot.net:30000** i comprova que pots accedir a l'aplicació.

d) Des d'una altra màquina física, modifica el fitxer hosts perquè apunti a la teva màquina virtual **node1**. Comprova que també es pot fer una connexió a **http:/ppv.clotfje.net:30000**.

4- Escalant manualment el desplegament

a) Torna a comprovar el nombre de **Pods** del desplegament que vas crear a l'aparat **3.3.3** executant dins de **ppv** la següent ordre:

```
microk8s kubectl get pods
```

i comprova novament que tens **3 Pods** en marxa.

b) Ara, fes que el deployment **depl-ppv** que vas crear a l'aparat **3.3.3** passi a treballar amb **10 Pods** executant dins de **ppv**:

```
micro8ks kubectl scale --replicas=10 deployment/depl-ppv
```

i si tot va bé, el sistema mostrarà el missatge: `deployment.apps/depl-ppv scaled`

c) Comprova novament el nombre de **Pods** del desplegament i que ara tens **10 Pods**.

d) Passa a treballar amb **5 Pods** executant:

```
micro8ks kubectl scale --replicas=5 deployment/depl-ppv
```

i comprova novament que el sistema indica que s'ha pogut escalar el desplegament i que ara treballes amb **5 Pods**.

5- Comprovació de la distribució dels Pods entre els nodes del cluster

a) Fes que el deployment **depl-ppv** que vas crear a l'aparat **3.3.3** passi a treballar amb **15 Pods** executant dins de **ppv**:

```
micro8ks kubectl scale --replicas=15 deployment/depl-ppv
```

b) Comprova el **Pods** del desplegament executant:

```
microk8s kubectl get pods -o wide
```

i troba el nombre de **Pods** que s'executen en el **node1**, **node2** i **node3**.

6- Comprovació de l'inici automàtic de nous Pods quan un altre deixa de funcionar

a) Fes que el deployment **depl-ppv** que vas crear a l'aparat **3.3.3** passi a treballar amb **10 Pods** executant dins de **ppv**:

```
micro8ks kubectl scale --replicas=10 deployment/depl-ppv
```

b) Mostra el **Pods** del desplegament executant: `microk8s kubectl get pods -o wide`

c) Escull el primet **Pod** i troba el seu nom.

d) Destrueix el Pod executant: `microk8s kubectl delete pod <nom_complet_del_pod> --now`

e) Torna a mostrar la llista de **Pods** i comprova que:

- El **Pod** escullit ja no existeix
- Encara hi ha **10 Pods**
- S'ha creat un nou **Pod** que abans no existia (a la columna AGE tindrà només segons de vida).
- S'ha creat el nou **Pod** en el mateix node que el **Pod** destruït.

Lliurament de l'activitat

- Des del sistema física executa **vagrant status** i mostra que les 3 màquines virtuals en marxa.
- Accedeix a **node1** (el control planer) i mostra que els **3 nodes** del cluster funcionen.
- Des de **node1** mostra el **deployment** creat**.**
- Des de **node1** mostra el **service** creat.
- Des de **node1** l'**Ingress** creat.
- Accedeix a l'aplicació web des de la màquina física i comprova que funciona correctament.
- Posa en marxa **12 Pods** de l'aplicació i mostra'ls.
- Mostra els **nodes** dins dels quals s'executen els **Pods** i comprova que estan distribuïts per igual.
- Destrueix un **Pod** i comprova que es crea un de nou en el mateix **node**.
- Data límit per obtenir el 100% de la nota: **dilluns 9-2-26** a les **19.10**. (posteriorment és el 50%).